Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Nikolay Savchuk
Lars Edvin Jonsson Hoff
Anders Brunsberg Mariendal

Webapp for visualisering av CIE-funksjoner

Bacheloroppgave i BIDATA/BPROG
Veileder: Ivar Farup
Mai 2024

NTNU
Kunnskap for en bedre verden

Nikolay Savchuk
Lars Edvin Jonsson Hoff
Anders Brunsberg Mariendal

Webapp for visualisering av CIE-funksjoner

Bacheloroppgave i BIDATA/BPROG
Veileder: Ivar Farup
Mai 2024

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU

Kunnskap for en bedre verden

Abstract

This bachelor's thesis focuses on the development of a web application for visualizing CIE functions. CIE functions are standardized mathematical functions that describe how an average person perceives colors based on various parameters such as age. The application is being developed as an extension for an existing application "CIE Functions,"created in collaboration between NTNU and the University of South-Eastern Norway.

The main goal of the project is to develop a web application that offers the same functionalities as the original application. The application should support calculations of various color matching functions for standard observers and represent the results in an easily understandable way. The project includes testing different frameworks to find the most suitable solution, as well as developing prototypes and discussion of the choice of technology.

The development has been carried out using Python for backend calculations and API design, and modern web technologies such as React and TypeScript for the frontend architecture. The project has also had a strong focus on user-centered design and a responsive interface to ensure accessibility and user-friendliness.

Through this work, we have gained a deeper understanding of both old and new forms of web technologies, as well as the importance of good development practices such as documentation and testing. The web application we have developed will facilitate research in color vision and colorimetry, making it more accessible.

Sammendrag

Denne bacheloroppgaven fokuserer på utviklingen av en webapplikasjon for visualisering av CIE-funksjoner. CIE-funksjoner er standardiserte matematiske funksjoner som beskriver hvordan et gjennomsnittlig menneske oppfatter farger basert på forskjellige parametere som for eksempel alder. Applikasjonen lages som en utvidelse for den eksisterende programvaren 'CIE Functions', utviklet i samarbeid mellom NTNU og Universitetet i Sørøst-Norge.

Hovedmålet med prosjektet er å utvikle en webapplikasjon som tilbyr de samme funksjonalitetene som den opprinnelige applikasjonen. Applikasjonen skal støtte beregninger av forskjellige fargematchfunksjoner for standardobservatører og representere resultatene på en lett forståelig måte. Prosjektet inkluderer testing av forskjellige rammeverk for å finne den mest egnede løsningen, samt utvikling av prototyper og diskusjon rundt valg av teknologi.

Utviklingen har blitt gjennomført ved bruk av Python for backend-beregninger og API-design, og moderne webteknologier som React og TypeScript for frontend-arkitektur. Prosjektet har også hatt et sterkt fokus på brukersentrert design og et responsivt grensesnitt for å sikre tilgjengelighet og brukervennlighet.

Gjennom dette arbeidet har vi fått en dypere forståelse av både gamle og nye webteknologier, samt viktigheten av god utviklingspraksis som dokumentasjon og testing. Webapplikasjonen vi har utviklet vil lette forskningen innen fargesyn og kolorimetri og gjøre den mer tilgjengelig.

Forord

Bacheloppgaven er utarbedid av to dataingeniører studenter og en programmerings student ved NTNU i Gjøvik. Denne bacheloroppgaven har blitt gjennomført i samarbeid med Fargelabben ved Institutt for Datateknologi og Informatikk (IDI) ved NTNU, med veiledning av Ivar Farup og Jan Henrik Wold. Prosjektet har hatt som mål å utvikle en webapplikasjon for visualisering av fargematchfunksjoner. Prosjektet tar utgangspunkt i et allerede eksisterende Python-basert programvare.

Vi ønsker å takke våre veiledere, Ivar Farup og Jan H. Wold, for deres uvurderlige støtte og veiledning gjennom hele prosjektet. Deres innsikt og råd om programmering og kolorimetri har vært avgjørende for prosjektet, og gjort det mulig for oss å navigere de betydelige utfordringene vi har møtt.

Gjøvik, Mai 2024
Nikolay Savchuk, Lars Edvin Jonsson Hoff og Anders Brunsberg Mariendal.

Innhold

Figurer

Tabeller

Kodelister

Akronymer

API Application Programming Interface. 16–18, 36, 41, 47, 54, 62, 65, 67, 68, 74, 92–94, 98, 101

ASGI Asynchronous Server Gateway Interface. 19, 39, 57, 58, 77

CIE Commission internationale de l'eclairage. 2, 10

CSS Cascading Style Sheets. 13, 14, 16, 66, 84, 85

CSSOM CSS Object Model. 14

DOM Document Object Model. 14, 36, 37, 46, 47

FFI Foreign Function Interface. 47

HATEOAS Hypermedia As The Engine Of Application State. 17

HCI Human Computer Interaction. 20

HMR Hot Module Replacement. 44

HTML HyperText Markup Language. 12–14, 16, 59, 61, 76, 85

HTTP HyperText Transfer Protocol. 17, 18, 53, 87

IDE Integrated Development Environment. 39

IDI Institutt for datateknologi og informatikk. 1, 5

JS JavaScript. 13, 14, 16, 18, 32, 36, 46–48, 71, 74, 85

REST Representation State Transfer. 17, 29, 53, 54, 59, 75, 76

SOAP Simple Object Access Protocol. 17, 18

SPA Single Page Application. 16

URL Uniform Resource Locator. xiv, 17, 52–54, 59, 67, 68, 73, 74, 76–79, 94

USN Universitetet i Sørøst-Norge. 1

Wasm WebAssembly. 18, 32, 35, 36, 46, 47, 49

WSGI Web Server Gateway Interface. 19, 35, 55–57

XML Extensible Markup Language. 18

Kapittel 1

Introduksjon

1.1 Bakgrunn

Fargelabben ved IDI (Institutt for datateknologi og informatikk)[1] hos NTNU Gjøvik driver med en rekke forskningsprosjekter innenfor fargelære og -styring, med spesiell engasjement innenfor flere felt som datasyn, medisin og bevaring av bilder. Blant annet har de også forskning innenfor fargesyn og hvordan ulike mennesker tar opp farger forskjellig - hvor det er da satt spesiell fokus på å bruke standardiserte matematiske funksjoner (også kalt 'fargematchfunksjoner') for å uttrykke noen sitt fargesyn, gitt relevante parametere som deres alder og feltstørrelse av eksperimenter.

Til denne hensikt ble det produsert en refereanseprogramvare kalt 'CIE Functions' [1] (se figur 1.1) i et bredt samarbeid mellom NTNU og USN (Universitetet i Sørøst-Norge), med bakgrunn i den tekniske komiteen 'CIE TC1-97'[2]. Applikasjonen klarer å beregne en rekke fargefunksjoner for et gjennomsnittlig individ med gitte parametere (også kalt en 'standardobservatør'), og representere resultatene i ulike former som gjør det enkelt å referere til. Applikasjonen er utviklet gjennom Python med åpen kildekode, og et brukergrensesnitt innenfor PyQt[3] biblioteket.

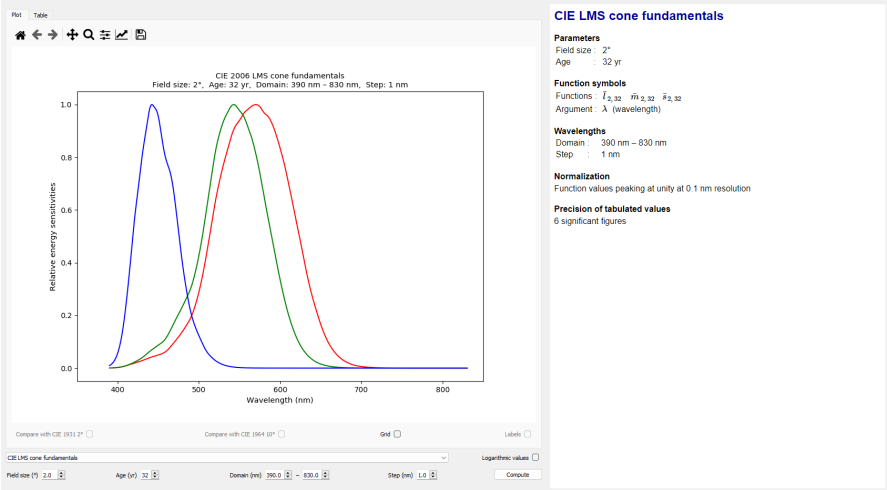
1.1.1 Oppgavedefinisjon

Vår oppgave er å ta for oss denne programvaren, og utvikle en webapplikasjon med de samme funksjonalitene som eksisterer i basisprogrammet. Dette inkluderer visuelle funksjonaliteter, hvor den skal inneholde ett brukergrensesnitt som virker likt som det i programmet. I og med at koden for utregning av fargematchfunksjonene allerede eksisterer i form av Python kode, så er det et viktig punkt å lage løsningen med basis i Python.

[1] `https://www.ntnu.edu/colourlab/view/about`, hentet 19.05.2024

[2] https://cie.co.at/technicalcommittees/age-and-field-size-parameterised-calculation-cone-fundamental-based-spectral, hentet 19.05.2024

[3] http://www.riverbankcomputing.com/software/pyqt/, hentet 19.05.2024



Figur 1.1: Skjermbilde av basisapplikasjonen.

Det ønskes også at det testes forskjellige rammeverk for programmet for å finne det som er best egnet til applikasjonen. Dette innebærer tidlig utvikling av prototyper for ulike rammeverk, og deretter testing med diskusjon for å velge den som passer best. Løsningen vi anser som best utvikles videre på for å oppnå alle funksjonaliteter påkrevd i prosjektet.

1.1.2 Avgrensning

Prosjektets rammeverktesting skal fokusere på rammeverket selv, og ikke inkludere testing eller diskusjon av hvordan maskinvare og annet programvare brukt ville ha påvirket applikasjonens ytelse. Alle tester gjøres derfor på samme maskinvare for å forsikre seg konsise forskningsresultater. Maskinvare er også utenfor rammet av oppgaven, og er avgrenset for oss.

Prosjektet skal ikke inkludere nye funksjoner som ikke allerede finnes i basisapplikasjonen. Hvis det oppstår behov for ekstra funksjonalitet, må dette avtales direkte mellom oppdragsgiver og gruppen. I slike tilfeller skal den nye funksjonaliteten også implementeres i basisapplikasjonen, slik at den oppdateres til å inkludere den samme funksjonaliteten.

1.2 Målgrupper & Mål

1.2.1 Målgrupper

Målgruppen for dette prosjektet blir fargesynsforskere rundt i hele verdenen. I og med at basisapplikasjonen ble laget med basis i en teknisk komité for den internasjonale kommisjon for belysning (CIE)[4], så kan det antas at dette prosjektets

[4]https://cie.co.at/

applikasjon vil gjelde den samme målgruppa - og derfor være laget spesifikt med dem i tankene. Denne gruppa vil også inkludere Fargelabben ved NTNU, for de inngår i gruppen 'fargesynsforskere'.

Rapporten, derimot, er spesifikt skrevet for Fargelabben og vår oppdragsgiver. Gjennom denne rapporten ønsker vi å formidle alt arbeidet som har gått inn i applikasjonen på en måte som både forteller om og forklarer prosjektet til lekfolk. Vi erkjenner at programmering og fargematematikk ikke nødvendigvis tilhører samme domene, så rapporten er skrevet med dette i tankene.

1.2.2 Mål

Vi ønsker å oppnå tre spesifikke mål med denne oppgaven:

Resultatsmål

Resultatet av dette prosjektet blir en ferdigutviklet webapplikasjon som kan hostes på en egen webserver. Webapplikasjonen skal inneholde alle av de samme funksjonalitetene som eksisterer i basisapplikasjonen, med et likt brukergrensesnitt men med videre adapsjon for responsivt design.

Effektsmål

Effekten prosjektet utgjør vil gjøre det enklere for fargesynsforskere å utføre beregninger relatert til fargesyn, og forhåpentligvis skape mer interesse innenfor forskningsemnet med å tilby enkel tilgang til beregningsfunksjonene. Den vil også gjøre det mer komfortabelt og brukbart for forskere som ikke kan bruke basisapplikasjonen med å tilby et alternativ.

Læringsmål

Vi har erklært følgende læringsmål som punkter av spesiell fokus med tanke på utdypelse og å lære mer om dem:

- Utvikling, testing og vurdering av både tradisjonelle nettrammeverk - men også nyere teknologier.
- Utvikling og diskusjon rundt relevante klientrammeverk med tanke på brukersentrert design.
- Profesjonell etikette angående programvareutvikling gjennom flere relevante metoder (praktisk bruk av arbeidsmetodikker, dokumentasjon, osv.).
- Mer erfaring innen gruppearbeid, hvordan vi skal kunne samarbeide for å skape et større prosjekt.
- Hvordan vi skal formidle våre funn i en rapport, og presentasjon, gjennom drøftinger og utdypelser.

1.3 Gruppe

Vår gruppe består av tre studenter; Nikolay Savchuk (BPROG[5]), Lars Edvin Jonsson Hoff (BIDATA[6]), Anders Brunsberg Mariendal (BIDATA) - og er derfor en blandet gruppe med folk fra forskjellige grader. Allikevel, har vi en rekke unike kompetanser som gjør oppgaven godt egnet for oss.

For våre dataingeniørstudenter, så tilbyr de stor kompetanse og innsikt i optimalisering av dataflyt og integritet av systemer, samtidig som de bringer frem en dypere forståelse av underliggende systemer, nettverk og databehandling fra deres fag.

Programmeringsstudenten har også ekstra erfaring innenfor Python. Inkludert fagene han har hatt over graden, så har han også jobbet som en læringsassistent for et fag innenfor kunstig intelligens for Python - og hatt ett større prosjekt som brukte Python for trening og testing av maskinlæringsmodeller.

Dette gjør at gruppen er godt rustet for å kunne designe og implementere en fullstendig løsning som vil oppfylle oppdragsgivers krav.

1.4 Rammer

1.4.1 Tidsrammer

- Kodeutviklingen, det vil si utviklingen av alle krav og funksjonaliteter, skal være ferdig innen 18. april 2024. Denne fristen gjelder kun for de essensielle funksjonalitetene. Endringer som følge av videre testing og/eller finpussing kan fortsatt utføres frem til den endelige tidsfristen.
- Den endelige tidsfristen for hele prosjektet, som inkluderer en ferdigutviklet programvare og en tilhørende prosjektrapport, er satt til 21. mai 2024.

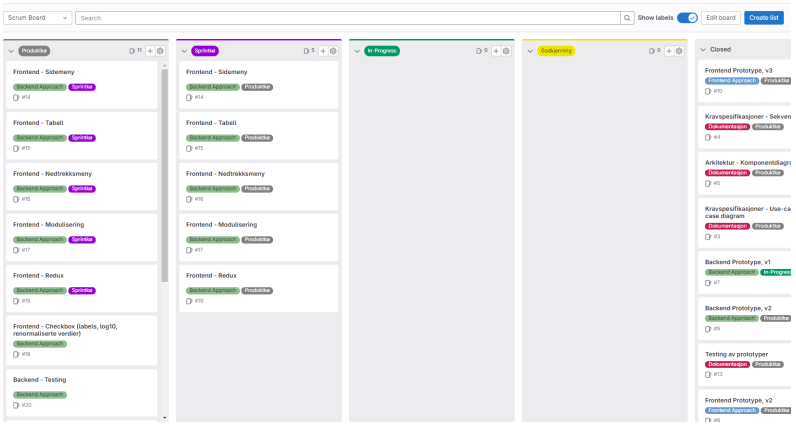
1.4.2 Teknologiske rammer

- Webapplikasjonen skal støtte `HTML5`, og være tilgjengelig gjennom de fleste populære nettlesere.
- I likhet med basisapplikasjonen skal webapplikasjonen ha åpen kildekode og falle under samme type lisens.

1.4.3 Andre rammer

- Det er bestemt at gruppen skal følge arbeidsmetodikken 'scrum', med en sprintperiode på to uker. Det skal brukes et scrumboard for å holde styr av oppgavene for en sprint og eventuelle oppdateringer til produktkø for prosjektet. Scrumboardet lages gjennom GitLab (1.2), hvor alle medlemene av gruppa er ansvarlig for å oppdatere oppgaver de selv jobber med.

[5]https://www.ntnu.no/studier/bprog, hentet 19.05.2024
[6]https://www.ntnu.no/studier/bidata/cybersikkerhet, hentet 19.05.2024



Figur 1.2: Utdrag av relevant scrumboard rett etter at en sprint hadde blitt planlagt.
Inneholder tabell for produktkø (1. fra venstre), sprintkø (2. fra venstre) og progresjon av oppgave (enten "in-progress", venter godkjenning", eller fullført/"closed").

- Det skal holdes et møte hver onsdag, ved klokken 11:15 til 12:00. Møtet skal holdes mellom alle gruppemedlemmer, sammen med produkteier og veileder. Møtet holdes fysisk på et angitt kontor ved NTNU Gjøvik, med mindre noe annet er bestemt.
    - Møter med produkteier og veileder kan avlyses dersom det ikke er noe nytt å rapportere. Dersom møter avlyses flere uker på rad har gruppen ansvar for å rapportere status for prosjekt til både produkteier og veileder gjennom epost.
- Møter mellom gruppemedlemmer skjer ved avtaler, og kan holdes både digitalt og fysisk avhengig av hva som avtales.
- Alle medlemmer av gruppa er utviklere for programvare. Dette betyr at de er selv ansvarlig for å lage effektiv og korrektkode, samt alle ansvar tilknyttet deres arbeid (testing, dokumentasjon, osv.).

1.5 Øvrige roller

- Vår oppdragsgiver er Jan Henrik Wold, som representerer Fargelabbens interesser ved IDI. Oppdragsgiver er ansvarlig for å representere bedriften, og å kommunisere deres tanker angående utviklingen av programvaren.
- Vår veileder er Ivar Farup. Han er ansvarlig for å gi akademisk veiledning til programvare og rapport, sammen med støtte angående administrative aspekter og kritisk tilbakemelding.
- Rollen som Scrum master har rullert fra sprint til sprint. Siden Scrum masterens og gruppeledreens ansvarsområder har en del overlapping fant vi det

hensiktsmessig å slå sammen disse to rollene. Scrum masteren er ansvarlig for å at alle Scrum-hendelser skjer innenfor de angitte tidsrammene, samt samarbeid innad i teamet.

- Utviklerrollen har blitt fylt av alle gruppemedlemene gjennom alle sprintene. Utviklerene er ansvarlige for å implementere oppgavene satt i sprintloggen uke til uke. Dette skal gjøres med ren og effektiv koding. Utviklerene er også ansvarlige for dokumentasjon og testing av skrevet kode.
- Redaktørrollen er fylt av Nikolay Savchuk. Redaktøren er ansvarlig for å overse dokumentasjonen av prosjektet. Dette inkluderer møtereferater, fremgangsrapporter og dokumentasjon nødvendig for rapporten selv. Han har også så ansvaret for levering av oppgaven og etterfølgende fremføring.

1.6 Rapportoppsett

1.6.1 Oppsett av rapport

Rapporten er oppsatt inni følgende kapitler:

1. **Kapittel 1 - Introduksjon** dekker bakgrunnen for prosjektet, hva oppgaven for prosjektet er, og eventuelle avgrensninger, mål og rammer. Den skal også introdusere gruppen og øvrige roller for prosjektet.
2. **Kapittel 2 - Teori** dekker den teoretiske bakgrunnen som er relevant for prosjektet. Dette handler kort om fargematchfunksjoner og den faglige teorien.
3. **Kapittel 3 - Kravspesifikasjoner** dokumenterer alle kravene vi har laget for prosjektet. Dette inkluderer funksjonelle, ikke-funksjonelle, operasjonelle og produktkø. Det har også blitt laget diagrammer.
4. **Kapittel 4 - Design & Arkitektur** tar for seg designet av det grafiske brukergrensesnittet, og introduksjon av arkitekturer. Det skrives om to arkitekturer, begge som potensielle løsninger å utføre oppgaven på.
5. **Kapittel 5 - Teknologier** omhandler teknologier som skal brukes i prosjektet, både de som var planlagt og de som ble tatt i bruk underveis. Disse nevnes etter at arkitekturen relevant for oppgaven er beskrevet.
6. **Kapittel 6 - Prosess** dekker hele utviklingsprosessen, organisert i sprinter. Det første delkapitlet omhandler diskusjonen og beslutningen mellom de to arkitekturene som tidligere er introdusert i rapporten. De påfølgende delkapitlene beskriver videreutviklingen av den valgte arkitekturen.
7. **Kapittel 7 - Implementasjon** fokuserer på det siste produktet etter utviklingsprosessen. Her utdypes ulike implementasjonsdetaljer. Spesifikt skal det diskuteres koden som ble utviklet og tankegangen bak hvorfor løsninger ble implementert på den måten de ble.
8. **Kapittel 8 - Testing & kvalitetssikring** beskriver hvordan programvaren sikrer kodekvalitet, gjennomfører testing, og bruker lisensiering.
9. **Kapittel 9 - Utrulling & Installasjon** beskriver hvordan programvaren ble iverksatt for tjenestegjøring på en webserver, og kort om hvordan den skal installeres.

10. **Kapittel 10 - Diskusjon & Konklusjon** inneholder en helhetlig diskusjon og drøfting av prosjektet. Det skal spesifikt utdypes om våre mål, bruken av KI, kritikk om det vi har laget, hvilket videre opplegg finnes det, og deretter en evaluering av vårt arbeid. Kapittelet slutter med en konklusjon.

1.6.2 Terminologi og praksis

Denne seksjonen inkluderer klargjøringer av terminologi brukt i rapporten, sammen med avklaringer angående bruk av font, styles og annet i rapporten.

- Alle fremtidge referanser til 'basisapplikasjon', 'basisprogrammet' (og lignende) refererer til 'CIE Functions' [1] programmet, som vårt prosjekt er basert på. Vårt prosjekt er en utvidelse av dette programmet. For denne grunn, gjenbruk av kode fra dette programmet er lov for å oppbevare konsistenthet med oppgaven.
- Det skal benyttes seg av `kodefont` for ting som typer og klasser. I tillegg kan det også benyttes for filnavn og variabelnavn dersom det passer.

Sammen med dette, anbefaler vi å konsultere vedlegg A for definisjoner om ord og uttrykk brukt i rapporten.

Kapittel 2

Teori

2.1 Fargematchfunksjoner

Når vi ønsker å måle noe, betyr det å fastlegge det på en kvantitativ måte slik at målingen kan senere brukes i sammenligninger og/eller utregninger [2]. Vi har flere typer målinger; meter for lengde, sekund for tid, og gram for vekt er kun noen allmennkjente eksempler. Et eksempel på et område hvor måling blir mer komplisert derimot, er innen fargevitenskap. Det er mulig å måle den kvantitative siden av lysbølger gjennom et spektrofotometer[1] for å se bølgelengdens frekvens, men denne målingen tar ikke hensyn til hvordan vi mennesker oppfatter fargen. Om vi skal måle denne siden av farger, bruker vi heller et visuell kolorimeter med standardiserte lyskilder, hvor brukeren justerer på en gitt farge inntil den er lik en annen referansefarge i verktøyet [3].

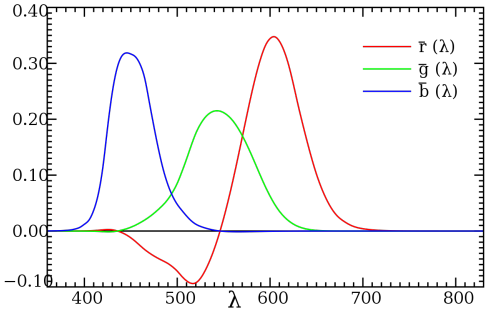
Denne typen tester danner grunnlaget for 'fargematchfunksjoner'. Disse funksjonene er basert på tappene i øynene våre og hvordan hjernen tolker signalene de sender. Tappene er reseptorer som gjør det mulig for oss å oppfatte kombinasjoner av rødt, grønt og blått lys, og dermed se hele spekteret av synlig lys. For å oppfatte farge, stimuleres disse tappene i varierende grad av forskjellige lys. Hvis det fantes en metode for å måle intensiteten for alle tre tappene for bestemte bølgelengder, ville det være mulig å lage en modell av hvordan et individs tapper reagerer under ulike lysforhold.

Dette ble oppnådd på 1930-tallet av forskeren William D. Wright, som utførte et eksperiment med en gruppe standardobservatører. Individer ble utsatt for et delt synsfelt adskilt av en rett linje. I det ene feltet var det en ukjent referansefarge, og i det andre feltet en kombinasjon av justerbare primærfarger. Deltakerne fikk justere intensiteten på primærfargene for å oppnå en matchende farge i begge feltene gjennom additiv fargeblanding. Målet var å justere fargene til begge feltene viste identiske farger ([4], 8.12).

Ved å gjennomføre dette eksperimentet med de samme parameterne på flere

[1] https://snl.no/spektrofotometer, hentet 19.05.2024

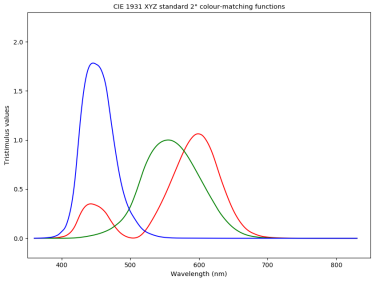
individer, ble det mulig å skape en fargematchfunksjon ved hjelp av tre vektingsfunksjoner for hver referansestimulus som ble testet i eksperimentet. Denne undersøkelsen dannet grunnlaget for den kjente 'CIE 1931' fargematchfunksjonen, som er utgangspunktet for all fargemåling og kolorimetrisk matematikk.



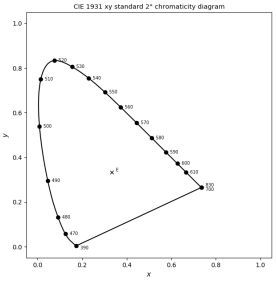
Figur 2.1: CIE 1931 (RGB) fargematchfunksjon, normalisert [5] .

Denne modellen (slik vist i figur 2.1), hadde klart å uttrykke monokromatiske fargestimuli som punkter bestående av r, g, b verdier korresponderende til deres respektive farger. Modellen ble snart videre utviklet til å ikke inkludere negative tall, og ble kjent som standardiseringsfunskjonen CIE 1931 (XYZ) (figur 2.2a) ([4], 8.12) når det ble snart innført av den internasjonale belysningskommisjonen (CIE). De tre verdiene av X, Y, Z ble virtuelle referansestimuli for fargekoordinater i deres fargevektorrom.

Denne modellen ble videre utviklet til å separere luminansen fra fargen, til å kun sitte igjen med kromasiteten - hvor sluttresultatet ble to punkter kjent som (x, y) - videre dannet det vi kaller et kromatisitetsdiagram for (x, y), karakterisert av dets hestesko form (figur 2.2b) ([4], 8.12).



(a) CIE 1931 (XYZ) fargematchfunksjon.



(b) CIE 1931 (x, y) kromatisitetsdiagram.

Figur 2.2: CIE 1931
Begge figurer er hentet fra basisprogrammet.

Bruken av disse modellene gjorde det mulig å representere farger som koordinatverdier i matematikk, noe som videre dannet grunnlaget for bedre represen-

tasjon av fargestimuli i teknologi. Et eksempel på deres betydning er ICC-profiler; spesielle profiler for enheter med kameraer eller skjermer som sikrer korrekt fargegjengivelse mellom dem[2]

2.2 Faglig teori

2.2.1 Webapplikasjon

En webapplikasjon kan beskrives som 'et program for nettleseren'. Mens vanlige applikasjoner er spesifikt laget for å kjøre individuelt på en enkelt datamaskin, er webapplikasjoner utviklet for å kjøre i nettlesere. De er avhengige av både en klientmaskin for å kjøre programmet og servere for å levere funksjonaliteter. Med andre ord, en webapplikasjon krever ikke bare brukerens maskin, men også minst en servermaskin for å håndtere funksjonaliteten og all interaksjon fra brukeren. Denne modellen er så vanlig at den har fått det tekniske navnet 'klient-tjener modellen' ([6] s. 67, kap. 2). Modellen består av to lag:

- **'Klient'/'Frontend'** - utviklingen av brukerinteraksjon *gjennom* det visuelle.
- **'Tjener'/'Backend'** - utviklingen av arkitektur og funksjonalitet *for* det visuelle.

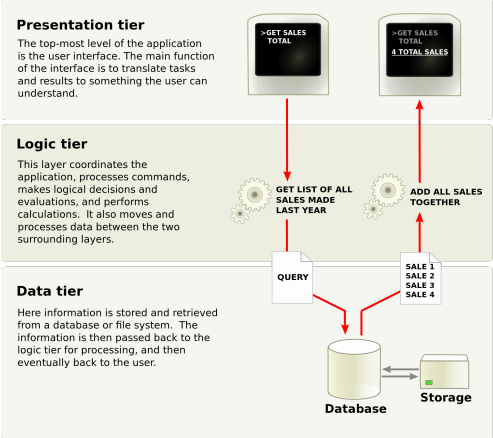
Disse skal dekkes mer i deres respektive kapitler nedenfor.

I tillegg til dette, tas det også opp ulike arkitektursmodeller for applikasjonen. Forskjellige applikasjoner krever forskjellige krav basert på deres påkrevde funksjonaliteter; enkelte må kanskje kunne lagre informasjon over lang tid, mens andre bare må vise frem et enkelt dokument. Gjennom dette ble det utviklet en serie av ulike modeller for hvordan utviklere skulle kunne strukturere deres webapplikasjoner, hvor noen relevante eksempler er:

- **'Tre-lag' arkitekturen (som vist på figur 2.3):** Et tradisjonelt og populært valg hvor webapplikasjonen sin struktur består av tre individuelle lag ([7], kap. 5).
  1. 'Presentasjonslaget' hos brukerens enhet, korresponderende til 'frontend' laget diskutert tidligere.
  2. 'Logikklaget' hos en webserver, korrespondere til hoveddelen av 'backend'.
  3. 'Datalaget' hos en separat database, også ofte omtalt som en del av 'backend'.

  Denne oppdelingen av lag (også kjent som prinsippet 'separasjonen av lag' ([8], s. 2)) er det som gjør arkitekturen et populært valg hos mange utviklere. Med å ha hvert lag uavhengig fra de andre, er det enklere å øke/redusere dets skalerbarhet i et system. Lagene er også strukturert på en måte hvor

[2] `https://www.color.org/faqs.xalter#wh2`, hentet 21.05.2024.



Figur 2.3: Et eksempel av en elementær '3-tier' arkitektur. [10]

brukere aldri snakker direkte med datalaget. De må passere logikklaget, noe som resulterer i en større grad av sikkerhet.

- **'To-lag' arkitekturen:** Et mindre brukt men allikevel populært valg for arkitektur [9]. Denne strukturen er veldig sammenlignbar med arkitekturen forklart over, men bruker kun to lag:

  1. 'Presentasjonslaget' slik beskrevet tidligere.
  2. 'Backend', hvor det kan bestå av en webserver, en database eller begge på samme maskin.

Denne arkitekturen består av færre komponenter, men gir til gjengjeld fordeler som at det er enklere å opprettholde, billigere, og dekker krav hos de fleste tjenester hvor en database og/eller funksjonalitet er ikke viktige aspekter.

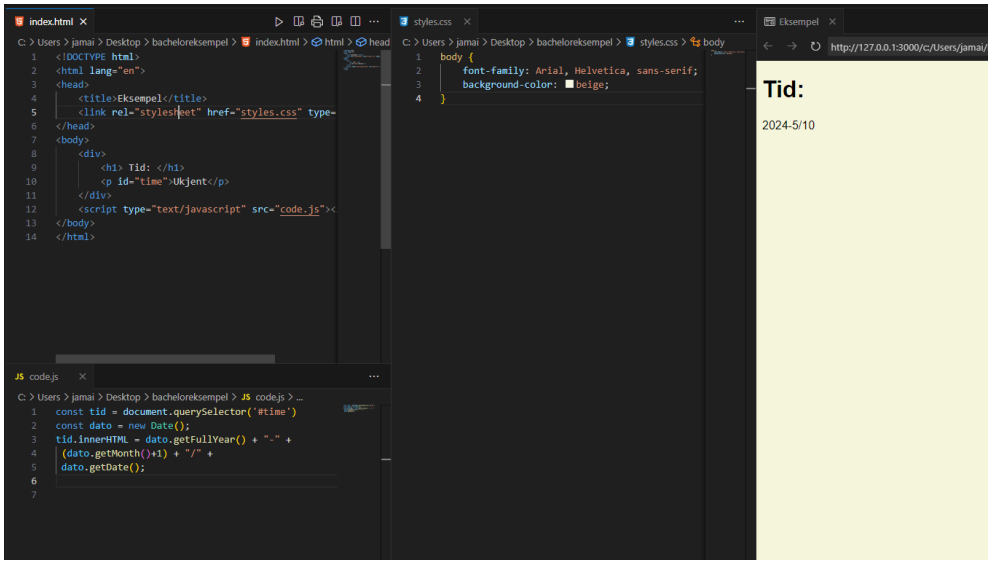
2.2.2 Frontend

Frontendutvikling er aspektet av webutvikling hvor det fokuseres på klienten, primært gjennom 'det visuelle' og 'det interaktive'. Dette gjøres gjennom utviklingen av et brukergrensesnitt som skal tillate en bruker å interagere med logikken til en applikasjon gjennom dets design. For hensikten av å formidle teorien bedre, deles dette kapittelet i to deler - en som forteller de tekniske detaljene, og en annen som går innom design.

Teknisk

Når det gjelder den tekniske siden hos frontendutvikling, snakker vi hovedsakelig om tre tekniske lag som bygger opp hele strukturen av hva en bruker ser og gjør på en webapplikasjon:

1. **Innholdslaget (HTML)**: Den essensielle strukturen og innholdet av en gitt nettside, med alt av tekst og semantisk struktur ([11], s. 20). Hver eneste



Figur 2.4: Enkel eksempel av frontend gjennom alle tre lag.

tittel, bilde og tabell som finnes på en nettside er der takket være bruken av HTML, også kjent som 'HyperText Markup Language' - ett dokumentasjonsspråk som bruker semantiske 'tagger' for å skille elementer fra hverandre.

2. **Presentasjonslaget (CSS)**: Presentasjon av innholdslaget utføres av presentasjonslaget, hvor regler for stil og design bestemmes ([11], s.229-230). Dette inkluderer også visuell struktur - annerledes fra den semantiske strukturen tilgjengelig gjennom HTML. Dette laget kontrolleres av stilspråket CSS ('Cascading Style Sheets').
3. **Interaksjonslaget (JS)**: Hvordan en bruker interagerer med innholdet blir behandlet gjennom interaksjonslaget, som både tilsetter indirekte respons (som dynamiske oppdateringer), og direkte (som umiddelbar tilbakemelding for input) til et ellers statisk innhold ([12], s. 4-5). Ulik de to foregående lagene, dette laget bruker faktisk et programmeringsspråk kalt JavaScript (JS).

Disse tre lagene utgjør grunnlaget for frontend. Innholdslaget er det eneste obligatoriske, mens de to andre kan betraktes som tillegg for å forbedre det visuelle aspektet eller legge til rette for brukerinteraksjon i et ellers statisk dokument. En demonstrasjon av disse lagene kan sees i figur2.4.

- Innholdslaget forteller dokumentet den semantiske strukturen av nettsiden. Det beskriver hvor det for eksempel skal finnes en tittel og et paragraf med innholdet 'ukjent'. Den lenker også opp presentasjons- og interaksjonslaget.
- Presentasjonslaget gir dokumentet reglene at bakgrunnsfargen for `body` taggen skal være beige, og at teksttypen skal være enten Arial, Helvetica, eller sans-serif.
- Interaksjonslaget tilbyr indirekte respons/interaksjon med å dynamisk opp-

datere innholdet av paragrafen fra innhold til å inneholde dagens dato.

En nettleser vil kunne tolke disse tre lagene i en serie av prosesser [13]:

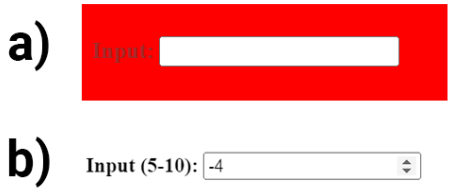
1. **Parsing:**
    - **Lasting av HTML:** Når en nettside, eller URL besøkes,henter nettleseren HTML-dokumenter fra serveren. Dette er startpunktet for å bygge en nettside[14].
    - **Analyse av HTML:** Parsetrinnet begynner med at nettleseren leser den rå HTML-teksten. Her brytes dokumentet opp i ulike elementer som for eksempel `<div>`, `<span>` og `<img>`. Formålet med dette er å forstå strukturen i dokumentet[15].
2. **Document Object Model(DOM):**
    - **Bygging av DOM-treet:** Etter parsing, bygger nettleseren DOM-treet. Dette er en strukturert representasjon av HTML-dokumenter. DOM representerer alle elementene som objekter. Dette gjør det mulig for JavaScript å sammhandle dynamisk med DOM[15].
    - **Scripting:** Når DOM-treet er ferdigstilt, kan JavaScript kjøre for å manipulere DOM ved å legge til, fjerne eller endre elementer og deres egenskaper[16].
3. **Rendering:**
    - **CSS Object Model:** Samtitid som DOM-treet bygges analyserer nettleseren også alle CSS filene tilknyttet HTML-dokumentet for å bygge CSS Object Model(CSSOM). CSSOM representerer alle stilene[15].
    - **Visuell rendering:** Når både DOM og CSSOM er på plass begynner nettleseren å tegne innholdet på skjermen. Dette innebærer beregning av layout får hvert element basert på dens dimensjoner og posisjon, etterfulgt av den faktiske tegningen av elementene[15].
4. **CSS og JavaScript:**
    - **Styling:** CSS-koden brukes for å definere hvordan HTML-elementene skal se ut på skjermen. Dette bidrar til å transformere det strukturerte innholdet for å gjøre det mer estetisk[17].
    - **Interaktivitet:** JavaScript bruker DOM for å legge til interaktivitet i nettsiden. Dette kan inkludere å håndtere brukerinteraksjoner som klikk og tastetrykk, gjøre API-calls og oppdatere brukergrensesnitt dynamisk[14].

Design

Hvordan en frontend er designet spiller vanligvis en stor rolle for brukere; dersom de skal interagere med applikasjonen på en fornuftig måte, så må applikasjonen først være laget fornuftig. På grunn av dette er design et veldig viktig felt i frontendutvikling. Dette er et stort felt med mye teori av alle former, men for hensikten

av vår rapport, så forteller vi heller om tre nøkkelord sentralt for designet: 'universell', 'validert' og 'brukersentrert':

- Det er viktig at designet er passet til å være '**universell**'. Det er viktig at applikasjonen er inklusiv og kan bli brukt av mest mulig folk ([18], s. 1.5), uavhengig av deres livssituasjoner. Det er naturlig at man ønsker at en tjeneste skal nå ut til flest mulig folk, og ikke kun til den gjennomsnittlige personen.
  Noen folk har dårlig syn som gjør det vanskeligere å se figurer med dårlig kontrast og store mengder av tekst - mens andre har vansker med interaksjon, og krever enklere tilgang til ressursene de trenger. Slik inklusivt design inkluderer også 'responsivt' design; prinsippet unikt til webutvikling hvor designet må passe brukerens enhet, uansett størrelse [19].
  Et eksempel på det motsatte av universell design vises frem på punkt a av figur 2.5; dårlig fargekontrast hos elementer gjør det vanskelig for folk med nedsatt synsevne til å se kanter (og derfor hva som er hva på bildet).
- Noe nesten like viktig er at designet er '**validert**'; trygg for både oss og brukeren. Det er ett kjent prinsipp innenfor programvareutvikling at '*en skal aldri stole på en bruker*' ([20], s. 1), for de kan enten ha ondsinnede tanker om å angripe programmet - eller angripe det med et uhell gjennom ukontrollert input. Derfor, har frontend også ansvaret for å forsørge seg å kun gjøre det mulig å ta inn validerte og spesifikke input fra brukeren, samt gi dem kun det nødvendige for å minimalisere fare for uhell.
  Et eksempel på det motsatte av slik kontrollert design er på punkt b) av figur 2.5; ugyldig input (enten med uhell eller med vilje) kan forårsake programfeil hos en applikasjon dersom de blir sendt videre uten validering.
- Til sist, må designet også være naturligvis **sentrert for brukeren**. Det finnes flere prinsipper på hvordan dette oppnås, men her tar vi for oss 'de fem dimensjonene av design', sammen med Don Normans prinsipper for interaksjonsdesign.
  - De fem dimensjonene sier at det skal være enkelt å forstå informasjon gjennom ord, at det brukes visuelle representasjoner for informasjonsformidling, at det finnes fysiske objekter som kan brukes (mus, telefon, tastatur), at det tar akseptabel tid å få informasjon, og sist at brukerens reaksjon på applikasjonen tas i betraktning mens de bruker den. [21].
  - Don Normans prinsipper mener at god synlighet, passende tilbakemeldinger, begrensinger (likt konseptet som 'kontrollert input' definert over), tydelig kartlegging, konsistenthet og tydelig markering av bruk er alle gode prinsipper å ha med seg når man ser for seg mulige interaksjoner innenfor design [22], s. 21.



Figur 2.5: Eksempel på dårlig design; a) viser frem 'eksklusiv' design, og b) viser frem ikke-validert input.

Rammeverk

Oppgjennom årene med webutvikling, har det blitt langt mer vanlig å bruke spesielle rammeverk som tar seg av frontend (istedenfor å bruke en kombinasjon av ren HTML, CSS og JS). Disse rammeverkene gjør det enklere å bygge kompliserte webapplikasjoner med å lage komponentmodeller som kan uttrykkes som nettsider - i tillegg til å lage effektive 'SPA' webapplikasjoner gjennom oppdateringer av kun en side. De tilbyr også en rekke fordeler over vanlig JS, som enklere modulisering og gjenbruk av kode, og greiere tilgang til flere verktøy skapt av arbeidsmiljøet rundt rammeverket. Kjente eksempler er Angular[3], Vue[4] og Ember[5] - som har hver av sine fordeler og ulemper relativ til hverandre.

2.2.3 Backend

Backendutvikling er det andre aspektet av webutvikling. Her fokuseres på alt av applikasjonslogikk, vanligvis med tanke på det som skjer hos frontenden av webapplikasjonen. Dette går langt mer inn på arkitekturen av hele webapplikasjonen, sammen med alle former av funksjonalitet som finnes.

I motsetning til frontend er det ingen fast og obligatorisk teknisk struktur for en backend; den kan skrives i flere ulike programmeringsspråk, hver på sine egne metoder og standarder. Vi tenker heller at dets tekniske struktur avhenger av funksjonalitene den må kunne dekke avhengig av strukturen til applikasjonen. I noen tilfeller kreves det bruk av en database, mens i andre situasjoner kreves det bruk av flere instanser av backend for lastbalansering.

Allikevel, finnes det en vanlig struktur for backend (som applikasjonslogikk) i form av det som kalles en 'API'.

Application Programming Interface

Som diskutert tidligere ved 2.2.1, ble backend beskrevet som '*utviklingen av funksjonalitet for det funksjonelle*' - noe som kan betyr at backendprogrammet vil utføre gitte aksjoner fra frontendprogrammet gjennom en brukers instruksjoner. Denne

[3]https://angular.io/, besøkt 19.05.2024
[4]https://vuejs.org/, besøkt 19.05.2024
[5]https://emberjs.com/, besøkt 19.05.2024

formen for kommunikasjon mellom to datasystemer danner grunnlaget for det vi kaller for en 'API'.

Et programmeringsgrensesnitt (kjent som 'Application Programming Interface' på engelsk) er i hovedsak et grensesnitt for kommunikasjon mellom to applikasjoner ([23]). På samme måte som at vi sier at brukergrensesnittet er hvordan vi som mennesker kommuniserer med data gjennom visuelle representasjoner - så er en API hvordan datamaskiner snakker med hverandre gjennom visse representasjoner av data.

Det finnes flere måter å konstruere et 'API' på. Man kan selvfølgelig lage en intern representasjon spesifikt for egne oppgaver, men det er også mulig å følge visse arkitekturer som kan være mer gunstige basert på kravene man har. Noen eksempler på slike arkitekturer er:

- **REST ('Representation State Transfer')**: Arkitekturstil som baserer seg på seks designprinsipper av Roy Fielding ([24], kap. 5). Stilen er kjent for å være enkel å implementere, har god bruk for klient gjennom `HTTP` metoder, og for fordelene hvert prinsipp tilbyr. For at en server skal følge arkitekturen (også kalt å 'være RESTful'), så må disse seks prinsippene følges:
  1. Separasjon av klient og server gjennom forskjellig logikk. Det skal implementeres en separasjon av bekymringsområder, hvor det som gis til klienten er kun det klienten skal bekymre seg for - og server har kun det *den* må bekymre seg for ([24], 5.1.2).
  2. Server lagrer ikke tidligere laget informasjon fra klientforespørsel (verken resultat av forespørsel eller informasjon om forespørsel).Det er tilstandsløs for å forbedre skalerbarhet og sikkerhet ([24], 5.1.3).
  3. Data hentet fra server må markeres enten implisitt eller eksplitt om det kan settes i klientens egen cache for å øke ytelse og minimalisere mengde av forespørsler fra en gitt bruker ([24], 5.1.4).
  4. Hele applikasjonen må være utviklet på en måte hvor komponenter ikke vet hvem de skal kommunisere med konkret - så derfor, må de utføre deres logikk uavhengig av hvem som skal bruke det ([24], 5.1.6).
  5. Dersom serveren sender ut kjørbar kode med hensikt om å øke funksjonalitet for tjenesten, så må koden kunne kjøre hos klienten ved mottakelse av ressurs. Dette prinsippet er valgfritt ([24], 5.1.7).
  6. Alle unike ressurser har en representasjon tilgjengelig på en unik URL ('Uniform Resource Locator') som kan brukes til å identifisere kun den gitte ressursen. Gjennom denne URL'en, så må en bruker kunne manipulere ressursen gjennom bruk av HTTP med riktig semantisk metode. En respons må også inneholde metainformasjon om forespørselen og dets status. Til slutt, må informasjonen en klient får fra ressursen være nok til å traversere og finne alle ressurser uten tidligere kjennskap til dets arkitektur ('HATEOAS') ([24], kap. 5.1.5).
- **SOAP (Simple Object Access Protocol):** SOAP er en annen arkitekturstil

som baserer seg på bruken av en meldingsprotokoll av samme navn[6]. Stilen baserer seg på tre komponenter; en server med SOAP, et register av alle servere, og en klient som bruker registeret til å finne en ønsket server. Klienten binderseg til serveren for en dialog hvor eventuelle kommunikasjoner foregår gjennom XML og HTTP ([25], kap. 2.1). SOAP har fordelen med at den har god sikkerhet med bedre støtte fra webstandarder ([26], s. 14).

- **GraphQL**: En egen arkitektur basert på sitt eget språk og kjøring[7]. Ideen er at brukere kan lage svært spesifikke forespørsler på eksakt hva de ønsker fra en tjeneste (gjennom en melding skrevet i språket). En API som kjører på GraphQL skal da kunne ta inn forespørselen, og gi kun det brukeren ønsket. Den har fordelene av å være enkel å bruke med å kun trenge en forespørsel fra en klient for å få alt av informasjon en ønsker, og har god bakgrunn som et prosjekt utviklet av Facebook i 2012.

2.2.4 WebAssembly

Det ble tidligere skrevet i rapporten at det er tradisjonelt å ha JavaScript som "interaksjonslagetfor en frontend. Dette skyldes en enkel grunn: Det har lenge vært det eneste programmeringsspråket en kunne ha i frontend. Dette er på grunn av nettstandarder og hva mesteparten av nettlesere støttet. For de fleste så var dette heller ikke nødvendigvis et problem; JavaScript var laget spesifikt for webutvikling, og tok nytte av mange innovasjoner som Chrome V8[8]. Det var kun i 2019, hvor Wasm (WebAssembly) ble introdusert som en offisiell standard av W3C [27].

WebAssembly er inspirert av programmeringsspråket Assembly, og er selv også et 'lavt nivå' språk spesifikt utviklet med tanke på frontend for webapplikasjoner[9].

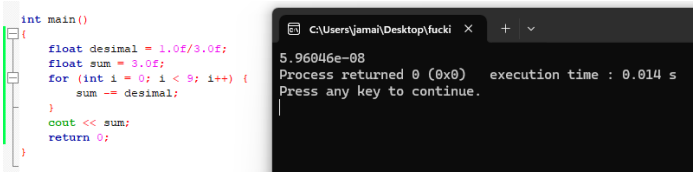
Wasm er ikke et språk man skriver man vanligvis skriver direkte, men heller et språk andre 'høynivå' språk blir kompilert til. Det åpner teoretisk sett for muligheten å kjøre flere språk gjennom nettleseren sammen med deres eventuelle fordeler og ulemper. Selv om teknologien er nyere og mindre brukt enn JS, så finnes det forskning som indikerer at Wasm er mer energieffektiv [28] og raskere enn det tradisjonelle valget for visse applikasjoner. Mange mener allikevel at begge burde brukes sammen for å oppnå full nytelse av hva en nettleser har å tilby. Et kjent eksempel på en kompilator for Wasm er Emscripten[10], som har evnen til å kompilere C/C++ med støtte for flere APIer som `SDL` og `pthreads`.

2.2.5 Beregningspresisjon

Noe viktig å vite om til denne oppgaven er det besynderlige fenomenet som oppstår hos datamaskiner når de regner med desimaltall. Ta eksemplet under, ved

[6] https://www.w3.org/TR/soap/, besøkt 19.05.2024
[7] https://graphql.org/, hentet 21.05.2024
[8] https://v8.dev/, besøkt 19.05.2024
[9] `https://webassembly.org/`, hentet 21.05.2024
[10] https://emscripten.org/

figur 2.6:



Figur 2.6: Eksempel på kalkulasjonspresisjonsfeil i C++.
Resultatet skulle bli 0, for $3-(10*1/3)=0$ - men programmet mener heller at det er lik $5.9604*10^{-}8$.

Datamaskiner utfører matematiske beregninger ved å representere tall binært. Dette er ikke er et problem for heltall fordi de kan defineres komplett i en binær form.Desimaltall er derimot vanskelige å uttrykke i binær form (for eksempel, 0.1 i binær form er lik en uendelig sekvens av 0.000110011001100...) [29]. De kan ikke uttrykkes nøyaktig som de faktisk er, så datamaskiner tilnærmer seg ved å kutteddenne uendelige sekvensen til en gitt presisjon. Dette resulterer vanligvis i et tall som er *omtrent* likt det opprinnelige desimaltallet. Om vi tar 0.0001100110011 med å kutte av eksemplet over, så ville vi ha fått desimaltallet 0.09997558593 når vi konverterer det tilbake - noe som er nært men ikke likt 0.1.

I mange tilfeller, ender datamaskinen opp med et tall som er så nært det opprinnelige tallet at det spiller lite rolle - men i felt som økonomi og fysikk, så kan dette fenomenet være katastrofalt dersom det oppstår. Dette problemet løses vanligvis med opp/nedrunding av tallet.

2.2.6 WSGI & ASGI

I et forsøk på å skape bedre grunnlag for nettrammeverk i Python, ble det skapt en konvensjon i 2010 kalt WSGI (kort for 'Web Server Gateway Interface') med detaljer for en slags abstraksjon mellom en webserver og applikasjon bygget opp i Python [30]. Målet var å standardisere deres kommunikasjon mellom hverandre, for å tillate mer allsidighet og uavhengighet fra servere.

Denne innsatsen hadde flere suksesser, og grunnla basis for flere nettrammeverk tilgjengelig i Python - inkludert kjente eksempler som Flask og tidligere versjoner av Django[11].

Med introduksjonen av asynkronitet i Python gjennom `await` og `async` ble det etterhvert laget en ny konvensjon i 2019 for å støtte dette for nettforespørseler. Dette ble gjort fordi WSGI kun støttet synkron takling av forespørsler. Denne konvensjonen kalles for ASGI (kort for 'Asynchronous Server Gateway Interface'), og introduserte bruken av asynkron kommunikasjon mellom applikasjon og serverlag tidligere diskutert hos WSGI for å kunne støtte asynkron takling av forespørsler

[11] `https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface`, besøkt 19.05.2024

[12].

Denne etterkommeren hadde stor suksess, og resulterte også i opprettelsen av flere kjente rammeverk, som FastAPI[13], nyere versjoner av Django[14], og Sanic[15].

2.2.7 Human Computer Interacton

Human Computer Interaction (HCI) er studiet av hvordan mennesker som brukere samhandler med datamaskiner. Gjennom studiet er det mulig for utviklere å lære hvordan de kan fjerne feil i design, og forbedre brukeropplevelsen. Den letteste måten oppnå dette på er å integrere HCI prinsippene i utviklingsprosessen. [31]

Brukervennlighet er en vesentlig egenskap ved HCI. Det handler om metoder for å gjøre applikasjonen mer intuitiv under designprosessen. Lærbarhet, effektivitet, minneverdighet, brukerfeil og tilfredshet er alle viktige faktorer. Lærbarehet defineres som hvor enkelt en bruker kan utføre sine oppgaver første gang de bruker programmet. Når en bruker har mestret grensesnitter, referer effektivitet til hvor enkelt det er for brukeren å utføre oppgaven brukeren har satt seg. Minneverdighet handler om hvor lett det er for brukeren å reetablere seg ferdigheter når brukeren kommer tilbake til applikasjonen etter å ikke ha brukt den på en stund. Feil henviser til hvor mange feil brukeren gjør og hvor enkelt det er for dem å korrigere dem. Tilfredshet betyr hvor behagelig designet er å bruke. [32]

[12]https://asgi.readthedocs.io/en/latest/specs/main.html, besøkt 19.05.2024
[13]https://fastapi.tiangolo.com/, besøkt 19.05.2024
[14]https://docs.djangoproject.com/en/5.0/howto/deployment/asgi/, hentet 19.05.2024
[15]https://sanic.dev/en/, hentet 19.05.2024

Kapittel 3

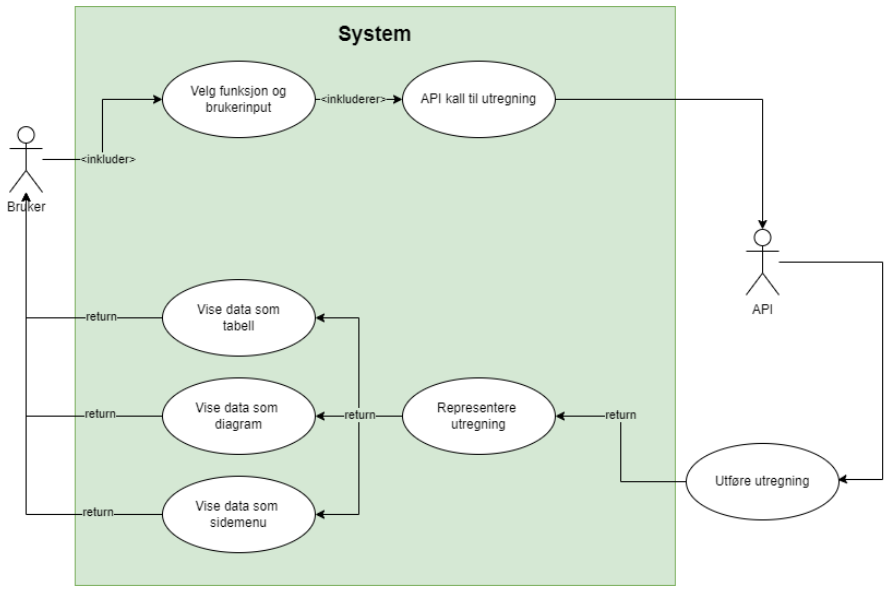
Kravspesifikasjon

For å takle denne oppgaven, har vi laget et sett med krav for hvordan resultatsmålet skal oppnås. Dette inkluderer funksjonelle krav, ikke-funksjonelle krav, og operasjonelle krav med innblandelse fra domenemodell og produktkø.

3.1 Funksjonelle krav

Fordi vårt prosjekt handler om å lage en derivert kopi av basisapplikasjonen, har følgende funksjonelle krav blitt satt med tanke på dette programmet. Dette er detaljert i følgende use-case diagram og use-cases:

3.1.1 Use-Case Diagram



Figur 3.1: Use Case-diagram

3.1.2 Use-cases:

I og med at de funksjonelle kravene skal være lik mellom basisapplikasjon og webapplikasjon, så er disse use-case scenarioene også direkte utførbare på basisprogrammet.

Use Case 1

- Navn: Interagere med graf for valgt funksjon.
- Aktør: Bruker.
- Mål: Bruke webapplikasjonen til å se på og interagere med graf tilhørend en spesifikk fargeromsfunksjon for et gitt sett med parametere.
- Beskrivelse: En bruker velger en av de tilgjengelige fargeromsfunksjonene i programmet. Deretter endrer de på input verdier hos webapplikasjonen, og trykker på en knapp som skal regne ut grafen. Etter det skal grafen vises i webapplikasjonen, hvor brukeren kan interagere med den.

Use Case 2

- Navn: Aktivere logaritmiske verdier for LMS.
- Aktør: Bruker.
- Mål: Bruke webapplikasjonen til å finne logaritmiske verdier for LMS fargeromsfunksjon.
- Beskrivelse: En bruker velger enten 'LMS' eller 'LMS base' fra programmet. Deretter, justerer de på input verdiene for valgt funksjon i webapplikasjon, før de trykker en knapp for utregning. Etter dette finner brukeren en knapp som er markert med 'logarithmic values', som de trykker på for å se at det aktivereres eller deaktivereres logaritmiske verdier for LMS.

Use Case 3

- Navn: Finne en spesifikk verdi i tabell for valgt funksjon.
- Aktør: Bruker.
- Mål: Bruke webapplikasjonen til å finne verdier korresponderende til en spesifikk verdi for en funksjonstabell.
- Beskrivelse: En bruker velger en av de akutelle fargeromsfunksjonene i webapplikasjonen. Deretter justerer de på input, og trykker på utregningsknappen for å få utregningene. Til slutt, trykker de på en knapp som bytter ut grafen med en tabell. I denne tabellen kan brukeren bla og lete etter en spesifikk verdi for funksjonens utregninger.

3.2 Ikke-funksjonelle krav

Vi har også laget følgende ikke-funksjonelle krav til applikasjonen vår:

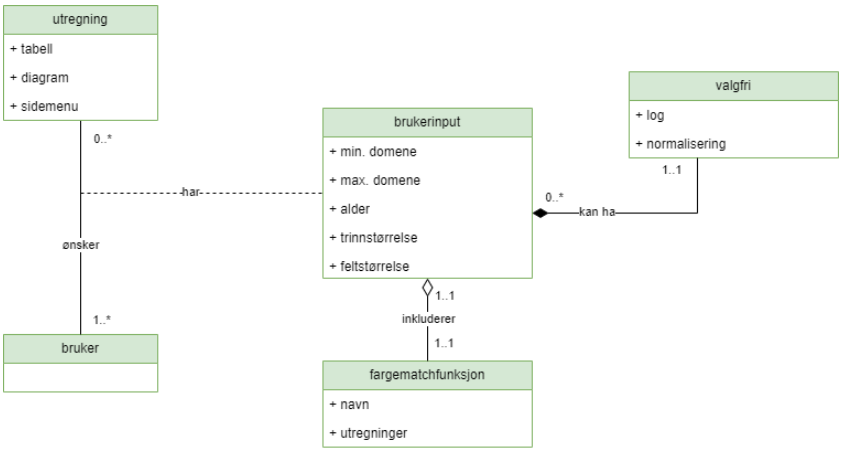
- Webapplikasjonen burde ha et brukergrensesnitt som er likt som det som finnes i basisprogrammet. Brukergrensesnittet burde være tilgjengelig for alle brukere, uavhengig av teknisk erfaring og/eller livssituasjon.
- Webapplikasjonen burde designes med tanke på ytelse. Det skal sørges for at beregninger og/eller visualiseringer utføres med så lite forsinkelser som mulig, og med raskest responstid mulig.
- Webapplikasjonen burde også designes med tanke på skalerbarhet, dersom det må skaleres opp i fremtiden.
- Webapplikasjonens kildekode skal være under åpen lisensiering. Den må også ha ordentlig dokumentasjon for planer om utrulling og gjenbruk av kode.

3.3 Operasjonelle krav

I og med at dette prosjektet skal tjenestegjøres og brukes som et produkt av oppdragsgiver etter den endelige tidsrammen, er det viktig for oss å detaljere operasjonelle krav. Men, fordi vi har avgrensningen at vi ikke skal fokusere på maskinvare, har vi laget følgende operasjonelle krav:

- Svartid på alle mulige forespørsler til system fra bruker må ligge under 1000ms.
- Systemet må kunne takle forespørsler fra flere brukere samtidig, med forslag som asynkronitet eller andre muligheter som kan være relevante for valgt arkitektur.

3.4 Domenemodell



Figur 3.2: Domenemodell

3.5 Produktkø

Produktkøet ble etablert ved starten av prosjektet, og oppdatert jevnlig gjennom dets bruk på Scrumboardet. Det er inkludert i vedlegg E.

Kapittel 4

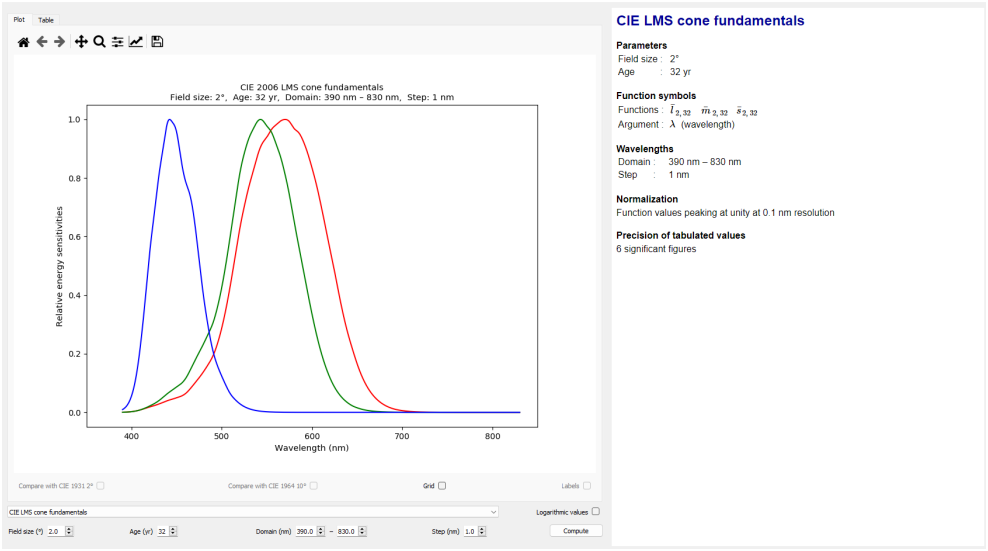
Design og Arkitektur

4.1 Grafisk grensesnitt

Oppdragsgiver uttrykket tidlig et ønske om at webapplikasjonen skulle holdes så lik som mulig til basis applikasjonen (vist i vedlegg D). For å oppnå dette, fokuserte vi på å opprettholde et uniformt grensesnittdesign og interaksjonslogikk mellom basis- og webapplikasjonen. Vi sørget for at alle elementene i brukergrensesnittet var like i både utseende og funksjonalitet. Dette vil bidra til å gjøre overgangen sømløs for brukeren.

I applikasjonen kan brukeren velge mellom ti forskjellige fargematchfunksjoner i en nedtrekksmeny. Under nedtrekksmenyen kan brukeren spesifisere ulike parametere, som feltstørrelse, alder, domene og skritt, dersom det er ønskelig. Etter dette kan brukeren trykke på knappen "compute" for å oppdatere en grafisk og tabulær fremstilling av utregningene, sammen med en informativ sidemeny med formler og annen nyttig informasjon benyttet for utregningene.

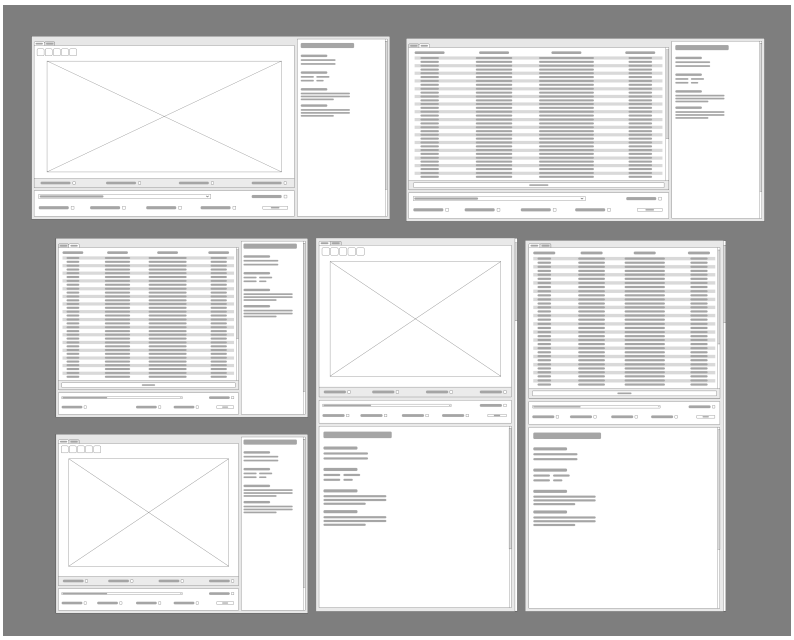
Øverst i applikasjonen finnes det en navigasjonslinje, hvor brukeren kan velge mellom tabulær eller grafisk visning. Under dette, er det plassert to iframes ved siden av hverandre. Den venstre iframen oppdateres ettersom brukeren bytter mellom tabulær eller grafisk fremvisning. Den høyre iframen viser formler og annen informasjon benyttet for for å regne ut de dataene som vises.



Figur 4.1: Basisapplikasjonen 'CIE Functions'

4.1.1 Wireframe

For å oppfylle de ikke-funksjonelle kravene til brukergrensesnittet ble det laget flere detaljerte wireframes. Dette vil hjelpe oss med å sikre intuitiv navigasjon. Brukere skal enkelt kunne finne og benytte alle funksjoner uten behov for omfattende opplæring.



Figur 4.2: Wireframe for brukergrensesnittet

Wireframen, vist på figur 4.2, tar hensyn til responsivt design, samtidig som det forsøker å forholde seg til utseendet til basisprogrammet. Wireframen viser applikasjonen ved fullskjerm (to øverste elementer), applikasjonen ved mindre skjerm (to elementer fra venstre bunn), og til sist, applikasjonen ved enda mindre skjerm (to elementer fra midten bunn og høyre bunn). Den minste skjermen skal ligne til dimensjonene av en mobilenhet.

Utseendet ble laget som en kopi av det som allerede finnes i basisprogrammet, med tanken at en god replikasjon vil gjøre det enklere for brukere å tilvende seg den nye applikasjonen. En ny funksjonallitet, som vi tilføyer med responsivt design i tankene, er å flytte sidemenyen til bunnen av grensesnittet når skjermen blir for smal. Dette finnes ikke i basisprogrammet, men vi mente at det var nødvendig å inkludere dette for å støtte flere enheter og dimensjoner for applikasjonen.

4.2 Systemarkitektur

Som skrevet tidligere i rapporten, så er en stor del av vår oppgave å prøve forskjellige rammeverk for å finne hvilket som passer vårt prosjekt best. For å lage en interessant oppgave med mye diskusjon, har vi valgt å gå for to helt unike arkitekturer med hver av deres fordeler og ulemper:

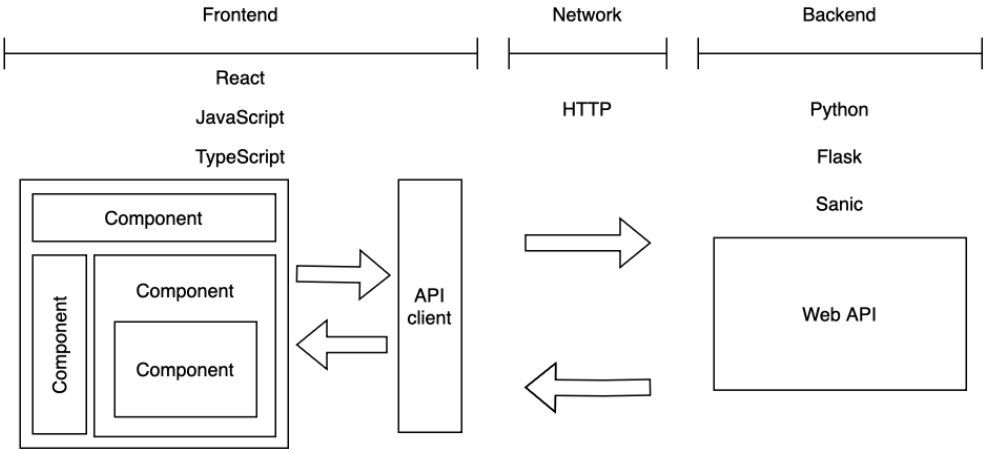
- En 'klient-tjener-basert' arkitektur, hvor utregninger utføres på backend(tjeneren), og klient refererer til brukergrensesnitt eller frontend.
- En 'frontend-basert' arkitektur hvor utregninger utføres heller på frontend, hos klienten. For videre referanser, skal denne arkitekturen bli referert til som 'alt-i-nettleseren'.

Planen er å se på disse arkitekturene, finne passende teknologier og eventuelt lage prototyper hvor vi ser på fordeler og ulemper. Deretter beslutter vi oss for en av dem og utvikler vårt produkt basert på den valgte arkitekturen.

4.2.1 Arkitektur 1: 'Klient-tjener'

I en klient-tjener basert arkitektur, mener vi at applikasjon blir laget med en to-lagsarkitektur, bestående av følgende lag:

1. Presentasjonslaget(frontend) hos klienten: Dette er det visuelle laget, og består av et grafisk brukergrensesnitt som brukerene kan samhandle med direkte.
2. Logisk applikasjonslag hos serveren(backend): Dette er hvor beregninger vil bli utført og APIen lages.



Figur 4.3: Arkitekturdiagram for frontend-backend løsning

Presentasjonslaget

Presentasjonslaget, ofte referert til som brukergrensesnittet, er der brukerene samhandler med applikasjonen. I vårt tilfelle syntes vi det var hensiktsmessig å benytte React og TypeScript. Dette gir flere fordeler:

1. Responsiv Design: Bruk av React gjør det mulig å lage en responsiv og dynamisk UI som kan tilpasse seg forskjellige skjermstørrelser og enheter[33].
2. Komponentbasert Arkitektur: Reacts komponentbaserte tilnærming tillater gjenbrukbare og isolerte komponenter. Dette gjør applikasjonen letter å vedlikeholde og skalere[34].
3. Typesikkerhet: TypeScript gir statisk typesjekking. Dette bidrar til å redusere feil under utviklingen, og gjør koden lettere å forstå og vedlikeholde[35].
4. Responsiv Ytelse: Brukergrensesnittet skal reagere raskt på brukerens handlinger med minimal ventetid.

Applikasjonslaget

Applikasjonslaget, også kjent som det logiske laget, er der kjernelogikken til applikasjonen blir implementert. I vårt system er dette laget utviklet i Python. Python ble valgt fordi basisapplikasjonen er laget på Python, samt at programmeringsspråket er spesielt egnet for beregninger og databehandling. Applikasjons laget er ansvarlig for å oppfylle de følgende funksjonelle kravene til applikasjonen:

1. Beregininger: Dette laget håndterer komplekse beregninger baser på parameter spesifisert av brukeren.
2. API Funksjonalitet: Applikasjonslaget eksponerer APIer som gjør det mulig for forntend å kommunisere med backend og hente nødvendige data.

API Design

Vi valgte å bruke REST-prinsipper for å strukturere vårt API. REST er et sett med retningslinje som sikrer skalerbarhet, enkelhet og effektivitet i kommunikasjonen mellom klient og server. Noen av fordelene med REST:

- Skalerbarhet: RESTful APIer kan enkelt skaleres over flere servere[36].
- Modularitet: Hver ressurs er klart definert. Dette gjør det enkelt å oppdatere og vedlikeholde systemet[36].
- Statelessness: Hver forespørsel fra klient til servern må inneholde all informasjonen som kreves for å forstå og behandle forespørselen. Dette bidrar til å forenkle serverens logikk[36].

Domenemodell

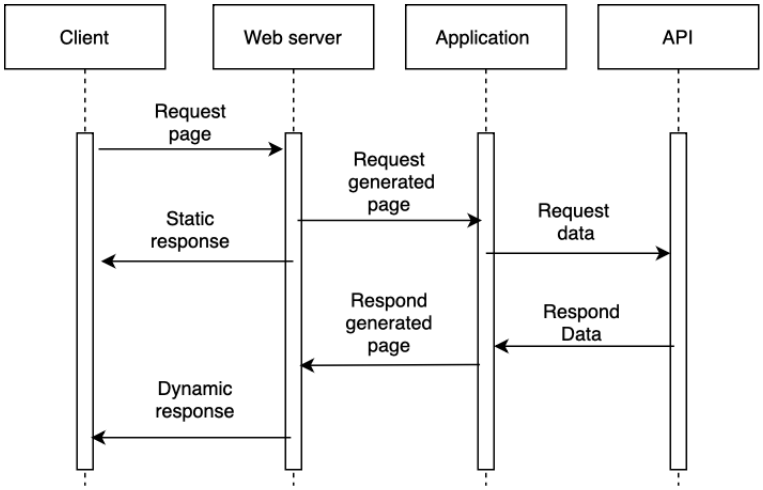
Diagrammet viser relasjonene mellom ulike elementer i systemt 3.2:

1. Bruker: En bruker kan ønske å utføre flere beregninger. Hver bruker kan initier en eller flere beregningsforespørsler.
2. Utregning: Etter representere selve beregningsprosessen. Hver utregning er koblet til brukerinput via en relasjon som indikerer at en utregning har en eller flere brukerinput.
3. Brukerinput: Dette inneholder flere forskjellige parametre. brukerinput inkluderer også en fargematchfunksjon som kan være enten en standardiseringsfunksjon eller en parameterbasert funksjon.
4. Valgfri Funksjonalitet: Brukerinput kan også ha valgfrie spesifikasjoner. For eksempel om de ønsker logoritmisk visning av data eller renormaliserte verdier.

Domenemodellen spiller en kritisk rolle i å opprettholde systemets integritet og funksjonalitet. Ved å implementere nødvendige regler og prosesser for datamanipulering legger domenemodellen grunnlaget for en robust og pålitelig applikasjon. Godt samarbeid mellom domenemodellen og REST APIet er essensielt for å sikre nøyaktig output fra applikasjonen basert på de spesifiserte parameterne.

Sekvensdiagram

Sekvensdiagrammet illustrerer hvordan en klientforespørsel håndteres sekvensielt gjennom ulike lag i denne webappliasjonen, fra forespørsel til respons. Diagrammet viser samspillet mellom klienten, webserveren, webapplikasjonen og APIet.



Figur 4.4: Sekvensdiagram for backend-basert løsning

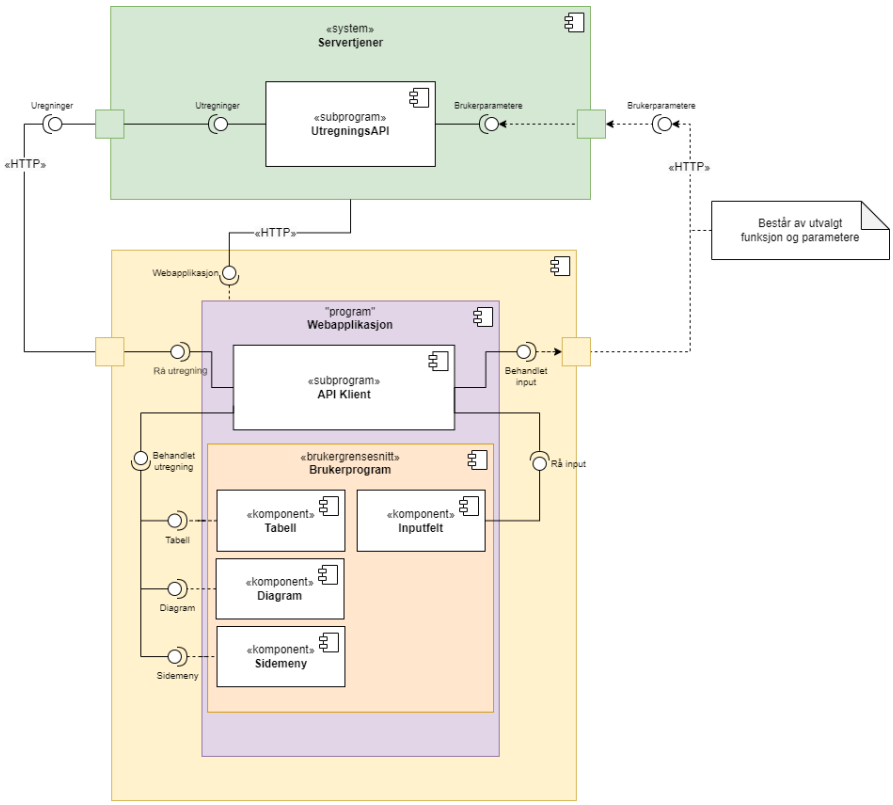
Når en klient, som typtisk er en nettleser, sender en forespørsel om en side til webserveren vurderer webserveren om innholdet er statisk eller dynamisk. statisk innhold, som HTML-filer, bilder eller CSS blir umiddelbart sendt tilbake til klienten som en statisk respons. Dette er rask og krever ingen videre behandling.

For dynamisk innhold må webserveren be applikasjonen om å generere innhold på siden.

APIet fungerer som et grensesnitt til å utførte beregninger. Når applikasjonen gjør en forespørsel til APIet, gjør APIet de nødvendige beregningene og returnerer dataen til applikasjonen. Applikasjonen bruker deretter disse dataene til å lage den dynamiske siden.

Komponentdiagram

Komponentdiagrammet viser hvordan de ulike delene av systemet henger sammen. Hovedkomponentene er API, nettleser, webapplikasjon og brukergrensesnitt. Hver komponent har definerte ansvarsområder:



Figur 4.5: Komponentdiagram for backend-basert løsning

1. **API:** Håndterer alle beregninger og brukerparametere. Dette inkluderer å utføre beregninger basert på brukerinput og returnere resultatene til webapplikasjonen.
2. **Nettleser:** 'Hoster' webapplikasjonen og presenterer den til brukeren. Den mottar også data fra APIet for å oppdatere brukergrensesnittet.
3. **Webapplikasjon:** Inkluderer API-klienten som kommuniserer med backend for å hente data og behandle input. Dette laget fungerer som en mellommann mellom frontend og backend.
4. **Brukergrensesnitt:** Består av ulike komponenter som tabell, inputfelt og diagram. Disse komponentene gir brukeren mulighet til å samhandle med applikasjonen og viser de nødvendige dataene på en strukturert måte.

4.2.2 Arkitektur 2: 'Alt-i-nettleseren'

Diskusjon

I en frontend-basert arkitektur flyttes all nødvendig logikk for beregninger til brukerens nettleser, i motsetning til den mer tradisjonelle arkitekturen beskrevet tidligere. Selv om dette ikke er like tradisjonelt, så er det heller ikke en uvanlig arkitektur for webapplikasjoner hvor umiddelbar respons er det viktigste; et godt

eksempel er nettleserspill[1] hvor spillet kjøres i selve nettleseren uten å måtte kommunisere med en tjener for å kjøre.

Det er viktig å merke at tjeneren eksisterer fortsatt i en slik arkitektur - for den er ansvarlig for å utlevere webapplikasjonen. På grunn av dette, kan vi fortsatt omtale denne arkitekturen som 'to-lag' - men det er riktigere å omtale den som en variant.

Vi kan se hvordan med å se på dets komponent- og sekvensdiagram, som vist i figur 4.6 og 4.7. Tjeneren vil fremdeles eksistere for å gi fra seg webapplikasjonen - men istedenfor at den skal selv utføre applikasjonslogikken, flyttes det over nativt inni nettleseren. Om vi skulle sammenligne det med den forrige arkitekturen, kan man egentlig si at de er identiske - kun at applikasjonslaget har blitt flyttet over til nettleseren.

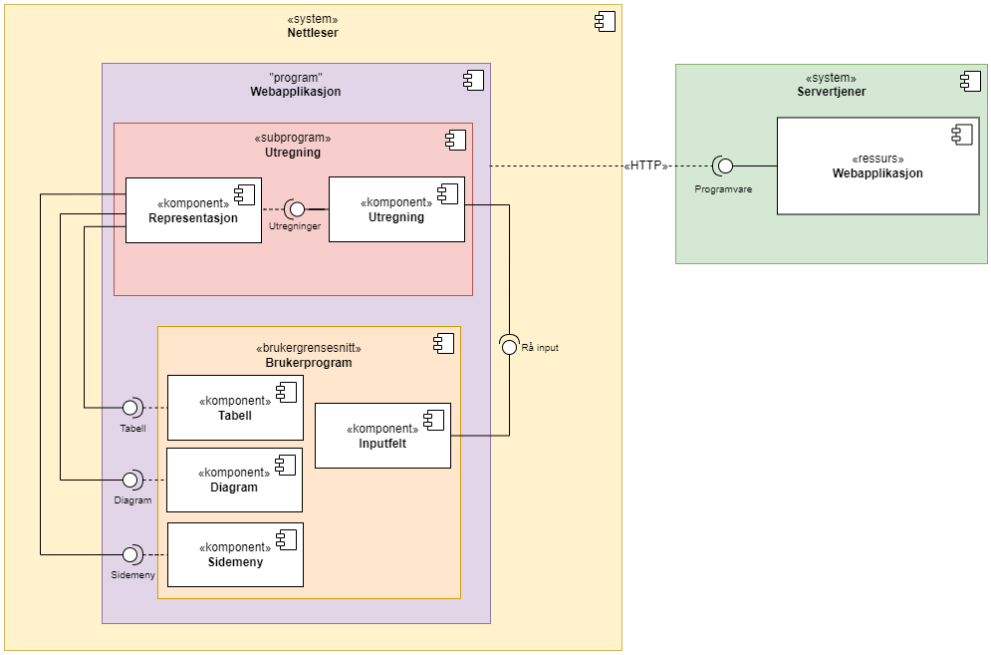
For noen år siden, ville en slik arkitektur ha vært vanskelig å utføre. Dette skyldes at det ville ha innebåret å oversette utregningskoden til JS - noe som kan være tidsintensiv, gi forskjellige resultater fra basiskoden[2], og muligens ha vært umulig på grunn av bruken av språkekslusive biblioteker. Men med ankomsten av WebAssembly, har denne barrier blitt brutt, og mulige funksjonaliteter som man kan utføre gjennom en webapplikasjon har økt drastisk. En arkitektur der beregninger utføres av et Wasm-basert rammeverk for Python, har blitt mulig, og er derfor et reelt alternativ for denne typen oppgaver.

Fordelene med en slik arkitektur er flere. Den viktigste er at alt flyttes til klienten. Serverens eneste oppgave blir å levere det statiske dokumentet med applikasjonen, noe som krever minimal ytelse fra serveren og dermed gjør det billig å opprettholde. Alt av utregninger vil avhenge på klienten sin enhet. Vi antar ikke at det skal være et problem for klientens enhet, da WebAssembly skal være rask og gi 'nesten-nativ' ytelse[3]. Med tanke på selve ytelsen til basisprogrammet, antar vi ikke at dette skal være et bekymringsområde for oss.

Komponentmodell

Vises på figur 4.6.

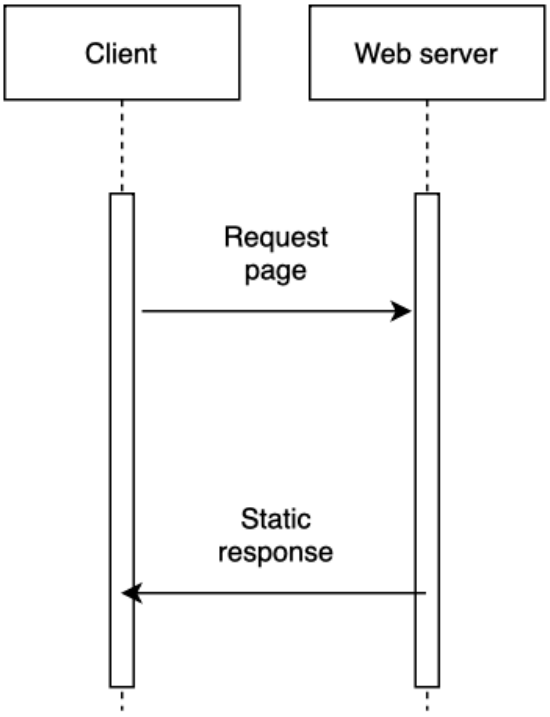
[1] https://no.wikipedia.org/wiki/Nettleserspill, besøkt 19.05.2024
[2] Med tanke på forskjellige språk sine typer; oversettelsen mellom dem.
[3] `https://developer.mozilla.org/en-US/docs/WebAssembly`, hentet 19.05.2024



Figur 4.6: Komponentdiagram for frontend-basert løsning.

Sekvensdiagram

Vises på figur 4.7.



Figur 4.7: Sekvensdiagram for frontend-basert løsning

Kapittel 5

Teknologier

5.1 Oppgavearkitekturer

Som diskutert tidligere, ønsker vi først å eksperimentere med to typer arkitekturer som er radikalt forskjellige fra hverandre, i håp på at vi kan finne den beste av dem gjennom testing og diskusjon senere gjennom utviklingsprosessen.

En av dem skal da være basert på 'alt-i-nettleser' arkitektruren, mens den andre skal være basert på et 'klient-tjener' forhold. For dem, har vi valgt følgende teknologier:

5.1.1 'Klient-tjener'- Flask

For den backend-baserte arkitekturen, gikk vi med rammeverket kalt Flask. Flask er et WSGI-basert rammeverk for å skape webapplikasjoner gjennom Python, utviklet av Armin Ronacher i 2004[1]. Senere ble det overlatt til Pallets i 2016 [37]. Rammeverket har åpen kildekode sammen med en BSD-3-Clause lisens på Github[2].

Vi valgte Flask over andre kjente rammeverk fordi vi hadde allerede jobbet med det tidligere i flere prosjekter gjennom våre grader, og vi syntes at det var godt å jobbe med. Rammeverket er også populært, enkelt tilgjengelig og godt dokumentert. Dette gjør det enklere for oppdragsgiver å oppdatere senere etter tidsrammen for prosjektet.

5.1.2 'Alt-i-nettleser' - PyScript

For den frontend-baserte arkitekturen, valgte vi det nye og moderne rammeverket kalt PyScript. PyScript ble skapt av 'PyScript Development Team' hos Anaconda, Inc.[3], og er et WebAssembly-basert rammeverk som gjør det mulig å kjøre Pyt-

[1] `https://flask.palletsprojects.com/en/2.0.x/`, hentet 21.05.2024
[2] https://github.com/pallets/flask, hentet 19.05.2024.
[3] https://pyscript.net/, hentet 19.05.2024

hon hos en klient sin nettleser. Rammeverket har åpen kildekode[4] med Apache 2.0-lisensen, og oppdateres ofte av flere utviklere med ønske om å forbedre rammeverket over tid.

PyScript i seg selv er ikke en Wasm-basert kompilator, men er heller et lag av abstraksjoner, bygget opp og API basert på Pyodide; ett rammeverk utviklet hos Mozilla med åpen kildekode som videre fungerer som en port av CPython til WebAssembly [5]

5.2 Klient-tjener frontend

5.2.1 React

React[6] er et lettvekts JavaScript bibliotek hvor hovedfokuset er å gi raske, skalerbare og enkle web applikasjoner. React ble utviklet av en liten gruppe utviklere hos Facebook for å løse utfordringene rundt å utvikle komplekse brukergrensesnitt med oppdatering av datasett [34].

I motsetning til de eksisterende MVC (Model-View-Controller)[38] rammeverkene som ofte brukes til webutvikling, så fokuserer React på 'view' delen av MVC. Det bruker modulære komponenter og virtuell DOM, som gjør at visningen kan endres individuelt kun for de komponentene som trenger å oppdateres. Med dette begrenses antallet DOM oppdateringer som trengs. Dette optimaliserer «rendering» og gjør at en React app oppfattes som rask og responsiv.

Som nevnt er React et lettvekts bibliotek, noe som betyr at det ikke inneholder alt av verktøy man gjerne forventer av et rammeverk for webutvikling[39]. Til gjengjeld kan eksisterende JavaScript biblioteker sømløst integreres. Et eksempel på dette er at React kan benytte seg av `Redux` dersom det trengs en mer kompleks håndtering av programtilstand i webapplikasjonen[40].

5.2.2 TypeScript

TypeScript er en utvidelse av JavaScript utviklet av Microsoft[7]. Det er designet slik at all kode skrevet i JavaScript kan benyttes med nøyaktig samme syntaks i TypeScript. TypeScript legger til en rekke funksjonaliteter til JavaScript for å forenkle utviklingen av større applikasjoner mer håndterbar.

Ett av disse funksjonalitetene er statisk sjekking av variabeltype. Dette lar utviklere sjekke typen til en variabel, parameter eller returverdi under utvikling. Dette gjør det betydelig lettere å fange opp feil før koden kjører og reduserer bugs og kjørtidsfeil.[41]

En annen element er et større fokus på objektorientert programmering gjennom bruk av klasser. Mens JS støtter en tidlig prototype av dette, så utvider Type-

[4] https://github.com/pyscript/pyscript, hentet 19.05.2024
[5] `https://pyodide.org/en/stable/project/about.html`, hentet 19.05.2024
[6] https://react.dev/, hentet 19.05.2024
[7] `https://www.typescriptlang.org/`, hentet 19.05.2024

Script dette til mer klassisk og kjent form. Dette tillater utviklere som kommer fra for eksempel Java eller C å benytte teknikker som klasser, interfaces og arv[42].

5.2.3 Visual Studio Code

Visual Studio Code (også kjent som VS Code) er et populært koderedigeringsprogram utviklet av Microsoft med støtte for Windows, macOS, Linux og nettlesere[8]. Programmet har støtte for utallige forskjellige kodespråk og har utviklingsverktøy som «intelligent code completion», «debugging», «syntax highlighting» og «code refactoring» innebygd[43].

I tillegg til dette, har VS Code et stort utvalg av forskjellige verktøy skapt både av Microsoft og av andre brukere, hvor brukeren kan laste ned ulike utvidelser for alle slags funksjonalitet. Dette gjør at utviklere kan tilpasse VS Code i større grader gjennom preferanser og funksjonalitet som vil hjelpe med det gjeldende prosjektet.

5.2.4 Plotly & Recharts

Plotly[9] og Recharts[10] er to ulike biblioteker for grafisk visualisering av data.

Plotly er et allsidig og sterkt bibliotek som legger til rette for å lage interaktive visualiseringer av data, som for eksempel grafer, av høy kvalitet. Det støtter mange ulike typer grafer og tilbyr en rekke innebygde funksjoner som hjelper med interaktiviteten og funksjonaliteten til grafen. Eksempler på dette er innebygd funksjonalitet for forstørrelse og panorering.

Plotly er originalt et JavaScript bibliotek for webutvikling, men støtter nå en rekke andre programmeringsspråk som Python[11] og MATLAB[12]. Plotly har støtte for både enkle og særdeles komplekse grafiske visualiseringer av data. Plotly er også et sterkt bibliotek med tanke på skalerbarhet, da det er bygd for å effektivt håndtere veldig store datasett.

Recharts er et bibliotek bygd på React komponenter og utnytter D3.js[13], et JavaScript bibliotek for å produsere dynamiske og interaktive datavisualiseringer i nettlesere. Dets komponenter er rene React komponenter som sømløst passer inn i rammeverkets økosystem. Dette betyr at det kan utnytte Reacts sine funksjonaliteter som `props` for tilpasning og `state` for dynamiske oppdateringer. Det er også designet for effektivitet innen rammeverkets virtuelle DOM, som gjør at det kan hurtig og effektivt håndtere oppdateringer og oppdatere 'rendering' hos webapplikasjonen.

[8] `https://code.visualstudio.com/`, hentet 19.05.2024
[9] `https://plotly.com/`, hentet 19.05.2024
[10] `https://recharts.org/en-US/`, hentet 19.05.2024
[11] `https://plotly.com/python/`, hentet 19.05.2024
[12] `https://plotly.com/matlab/`, hentet 19.05.2024
[13] `https://d3js.org/`, hentet 19.05.2024

Recharts har god støtte for enkle graftyper som linjegraf, stolpediagram med mer, men kommer gjerne for kort når det er krav om mer komplekse visualiseringer av data. Den har også en begrensning når det kommer til håndtering av veldig store datasett, da dette vil påvirke ytelsen i en større grad.

5.3 Klient-tjener backend

5.3.1 Python

Som ønsket av oppgaven, så skal Python brukes som programmeringsspråket for utregninger relevant til prosjektet. Språket er et populært høyt-nivå objekt-orientert språk [14] med åpen kildekode [15], kjent for dets enkle syntax og populære fagmiljø. Det har lenge vært et av de mest populærte språkene innenfor programmering (med noen undersøkelser som sier at det er det tredhe mest populære språket[16], hvor det har spesiell erkjennelse innenfor maskinlæring.

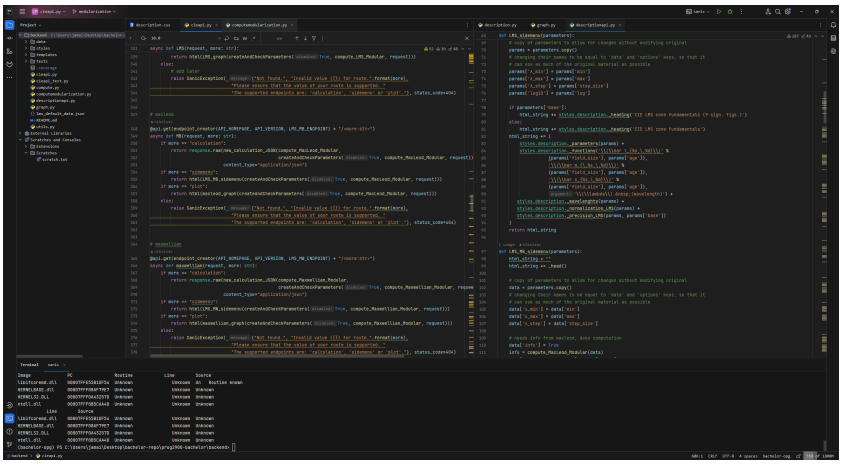
Den tilbyr mange funksjonaliteter, som dynamisk variabeltypesetting, enkel og rask debugging, kan kjøre uten kompilering, og er veldig universell blant forskjellige systemer. Den har også et veldig rikt økosystem av mange biblioteker og rammeverk for alle mulige oppgaver.

For dette prosjektet, fant vi følgende biblioteker relevant:

- **Pandas**[17]: Et populært bibliotek for manipulering og analyse av datastrukturer. Dets gode ytelse med diverse datastrukturer, sammen med fleksible og enkle metoder gjør det anerkjent i språkets økosystem. Biblioteket har åpen kildekode med BSD-3-Clause lisens[18].
- **NumPy**[19]: Et annet populært bibliotek for datastrukturer, kun at det fokuserer mer på multidimensjonelle `array` og matematikk. Biblioteket er også anerkjent godt i økosystemet, og er integrert med flere andre biblioteker tilgjengelig i språket. Denne har åpen kildekode med egen lisens[20]
- **Unittest & Pytest**: Dette er to populære testingbibliotek i applikasjonen, hvor Unittest er tilgjengelig i standardbiblioteket[21] mens PyTest[22] er enkel og separat med åpen kildekode og MIT-lisens[23].

[14] `https://www.python.org/doc/essays/blurb/`, hentet 20.05.2024
[15] `https://docs.python.org/3/license.html`, hentet 20.05.2024
[16] `https://survey.stackoverflow.co/2023/#technology`, hentet 20.05.2024
[17] `https://pandas.pydata.org/`, hentet 19.05.2024
[18] `https://github.com/pandas-dev/pandas`, hentet 19.05.2024
[19] `https://numpy.org/`, hentet 19.05.2024
[20] `https://github.com/numpy/numpy`, hentet 19.05.2024.
[21] `https://docs.python.org/3/library/unittest.html`, hentet 19.05.2024.
[22] `https://docs.pytest.org/en/8.2.x/`, hentet 19.05.2024.
[23] `https://github.com/pytest-dev/pytest`, hentet 19.05.2024

5.3.2 PyCharm



Figur 5.1: Skjermbilde av PyCharm som utviklingsmiljø.

For utvikling av backend, blir det brukt PyCharm[24] som hovedutviklingsmiljø. PyCharm er en høykvalitets IDE laget av JetBrains, et firma anerkjent for både utviklingen av flere sofistikerte IDE, i tillegg til å lage programmeringsspråket Kotlin for Androidenheter [44].

Miljøet tilbyr flere funksjonaliteter som vi mener gjør det overlegent over andre utviklingsmiljøer. I tillegg til at den tilbyr et enkel installering, et sofistikert debuggingsystem med brekkepunkter og omfattende deteksjon av programproblemer. Den har også innebygd støtte for flere biblioteker ved å tilby spesielle funksjonaliteter for dem.

5.3.3 Sanic

Et annet webapplikasjonrammeverk for Python (som skal tas opp senere i rapporten) heter Sanic[25]. Sanic er et ASGI-basert rammeverk, skapt spesifikt for hensikten om å 'være rask' gjennom bruk av spesiell asynkron kode i Python. Den var først laget i 2018 med åpen kildekode og MIT lisens[26]. Ved dets lansering, virket det rask som en stor inspirasjon til andre rammeverk for dets gode hastighet (som for eksempel FastAPI[27] som selv anerkjente rammeverket som en inspirasjonskilde[28]).

Rammeverket er kjent for å være lett, fleksibel og å ha et 'Flaskaktig syntaks'[29], samtidig som det har et stort økosystem av biblioteker dedikert til å tilsette flere

[24] `https://www.jetbrains.com/pycharm/`, hentet 19.05.2024
[25] `https://sanic.dev/en/`, hentet 19.05.2024.
[26] `https://github.com/sanic-org/sanic`, hentet 19.05.2024.
[27] `https://fastapi.tiangolo.com/`, hentet 19.05.2024
[28] `https://fastapi.tiangolo.com/alternatives/#sanic`, hentet 19.05.2024.
[29] `https://sanic.readthedocs.io/en/18.12.0/`, hentet 19.05.2024

funksjonaliteter til rammeverket. Selv som et eget rammeverk, tilbyr den funksjonaliteter spesifisert til å 'få jobben ferdig'; dette inkluderer god debugging, automatisk TLS for sikkerhet og mye mer. En annen god side ved Sanic er dets skalerbarhet, hvor det kan tillate flere arbeidsprosesser til å kjøre og tjenestegjøre for en webapplikasjon.

5.3.4 Docker

For utrullingen av prosjektet, planlegges det å bruke Docker[30]. Docker er en plattform som gjør det mulig å automatisere utrulling av applikasjoner som selvstendige konteinere [45]. Plattformene det består av har åpen kildekode med Apache-2.0 lisensen[31]

Som sagt, så benytter plattformen seg av konteinere, som er enkle og isolerte miljøer med alt av data nødvendig for at en gitt applikasjon skal kjøre. Disse konteinere er selvstendig og kan kjøres likt uavhengig av operativsystem, dette er en fordel som gjør Docker et godt valgt til portabilitet av programvare mellom systemer [**docker.docs**]. De kan også kjøres i plenum for å øke skalerbarhet for en tjeneste.

Vi ser at Docker inneholder en rekke viktige komponenter, som:

- **Docker Engine**: Komponenten som står for å bygge og kjøre konteinere.
- **Docker Image**: Et uforanderlig miljø som har alt man trenger for å kjøre en applikasjon[46].
- **Docker Kontainer**: Kjører instanser av en 'image' fortalt over.
- **Docker Compose**: Verktøy for å definere og kjøre applikasjoner bestående av flere konteinere ved hjelp av YAML-filer.

5.4 Annet

5.4.1 Maskinvare

For dette prosjektet, ble det brukt to typer av maskinvarer for ytelsestester og tjenelsegjørelse av applikasjonen. Deres spesifikasjoner er vist frem i tabell 5.1, og er:

1. Maskinvare 1: En Acer Swift SFX14-41G brukt av en student i gruppa for utvikling og kjøring av umiddelbare ytelsestester kun for utviklingsperioden.
2. Maskinvare 2: En dedikert virtuell maskin hos SkyHiGh[32] for siste ytelsestester diskutert ved kap. 8.1.2, i tillegg til videre tjenestegjørelse av applikasjonen.

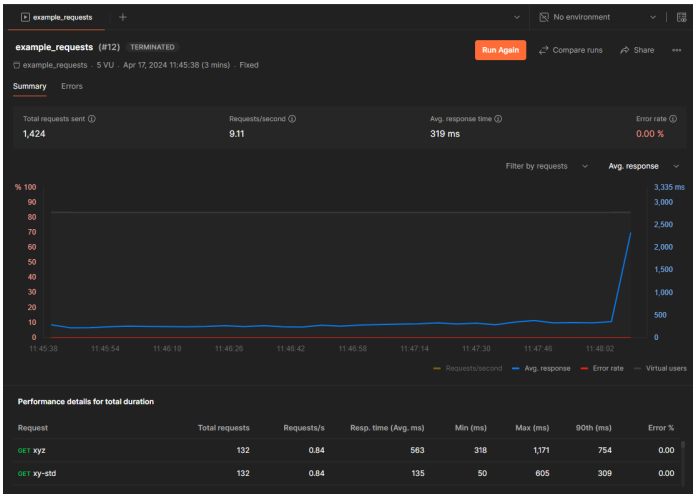
[30] `https://www.docker.com/`, hentet 19.05.2024.
[31] `https://github.com/docker`, hentet 19.05.2024.
[32] `https://www.ntnu.no/wiki/display/skyhigh`, besøkt 19.05.2024

	CPU	RAM	Minne
Maskinvare 1	8 kjerner	16 GB	50 GB+
Maskinvare 2	2 kjerner	2 GB	5-10 GB

Tabell 5.1: Maskinvarespesifikasjoner

5.4.2 Postman

Postman[33] er en plattform for avansert testing av API. Verktøyet tillater for flere metoder av testing og overvåking av tjenester, som elementære enhetstester av respons - til sofistikerte integrasjons- og ytelsestester. For dette prosjektet, ble det brukt til å teste responsinnhold fra backend, i tillegg til diverse ytelsestester utført over helheten av prosjektet.



Figur 5.2: Skjermbilde av Postman brukt i prosjektet for trafikktesting.

5.4.3 Overleaf

Overleaf[34] er en webapplikasjon for skriving av dokumenter gjennom LaTeX(LaTex). Applikasjonen tillater for flere medlemmer å jobbe sammen på et dokument, og har vært en standard for oss studenter i flere år. For dette prosjektet, brukes det til å skrive rapporten for oppgaven, samt andre tekster relevant for oppgaven.

5.4.4 Figma

Figma[35] er en webapplikasjon for designing og prototyping av diagrammer og wireframe. Programmet har blitt brukt av oss studenter jevnlig gjennom vår grad. I dette prosjektet, skal det brukes for å generere wireframe for prosjektet.

[33] `https://www.postman.com/`, hentet 19.05.2024
[34] `https://www.overleaf.com/`, hentet 19.05.2024
[35] `https://www.figma.com/`, hentet 19.05.2024.

5.4.5 Drawio

Drawio[36] er en webapplikasjon for designing av UML-diagrammer. Likt Figma, så er dette programmet også godt kjent med oss, og derfor bruker vi det med stor troverdighet. I dette prosjektet, var det brukt til å lage use-case diagram, domenemodell, og flere andre diagrammer for rapporten.

[36] `https://www.drawio.com/`, hentet 19.05.2024.

Kapittel 6

Prosess

Dette kapittelet skal gjennomgå utviklingsprosessen vi har hatt gjennom prosjektperioden. Fordi prosjektet vårt gjennomgikk flere drastiske endringer i løpet av våre sprinter, mener vi at det er best for rapportens skyld å først formidle prosjektet gjennom utviklingsprosessen med bruken av sprinter. Dette vil følges opp av et eventuelt kapittel som tar det endelige produktet fra alle sprinter, og går mer innom detalj på implementasjon.

6.1 Sprint 1-2

De første to sprintperioder gikk til valget vi måtte gjøre mellom våre arkitekturer; 'alt-i-nettleser' og 'klient-tjener'. Selv om begge er reelle måter å løse oppgaven på, kunne vi velge kun en. Derfor, bestemte vi oss at vi skulle finne ut hvilken var best for oppgaven med å lage prototyper som skulle demonstrere spesifikke funksjonaliteter.

Disse funksjonalitene ble bestemt til å være:

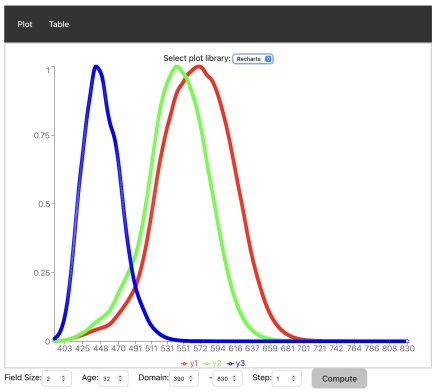
- Utføre utregning fra basisprogrammet.
- Vise frem et diagram som finnes i basisprogrammet.
- Få input fra bruker gjennom parametere i prototype.
- Vise frem og/eller ha en form av en tabell.

Etter deres utvikling, var planen å ta dem opp til diskusjon med å fokusere på deres fordeler og ulemper innenfor deres utviklingsprosess og prototypen selv. Kun etter det, mente vi at vi var klar til å gjøre et endelig valg.

6.1.1 Prototype for 'klient-tjener':

For utvikling av 'klient-tjener' prototypen, valgte vi å benytte Flask som vårt nettrammeverk slik tidligere skrevet. I tillegg til dette, bestemte vi oss å også bruke React, Vite og Typescript for å utvikle brukergrensesnittet. React gjør det mulig for oss å lage det dynamisk og responsiv gjennom Typescript, mens Vite sørger

for rask byggetid og bruk av HMR[1]. Med bruken av dette, utviklet vi en prototype med de nødvendige funksjonalitetene.



Figur 6.1: Diagram gjennom Recharts

Figur 6.2: Diagram gjennom Plotly

Prototypen ble utviklet med tanke på dets tidligere diskuterte arkitektur, hvor brukeren sender en forespørsel til serveren via et endepunkt, serveren behandler forespørselen og utfører nødvendige beregninger, før det sender resultatene tilbake til klienten. Dette ble gjort gjennom bruk av en forespørsel med metoden `POST`, og innmat av JSON for å håndtere innkommende data og input fra brukeren. Innmaten ble sendt som en `Map` av både `result` og `plot` verdier fra utregningene direkte.

Vi testet også to biblioteker for diagramvisualisering, som er da Plotly og Recharts slik tidligere introdusert. Hensikten var å finne hvilken var best for å visualisere diagrammet.

Under utviklingen av backend prototypen møtte vi også på en del utfordringer. Den største av disse var 'beregningsfeil' slik tidligere fortalt om i løpet av kapittel 2.2.5. Under testing oppdaget vi at ikke alle beregningene ga de forventede resultater, og at flere viste den klassiske karakteristikken av tall preget med feilen (slik vist i figur 6.3). I basisapplikasjonen var det implementert egendefinerte avrunding og 'kutte' funksjoner som sørget for at tallene ble korrekte, men allikevel med deres bruk, hadde vi fortsatt disse resultatene.

X	Y 1	Y 2	Y 3
390	0.000415003	0.00036834899999999997	0.00954729
391	0.0005026500000000001	0.000448015	0.0114794
392	0.000607367	0.0005439650000000001	0.013798600000000001

Figur 6.3: Tabell med beregningsfeil.

[1]https://webpack.js.org/concepts/hot-module-replacement/, hentet 19.05.2024

Det ble også utført ytelsestester av denne tjenesten for å evaluere ytelsen under moderat belastning med 5 virtuelle brukere i Postman, slik dokumentert i mer detalj hos vedlegg F. For hensikten av enklere lesing, har relevant informasjon også blitt tilsatt her:

	Verdi
Forespørsler per sekund	2.26
Gjennomsnittlig respons tid	1211 ms
Totalt antall forespørsler	421

Tabell 6.1: Ytelsestest av API

Vi ser at modellen bruker 1211 ms for å få svar fra en forespørsel, noe som kan virke ekstremt; men det må klargjøres at endepunktet som ble testet utregnet alle utregningsfunksjoner fra basisprogrammet. Med dette i tanke, er denne responstiden faktisk en fordel for prototypen - for det viser til at denne tjenesten bruker kun 1.2 s til å gi en klient utregningene (og kan reduseres gjennom modulisering av utregningene).

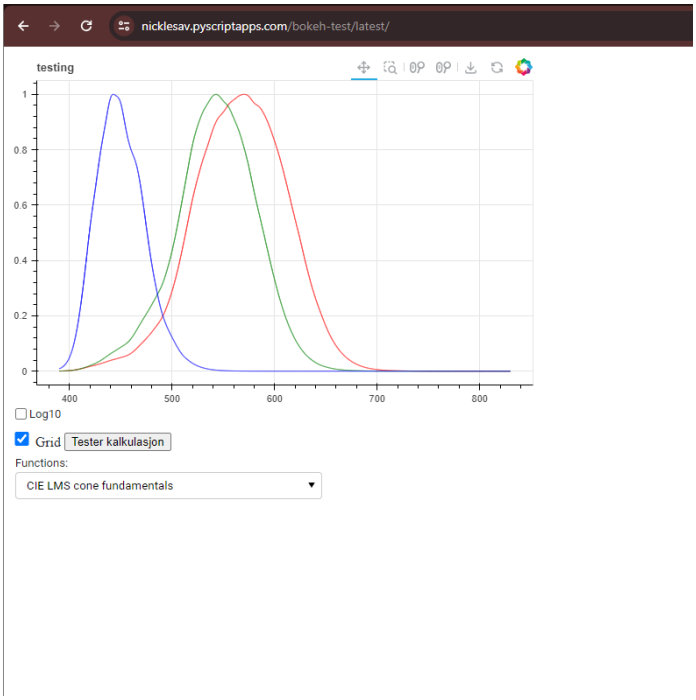
For å videre få innsikt i dette, brukte vi Postman igjen for å sende en enkel forespørsel for å se mer på den, hvor vi tok med viktige punkt i tabell 6.2.

	Verdi
Respons tid	883 ms
Nedlastet data	3.44 MB
Nedlastningshastighet	3.90 MB/s

Tabell 6.2: Enkel test av API

Responstiden uten stress var satt på 883 ms, et tall vi mener kan igjen reduseres gjennom modulisering. Vi ser også at den nedlastede dataen er høy - igjen en sideeffekt av at prototypen regner ut alle på en gang.

6.1.2 Prototype for 'alt-i-nettleser':



Figur 6.4: Skjermbilde av frontend-basert prototype med flere funksjonaliteter.

Utviklingen av 'alt-i-nettleser' prototypen foregikk gjennom en dedikert utviklingstjeneste av PyScript selv, hvor registrerte brukere fikk tillatelse til å enklere lage demoapplikasjoner[2]. Selv om det er mulig å lage en lokal prototype, så ble heller dette valgt for fordelen av å teste om prototypen fungerte andre enheter. Det var jo teknisk sett 'nytt', og kunne ha problemer med forskjellige nettlesere.

Prototypen selv (som vist på figur 6.4) har funksjonalitene for ett funksjonelt diagram, og brukerinteraksjon gjennom avmerkningsbokser og nedtrekksmeny for flere funksjoner. En tabell ble ekskludert fra utviklingen, for vi fant raskt ut at dette rammeverket hadde en god del ulemper, og kun noen få fordeler.

Det første problemet vi fant, var en direkte forskjell i evner mellom den tradisjonelle JS, og alle Wasm-baserte rammeverk. Mens JavaScript hadde direkte evne til å manipulere DOM (og derfor direkte generere dynamisk innhold), så har det ikke blitt implementert innenfor WebAssembly ennå, med fremtidige planer til det[3]. Selv om dette ble ikke ett direkte problem for utregningskoden nødvendig for figur og tabell for Python kunne fremdeles kjøre, så ble det heller to problemer med tanke på kompatibilitet:

[2]https://pyscript.com/, hentet 19.05.2024
[3]https://developer.mozilla.org/en-US/docs/WebAssembly/Concepts, hentet 19.05.2024

1. For å kunne vise resultatene det utregner, må Python kommunisere gjennom JS som proxy til dokumentet. Selv om rammeverket tilbyr en FFI med tilgang til standard webAPI som DOM[4], så kommer problemet inn når man må kombinere data mellom begge språkene. Et eksempel er at unike typer kreve rundveier for å være kompatibel, som bruk av JSON for å passere på informasjon mellom språkene. Det er også spesielt vanskelig å finne ut hvor det oppstår programfeil.
2. I det tilfellet hvor denne arkitekturen velges, så er det sannsynlig at den må gjennomgå en radikal endring i fremtiden. I det tilfelle hvor Wasm oppdateres til å bruke DOM innebygd, da finnes det grunnlag for å måtte skifte koden til å tilpasse dette. Dette kan være komplisert for oppdragsgiveren i fremtiden, noe vi ønsker å unngå helst.

Et annet problem ble ytelsen. Fordi arkitekturens ytelse avhengte av enheten til brukeren, så mente vi først at dette var noe som avhengte av brukerne selv. Så lenge det falt innen våre forventninger vi hadde for våre maskiner, så ville vi ha ansett det som godt nok til å fortsette. Denne tanken var dessverre endret når vi så at prototypen selv tok ca. 20 sekund på å laste inn i nettleseren.

Dette var klart et bekymringsområde for oss nå, så vi sjekket både hastighet og minnebruk av vår prototype og to andre demoapplikasjoner (tre-på-rad[5], K-means[6]) for PyScript, hvor vi fant følgende resultater:

	Prototype	Tre-på-rad	K-means
Gjennomsnitt	582 MB	127 MB	1012 MB
Median	517 MB	110 MB	1200 MB

Tabell 6.3: PyScript - Minnebruk

	Prototype	Tre-på-rad	K-means
Gjennomsnitt	21.946s	3.094s	20.947s
Median	22.71s	3.06s	20.79s

Tabell 6.4: PyScript - Lastehastighet

Testene sjekker for minnebruk og hvor lang tid det tar for nettsiden å laste inn. Den sjekker også hvor mye av dette er rammeverket selv, med å ta i faktor en demo uten mye kode ([5]), og en annen med flere biblioteker og utregninger ([6]). Minnebruket ble målt gjennom funksjonalitet i Google Chrome, og lastehatighet med en stoppeklokke. Det ble utført 10 tester av hver kandidat, hvor gjennomsnitt og median ble tatt.

[4] `https://docs.pyscript.net/2024.5.2/user-guide/dom/#FFI`, hentet 19.05.2024
[5] `https://examples.pyscriptapps.com/tic-tac-toe/latest/`, hentet 21.05.2024
[6] `https://examples.pyscriptapps.com/kmeans-in-panel/latest/`, hentet 21.05.2024

Gjennom bruk av disse tester, bekrefter vi våre bekymringer - og ser at PyScript *kan* være ressursintensiv for klienten, hvor det avhenger av typen av applikasjon. For enkle applikasjoner brukes det relativt vanlig tid og minne sammenlignet med vanlige nettsider - men for avanserte situasjoner (som vårt prosjekt), så drar både minnebruk og lastehatighet høyt opp - med nesten 600 MB og 20 sekund for å laste inn. Dette gikk langt over våre forventninger av en slik arkitektur, og vil virke negativt mot det senere i vår beslutning.

Det er også viktig å nevne at det ble funnet at dette rammeverket hadde problemer med noen nettlesere (som Safari[7]), noe som kan peke til at denne løsningen støttes ikke av alle aktuelle nettlesere - en annen negativ ulempe til dette.

Til tross av disse ulemper, fantes det fortsatt en rekke fordeler med denne prototypen:

- **Klient-basert Python:** Som tidligere skrevet, så er fortsatt en stor fordel av en 'alt-i-nettleser' løsning at det kjøres hos klienten - til tross for våre resultater diskutert ovenfor. Selv om det tar lang tid å laste inn, så kan det være fordelsaktig dersom 'klient-tjener' modellen har også høy responstid for forespørsler.
- **Pythons økosystem i nettleser:** Fordi vi kan kjøre Python i nettleseren, betyr det at vi kan kjøre alle av dets biblioteker og rammeverk direkte hos en klient - noe som er en stor fordel når det kan kombineres med JavaScript i nettleseren for å oppnå utrolig mye funksjonalitet.

6.1.3 Diskusjon & utvalg av arkitektur

La oss starte diskusjonen med å først se på tre punkter av relevans for diskusjonen; testene brukt, hvordan utviklingen føltes, og fordeler/ulemper mellom dem:

1. **Testene:** Vi ser at testene indikerer høyere tilgjengelighet og mindre ressursbruk hos 'klient-tjener' enn 'alt-i-nettleser'. Mens en av dem bruker omlag 20s på å laste inn og samtidig bruker litt over 500 MB av midlertidig minne, så ser vi at den andre prototypen utgir resultatene relativt øyeblikkelig og med mindre data brukt - sammen med evnen at dette kan optimaliseres gjennom modulisering. Med dette i tanke, mente vi at 'klient-tjener' vant enkelt når det gjaldt testene.
2. **Utvikling:** Vi mener også at det føltes tryggere å programmere gjennom 'klient-tjener' prototypen enn den andre. Vi erkjenner at 'alt-i-nettleser' er fortsatt 'ny', men det føltes rat å programmere både JS og Python gjennom bibliotekene tilgjengelig i rammeverket. Det var også vanskeligere med tanke på programfeil. Imens hos den kjente 'klient-tjener' prototypen med Flask og React, følte vi oss langt mer sikker med rammeverkets anerkjennelser og innebygd støtte.

[7] `https://www.apple.com/safari/`, hentet 21.05.2024

3. **Fordelene og ulempene:** Med å sammenligne dem, ser vi allerede en stor kontrast. 'Alt-i-nettleser' har flere ulemper utenom de diskutert tidligere, som tydeligvis ustøttet bruk i noen nettlesere. Hos vår andre arkitektur, ser vi kun beregningsfeilen, men dette er noe vi antar kan fikses raskt når det undersøkes nærmere på. Med dette, vant 'klient-tjener' igjen.
   Det er også mulig å si at 'klient-tjener' som en arkitektur har bedre bakgrunn enn den andre arkitekturen. Selv om 'alt-i-nettleser' er laget av svært anerkjente utviklere, så er det fortsatt ikke sikkert at Wasm blir den nye normen i den snare fremtiden. Vi vet ikke om klientbaserte webapplikasjoner gjennom dette blir noe stort i fremtiden. Dette er ikke et problem for 'klient-tjener', for å bruke rammeverk som Flask og React har dannet flere tilgjengelige nettsider på internettet allerede. De er svært populære, og er essensielt normen for flere utviklere. Vi kan si oss mer sikre for oppdragsgiver å bruke denne over noe som er mer 'eksperimentelt' som 'alt-i-nettleser'.

En av de største fordelene med 'klient-tjener' prototypen var at beregningene ble utført på serveren. Med dette unngår vi at brukerens datamaskin blir en flaskehals for ytelse. Dette betydde at vi kunne håndtere tyngre beregninger og mer komplekse oppgaver uten å bekymre oss for brukerens maskinvare. Dette ville være spesielt viktig for vårt prosjekt, som krevde omfattende beregninger.

Vi mener allikevel at det finnes noe potensiale for 'alt-i-nettleser' prototypen om fremtiden. Som sagt, så er PyScript og WebAssembly relativt nye teknologier som fortsatt venter på komplette implementasjoner hos nettlesere. Når dette er komplett og fullført med flere optimaliseringer som reduserer innlastingstiden og minnebruk, antar vi at rammeverket kan da være mer ideell for oppgaven.

Med alt dette i tanke og de andre ulempene diskutert, måtte vi si at 'klient-tjener' prototypen var derfor den beste arkitekturen for oss å velge, og med dette, fant vi vår arkitektur for oppgaven.

6.2 Sprint 3-5

Dette delkapittelet representerer de neste tre sprintperioder etter vår prototypefase av mulige løsninger - hvor vi videreutvikler på 'klient-tjener' prototypen vi hadde laget tidligere for å virkelig begynne prosjektets utvikling til resultatmål.

6.2.1 Utregningsfunskjoner

Det første steget var implementasjon av moduliserte utregningsfunksjoner basert på de allerede eksisterende hos `compute.py`[8]. I deres opprinnelige former hos basisprogrammet (og slik prototypen hadde gjort det), så hentes alt av relevante utregninger fra den kollektive `compute_tabulated()`, ansvarlig for å utføre alle

[8] `https://github.com/ifarup/ciefunctions/blob/master/tc1_97/compute.py`, hentet 21.05.2024

	Modulariserte funksjoner			Direkte fra compute.py		
	P1	P2	P3	P1	P2	P3
LMS	29 ms	27 ms	24 ms	27 ms	29 ms	31 ms
MacLeod	49 ms	39 ms	41 ms	226 ms	265 ms	232 ms
Maxwellian	31 ms	36 ms	30 ms	214 ms	216 ms	223 ms
XYZ	130 ms	374 ms	379 ms	180 ms	489 ms	487 ms

Tabell 6.5: Vurdering av modulariserte utregningsfunksjoner mot de tilgjengelig fra compute.py

`compute_x` tilbehørene til deres korresponderende fargematchfunksjoner. Vi mente at dette var en dårlig løsning for oss, for utregningen av alle ville ha kun gjort applikasjonen saktere. Derfor, en moduliseringen av funksjonene ville ha gjort den raskere. Det er også ikke lov for oss å lagre tidligere resultater (med tanke på REST sitt prinsipp om statsløshet), så vi kunne enten bruke `compute_x` funksjonene individuelt, eller lage våre egne moduliserte.

Vi mente at det fantes noe god praksis i å lage våre egne moduliserte versjoner med tanke på ytelsen. Noen av de opprinnelige funksjonene vil ta mer tid for å produsere utregninger fordi de måtte utføre det både for en `result` og en `plot`. Men vår løsning på dette punktet kunne kun gi en av dem om gangen, så det var lite mening i å regne begge ut. For å teste om denne tanken hadde mening, laget vi moduliserte versjoner av et par fra `compute.py`. De ble sammenlignet med deres opprinnelige motparter med tanke på hastighet.

Dette ble gjort for endepunktene tilbehørene LMS, Macleod-Boynton, Maxwellian og XYZ, hvor det ble testet med tre parametere for hver funksjon. Resultatene av dette vises frem i tabell 6.5.

Vi ser i tabellen at for det spilte ingen rolle for LMS, og at tiden forble den samme. Men for MacLeod og Maxwellian spesielt, undersøkte vi en stor reduksjon i responstid. Det fantes også en merkbar reduksjon hos XYZ, en utregningsfunksjon kjent til oppdragsgiver å ha tatt relativ 'lang' tid. Dette var nok for oss å bestemme at vi ønsket moduliserte funksjoner, så en del av sprinten gikk til videreutviklingen av dem.

For å gjøre det enkelt å finne feil i våre moduliserte utregninger og/eller programfeil, laget vi også en improvisert plattform for testing i form av et endepunkt (slik vist frem i figur (6.5)). Dette ble utført med å bruke basisprogrammet til å eksportere forskjellige utregninger til `csv` filer - hvor filene ble etterpå sammenlignet til utregningene vårt program laget med hjelp av Pandas igjen.

```
{
  "LMS": true,
  "LMS-LOG10": true,
  "LMS-BASE": true,
  "LMS-PLOT": true,
  "MB": false,
  "MB-PLOT": false,
  "MW": true,
  "MW-PLOT": false
}
```


Figur 6.5: Skjermbilde av resultat fra plattformen for testing.

6.2.2 Utregningspresisjon

Som skrevet tidligere, var et av problemene som fantes i prototypen feil med beregningspresisjon. Fordi vår målgruppe er forskere som krever nøyaktige tall fra utregninger, så er dette et problem som må takles seriøst og riktig. Derfor, ble det utført en stor innsats til dette over perioden.

Vi visste at det var et problem med selve eksportingen av utregningene. Fordi vi brukte de opprinnelige funksjonene i prototypen, kunne dette umuligvis ha vært forårsaket av egen kode. Derfor, så måtte det ha vært noe i hvordan den blir til JSON som forårsaker dette til tallene. Med dette, var det vurdert flere løsninger, men ingen av dem virket ideell for oss:

- Direkte bruk av json[9] fra standardbiblioteket ga programfeil, for den hadde ikke evnen til å eksportere en `numpy.ndarray`[10] (multidimensjonell array) til JSON.
- En egen JSON-enkoder var implementert og testet, men presisjonsproblemet fortsatte, for kodingen skrev tallene som direkte som de fantes i programmet.
- Å lagre tallene i en mini database som SQLite[11] momentant var en vurdert ide, men det var dømt for ressursintensiv å lage for denne hensikt.

Kun etter at vi fant Pandas, fant vi en mulig løsning for dette problemet. Vi kunne først konvertere utregningene om til en dedikert `pandas.DataFrame` type tilgjengelig i biblioteket. Gjennom datastrukturen, fikk vi tilgang til mange sterke metoder. En av dem var `.to_json()`[12] hvor spesifikke verdier av parameteren `double_precision` forandret på mengden av desimaltall hos tallene i utregningene. Med denne metoden, la vi merke til at flere tall ble uttrykt riktigere i den resulterende JSON innmaten hos responsen.

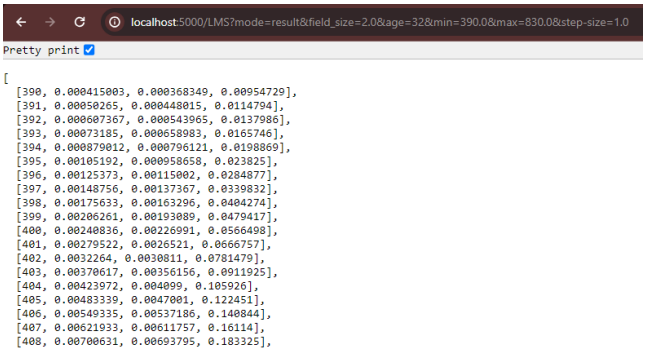
[9] `https://docs.python.org/3/library/json.html`, hentet 20.05.2024

[10] `https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html`, hentet 20.05.2024

[11] `https://docs.python.org/3/library/sqlite3.html`, hentet 20.05.2024

[12] `https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_json.html`, hentet 20.04.2024

Denne løsningen hadde initielt kun en dårlig side, og det var at vi måtte forlate eksporteringen med `dict`. Pandas støtter kun spesifikk eksportering, og kan ikke utføre metoden på en `dict` med flere datastrukturer i seg. Datastrukturen vi hadde fra prototypen måtte derfor forlates for denne nye metoden. Som et følge, måtte en eventuell bruker ha sendt flere forespørsler for å få all utregningsinformasjon, noe vi ser på som et negativt problem.



Figur 6.6: Skjermbilde av utregningsresultater etter Pandas løsning.

Allikevel, godtok vi risikoen og mente at dette måtte gjøres for å få de riktig tallene. Dette var inntil vi fant ut at det allikevel fantes problemer med denne metoden. Fordi metoden kun øker mengden av desimaltall (og indirekte presisjonen av et tall) så betyr det at tall med mange desimaler ble påvirket positivt, men tall med mange signifikante sifre og heltall ble derimot påvirket negativt.

Alle tall med en mengde av signifikante sifre som var over terskelen av desimaltall tillatt i en `pandas.DataFrame` eksportering ble rundet av, og derfor ikke utlevert i riktig form. I tillegg, enkle heltall i utregningene begynte å selv få de samme presisjonsproblemene man finner i utregning med desimaltall, noe vi fant ut oppstod når parameteren var satt for høyt. Begge av disse er alvorlige problemer som måtte tas opp videre i utviklingen.

Det ble også funnet et seriøst problem med testplattformen vi hadde brukt inntil nå for å forsikre oss at utregninger var riktige. Fordi `pandas.DataFrame` kan kun støtte en viss mengde av desimaler slik fortalt over, så ble det snart klart at den kunne ikke muligens ha de riktige tallene for de utregningene. Vi fant flere problem når vi undersøkte dette videre. Med bruk av debuggingsverktøy i PyCharm, fant vi dessverre ut at selve innlastingen av filene for testing var preget av kalkulasjonsfeil. Dette betydde at vi måtte også revurdere og restrukturere dette i fremtiden.

6.2.3 URL & parametere

En annen ting utviklet i denne perioden, var skiftet fra å bruke `POST` forespørsler med JSON innmat, til å heller bruke URL med betydningsfulle søkestrenger.

Dersom vi skulle lage en RESTful tjeneste, var det viktig å følge prinsippene til det.

Vår prototype følgte ikke det sjette prinsippet; dersom vi tenker på alle utregninger som unike ressurser, så ville deres fellesbruk av kun en URL ha brutt prinsippet. I tillegg, så er bruken av `POST` forespørsler for 'å hente informasjon' ikke en norm [13], for det er designert at den metoden er spesifikt for når brukere skal tilsette informasjon til en tjeneste med hensikt på å oppdatere den.

```
.../api/v1/funksjon?parameter1=x&parameter2=x&...
```

Figur 6.7: Struktur av URL ved første sprint.

Derfor, forandret vi først tjenesten til å heller kun ta `GET` forespørsler, for det var den riktige metoden spesifisert i HTTP for å hente informasjon[14]. For å passere videre informasjon om brukerparametere, valgte vi å gå for søkestrengene som en del av URL for å følge det sjette prinsippet, og gi hver unike beregningsressurs en unik URL. Vi endte opp eventuelt med et system som likner det funnet i figur 6.7, og var fornøyde med dette systemet for nå.

6.2.4 Stil & utforming

For frontend, ble fokusert en stor del på av perioden på stil og utforming for å sikre at applikasjon ikke bare fungerte som den skulle, men også at den var estetisk og brukervennlig.

Sammen med bruken av React og Typescript, brukte vi også og for å designe komponentene både visuelt estetisk og visuelt strukturelt for å holde oss innenfor kravene vi hadde laget tidligere i prosjektet.

For å gjøre dette, eksperimenterte vi med flere teknologier for å både plassere komponentene på riktige steder hos nettsiden - samtidig som at det skulle være enkelt å oppdatere for fremtiden. Derfor, gikk en stor del av sprinten til dette - hvor gjennom flere iterasjoner, landet vi med `Flex`[15] hos CSS. Vi hadde vurdert og testet `Grid`[16] og Bootstrap[17], men `Flex` ble utvalgt fordi det var enkelt å sette opp og bruke når man ble mer vant til det.

6.2.5 Modulisering

En annen stor del av denne sprinten ble brukt på å dele opp koden hos frontend inni moduler. Ved å modulisere prosjektet, kunne vi gjøre koden mer organisert

[13] https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST, hentet 19.05.2024

[14] https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET, hentet 19.05.2024

[15] `https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Flexbox`, hentet 20.05.2024

[16] `https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Grids`, hentet 20.05.2024

[17] `https://getbootstrap.com/`, hentet 20.05.2024

og lettere å vedlikeholde for fremtidige utviklere av dette prosjektet. Å modulisere koden ville også ha skapt et godt grunnlag for resten av applikasjonen. Dette var en kontinuerlig prosess som vedvarte gjennom hele utviklingsfasen av brukergrensesnittet, men som vi startet med allerede nå.

6.2.6 Konstruksjon av URL i frontend

Etter som tjeneren ble mer og mer RESTful, måtte vi utvikle et nytt og bedre system for å kommunisere med den. Dette ble gjort med et samarbeid mellom en `StringBuilder`, nedtrekksmeny og brukerspesifiserte parametre. Når brukeren velger en ny funksjon i nedtrekksmenyen, vil `StringBuilder` hente informasjonen sammen med eventuelle parametere til å bygge en ny URL for å passe det nye systemet.

6.3 Sprint 6

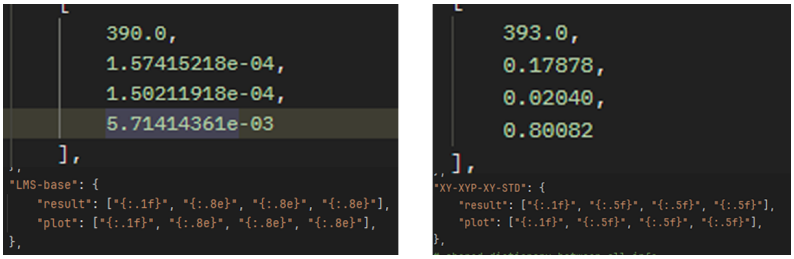
6.3.1 Stringformatering for JSON

For backend i denne sprinten, ble det ettersøkt videre hvilke metoder kunne fikse presisjonsproblemet fra den forrige sprinten. Selv om det var funksjonelt til en viss grad, så var det ikke tillatt av oss å la det være slik, spesielt med tanke på at vår målgruppe er forskere som trenger presise tall. Derfor, fortsatte søket inntil vi så nærmere på problemfeltets teori, og fant et interessant funn for prosjektet.

Denne formen av feil er overalt hos datamaskiner som tar i bruk desimaltall - og i felt hvor det er viktig å ha de riktige tallene (som økonomi), så har det blitt utviklet en rekke triks mot det som vi hadde ikke sett nærmere på tidligere. Et av dem er å uttrykke tallene som `string` slik vi hadde forsøkt tidligere, men i tillegg også spesifisere presisjon gjennom formatering. Et relevant eksempel er PayPal sin API[18]. Vi mente først at dette var en blindvei, for vi hadde ingen slike formater tilgjengelig direkte. Å forsøkte å gjette de var heller ikke et godt tegn. Det var kun når vi så nærmere på koden til basisprogrammet, at vi fant at dette var eksakt hvordan de opprinnelige utviklerne hadde taklet oppgaven, sammen med spesifikke formater oppstilt i koden[19]. Kun ved dette funnet, fant vi ut at dette var løsningen son både kunne brukes for å oppnå riktige tall, men også måtte for å oppnå komplett konsistenthet med basisprogrammet.

[18] `https://developer.paypal.com/docs/api/payments/v2/`, legg merke til `currency|medlemmet`, hentet 20.05.2024

[19] `https://github.com/ifarup/ciefunctions/blob/master/tc1_97/table.py#L155`, eksempel her, hentet 21.05.2024.



Figur 6.8: Eksempler av utregninger fra backend sammen med deres formateringsstruktur under.
I det første bildet, er det deklarert bruken av standardform med de første 8 desimaltall. I den andre, er det deklarert bruken av kun de første 5 desimaltall.

Det var opprinnelig forsøkt å se etter en tilgjengelig enkoder som tillate spesifikk formatering. Dette søket kom med ingen resultat, for det var et nisje ønske og ble ikke funnet. Dette preget oss til å igjen ta opp ideen om en egen JSON-enkoder, kun med evnen til å bruke en viss format når ønsket. Denne teknikken endte opp med å fungere så godt at den er fortsatt implementert i programmet. Sammen med dette, tillot dette oss å ekspandere ressursene vi utga fra kun en `numpy.ndarray`, tilbake til `dict` slik vi hadde det i prototypen. Dette er fordi vi kunne videre utvikle på enkoderen til å inkludere dette enkelt. Gjennom dette, var presisjonsproblemet løst på en måte som indirekte gjenbruker og oppnår konsistenthet med basisprogrammet.

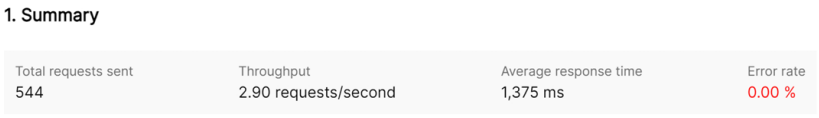
6.3.2 Vurdering & skifte av rammeverk

Etter at vi fikk de riktige resultatene hos utregningene i kapitlet over, antok vi at vi hadde kommet til et punkt hvor backend var nesten helt fullført for prosjektet. Derifra, mente vi at det eneste som gjenstod var ordentlig implementasjon med det for frontend (og eventuelle endringer det forårsaker). Med basis i denne antagelsen, ble det utviklet tester for å teste applikasjonen; først enkle enhetstester for kodekvalitet, men deretter stresstesting for å teste dets skalerbarhet. Dette var hvor vi innså at vi hadde gjort en kritisk feil i utviklingsprosessen.

Ett av våre krav var at applikasjonen måtte være asynkron - men fordi Flask er et WSGI, støttet det ikke innebygd asynkronisme. Dette kravet kunne ha blitt argumentert mot dersom applikasjonen oppfylte kravet om '*maksimalt 1000 ms responstid*' i stresstestet. Om programmet hadde klart å passere kravet for stresstesten, kunne vi ha argumentert at det fantes ikke lenger noe grunnlag for optimaliseringer som det. Dette betydde at vi kunne se vekk fra asynkronismen, men vi fant snart at det var nødvendig allikevel, som vist i resultatene hos figur 6.9.

I løpet av testen, fant vi ut at den gjennomsnittlige responstiden var 1.4 sekund - hvor noen utregningsfunksjoner[20] hadde gjennomsnittlig responstid på 3.5—4.3 sekund. Grunnen til dette var klar når man tar i faktor at en stresstest sender ut

[20] Spesifikt hos endepunkter /xyz og /xy.

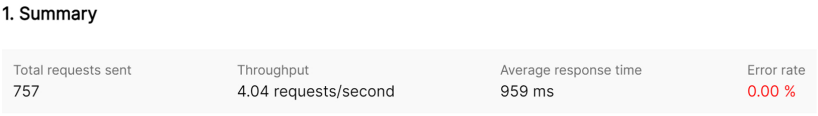


Figur 6.9: Resultater fra stresstest hos vanlig Flask rammeverk.

flere forespørsler på samme tid: Flask sine synkrone evner kan kun takle en forespørsel om gangen, så flere ender opp med å vente i en kø. Selv om dette var en stresstest med større mengder forespørsler enn det som er forventet, mente vi at dette var ikke akseptabelt.

For denne grunn, gikk resten av sprinten til å undersøke mulige løsninger for å gjøre rammeverket raskere. Vi mente at det var best å se på andre teknologier og utvikle ufullstendige demovarer av dem, kun med utregningene vi hadde nå. Gjennom det, kunne vi teste dem på lik grunnlag som vanlig Flask, finne ut hvilken ga best ytelse, og fortsette med det. Vi valgte tre mulige løsninger, og undersøkte dem videre:

Flask Aysnc



Figur 6.10: Stresstestresultater for flask async.

Problemet vi diskuterte over, var at ventetiden var for lang og at rammeverket tok kun vare av en forespørsel om gangen. Selv om rammeverket kan ikke endres på hvordan det takler forespørsler (fordi det er ikke en del av WSGI sin struktur, og alle forespørsler vil ta en lik mengde tid), noe som *kan* endres på er den direkte ventetiden applikasjonen bruker. Med å ta vare av en annen forespørsel mens systemet venter på svar fra seg selv om utregningene, så er det mulig å minimalisere ventetiden. Dette er prinsippet bak `flask async`[21]. Med å implementere `async` og `await` inni rammeverket, åpner vi opp til at den kan takle flere forespørsler på samme tid gjennom metoden over.

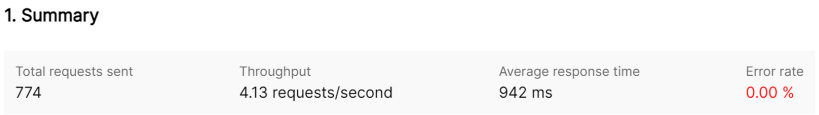
Gjennom dette, kan vi bekrefte at ventetiden var hovedproblemet vi hadde hos det opprinnelige rammeverket gjennom stresstester utført her (vist på figur 6.10). På testene, ser vi en umiddelbar reduksjon i responstid ned til 959 ms, sammen med en 139.3% raskere prosessering av forespørsler i sekundet. Dette er

[21] `https://docs.python.org/3/library/asyncio-task.html`, hentet 20.05.2024

i tillegg til at det var enkelt å implementere, og har sikkert videre potensiale for mer optimalisering gjennom bruk av flere `await` for spesielt intensiv kode.

Allikevel, er det viktig å peke ut at de samme utregningsfunksjonene som tok lang tid hos det opprinnelige rammeverket, tok allikevel over 1000 ms her også. De hadde gjennomsnittlig responstid innenfor 2.4 — 2.9 sekund. Dette var ikke kravsoppfyllende, men allikevel en stor forbedring og god start.

Flask med Gevent



Figur 6.11: Stresstestresultatet for Gevent.

En annen løsning innenfor Flask for asynkron takling av oppgaver, er bruken av biblioteket Gevent[22], med åpen kildekode og MIT lisense [23] . Dette biblioteket kombinerer bruken av korutiner sammen med spesielle 'grønne tråder' fra Greenlet[24] for å oppnå asynkron takling av forespørsler - lik hvordan `async` og `await` takler det oppe. Fordelen denne har over dem, er at det kreves *enda* mindre kode å implementere dette inni en allerede eksisterende Flask applikasjon.

Allikevel, med basis på figur 6.11, ser vi at ventetiden forblir hovedsakelig den samme. Dette forårsakes av ingen stor endring i rammeverket fra den tidligere sett `async` løsningen, i og med at begge er kun veier man kan takle ventetiden mellom utregninger på. I tillegg, ser vi en liten minimal reduksjon i responstid for de utfordrende utregningsfunksjonene, hvor deres responstid tok nå 2.2 — 2.8 sekund i gjennomsnittet.

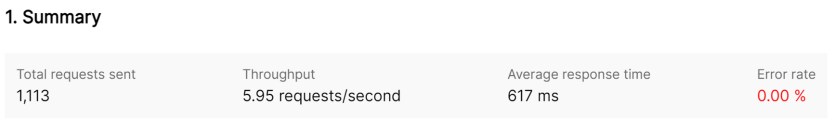
Sanic

Vi ser i de to tidligere løsninger at de *hjelper*, men ikke så mye. Dette er fordi uansett hva man gjør, så er Flask fortsatt ett WSGI. Dets hele struktur og basis er spesifikt for synkrone forespørsler. Om vi skal takle asynkrone forespørsler, er det derfor naturlig å velge et asynkront rammeverk.

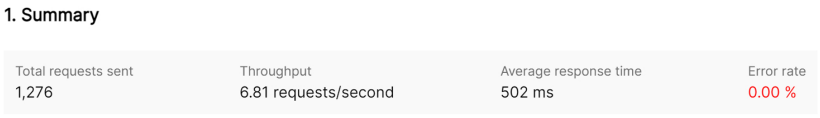
Det finnes et stort utvalg av disse (som FastAPI og nyere versjoner av Django), men gjennom en anbefaling fra en venn og videre undersøkelse i det, ble rammeverket kalt Sanic valgt. Vi ønsket ikke å undersøke hvert eneste slikt rammeverk, så vi valgte dette som vår eneste representant av ASGI-slaget. Rammeverket ble også valgt fordi den hadde lik syntaks som tjenesten vår hadde allerede, så det var enkelt å implementere.

[22] `https://flask.palletsprojects.com/en/3.0.x/deploying/gevent/`, hentet 20.05.2024
[23] `https://github.com/gevent/gevent`, hentet 20.05.2024
[24] `https://greenlet.readthedocs.io/en/latest/`, hentet 20.05.2024

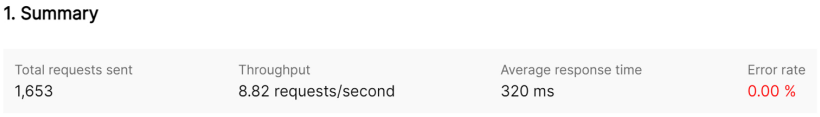
For testene, tok vi bruk av dets evne til å øke skalerbarhet, og kjørte tre tester med forskjellige variasjoner i arbeidsprosesser. Ett av testene hadde kun en arbeidsprosess for å rettferdig sammenligne det med tidligere tester.



Figur 6.12: Stresstestresultat for Sanic (1 arbeidsprosess).



Figur 6.13: Stresstestresultat for Sanic (2 arbeidsprosesser).



Figur 6.14: Stresstestresultat for Sanic (4 arbeidsprosesser).

Vi ser at alle tester gjør det merkbar bedre i hastighet enn demoene fortalt tidligere. Vi mener at dette forårsakes av den innebygde asynkroniteten i rammeverket, og dets bruk istedenfor kun enkle korutiner. Dette kan bevises gjennom den første testen, hvor det brukes kun en arbeidsprosess, og allikevel er den raskere på responstid enn alle implementasjoner med Flask. På den spesifikt, ser vi at dets gjennomsnittlig responstid på de utfordrende utregningsfunksjonene sank til $800 - 912$ ms; godt innenfor våre krav.

Denne responstiden kan forbedres ved økning av arbeidsprosesser for økt horisontal skalerbarhet (gitt at maskinvare har nok ressurser). Gjennom dette, ser vi enda mer reduksjon i tid, først fra 502 ms (2 arbeidsprosesser), og deretter 320 ms (4 arbeidsprosesser). For den siste, reduseres responstiden hos de utfordrende funksjonene helt ned til $619 - 837$ ms, en storartet forbedring i ytelse sammenlignet med de tidligere undersøkt biblioteker.

Skifte av rammeverk

Det er klart å se (selv med videre implementasjoner fra biblioteker med korutiner), at Flask kan ikke forbedres til å takle kravet holdt av denne oppgaven. Det ble utkonkurrert komplett av ASGI-baserte rammeverk. Selv om det finnes basis i å dra videre med sammenligninger av flere slike rammeverk for å se hvilken er

raskest, så mener vi at vi har sammenlignet nok rammeverk for helheten av oppgaven. Vi kutter det der, og bare fortsetter med Sanic som vårt nye rammeverk.

Hele følgende skifte og implementasjonen til Sanic gikk veldig raskt, takket være dets fordel av et Flaskaktig syntax. For sikkerhetens skyld for å se om våre resultater var nøyaktige, kjørte vi en siste stresstest med alle funksjonaliteter, hvor vi igjen fikk like resultater.

6.3.3 Annet hos backend

Sidemeny & URL

Det var også i løpet av denne perioden at det ble bestemt at det er best å la sidemenyet være en addisjonell ressurs som avgis av backend. Dette ble gjort med tanke på å redusere mengden av forespørsler nødvendig til serveren for å lage det gjennom frontend selv. Vi ser at det kan kreve alt fra 1-3 innkallinger til forskjellige funksjoner for frontend å gjøre det selv. Derfor, for å redusere dette til kun en, utfører vi heller ressursutdelingen hos backend selv som en optimalisering i prosjektet.

Vi visste at dette var mulig gjennom bruken av `<iframe>` elementet i HTML semantikken. Dersom vi fikk tjenesten til å avgi et HTML dokument som var dynamisk generert til en gitt URL, så ville det ha vært mulig for frontend av applikasjonen kalle til det gjennom URL. Denne implementasjonen ble hvordan vi endte opp å gjøre det i den resulterende programvaren, og dekkes derfor mer i kap. 7.2.2.

Sammen med dette, måtte vi endre på strukturen av våre URL. Inntil nå, har alle av våre URL til backend vært i form av figur 6.7, hvor den avgir kun utregningsresultater som den eneste ressurs. Men med introduksjonen av et nytt ressurs, måtte vi restrukturere på dette. Vi valgte å gå for å bruke funksjonen som en del av stien, hvor man deretter spesifiserer formen av ressurs (nå enten `/calculation` eller `/sidemenu`). Med dette i tanke, restrukturerte vi URL'ene våre til se slik ut (figur 6.15):

```
.../api/v2/funksjon/ressurs?parameter1=x&parameter2=x&...
```

Figur 6.15: Struktur av ny URL for ny ressursutgivelse.

Dette gjør det mer riktig med tanke på REST, samtidig som det ikke forandrer alt for mye på systemet vi har på plass allerede. Etter at dette ble laget, var implementasjonen av sidemeny startet og avsluttet på samme dag.

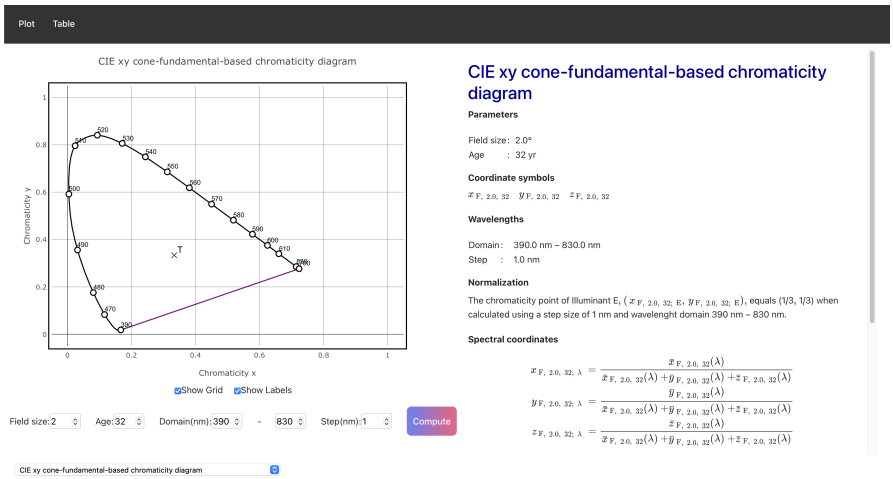
Plattform for testing

Testplattformen vi hadde laget tidligere (selv om den var preget av problemer) var allikevel svært viktig å ha med oss. Å kunne sammenligne utregninger fra vårt program og basisprogrammet underveis i utviklingen er et nødvendig steg vi mente var absolutt kritisk å inkludere. Derfor, så var det også fokusert noe innsats her for å finne en løsning til dette.

Vi flyttet først testplattformen til en egen enhetstestingmodul gjennom Unittest slik at de kunne både sjekke resultatene og øke kodekvaliteten. Etter det, forsøkte vi et par mulige løsninger. Vi bestemte oss om å fortsette bruken av Pandas, men å umiddelbart konvertere det til et `numpy.ndarray` fra Numpy. Dette ga oss ikke de samme feilene til tross fortsettelsen av Pandas bruken, så vi antok at den oppbevarer kvaliteten av tallene til en nok grad til at det ikke påvirkes av kalkulasjonsfeil. Dette ble vår endelige løsning for testplattformen, og den vi fortsetter å bruke i våre tester videre i utviklingsperioden, med mer detaljer tilgjengelig i kapittel 8.1.1.

6.3.4 Plotting av diagrammer

Mye av utviklingen i frontend for denne sprinten gikk til å utvikle brukergrensesnittet, sammen med visualiseringen av diagrammer og data gjennom Plotly. Målet var å lage og forbedre diagrammene slik at de ble mer informative. De eksisterende funksjonene for datahåndtering ble oppdatert slik at de kunne hente viktige plotpunkter fra vår tjeneste. Dette inkluderte utregninger for elementer som hvitpunktet til grafene, plotpunkter for spesielle bølgelengder og endepunktene til purpurlinja hos diagrammet.



Figur 6.16: Brukergrensesnittet etter sprint 6
Purpurlinja er den lilja-fargede linja på bunnen av hesteskokurven. Hvitpunktet er punktet i midten av diagrammet. Plotpunktene for bølgelengdene er punktene på selve kurven.

Disse dataene forberedes og manipuleres med funksjoner som `AddSpecificWavelengthPoints` for punktene på buen i grafen og `prepareChartData` for hvitpunktet og purpurlinjen. Disse funksjonene sikrer at alle nødvendige punkter blir plottet riktig på grafen.

6.4 Sprint 7 - Slutt

6.4.1 Nytt ressursystem

Det var nå antatt at vår backend kunne avgi alle utregninger nødvendig for diagrammene å illustreres hos frontend, slik vist på figur 6.16. Allikevel fantes det fortsatt et stort problem vi igjen ikke innså.

Mens noen av våre funksjoner krevde kun en forespørsel for å få alt av utregninger nødvendig (som `/lms`), så krevde andre flere utregninger fra andre funksjoner (for eksempel `xy-p`, trenger `xy` for hesteskokurven). I basisprogrammet, ble alt for ett gitt sett av parametere utregnet på samme tid og lagret til et globalt `dict`, så man kunne enkelt referere til utregningene tidligere laget hos datastrukturen. Dette var ikke mulig for oss fordi vi moduliserte funksjonene for å unngå eksakt dette. På grunn av dette, krevde frontend å sende flere forespørsler. Vi mente at dette var ikke en intuitiv løsning, og ønsket gjerne å redusere denne mengden til et minimalt.

Som en rask løsning, ble derfor ressurssystemet for utregningene forandret fra å kun inneholde direkte resultater fra `computemodularized.py`, til å også inneholde utregningene inkludert de for valgfri parametere. Det ble også iverksatt at alle nødvendige komponenter fra andre funksjoner (som hesteskoen nevnt over) var også utregnet og inkludert i ressursene. Vi essensielt gjorde det mindre modulisert, og mer likt basisprogrammet. Dette skiftet gjorde det mulig for frontend å kun sende *en* forespørsel for alt av informasjon, noe vi anså som positivt.

Men, denne endringen forårsaket et nytt problem i form av store forandringer; en stor del av parametersystemet måtte endres til å stoppe bruken av valgfri parametere. Endringen betydde også at flere av de moduliserte funksjonene mistet deres mening i å være modulisert. Dette påvirket ytelsen og tjenestens evne til å kjøre. Dette ga oss stress, sammen med tanken at frontend var ikke ferdig med alle diagramfunksjonaliteter ennå, og tidsrammen for utvikling var nærme.

6.4.2 Tilbakerulling & diagramendepunkt

Som både en løsning på systemskiftet oppe *og* for å få flere funksjonaliteter for diagrammer enkelt, så ble det oppdaget at backend kunne selv ta vare av den oppgaven. Med tanke på hvordan sidemeny var en ressurs i form av representasjonen HTML som ble tilsatt til en nettside gjennom `<iframe>`, så kom vi til ideen at diagrammet selv kan uttrykkes også som en slik ressurs.

En slik løsning ville ikke kun ha gjort det raskere enn før å starte diagrammer (med å direkte generere koden for dem gjennom serveren), men også ville ha

gjort skiftet av ressurser unødvendig. Vi kunne dra tilbake til det gamle systemet med de moduliserte utregningsfunksjonene. Gjennom bruken av diagramkoden allerede laget, ble det også veldig enkelt og raskt å implementere grafene med de manglende funksjonaliteter.

Det negative med dette, er at dette var trolig 'unødvendig'. Selv om det er mulig for tjeneren å avgi diagrammet som en ressurs, så er det best å la frontend gjøre det. Dette ville ha vært enklere og mer ytelseseffektiv for alle. Vi gjorde dette allikevel for å tilsette flere funksjonaliteter som hadde tidligere manglet.

Gjennom mye innsats over kort tid, ble diagramendepunktet snart utviklet inn i applikasjonen. Dette endte opp med å være implementasjonen vi brukte i det siste produktet, og er derfor vi ikke drar i mye detalj om det. Helle skal dets implementasjonsdetaljer dekkes i det neste kapittel.

6.4.3 Integrering av <iframe>, sidemeny & design

Ettersom det nå var utviklet en ferdigstilt løsning for grafene gjennom `<iframe>` levert av backend, måtte dette implementeres hos frontend. Det ble først gjort en opprydding i frontend delen av prosjektet, hvor alt med tilknytning til gammel implementasjon av plotting ble slettet. Det ble deretter opprettet nye React komponenter for å fremstille både plot og sidemeny gjennom `<iframe>` som hentes fra deres respektive endepunktene i APIet.

Denne løsningen er ressurseffektiv med tanke på antall API kall og håndtering av de ulike dataene tilknyttet tilleggsfunksjonene som logaritmiske verdier, sammenligning med standarder. Disse tilleggsvalgene ble tidligere returnert separat gjennom egne API kall til sine respektive endepunkter. Slik det er nå slipper man håndtering og lagring av disse relativt store utregningene hver gang man endrer fargematchfunksjon eller gjør en ny beregning med nye parametere.

Men selv om dette er en fordel med tanke på antall API kall og simplifisering av logikk for frontend, har det også noen negative sider for applikasjonen i en helhet. Det første er at det begrenser hvor tilpasselig `<iframe>` er til komponentene sammenlignet med om det ble plottet direkte som React komponent. Det andre er at siden `<iframe>` er ferdig bygd, er det også forhåndsbestemt hvilke tilleggsvalg som er aktive eller inaktive. For eksempel, om man slår av grid for så å endre en parameter og gjøre en ny kalkulasjon vil grid automatisk aktiveres igjen. Man vil da måtte bli nødt til å skru av eller på et gitt tilleggsvalg hver gang man gjør en endring som gjør at `<iframe>` blir lastet inn på nytt.

Det ble også implementert funksjonalitet for at sidemenyen blir vist under plot/tabell og parameterskjema dersom bredden på nettleservinduet beveger seg under en gitt størrelse. Dette gjør at applikasjonen er brukbar for flere typer skjermer og konfigurasjoner av nettleservinduet. Gjennom dette, oppnår vi det opprinnelige designet vi hadde planlagt gjennom vår wireframe, og implementert god responsiv design. Vi skal ta dette mer opp i neste kapittel gjennom figurer.

6.4.4 Utrulling

Etter dette, var det på tide å starte med utrulling. Hensikten med dette, er at det ikke skal være nødvendig å kjøre testmiljøene og starte de individuelle lokale serverne på lokal maskin hver gang man ønsker å kjøre applikasjonen. Vi ønsket å effektivisere prosessen for å starte applikasjonen, og å gjøre det mulig å få tilgang til den uten noen forberedelser på lokal maskin.

`requirements.txt` ble opprettet i backend og inneholder alt av avhengigheter som trengs å at apiet skal kjøre og utregning skal skje. Tilsvarende eksisterer det en fil: package.json i front en som brukes til det samme.

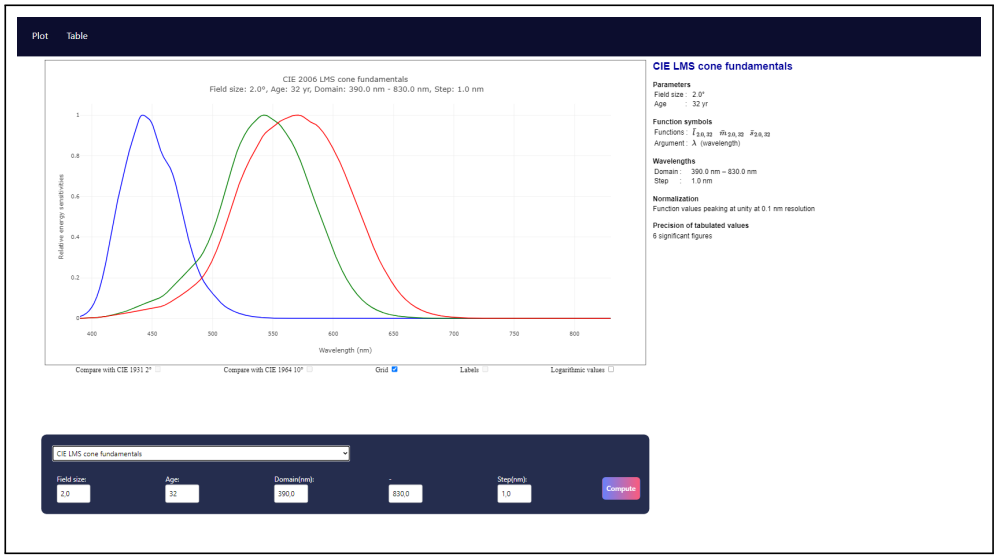
Dette ble gjort når vi ble ferdig med alle andre funksjonaliteter, og alt var klart til utrulling. Vi spesifiserer mer i detalj hvordan utrulling foregikk hos kap. 9.1.

Kapittel 7

Implementasjon

7.1 Frontend

Applikasjonens frontend har som oppgave å ta imot parametere og valg av fargematchfunksjon fra brukeren. Disse benyttes for hente data fra backends API, for så å grafisk fremstille dette for brukeren i form av plot, tabell og en sidemeny. Frontend er bygd opp av ulike React komponenter som alle tjener et spesifikt formål for en ønsket funksjonalitet. Disse utnytter seg av React sin virtuelle DOM for å raskt og effektivt oppdatere visning og tilstand. De ulike komponentene er designet for å være tilspissede og avgrenset til å kun håndtere en spesifikk utfordring. På denne måten blir koden lettere leselig, får en løsere kobling og høyere samhørighet. Ved å splitte funksjonalitet som ikke er direkte knyttet til en komponent til en egen klasse blir koden også langt mer modulær ved at ulike komponenter som alle avhenger av felles funksjonalitet kan importere og benytte seg av denne funksjonen. Man trenger da ikke å duplisere kode for hver komponent som avhenger av denne funksjonaliteten.
De ulike komponentene benytter seg av et sett hjelpefunksjoner og tredjepartisbiblioteker for å kunne oppnå ønsket funksjonalitet på mest mulig effektiv og ryddig måte. I App.tsx settes routing til graf og tabell. Her blir de ulike kompeonentene pakket inn i ParameterLayout, ParametersProvider og UseContentControllerProvider for å tilrette legge for manipulering av parametere og dynamisk rendering.

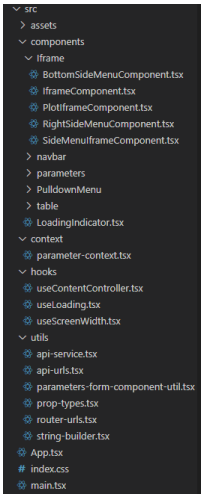


Figur 7.1: Slutt resultatet av frontend.

7.1.1 Kodestruktur

Kodestruktur for frontend kan ses i figur 7.2. Det er forsøkt å strukturere koden på en slik måte at det er intuitivt for utenforstående å navigere seg i applikasjonens filer. Alle filer faller under kildekode mappen `src`. Her har man en rekke underliggende mapper:

- `assets`: Alle ressurser som er knyttet til applikasjonen. Her finner man applikasjonens logo.
- `components`: Her finnes alle React komponentene. Disse er igjen inndelt i underliggende mapper for å gruppere hver komponent sammen med sin tilhørende CSS-fil.
- `context`: Filer som har ansvar for å håndtere kontekst og tilstand i applikasjonen.
- `hooks`: Egenkomponerte hjelpefunksjoner som har integrasjon av Reacts innebygde hooks. Logikken for disse er separert fra komponentene slik at de lett kan benyttes av alle komponentene som måtte trenge dem.
- `utils`: Akkurat som for funksjonene i `hooks`, er funksjonene man finner i `utils` hjelpefunksjoner med logikk som er adskilt fra den enkelte komponent for å oppnå en løsere kobling og høyere modularitet. Forskjellen mellom dem koker ned til at `utils` funksjonene er tilstandsløse.



Figur 7.2: Filstruktur for frontend.

I tillegg til et bevisst oppsett av filstruktur er det benyttet navnkonvensjon har til hensikt å gjøre det enkelt å forstå hvordan applikasjonen virker:

- React komponenter: PascalCase -> `MyComponent`
- Css- og typescript filer: kebab-case -> `my-component.css`
- Hooks og typescript funksjoner: camelCase -> `myComponent`.
  Hooks har i tillegg prefiks 'use', `useMyHook`.

Hensikten med dette oppsettet er som nevnt å gjøre koden så enkel og intuitiv som mulig for utenforstående, men også å få en applikasjon med høy modularitet, løs kobling og høy samhørighet.

7.1.2 Integrasjon med tjenesten

All integrasjon av interaksjoner med API er sentralisert i en egen fil kalt `api-service`, som man igjen finner i `utils` mappen. Funksjonen `fetchApiData()` tar inn parametere for endepunkt, som reffererer til valgt fargematchfunksjon, og brukerspesifiserte parametere. Den bruker så hjelpefunksjonen `stringBuilder()` for å bygge URL strengen for å hente ønsket data fra backend. Den bruker så en `fetch` funksjon med den bygde URL'en og spesifiserer at det er en GET metode som ønsker å hente et JSON svar. Dette svaret blir så lagret i en variabel kalt `response`. Det er også bygget inn feilhåndtering hvor det blir gjort en sjekk på `response`; om den er av riktig type, altså JSON. Om formatet er riktig blir dette returnert som data for `result`, som så kan brukes for å fylle inn data i tabellen. Om formatet er feil eller det skjedde en feil ved henting av data fra backend, så vil API returneres henholdsvis en beskrivende feilmelding på feil struktur på respons eller en HTTP status kode slik beskrevet senere i kap. 7.2.2.

7.1.3 React komponenter

Applikasjonen består av en rekke react komponenter som til sammen utgjør det grafiske brukergrensesnittet.

Navbar

Navigasjonsbaren er festet til toppen av nettleservinduet. Denne benytter seg av React Router DOM for å håndtere navigasjon mellom de forskjellige rutene i applikasjonen. Disse er plot og table.

Kodeliste 7.1: TypeScript kode for navigasjonsbaren

```
const Navbar = () => {
    return (
        <nav className="navbar">
            <ul className="navbar-list">
                <li className="navbar-item"><Link to={PLOT_ROUTE}>Plot</Link></li>
                <li className="navbar-item"><Link to={TABLE_ROUTE}>Table</Link></li>
            </ul>
        </nav>
    );
}

export default Navbar;
```

Som man kan se fra kodeliste 7.1, vil man ved å trykke på den ønskede grafiske fremstillingen blir man rutet til det respektive endepunktet; å trykke på plot ruter brukeren til URL/plot.

Plot- og tabell komponent

Komponentene for plot og tabell er plassert på samme sted i applikasjonen. Det er gjennom Navigasjonbaren og den resulterende rutingen det blir bestemt hvem av de to komponentene som vises. Plot komponenten er en React komponent som returnerer en iframe med en ferdig plottet graf basert på hvilken URL-streng den mottar.

Tabell komponenten mottar data fra et kalkulasjonskall til API. Denne dataen blir så puttet inn i en tabell for en oversiktlig representasjon av dataene.

Sidemeny

I likhet med plot komponenten, mottar sidemeny komponenten en iframe med ferdig stylet html kode basert på valgt fargematchfunksjon og parametere. Plasseringen av denne komponenten avhenger av bredden på nettleservinduet. Hvis den er under 910 piklser vil sidemenyen bli plasser under plot/tabell og parameterskjema. Dette gjør det lettere å operere applikasajonen på enheter med smalere bredde på skjerm. Implementeringen av dette skjer gjennom hjelpefunksjonen useScreenWidth(). Bredde verdien for skjermen definerer to ulike kompo-

nenter for sidemenyen RightGridinformationIframe og BottomGridInformation-Iframe. Hvordan dette er implementers vises under i kodeliste 7.2

Kodeliste 7.2: Funksjon som endrer plassering av sidemenyen basser på vindustørelse

```
const BottomSideMenuComponent: React.FC = () => {
  const screenWidth = useScreenWidth();
  return (
    <>
      {screenWidth <= 1200 && (
        <div className="sid">
          <SideMenuIframe />
        </div>
      )}
    </>
  );
};
export default BottomSideMenuComponent;
```

Parameterskjema og tilstandshåndtering

Parameterskjemaet er den mest avanserte komponenten i applikasjonen. Den krever tilstandshåndtering og må utnytte React virtual DOM for å oppfatte ulike endringer. Denne komponenten er ansvarlig for å vise og håndtere endring av parametere basert på brukerens input for ulike fargematchfunksjoner. Koden viser hvordan komponenten tar imot parametere fra brukeren og oppdaterer tilstanden basert på disse.

Tilstandshåndteringen skjer ved hjelp av Reacts innebygde hooks som 'useState' og 'useEffect' kombinert med egenkomponerte funksjoner. Siden applikasjonens kompleksitet og krav til tilstandshåndtering ikke var veldig omfattende, ble det besluttet å bruke Reacts innebygde hooks fremfor tredjepartsbiblioteker som Redux.

Alle fargematchfunksjonene benytter i stor grad de samme parameterne: feltstørrelse, alder, minimums- og maksimumsverdi for domenet av bølgelengder, og trinnstørrelse. For å sikre at applikasjonen husker disse parameterne når brukeren bytter mellom visninger, er tilstandshåndteringen implementert slik at parametrene bevares ved visningsbytte. Dette gjør det mulig for brukeren å utføre en beregning og sømløst bytte mellom tabell og plot uten å måtte utføre beregningen på nytt.

To variabler, 'generalFieldSize' og 'dropDownFieldSize', er implementert for å håndtere forskjellene mellom de generelle fargematchfunksjonene og standardfunksjonene som kun tar hensyn til feltstørrelser på 2 eller 10 grader. 'dropDown-FieldSize' lagrer den bruker-valgte feltstørrelsen fra nedtrekksmenyen for standardfunksjonene, mens 'generalFieldSize' lagrer den brukerspesifiserte feltstørrelsen for de øvrige fargematchfunksjonene. Gjennom funsksjonene som kan ses i kodeliste 7.3, settes feltstørrelsen i henhold til valgt funksjon, slik at en endring i parameter for en funksjon ikke påvirker andre funksjoner unødig.

Kodeliste 7.3: Håndtering av ulik field$_s$*izetypeogtilleggsparameter'base'*

```
useEffect(() => {
    const methodNumber = parseInt(selectedOption.replace('method', ''));
    setParameters(prev => {
      const updatedParams = { ...prev };

      // Handle optional parameter for method2
      if (selectedOption === 'method2') {
        updatedParams.optional = 'base';
      } else {
        delete updatedParams.optional;
      }
      // Handle field size based on method number
      if (methodNumber >= 1 && methodNumber <= 8) {
        updatedParams.field_size = generalFieldSize;
      } else if (methodNumber >= 9 && methodNumber <= 10) {
        updatedParams.field_size = dropdownFieldSize;
      }
      return updatedParams;
    });
    setParamsUpdated(true);
  }, [selectedOption, dropdownFieldSize, setParameters]);

// Handles parameter change for every function except xyz
const handleParameterChange = (event: ChangeEvent<HTMLInputElement>): void => {
    const { name, value } = event.target;
    const numericValue = parseFloat(value);

    // Validates the user specified input
    parameterSchema.validateAt(name, { [name]: numericValue })
      .then(() => {
        if (name === 'field_size') {
          setGeneralFieldSize(numericValue);
        }
        setParameters(prev => ({ ...prev, [name]: numericValue }));
      })
      .catch(err => {
        console.error(err.errors);
      });
};

// Handles parameter change for xyz functions,
// where field size of either 2 or 10 degrees are the parameters
const handleDropdownChange = (event: ChangeEvent<HTMLSelectElement>): void => {
    const selectedDegree = parseFloat(event.target.value);
    setDropdownFieldSize(selectedDegree);
    setParameters(prev => ({ ...prev, field_size: selectedDegree }));
    setParamsUpdated(true);
};
```

Denne fleksibiliteten og dynamikken gjør parameterskjemaet til en kritisk del av applikasjonens frontend, som effektivt håndterer brukerinput og tilstandsoppdateringer basert på valgte parametere og funksjoner.

7.1.4 Validering av brukerinput

Validering av brukerspesifisert inndata for parametere skjer ved hjelp av JS biblioteket 'Yup'. Dette implementeres ved at man først definerer et skjema som beskriver reglene som gjelder for valideringen.

Kodeliste 7.4: Yup skjema som definerer regler for de ulike parameterne

```
export const parameterSchema = yup.object().shape({
    field_size: yup.number()
      .required('Field size is required')
      .min(1.0, 'Field size must be greater than 1.0')
      .max(10.0, 'Field size must be less than 10.0'),
    age: yup.number()
      .required('Age is required')
      .min(20, 'Age must be greater than 20')
      .max(80,  'Age must be less than 80'),
    min: yup.number()
      .required('Min domain is required')
      .min(390.0, 'Min domain must be greater than 390.0')
      .max(400.0, 'Min domain must be less than 400.0'),
    max: yup.number()
      .required('Max domain is required')
      .min(700.0, 'Max domain must be greater than 700.0')
      .max(830.0, 'Max domain must be less than 830.0'),
    step_size: yup.number()
      .required('Step size is required')
      .min(1.0, 'Step size must be greater than 1.0')
      .max(5.0, 'Step size must be less than 5.0')
  });
```

I skjemaet som er tatt i bruk(kodeliste 7.4) for dette prosjektet brukes en .required(), .min() og .max() regel for alle feltene for parameter inndata. Navnene på disse regelfunksjonene er nokså selvforklarende; .required() spesifiserer og påser at det må finnes en verdi mens .min() og .max() definerer en minimums- og maksimumsverdi og definerer med det et tillat spenn for verdier for den gjeldende parameteren.

Applikasjonen initialiseres med standardparametere, så disse trenger ikke å valideres. Derfor implementeres sjekken i funksjonen som håndterer parameterendringer. Her brukes en innebygd funksjon fra Yup biblioteket som utfører sjekken på om de definerte reglene er fulgt. Hvis den nye verdien oppfyller kravene stilt av reglene vil verdien oppdatere seg, men om den nye verdien ikke oppfyller kravene vil endringen stoppes i sanntid.

Man har valget for hvordan man vil håndtere en feilsituasjon. Feilhåndteringen er at verdiendring blokkeres dersom den faller utenfor det tillatte spennet, og at det skrives en beskrivende feilmelding til konsollen. Det kan ofte være god praksis å gi tilbakemelding til brukeren, hvor det forklares hva som er feil. Da sluttbrukerne av denne applikasjonen er godt inneforstått med disse begrensningene i parameterverdi, kan det anses som intuitivt at man er utenfor det tillatte verdiområdet dersom man ikke får økt eller senket en verdi ytterligere.

7.1.5 Responsivt design

Som det kom av wireframes var det tenkt at applikasjonen skulle være utviklet med responsivt design til grunne. Dette for å sørge for at applikasjonen kan brukes på en rekke ulike enheter med forskjellige skjermstørrelser. Dette ble lagt litt på hyllene gjennom de fleste av sprintene, da det var utviklingen av funksjonalitet som var hovedfokus. Graden av hvorvidt responsivt design er implementert kan derfor diskuteres. Det kommer også en begrensning gjennom at grafene og tabellen må være stor nok til at det er mulig å interagere med, for ikke å snakke om å i det hele tatt se dem.

Med dette tatt i betraktning ble det bestemt at det måtte være en minste verdi på bredde på komponentene som viser graf og tabell. Det ble også bestemt at dersom skjemen er over en gitt høyde og bredde, skal scrollbar'en gjemmes. Dette anses da som å kjøre applikasjonen i fullscreen, og hele applikasjonen skal være innenfor visningen. Dersom enten høyden eller bredden blir mindre enn den definerte verdien dukker scrollbar opp igjen slik at man har mulighet til å scrolle i lengde eller bredde retning for å allikevel kunne bruke applikasjonen. Hvordan dette er integrert kan ses i kodeliste 7.5, som viser et hvordan dette implementeres med 'media queries'.

Kodeliste 7.5: Media Queries som brukes for å bestemme oppførsel ved gitte dimensjoner

```
.plo{
  width: 66.33%;
  min-width: 900px;

....

@media (max-height: 910px) {
  :root{
    overflow: auto;
  }
}
@media (max-width: 1200px) {
  :root{
    overflow: auto;
  }
  .plo,
  .tab {
    width: 100%;
    margin-right: 5%;
  }
  .sid {
    width: 100%;
    margin-top: 20px;
  }
}
```

Både som det fremkommer i kodeliste 7.5 og som nevnt i 7.1.3, blir også sidemenyen plassert under parameter skjemaet når bredden på skjermen blir under 910 piksler. Dette kan ses i figur 7.3.

Figur 7.3: Sidemeny skyves ned.

7.2 Backend

Dette kapitlet skal fokusere på implementasjonsdetaljer for backenden hos applikasjonen. Vi ønsker først å deklarere at det blir gjenbrukt kode fra basisprogrammet [1]. Dette var for å opprettholde likheten med programmet, og forsørge seg for de samme resultatene. For å forsikre seg korrekt dokumentasjon, har all gjenbruk av kode blitt dokumentert tydelig og klart i dokumentasjonen hos koden.

Det skal nå dras inn på strukturen av tjenesten backend tilbyr. før strukturen av koden og hvorfor den er laget slik den er. Dette er noe vi mener skal først, fordi det er hensiktsmessig for rapportens struktur og formidling.

7.2.1 Tjenestestruktur

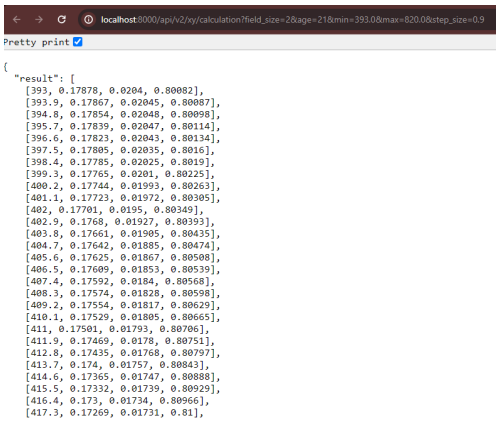
Backend tilbyr en rekke tjenester på dets flere URL for å tilby komplett funksjonalitet sammen med frontend til enhver bruker av applikasjonen. Dets primære tjeneste er å avgi utregninger i form av diverse representasjoner, tilgjengelige på URL som er logisk strukturert for å gi enkel mening til brukere - slik de er vist frem på figur 7.4.

```
- ../api/v2/xy/calculation?field_size=2&age=21max=820.0&step_size=0.9
- ../api/v2/lms/sidemenu?field_size=2.0&age=23&log10&max=700&min=400
- ../api/v2/lms-mb/plot?field_size=1.5&age=51&max=700
```

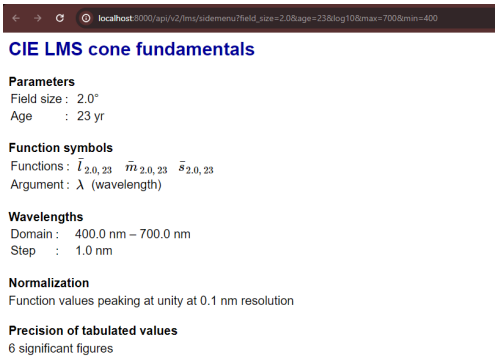
Figur 7.4: Eksempeler av URL for finalisert webapplikasjon.

Alle ressurser som avgis av APIet, trenger tre ting:

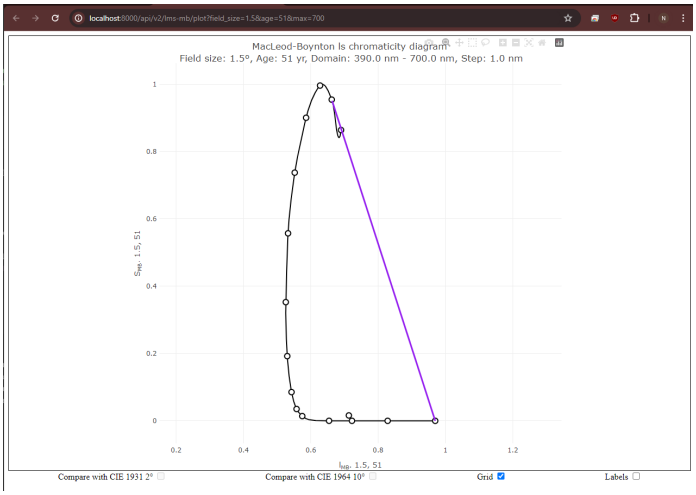
- En funksjon tilgjengelig i programmet representert i stien til lenka, hvor eksempler over er `/xy`, `/lms-mb` og `/lms`.
- En valgt representasjon av utregningsressursen, noe som kan ta tre former:
    - `/calculation` (som vist i figur 7.5) vil utgi utregningene for funksjon og gitte parametere direkte til brukeren i form av `application/json`.
    - `/sidemenu` (figur 7.6) vil representere en informasjonsside for funksjonen, lik til sidemenyet som finnes i basisprogrammet. Denne er i form av `text/html`, hvor det genereres dynamisk fra applikasjonen.
    - `/plot` (figur 7.7) vil representere diagrammet for funksjonen, sammen med funksjonaliteter tilgjengelig hos funksjonen. Denne er også i form av `text/html`, hvor det består av innebygd JS kode for å implementere det funksjonelle diagrammet.
- Et sett av URL baserte parametere for å representere brukerinput gjennom bruk av URL queries.



Figur 7.5: Eksempel på resultat fra /calculation endepunkt.



Figur 7.6: Eksempel på resultat fra /sidemenu endepunkt.



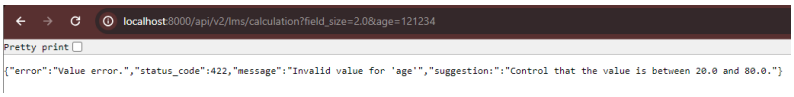
Figur 7.7: Eksempel på resultat fra /plot endepunkt.

Implementasjonsdetaljer og nærmere diskusjon til alle tre former av ressurser over vil tas opp videre i 7.2.2. Men, i tillegg til alt dette, finnes det en addisjonelle endepunkter for å øke kvaliteten av tjenesten, i form av et statusendepunkt. `/status` vil utgi informasjon om selve tjenesten, inkluderende hvor lenge serveren har vært oppe, og hvilken versjon er den nyeste.

RESTful

Et sentralt punkt for oss er å forsikre at vår tjeneste følger REST for å være RESTful slik vi hadde tenkt i kap. 4.2.1 - noe vi skal nå deklarere med tanke på prinsippene deklarert tidligere i kap. 2.2.3. Rekkefølgen de deklareres i følger samme rekkefølge som de ble introdusert.

1. Vi mener at tjenesten oppfyller prinsippet om 'separasjon av klient og server', men ønsker å utdype angående noe spesifikt. Som vi vet, så gir tjenesten også vekk ressurser i form av representasjon `text/html`, hvor den er da dynamisk generert hos backend. Vi erkjenner at dette kan ses ut som at tjenesten drar inni denne bekymring som skal opprinnelig ligge hos frontend - men vi mener at vi følger prinsipper allikevel, for vi kun avgir en ressurs i form av et HTML dokument - og selve visningen av det er opp til frontend.
2. Tjenesten lagrer ingen informasjon fra tidligere klientforespørsler, og derfor enkelt oppfyller kravet om statsløshet.
3. Alle av svarene som sendes fra server markeres eksplisitt om kan lagres gjennom bruk av `cache-control header` i svarene tjenesten genererer. Alle utregningsresultater har fått verdien `private` (som gjør at de kan lagres i privat cache hos nettlesere), mens statusendepunkt har fått en `no-store` - aldri cachet.
4. Det finnes ingen kode i applikasjonen som spesifiserer at det krever noe fra et spesifikk arkitekturkomponent. Arkitekturen vi har brukt krever kun to komponenter som kjører separat fra hverandre slik det er implementert nå.
5. Tjenesten implementerer bruken av dette prinsippet med å levere kjørbar kode hos noen av ressursene den utgir (gjennom endepunktet `/plot`). Koden kjøres på nettleseren til brukeren for å tilsette ekstra funksjonaliteter (funksjonelle avmerkningsbokser, diagrammer).
6. Alle ressurser som tilbys av tjenesten finnes på unike URL, hvor de kan kun hentes gjennom bruken av `GET`, som er da semantisk riktig. Det brukes også statuskoder for å inneholde informasjon om respons, i tillegg til at det finnes en hjemmeside med nok informasjon for en bruker å kunne lære å traversere seg rundt i tjenesten - sammen med detaljerte feilmeldinger som forteller mulige forslag til å fikse en gitt feil (7.8). Med dette, kan vi si at tjenesten oppfyller dette kravet.



Figur 7.8: Eksempel på feilmelding fra tjeneste.

Fordi vi mener at applikasjonen vår faller innom disse kravene, så kan vi kalle det for en RESTful applikasjon - og være fornøyd med dets evner.

7.2.2 Kodestruktur

Før vi drar inn på spesifikke endepunkter, ønsker vi først å belyse programmets generelle struktur. Dette er fordi rammeverkets syntaks er veldig likt andre rammeverk i andre språk fra vår egen erfaring, hvor vi ser følgende fellestrekk:

- Det lages en instans av en `app` for å representere en webapplikasjon.
- Til instansen, lages det kodefunksjoner for å representere endepunkter.

- Etter deklarasjonen av alle endepunkter, kjøres applikasjonen slik at den kan tjene til endepunktene.

Sanic passer inn her, om vi referer til kodelisten under. Vår hovedinstans er en `sanic.app.Sanic` gjennom `Sanic(__name__)`. Det er mulig å tilsette kontekster til denne applikasjonen (blant annet databaser eller tilgjengelige andre Sanic applikasjoner)[1], men for vår oppgave, tilsatte vi kun CORS[2] beskyttelse for å tillate andre maskiner å hente data fra tjenesten. CORS er ikke tilgjengelig innebygd i Sanic, men rammeverket har et stort økosystem med biblioteker for flere funksjonaliteter, hvor dette er kun en av dem[3].

Etter at applikasjonsinstansen er deklarert, tilsetter vi endepunkter gjennom funksjonen `@api.get()` sammen med en påkrevd `string` parameter for å representere URL for endepunktet. En slik tilkalling *må* alltid følges opp enten av en annen `@api.get()` for å definere flere URL, eller en funksjonsdeklarasjon med kode for å takle forespørsler sendt til de gitte URL - hvor det må kunne sende en respons. Fordi rammeverket støtter ASGI, kan vi tilsette `async` foran endepunktsfunksjonene for å gjøre dem asynkrone i rammeverket.

Kodeliste 7.6: Backend - Generell struktur

```
# hovedinnstans
api = Sanic(__name__)
CORS(api)

...
# utsnitt av endepunkt
@api.get(endpoint_creator(API_HOMEPAGE, API_VERSION, LMS_ENDPOINT) + "/<more:str>")
async def LMS(request, more: str):
    # behandling av endepunkt kommer her ...

...
# oppstart av applikasjon
if __name__ == '__main__':
    api.run(debug=True, port=8080)
```

Oppbygningen av URL hos alle endepunkter foregår gjennom en dedikert funksjon kalt `endpoint_creator(...)`, som tar inn `string`, og konstruerer et URL-sti for ett endepunkt basert på parametere. For dette, følger vi god praksis med å bruke konstanter med navn i store bokstaver deklarert øverst i filen. Det brukes også en `/<more:str>` for å gjøre det mulig å hente alle videre stier relatert til endepunktet fra denne URL.

I kodelista under, ser vi et eksempel på innmatet hos et endepunkt for utregning. Alle slike endepunkt deler på denne strukturen, hvor deres eneste forskjeller er deres URL og hvilke fargematchfunksjon de utregner for. I dette eksemplet, brukes det `API_HOMEPAGE, ("/api")` og `API_VERSION, ("v2")`, sammen med en `LMS_ENDPOINT, ("lms")` for å definere endepunktet for fargematchfunksjonen 'LMS'.

[1] `https://sanic.dev/en/guide/basics/app.html#application-context`, hentet 20.05.2024
[2] `https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS`, hentet 20.05.2024
[3] `https://sanic.dev/en/plugins/sanic-ext/http/cors.html`, hentet 20.05.2024

Sammen med bruken av `more`, kan sjekke en eventuell klient sin URL for å se hva de forsøker å hente. Verdien av denne variabelen sjekkes og rutes videre til ett av fire mulige valg.

Kodeliste 7.7: Backend - Eksempel av utregningsendepunkt.

```
@api.get(endpoint_creator(API_HOMEPAGE, API_VERSION, LMS_ENDPOINT) + "/<more:str>")
async def lms(request, more: str):
    if more == "calculation":
        return response.raw(new_calculation_JSON(compute_LMS_modular,
                                                 create_and_check_parameters(
                                                     True,
                                                     compute_LMS_modular,
                                                     request)),
                            content_type="application/json",
                            headers={"cache-control": "private"})
    if more == "sidemenu":
        return html(LMS_sidemenu(create_and_check_parameters(
            True,
            compute_LMS_modular,
            request)),
            headers={"cache-control": "private"})
    if more == "plot":
        return html(LMS_graph(create_and_check_parameters(
            True,
            compute_LMS_modular,
            request)),
            headers={"cache-control": "private"})
    else:
        not_found(more)
```

Brukeren kan enten sendes til å hente de tre tilgjengelige representasjoner av ressurser introdusert tidligere, eller så får de en feilhåndteringsmelding. Alle av disse er videre modulisert og delt inni passende filer. Mer informasjon om disse skal ses på i deres egne kapitler eventuelt. Før vi gjør det, ønsker vi å introdusere det underliggende systemet delt gjennom hele systemet med evnen til å behandle brukerinput og lagre det i en praktisk form; parametersystemet.

Parametersystemet

Rett etter selve webapplikasjon, ligger parametersystemet vi hadde laget som det nest viktigste systemet i hele programmet. For programvaren, så er det en `dict`-variabel med all brukerinformasjon hentet fra en gitt URL. Men, i hele systemet, er denne variabelen ansvarlig for all inntak av brukerinformasjon, validering av det (og tilstrekkelig feilhåndtering), og til sist dets universelle anvendelse i alle utregningsfunksjoner videre i programmet.

For enklere formidlingen, skal systemet diskuteres med tanke til kodelistene inkludert hos vedlegg G. Om vi ser på kodeliste G.1, så ser vi at vi starter opp hele parametersystemet gjennom funksjonen `createAndCheckParameters(...)`. Denne funksjonen eksisterer hos alle endepunkter, for alle ressurser. Funksjonen tar inn selve utregningsfunksjonen, sammen med et objekt som representerer klientforespørselen kalt `request` og en `bool` for sjekking om dette er for en standardise-

ringsfunksjon (hvor kun en nødvendig parameter sjekkes). Funksjonen vil enten returnere en `dict` av validerte parametere, eller så vil den utføre en `raise` og sende et feilsvar til klienten umiddelbart uten å kjøre resten av koden.

G.1 introduserer bruken av en indre hjelpefunksjon kalt `string_to_type_else` basert på `<Option>`[4] og `Maybe`[5] monadtypene fra programmeringsspråkene Rust og Haskell. På samme måte disse typene kan takle potensielle feil hos programmet på en grasiøs måte [47] (med å ha gitte verdier være abstrakte enten 'noe av type' eller 'ingenting') - så tar denne funksjonen og returnerer enten en verdi, `None` eller en spesifisert verdi (som vi skal se på i del 2/kodeliste G.2). Den oppnår dette med bruken av `try` og `except`; kode som gjør det mulig å forsøke potensielt feilaktig kode som innmat, hvor den vil fange programfeil i koden og kjøre spesifikk kode for en gitt type feil. Funksjonen tar inn en `string` (kalt 'string'), en datatype kalt 'type' som 'string' skal forsøkes å konverter til, og noe som 'other'. Resten av del 1 bruker funksjonen for å finne ut om de obligatoriske parameterne `field_size` og `age` er tilgjengelig i URL hos forespørselen. Den lager en `dict` med representanter for dem, hvor deres verdier er da resultatene fra hjelpefunksjonen. Om de er innenfor URL, så hentes dem og omgjøres til den ønskede `float` typen - ellers, vil hjelpefunksjonen utføre en feil, og feilen blir fanget av funksjonen - hvor den vil da returnere en `None`. Etter dette, vil alle parametere sjekkes om de har en `None` verdi, der om en har det, så vil programmet umiddelbart utføre en `raise` av typen `SanicException` med parametere for en egen feilhåndteringsfunksjon diskutert senere.

G.2 viser en videre bruk av hjelpefunksjonen, hvor den brukes sammen med en alternativ verdi kalt 'other'. Dette brukes for at hjelpefunksjonen skal kunne forskjelliggjøre mellom feil av *ingen* verdi og med *feil* verdi slik tidligere sagt. Når vår hjelpefunksjon forsøker å lese inn noe, så kan det oppstå kun tre utfall: Den vil enten klare å lese det korrekt, ikke kunne lese det fordi det er ikke av riktig form - eller ikke lese det for ingenting er oppgitt. For alle tre situasjoner, takler den det forskjellig for å gjøre ruting mulig til hver scenario mulig, og derfor unike behandlinger av dem - hvor ett eksempel er i den oppgitte kodelista, hvor den blir brukt til å gi standardverdier til parametere som ikke er oppgitt, men også kunne gi en feilmelding om de er oppgitt med feil verdi.

Om parameterne passerer disse sjekkene og drar videre, så begynner den andre fasen av feilvalidasjon - hvor de sjekkes om de er av riktige verdier (med ett eksempel i de siste 5 linjer av kode i del 2 for sjekking av `field_size`). Dersom alle parametere klarer dette sjekket, så drar de videre til en ekstra vurdering for `optional` parameteren, en egen dedikert parameter for valgfri input (som `log` hos `/lms`). Etter alt av dette, burde funksjonen gi et brukbart `dict` med kompatibilitet i alle endepunktfunksjoner.

[4] `https://doc.rust-lang.org/std/option/`, hentet 20.05.2024
[5] `https://wiki.haskell.org/Maybe`, hentet 20.05.2024

Endepunkt for utregning

Hovedressursen som skal avgis av alle utregningsendepunkter, er selvfølgelig utregningene selv. Prosessen for dette er abstrakt delt mellom alle endepunkter, og innebærer to steg:

- **Utregningen:** Det utføres et kall til en modulisert versjon av en `compute_x()` funksjon fra `compute.py`[6] (hvor `x` er en fargematchfunksjon fra basisapplikasjonen). `compute.py` er fra kodelageret til basisprogrammet, hvor utregningskoden for de moduliserte versjonene er direkte basert på dem. Den moduliserte funksjonen vil da gi utregningene som et resultat.
- **Formatering:** Utregningene deretter pakkes inn i en format de ikke kan bli påvirket videre i, i form av en JSON-basert enkoder med tilgang til stringformatering for spesifisering av presisjon.

Vi skal ikke dra i detalj på alle utregninger for de er unik til hver funksjon i applikasjonen. Allikevel, mener vi at det er best å nevne et eksempel og hva vi har gjort. Om vi ser på kodebiten E.3 tilgjengelig hos vedlegg G, ser vi et utsnitt av den moduliserte utregningsfunksjonen for `/lms` endepunktet. Den deler struktur med alle andre utregningsfunksjoner, hvor:

- Den tar inn kun en parameter kalt `parameters`, som representerer resultatet fra parametersystemet forklart tidligere. Denne variabelen vil inneholde alt av nødvendig brukerinput for å utføre utregninger basert på dem - og er veldig brukbar når den brukes universelt mellom alle endepunktfunksjoner.
- Alle slike utregningsfunksjoner vil returnere en `dict`, bestående enten av `result` & `plot` medlemmer - eller av andre medlemmer dersom bruken av `info` i brukerparametere aktiveres. Noen funksjoner støtter ikke bruken av `info` (slik dette eksemplet), men dette tas vare av i parametervalidasjon.
- Alle linjer i funksjonen er direkte basert på linjer hos `compute.py`. Som vi har sett tidligere hos 6.5, så har vi bevist at det gir det raskere resultater å modularisere istedenfor å bruke de opprinnelige funksjonene. Selv om den løsningen er gammel og gjelder ikke her lenger, så kan den fortsatt gjelde for visse situasjoner (som bruken av `'info'`, hvor den må utlevere forskjellige utregningsresultater enn de vanlige `result` og `plot`).
    - For å direkte vise at dette er gjenbruk, har modulen til koden selv deklarert dette klart - sammen med at vi har tilsatt direkte kommentarer til hvilke linjer korresponderer til hva i det opprinnelige.

Det som er heller mer interessant, er formatering av resultatene. Som vi husker fra kap. 6, så var det et stort fokus av oss å utlevere de riktige tallene uten påvirkning fra feil i presisjon - inntil vi fant en løsning basert på en egen JSON-enkoder med bruk av spesifikke formateringer for gitte utregninger.

Dette systemet starter opp med en hardkodet `dict` i selve programmet, av slik

[6] `https://github.com/ifarup/ciefunctions/blob/master/tc1_97/compute.py`, hentet 21.05.2024

utseende:

Kodeliste 7.8: Backend - Utdrag av stringformateringer

```
calculation_formats = {
    "LMS-base-log": {
        "result": ["{:.1f}", "{:.8f}", "{:.8f}", "{:.8f}"],
        "plot": ["{:.1f}", "{:.8f}", "{:.8f}", "{:.8f}"],
    },
    ...,
    "XYZ-XYZP-XYZ-STD": {
        "result": ["{:.1f}", "{:.6e}", "{:.6e}", "{:.6e}"],
        "plot": ["{:.1f}", "{:.6e}", "{:.6e}", "{:.6e}"],
    },
    # shared dictionary between all info
    "info-1": {
        "norm": ["{:.8f}", "{:.8f}", "{:.8f}"],
        "white": ["{:.6f}", "{:.6f}", "{:.6f}"],
        ...
        "trans_mat": ["{:.8f}", "{:.8f}", "{:.8f}"],
        # "trans_mat_N": ["{:.8f}", "{:.8f}", "{:.8f}"],
    }
}
```

De er hentet direkte fra basisapplikasjonen sin kildekode, og er organisert i `list` av string med forskjellige formateringer i seg. Mengden av formateringer korresponderer direkte til lengden av en rad i utregningene - og derfor, så er en formatering direkte korresponderende til et medlem i raden. For eksempel, i `LMS-base-log`, så er det forventet fire elementer per rad i utregningen - og at det første medlemmet formateres gjennom `{:.1f}`, et desimalltall med kun en desimal. Dette er viktig å huske på når vi fortsetter.

Dette systemet brukes primært i en rutingsfunksjon kalt `new_calculation_JSON(...)`, som både starter utregningen samtidig som det passerer det (og det utvalgte stringformatet) til vår JSON-enkoder gjennom koden under.

Kodeliste 7.9: Backend - Utdrag av rutingsfunksjon for JSON

```
def new_calculation_JSON(calculation, parameters):
    if parameters['info']:
        return write_to_JSON(calculation(parameters),
            calculation_formats['info-1'])
    if calculation is compute_LMS_Modular:
        if parameters['base']:
            if parameters['log']:
                # log10 base LMS
                return write_to_JSON(calculation(parameters),
                    calculation_formats['LMS-base-log'])
            else:
                # base LMS
                return write_to_JSON(calculation(parameters),
                    calculation_formats['LMS-base'])
        ...
```

Videre kalles `write_to_JSON()` for å starte enkodingen, men selv det er kun en hjelpefunksjon for viderebehandling av resultater fra funksjonen `ndarray_to_JSON`; hvor er faktisk der formateringen foregår.

Kodeliste 7.10: Backend - Formateringsfunksjonen

```
def ndarray_to_JSON(body, formatta):
    # an 1D-array, or by an ndarray recursively calling this function for each row)...
    if body.ndim == 1:
        # Double check if the length of the row is equal to the length of the format
        if len(body) == len(formatta):
            # Format each row respectively as they should be, through indexed for-loop:
            all = []
            for index in range(len(body)):
                # Retrieve the format and chopped body.
                tempar = chop(body[index])
                asda = formatta[index]
                # The '-inf' produced by LMS (log10) is something
                    £ that cannot be parsed by JSON;
                # it gets remade into null instead.
                if math.isinf(body[index]):
                    tempar = "null"
                    asda = "{}"
                # Append the formatted string to the 'all' array,
                    # symbolizing 'all' elements of current row.
                all.append(asda.format(tempar))
            # Join them all with commas, and then output it with [ ]
                # on either side, succesfully making it into
                # a JSON array.
            output = ','.join(all)
            return '[' + output + ']'
        else:
            # should only happen if the row format of a calculation does not match
            # the given format (example, [x, y, z] =/= [f1, f2]
            raise SanicException(
                ("PROCESSING␣ERROR",
                 "There␣has␣been␣an␣error␣inside␣of␣the␣server.",
                 "Contact␣system␣administrator␣for␣server,␣or␣try␣again␣later."),
                status_code=500)
    else:
        # create empty array
        temp = []
        # for each array in current array, recursively call to each
            # and append them to array above
        for row in body:
            temp.append(ndarray_to_JSON(row, formatta))
        # join them with commas, and output with [ ]
        output = ','.join(temp)
        return '[' + output + ']'
```

Formateringsfunksjonen tar inn en `numpy.ndarray` for all utregning (tilkalt tidligere), sammen med stringformatene beskrevet tidligere. For alle medlemmer hos den gitte `numpy.ndarray` som er over enn *en* dimensjon, vil den utføre et rekursivt kall til hver av sine barn inntil det møtes på en endimensjonal medlem. Når den kommer til en `body` av kun en dimensjon, så vil den formatere alle medlemmer av dette sammen med formateringen gitt, lenke alle til en singulær string gjennom `','`, og deretter sette `'[' og ']'` hos start og slutt. Når dette kombineres med dette samme for alle tidligere rekursive kall, ender vi opp med en riktig representasjon av `ndarray` til JSON i form av en `string` som ligner strukturen av en JSON fil.

`write_to_JSON()` tar resultatene, og tilsetter en `map` struktur til `string`en produ-

sert, før den sendes tilbake som resultatet for `new_calculation_JSON()`. Fordi dette er en rå string som kun representerer en JSON, så sendes eksplisitt ut som en rå fil gjennom metoden `.raw()`, men med en `header` som forteller at filen er av en `application/json` type.

I utviklingsprosessen, fortalte vi at vi forsikret oss at tallene var riktig med å sammenligne dem med programmet. For å tilsette mer sikkerhet til funksjonen, ble det også utviklet tester som sjekker lint av sluttresultatet - hvor vi kan si at vi har hatt ingen problemer med en feilstrukturert JSON.

Endepunkt for sidemeny

Som skrevet tidligere i 6.3.3, så måtte sidemeny implementeres som en ressurs i håp på å redusere mengden av forespørsler nødvendig for den vanlige brukeren. Men istedenfor å diskutere den direkte implementasjonen, skal vi heller først faktisk fortelle om basisprogrammet sin implementasjon[7]:

Kodeliste 7.11: CIE Functions - Utdrag fra description.py, linje 1161-1196

```
def lms_mb(data, heading, options, include_head=False):
    html_string = ""
    if include_head:
        html_string += _head()
    html_string += (_heading(heading) +
                    _parameters(data) +
                    _coordinates('\\(l_{\,\mathrm{MB},\,%s,\,%d}\\)' %
                                 (data['field_size'], data['age']),
                                 '\\(m_{\,\mathrm{MB},\,%s,\,%d}\\)' %
                                 (data['field_size'], data['age']),
                                 '\\(s_{\,\mathrm{MB},\,%s,\,%d}\\)' %
                                 (data['field_size'], data['age'])) +
                    _wavelenghts(data) +
                    _normalization_lms_mb(data) +
                    _LMS_to_lms_mb(data, options) +
                    _precision_lms_mb() +
                    _illuminant_E_lms_mb(data) +
                    _purpleline_tangentpoints_lms_mb(data))
    return html_string
```

Basisprogrammet er selv utviklet i Python, hvor den bruker `PyQt` for dets brukergrensesnitt - men dette selv bruker HTML for å uttrykke sidemeny. Det var derfor mulig å direkte gjenbruke deres funksjoner for implementasjonen av sidemenyet[8] - i tillegg til å faktisk bruke det allerede eksisterende parametersystemet til å inneholde alt av det `data` skulle i koden over, og enkelt kalle til utregninger når den trenger det gjennom det.

Gjennom dette, var det faktisk veldig enkelt å implementere deres kode inni vår applikasjon. Det var så lik at vi klarte å gjenbruke en del kode direkte fra

[7] `https://github.com/ifarup/ciefunctions/blob/master/tc1_97/description.py` , hentet 21.05.2024

[8] `https://github.com/ifarup/ciefunctions/blob/master/tc1_97/description.py`, hentet 21.05.2024

modulen. Dette er fint for konsistenthet mellom applikasjonene. Nedenfor er en seksjon av vår kode som er da motparten til den over, slik at leseren selv kan se hvor like våre implementasjoner er:

Kodeliste 7.12: Utdrag fra vår kode, direkte basert på koden over.

```
def LMS_MB_sidemenu(parameters):
    html_string = ""
    html_string += _head()
    data[' _min '] = data['min']
    data[' _max '] = data['max']
    data[' _step '] = data['step_size']
    data['info'] = True
    info = compute_MacLeod_Modular(data)
    data['norm_coeffs_lms_mb'] = info['norm']
    data['lms_mb_white'] = info['white']
    data['lms_mb_tg_purple'] = info['tg_purple']

    html_string += styles.description._heading(u'MacLeod\u2013Boynton␣ls␣chromaticity
␣␣␣␣␣␣␣␣diagram')
    html_string += (
        styles.description._parameters(data) +
        styles.description._coordinates('\\(l_{\,\mathrm{MB},\,%s,\,%d}\\)' %
                                (data['field_size'], data['age']),
                                '\\(m_{\,\mathrm{MB},\,%s,\,%d}\\)' %
                                (data['field_size'], data['age']),
                                '\\(s_{\,\mathrm{MB},\,%s,\,%d}\\)' %
                                (data['field_size'], data['age'])) +
        styles.description._wavelenghts(data) +
        styles.description._normalization_lms_mb(data) +
        styles.description._LMS_to_lms_mb(data, data) +
        styles.description._precision_lms_mb() +
        styles.description._illuminant_E_lms_mb(data) +
        styles.description._purpleline_tangentpoints_lms_mb(data) )

    return html_string
```

En liten endring å markere er skiftet fra lokalt til nett. I basisprogrammet var det installert et bibliotek som heter `MathJax`[9] for formatering av matematikk i tekst - men i og med at vår applikasjons struktur gjorde det umulig å passere dette videre med hver ressurs, så ble bruken av biblioteket skiftet til å heller kalle til biblioteket over en lenke gjennom `<script>`, noe som ser ut som en vanlig metode. Vi hadde også implementert filens CSS som en innebygd `<style>` for å minske kompleksitet med å tjenestegjøre for den individuelt.

Endepunkt for diagram

Det var opprinnelig satt opp til at frontend selv skulle utføre koden og logikk for diagrammene i webapplikasjonen, men med strukturen av ressurser fra backend fikk det til å kreve flere forespørsler, så ble det vanskelig å få det rett og i tid til prosjektets slutt. Derfor som en midlertidig løsning til dette (og basert på det tidligere arbeidet hos endepunkt for sidemeny), så ble det avgjort seint at backend

[9] `https://www.mathjax.org/`, hentet 20.05.2024

skulle utgi et HTML av diagrammet som en ressurs, med tanke at den kunne enkelt lastes inn gjennom bruken av et `<iframe>` hos frontend. Fordelen her var at vi kunne enkelt redusere forespørselsmengden helt ned til 1, men på bekostningen at dette var ment til å være den 'gamle' metoden å gjøre det på - og at arbeidet burde heller ha vært opptil frontend.

Implementasjonen av dette innebærer to steg:

- **Utregninger:** Først må alle relevante utregninger utføres slik at de kan bli uttrykt på diagrammet i det første.
- **Innebygd kode:** Etter at alle utregninger er laget, genereres det et HTML dokument i form av en stor string. Det tilsettes relevant CSS og JS innebygd inni dokumentet, sammen med utregningene på en form de kan tolkes av JS.

Vi skal referere til vedlegg H for videre forklaringer med eksempler fra kode. Det anbefales sterkt at resten av dette leses sammen med det, for det skal ikke tilsettes flere kodelister for resten av dette delkapittelet.

Hos den første kodelisten for utregningsdelen av implementasjonen, ser vi at vi utfører utregningene nødvendig for at diagrammet skal vises - og at de lagres i JSON enn i en `dict` denne gangen. Dette er for å gjøre det mulig å overføre utregningene til den innebygde JavaScript koden hos dokumentet uten noe risiko for tap eller feillesing i passeringen. Vi lager også eventuelle titler for diagrammet gjennom Python for å ta bruk av dets formateringer med brukerparametere, men disse lagres kun som `string`. Etter dette, begynner genereringen av den innebygde koden med å først refereres til en `head()` funksjon sammen med utregningsfunksjonen vår.

`head()` spiller en stor rolle i denne modulen, for den bygger både opp strukturen av HTML dokumentet sammen med nødvendige importeringer av biblioteker (som `Plotly`) - men lager også interaktive avmerkningsbokser inni det gjennom `checkboxes()`. Denne funksjonen tar utregningsfunksjonen, og bruker den til å enten aktivere eller deaktivere visse bokser basert på funksjonen den er gitt som parameter (derfor vi inkluderer utregningsfunksjonen inni `head()`). Den tilsetter deretter det inni dokumentet som HTML elementer i en boks med `Flex` med `<span>` for å holde dem horisontal i applikasjonen.

Når dette er ferdig, begynner prosessen av å sette sammen den innebygde JS koden inni dokumentet. Vi starter med å tilsette utregningene inni koden i form av `string` med JSON innhold - slik at de kan lastes inni koden uten noe konflikter fra dets opprinnelige kode. Utregningene må transponeres slik at strukturen endres til kun fire `array` - hvor den første korresponderer til alle bølgelengder, og resten er for relaterte grupper av tall. Med dette, blir våre utregninger klare for bruk i et diagram.

Vi følger opp dette med å initialisere innstillinger for et `Plotly` diagram med bruken av `config` og `layout` variabler. Innen disse, tilsettes det våre formaterte

`string` med bruk av `'` for å plassere det riktig inn, og ikke ha det reagere med `"""` i Python. Det settes også en konstant `width` og `height` for å forsikre riktig sideforhold (1:1) for kromatisitetsdiagram.

Etter dette, gjør vi klar våre diagram medlemmer med å gjøre dem også om til `Map`, hvor vi detaljerer eksakt hvilke deler av utregninger skal vises på diagrammet sammen med eventuelle andre justeringer for dets utseende. Etter deres deklarasjoner, tilsetter vi funksjonaliteter til avmerkningsboksene laget tidligere gjennom bruken av `addEventListener`. De lages sist i koden for å kunne korrespondere med navnene gitt til diagram medlemmer, hvor justeringer til dem endres på avhengig av status hos boksen - og endelig med alt dette ferdig, tegner vi (og oppdaterer diagrammet gjennom `eventHandler` diagrammet gjennom `react()` metoden fra Plotly.

Noe å legge merke til, er at denne koden kunne ha enkelt blitt delt inni biter og modulisert videre til funksjoner. Den kunne ha vært mer optimalisert med tanke på utregningene. Dette ble gjort med vilje, og skal tas opp senere i rapporten ved diskusjon.

Feilhåndtering

Den siste tingen vi ønsker skrive om for backend, er dets vei på å håndtere programfeil. Vi har allerede sett instanser av dette tidligere i teksten, men det er virkelig viktig for nettrammeverk å avgi meningsfulle feilmeldinger. For denne grunn, ble følgende kode skapt:

Kodeliste 7.13: Utdrag fra vår kode, direkte basert på koden over.

```
@api.exception(SanicException)
async def error_handler(request, exception):
    [title, message, suggestion] = exception.args[0]
    json_error = {
        "error": title,
        "status_code": exception.status_code,
        "message": message,
        "suggestion:": suggestion,
    }
    return json(json_error, status=exception.status_code)
```

Dette er en egen feilhåndteringsfunksjon som vil ta å fange alle instanser av `SanicException`. Lesere kan kanskje huske at vi utføre spesifikt `raise` med denne typen eksempsjon, sammen med meldinger som beskrev programfeil og hva kan gjøres bedre. Det er ikke mye å si om dette, men det er fint å nevne med tanke på HATEOAS fra . I tillegg til at brukeren skal kunne vite alt nødvendig fra hjemmesiden, hjelper disse meldingene en potensiell bruker med å navigere seg rundt gjennom riktigere bruk av parametere.

Det er også tilsatt spesielle feilhåndteringsmeldinger for visse situasjoner:

- Dersom brukeren forsøker å dra til den tidligere versjonen av tjenesten, vil de få en feilmelding som forteller at tjenesten er ikke aktiv.

- Dersom brukeren forsøker å utføre en forespørsel med en HTTP metode som er annerledes enn `GET`, vil tjenesten fortelle at metoden er ikke støttet av tjenesten.

Kapittel 8

Testing & kvalitetssikring

Dette kapittelet drar til testing og kvalitetssikringen utført på ...

8.1 Testing

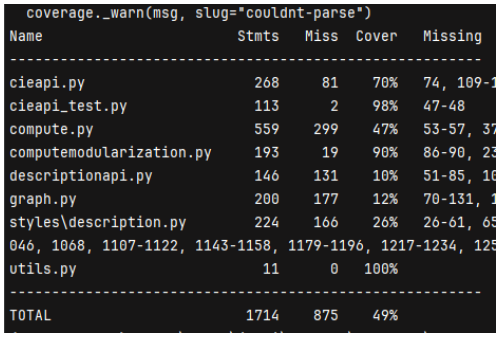
8.1.1 Enhets- & Integrasjonstesting

Over utviklingen av backenden hos tjenesten, har det vært utviklet flere enhets- og integrasjonstester. Vi startet med en testplattform i 6.2.1, før den ble oppgradert til å være en del av enhetstestene vi hadde - og nå er et komprehensiv plattform som dekker både enhets og integrasjonstester tilgjengelig gjennom `cieapi_test.py` filen i repositoriet.

Våre tester i backend forsørger seg at følgende er testet:

- Om vår JSON-enkoder gir data som kan ordentlig tolkes i JSON, gjennom bruken av `json_lint_test()` og korresponderende funksjoner (`test_ndarray_to_JSON()`, `test_advanced_ndarray_to_JSON()`, `test_write_to_JSON()`).
- Om alle av våre endepunkter hos tjenesten gir både de forventede resultater og riktige statuskoder. Dette inkluderer:
    - Nesten alle utregningsendepunkter, hvor det sendes en forespørsel til seg selv for å hente utregninger, laster det inn som JSON - og deretter sammenligner det med en `csv` fra basisprogrammet for de samme brukerparameterne. Koden klarer å laste begge inn uten feil i presisjon, før det sammenligner innholdet og struktur av begge. Denne testen inkluderer ikke `/xyz-p` endepunktet, for basisprogrammet klarer ikke å produsere `csv` filen nødvendig.
    - Alle andre endepunkter testes også, hvor de skal returnere en ønsket statuskode.

Dette ble utført både for det tidligere Flask rammeverket (gjennom Unittest biblioteket) - og en til gang for det nye Sanic rammeverket (gjennom Pytest for å støtte asynkrone forespørsler til seg selv). Med bruk av `coverage` for den aller siste versjonen av programmet, ble det funnet ut at `cieapi.py` har 70% dekning



Figur 8.1: Skjermbilde av testdekning for utrullet versjon av applikasjon.

av testene - mens `computemodularization.py` har 90%. Et skjermbilde av kodedekningen finnes på figur 8.1.

I tillegg til at dette dekker enhetstester, utfører det også integrasjonstester i og med at den sender forespørsler til seg selv. Gjennom denne prosessen, testes integrasjonen av utregninger sammen med alle andre systemer satt på plass (parametervalidering, feilhåndtering, formatering) - noe vi ser fungerer helt greit for applikasjonen.

Naturligvis, så er testene også godt dokumentert. Alle dekninger underveis for applikasjonen er tilsatt i et eget dokument i direktoriet for testing - hvor den inkluderer også brukerparametere for `csv` filene slik at det kan reproduseres.

Kodeliste 8.1: Backend - Eksempel av endepunktstestfunksjon for enhets- og integrasjonstest.

```
# test for macleod
@pytest.mark.asyncio
async def test_lmsmb_endpoint():
    truth_data = load_csv_to_array("./tests/LMS-MB-5-63-1.csv")
    req, response = await cieapi.api.asgi_client.get(
        "/api/v2/lms-mb/calculation?field_size=5&age=63"
        )
    assert np.all(np.array(json.loads(response.body)['result'])
        == truth_data) == True
    assert response.status == 200
```

8.1.2 Belastnings- & Stresstest

Gjennom bruk av belastnings- og stresstester, ønsket vi å undersøke hvor god ytelse tjenesten hadde i realistiske situasjoner. For denne grunn ble Postman brukt til å sende trafikk til den utrullede applikasjonen - hvor mengde av 'virtuelle brukere' (hvor mange sender forespørsler konstant gjennom testen) ble justert for å tilsette variabelen av belastning.

Vi kjørte først en stresstest for å se hvor mange forespørsler tjenesten skal kunne ta hvert sekund, og fortsatt være innenfor vårt operasjonelt krav om 'responstid under 1000 ms'. Denne testen sender forespørsler til alle tre former av ressurser

tilgjengelig hos hvert utregningsendepunkt, så vi ser på den gjennomsnittlige responstiden for å få en sjekke om vi faller fortsatt innenfor kravet. Det vi kom frem med, vises i vedlegg I.

Vi ser at tjenesten kan ta inn 9.73 forespørsler i sekundet og fortsatt operere innenfor våre krav om vi ser kun på gjennomsnittstiden for hver forespørsel. Det er viktig å merke at en del av endepunktene var over dette kravet, men disse er de intensive utregningsfunksjonene vi var allerede kjent med til å være vanskelig å holde innenfor kravet. Allikevel, ser vi at gjennomsnittstiden ligger på 958 ms dersom 11 virtuelle brukere sender forespørsler konstant til tjenesten.

Når det gjelder belastningstester, tok vi basis i dette, og utførte en test med to virtuelle brukere. Denne testen vil ikke se på gjennomsnittstiden slik den forrige, men heller om tjenesten kan sende forespørsel under 1000 ms hver for seg, slik at kravet kan i hvert fall oppholdes for dem. Resultatene av denne testen vises i vedlegg J.

Vi ser at testen klarer å takle mengden av brukere helt fint under de operasjonelle kravene. Den gjennomsnittlige responstiden er 307 ms, hvor den lengste forespørselen (`/xy?plot`) tok 884 ms gjennomsnittlig. Tjenesten behandler 4.55 forespørsler i sekundet, og med tanke på hvordan frontend trenger 3 for hver gang en bruker skal bruke den tjenesten, så betyr det i teorien at de kan enkelt hente inn alle ressurser de trenger innenfor kravet vårt i lett nettverkstrafikk.

Vi mener at vi er fornøyd med resultatene fra ytelsestestene vi har utført her. Det er allikevel viktig å nevne at tjenesten ble utrullet med fire arbeidsprosesser - og at det vil finnes potensiale å gjøre ytelsen bedre med å enten øke denne mengden, eller med å utrulle flere servere med applikasjonen; noe mer relevant for det neste kapittelet.

8.2 Kodekvalitet

I tillegg til testene beskrevet tidligere, har vi forsikret oss god kodekvalitet i en rekke andre aspekter. De ligger beskrevet under:

8.2.1 Frontend

Tiltak for å sikre god kodekvalitet i frontend delen av prosjektet omfatter ESlint, React.Strictmode og statisk typesjekking gjennom TypeScript. ESlint er et fleksibelt verktøy som legger til rette for å inkludere anbefalte regler fra TypeScript, JavaScript og React, samt å definere egne regler for kodepraksis. Ved brudd på de definerte reglene vil man få advarsler eller feil, noe som gjør det lettere å fange opp og rette opp i dårlig kodepraksis og struktur.

React.Strictmode er et verktøy som aktiverer flere sjekker og advarsler når det oppdager potensielle problemer knyttet til livssyklus, bruk av avviklede tredje-

partsbiblioteker og bruk av funksjoner som er frarådet i moderne React applikasjoner [48].

TypeScript har, som nevnt i kapittel 5.2.2, funksjonalitet for statisk sjekking av variabeltype, noe som sikrer at variabler og returverdier er av riktig type under utvikling, og hindrer bugs og kjøretidsfeil.

8.2.2 Backend

- Det ble brukt PEP8 som en kodingsstandard i backend [49]. Denne standarden detaljerer regler angående innrykk av koden, navn, mellomrom og flere andre for å forsikre seg at koden en lager er fin og god. Dette ble forsikret seg gjennom PyCharm sine innebygde funksjonaliteter for det, sammen med bruk av biblioteket autopep8[1] for å automatisk formatere det.
- Det ble brukt versjonskontroll av ferdig utviklede iterasjoner av API, hvor Flaskrammeverket var `v1`, og Sanic er `v2`.
- Det finnes detaljerte kommentarer som beskriver hvorfor noe gjøres, sammen med hva en implementert bit er i tilfeller hvor kode kan være vanskelig å forstå.
    - Alle kommentarer har også med ordentlig henvisning av gjenbrukt kode (mest primært for kode fra basisprogrammet, hvor det oppgis lenke, modul og kodelinjer), men også en instans av noe basert på et svar fra nettsider som StackOverflow.

8.3 Lisensiering

Selv om dette er ikke en del av testing og kvalitetssikring, ønsker vi allikevel kort å nevne bruken av lisensiering. Som en del av dets rolle som en kontinuering av basisapplikasjonen, så vil vårt prosjekt også få en lisensiering. Denne lisensen er av GNU General Public License [2], og forsikrer seg programvare er fri programvare[3].

Gjennom denne lisensieringen, forsikrer vi oss at applikasjonen er fri, noe vi mener kontribuerer til å øke programmets brukbarhet i fremtiden. Dette kan gjøre programmet bedre å jobbe med i fremtiden dersom oppdragsgiver ønsker å ekspandere på prosjektet etter den endelige tidsrammen.

[1] `https://pypi.org/project/autopep8/`, hentet 20.05.2024
[2] `https://www.gnu.org/licenses/gpl-3.0.en.html`, hentet 20.05.2024
[3] `https://no.wikipedia.org/wiki/Fri_programvare`, hentet 20.05.2024

Kapittel 9

Utrulling & Installasjon

9.1 Utrulling

For utrullingen av tjenesten, bestemte vi oss å bruke Docker og dets kontainerbaserte løsninger. Vi ønsket å pakke inn vår applikasjon med alle dets avhengigheter til tredjepartisbiblioteker inni kontainere. Sammen med en `Dockerfile` for å inneholde operasjoner, kunne vi enklere starte opp system for det, og derfor enklere for oppdragsgiver.

Den første ideen var å bruke en kontainer for både tjeneren og frontend, hvor backend skulle tjene frontend gjennom seg selv. Dette var vurdert i møter, men for oppgaven valgte en annen løsning: Å videre dele opp komponentene som skulle tjenestegjøres, med både backend og frontend som egne kontainere. Mens backend skulle kjøres av Sanic sin egne webserver med fire arbeidsprosesser, skulle frontend tjenes av en webserver tilgjengelig i biblioteket Serve[1]. Det kan menes at dette er unødvendig og tilsetter kompleksitet til arkitekturen, men vi mente at denne separasjonen økte tilgjengeligheten av hele tjenesten. Med å ha dem som separate komponenter, kan brukere i hvert fall fortsatt ha nettsiden fremme dersom det er høy trafikk hos backend.

En annen fordel med denne oppdelingen er at det skal være enklere å øke skalerbarheten for backend; med å ha dem separat, så kan en eventuell bruker av programvaren lage flere instanser av kontaineren for backend.

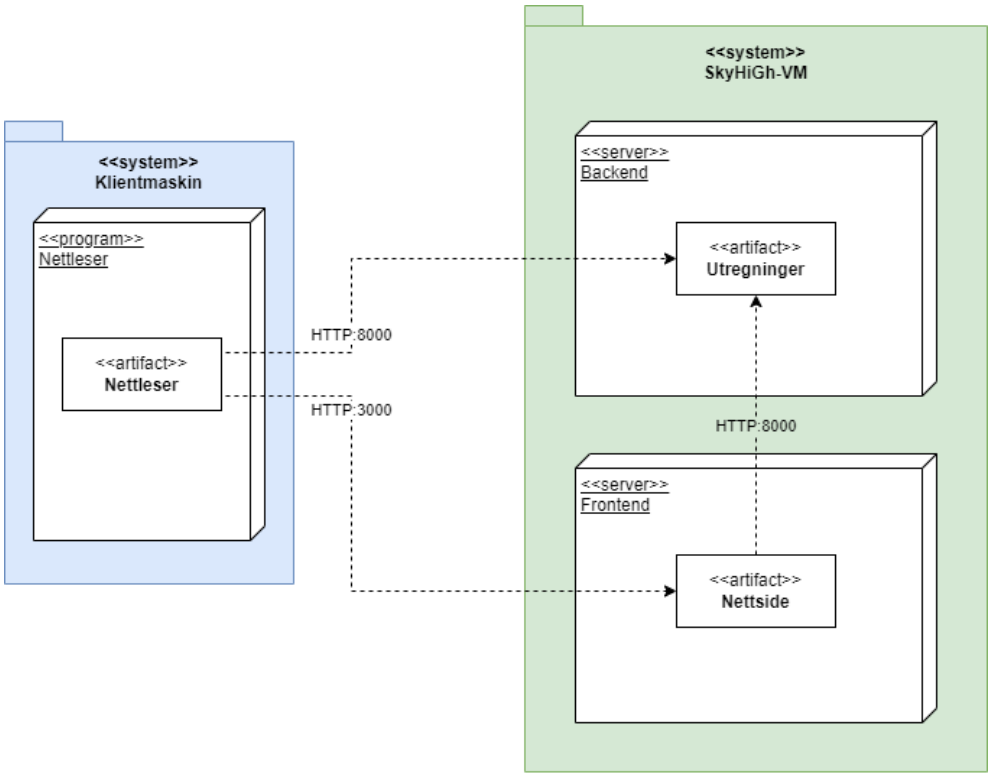
Bruken av dette oppsettet dannet to `images` fra Docker. For å få dem til å begge kjøre, brukte vi `docker-compose` for å kjøre en multi-container Docker applikasjon. På en slik måte, utfører vi at sluttbruker har separat tilgang til både applikasjonen i frontend, og API i backend. Vi benytter oss av 'port mapping'[2] for å justere nettverksportene for hver av komponentene. Backend-tjenesten kjører på port `:8000` inne i containeren og er tilgjengelig på samme port på vertsmaskinen. Frontend-tjenesten kjører på port `:3000` inne i containeren, men er

[1] `https://www.npmjs.com/package/serve`, hentet 20.05.2024

[2] `https://docs.docker.com/network/#published-ports`, hentet 20.05.2024

tilgjengelig på port `:80` på vertsmaskinen. Dette gjør at backend kan nås via `http://<server-ip>:8000`, og frontend kan nås via `http://<server-ip>:80`.

En enklere presentasjon av hele utrullingen finnes i figur 9.1.



Figur 9.1: Utrullingsdiagram for hele tjenesten
Klientmaskinen kommuniserer gjennom HTTP og porter til enten tjeneren til frontend, eller tjeneren til backend. Frontend vil selv kommunisere med tjener til backend.

I sammenheng med denne bacheloroppgaven blir utrullingen et bevis av konsept. Det er produkteier som må ta den offisielle utrulling videre. Det ble i denne sammenheng anskaffet en virtuell maskin gjennom NTNUs SkyHiGh tjeneste for virtuelle maskiner og servere. Hos en utgitt virtuell maskinen, ble vårt GitLab prosjektet klonet og satte på maskinen. Etter at vi hadde skapt våre kontainere på våre lokale maskiner, var nå `docker-` og `docker-compose` filer ferdig satt opp hos maskinen. Den eneste endringen som trengtes var å endre konstanten for APIets URL. Denne bruker nå den virtuelle maskinens IP adresse istedenfor `localhost`, som benyttes når applikasjonen kjøres lokalt. Når dette var gjort var det bare å bygge Docker, og vi fikk det til å starte. Applikasjonen og API er nå tilgjengelig for en hver bruker som er tilkoblet NTNUs nettverk.

Vi hadde skrevet tidligere at maskinvaren brukt var utenfor vår oppgave, med

tanken at oppdragsgiver skulle selv velge det som passet best for dem å tjenestegjøre applikasjonen i fremtiden med - men for å vise kompetanse, ønsker vi også å skrive noen anbefalinger.

Som sagt, benytter Sanic flere arbeidsprosesser for å øke skalerbarheten til tjenesten. Denne prosessen krever flere CPU-kjerner, hvor vi kan anta at to arbeidsprosesser passer per kjerne. I serveren vi benytter finnes det to kjerner, så vi har satt tjenesten til fire arbeidsprosesser, noe som gir gode resultater fra 8.1.2. Dersom oppdragsgiver ønsker å øke skalerbarheten, er det første gode steget å øke antall kjerner på hovedprosessoren til en virtuell maskin, og deretter kjøre containeren med backend med nye innstillinger for arbeidsprosessene.

De andre spesifikasjonene er ikke så viktig, utenom at det burde være minimum 2 GB RAM og nok lagringsplass til tjenesten. Operativsystemet til programmet er heller ikke viktig, for Docker sine konteinere er designet for å kjøre uavhengig av operativsystem. Allikevel, anbefaler vi Linux-baserte distribusjoner for at Docker skal kjøre innebygd i systemet (for det bruker virtuelle maskiner på andre operativsystemer). Linux tilbyr også et sterkt miljø for nettverksadministrasjon og sikkerhet, med enklere oppsetting og vedlikehold av maskinene.

9.2 Installasjon

Programmet har blitt om til kontainere gjennom Docker, og skal derfor være enkelt å installere på flere typer enheter. Sanic støtter også Docker[3], så man kan kjøre følgende prosedyre for å sette opp miljøet:

1. Last ned Docker hvis det ikke allerede er installert.
2. Klone prosjektet fra fra GitHub gjennom disse lenkene:
    - SSH: `git@github.com:group8ntnu2024/ciefunctionsweb.git`
    - HTTPS: `https://github.com/group8ntnu2024/ciefunctionsweb.git`
3. Naviger til rotmappa til prosjektet
4. Kjør:

```
docker-compose up --build
```

For at en klient skal kunne kommunisere med Sanic-konteineren, må portene 8000 og 80 være åpne for trafikk i begge retninger. Dette sikrer at forespørsler og svar kan flyte fritt mellom nettleseren og Sanic-tjeneren og webserveren.

9.3 Demo

Vi har allerede installert programmet på en Ubuntu-server, og den kan besøkes på følgende nettadresse:

[3] `https://sanic.dev/en/guide/deployment/docker.html`, hentet 20.05.2024

http://10.212.136.66/

Linken gir tillgang til tjenesten som er rullet ut på serveren slik at den som ønsker kan navigere og teste funksjonalitetene som har blitt utviklet og implementert. Vær oppmerksom på at denne lenken kun vil fungere innenfor NTNU sitt nettverk. Dette betyr at tjenesten er tilgjengelig for testing og bruk for alle enheter som er koblet til NTNUs interne nettverk, og den gir en mulighet for å se løsningen i aksjon.

Kapittel 10

Diskusjon & Konklusjon

Gjennom prosjektperioden, har vi gjennomgått både utfordringer og suksesser som har vært med på å forme vårt endelige produkt. I dette kapittelet vil vi reflektere over vår hvordan vi har arbeidet, diskutere resultater og trekke konklusjoner basert på det vi har gjort og lært.

10.1 Drøfting

10.1.1 Resultatsmål

Vi mener at vi har klart å oppfylle det resultatsmålet forventet av oss. Gjennom dette prosjektet, mener vi at vi har klart å lage den ferdigutviklede webapplikasjonen slik vi har fortalt om den tidligere i kap. 7.

Programmet dekker alle kravene vi har satt for oss slik skrevet i kap. 3.

- Mesteparten av våre funksjonelle krav er dekket, slik de er beskrevet gjennom use-case og deres diagram. Det er mulig av en bruker å 'velge funksjon og brukerinput', for å da få representasjoner av utregninger basert på det de ga. De kan også interagere med grafer, finne spesifikke verdier, og aktivere valgfri parametere, i tillegg til flere andre funksjonaliteter som finnes i basisprogrammet.
    - Noen ting forblir uferdig, men de tas opp i kap. 10.3.
- Programmet dekker også alle ikke-funksjonelle krav;
    - Applikasjonen har et brukergrensesnitt som er likt til det hos basisprogrammet.
    - Applikasjonen følger ytelseskravet vi har satt her, som bevist i våre tester hos kap. 8.1.2. Vi ser at ved vanlig belastning, så klarer den å gi responstider raskt, gjennom bruken av moduliserte funksjoner slik beskrevet hos kap. 6.2.1 og gode tiltak for skalerbare tjenester slik fortalt i kap. 9.1. Dette punktet gjelder også for det tredje kravet om skalerbarhet.

    - Applikasjonen skal ha åpen lisensiering slik vi hadde dekket det i 8.3. Dette dekker også gjenbruken av kode. Vi har også sørget for dokumentasjon om utrulling hos `README` filen på kodelageret.
- Vi har også sett gjennom kap. 8.1.2 at våre operasjonelle krav dekkes i det siste produktet, mesteparten. Tjenesten vår klarer å takle flere brukere samtidig gjennom Sanic sine asynkrone evner, samtidig som at gjennomsnittstiden for alle forespørsler fra tjenesten ligger under 1000ms.

10.1.2 Effekt- & Læringsmål

Med tanke på lærings- og effektmålene vi hadde deklarert tidligere i prosjektet i kap. 1.2.2, mener vi at vi har klart å utføre det vi hadde ønsket der.

Når vi ser på effektmålene, mener vi at vi har klart å oppfylle dem til en stor grad. Gjennom vår raske og høyt-tilgjengelige (kap. 8.1.2), mener vi at vi har definitivt gjort det enklere for fargesynsforskere å få de beregningene de trenger. Dette gjør også opplevelsen mer komfortabel til forskere - hvor alle beregninger er tilgjengelig på en enkel nettside, istedenfor en egen skrivebordsapplikasjon. Vi har også lagt merke til at vi selv har blitt mer interessert i forskningsemnet. Vi satte ekstra oppmerksomhet i kap. 2.1 med omfattende møter med og tilbakemeldæinger fra oppdragsgiver og veileder - som begge er medlemmer av Fargelabben.

Når det gjelder våre læringsmål, tenker vi også at vi har fått mye ut av dette prosjektet.

Vi har definitivt lært mye om de tradisjonelle nettrammeverk sammen med nyere teknologier, spesielt fra testingen og vurderingen av rammeverk. Vi kan egentlig si at prosjektets essens baserer seg rundt dette: Hvordan vi gikk fra bestemmelsen mellom PyScript og Flask og deretter til å revurdere Flask med andre rammeverk. Dette har gitt oss et stort innsikt i utviklingen av API og webutvikling generelt.

Vi har også lært mer om relevante og aktuelle klientrammeverk innen webutvikling. Gjennom våres grader, lærer vi vanligvis om det elementære - så vi har ikke fått en reell sjanse å se på slike teknologier før. Selv om det var en risiko med tanke på oppgaven, gikk vi fremdeles med å bruke ny teknologi som er relevant i arbeidslivet, slik at vi kunne spesifikt lære mer om det. Når utviklingsprosessen var konkludert innså vi hvor lurt dette var. Vi har alle oppnådd nye høyder innen kodeforståelse og webutvikling .

Det har også blitt lært mye om programvareutvikling og erfaring med å jobbe i grupper. Selv om vi har hatt gruppeprosjekter for store programmer jevnlig gjennom våre grader, så er en bacheloroppgave langt større enn dem. Derfor satte vi ekstra fokus på å utføre profesjonelle etikette og få så mye erfaring fra dette som mulig, noe vi mener vi har lykkes i både gjennom ulemper og fordeler. Dette tas opp mer i diskusjon i evalueringen av gruppearbeidet, hos kap. 10.4.

Til slutt, mener vi at vi har lært flere ting for det siste målet; om hvordan vi skal

formidle dette til en rapport og presentasjon. Etter omfattende tilbakemeldinger fra vår veileder, ble det satte inn mye innsats av alle parter i å søke forbedring. Vi lærte mye fra dem, som blant annet riktigere struktur av diagrammer og bedre ordbruk for å unngå misforståelser i teksten.

10.1.3 Bærekraft

Vi mener også at prosjektet dekker flere aspekter innenfor bærekraft, noe vi skal ta for oss gjennom følgende delkapitler innom det miljømessige, det økonomiske og det sosiale.

Sosial bærekraft

Et felt innen bærekraft som vårt prosjekt falle innunder, er sosial bærekraft. Vi bidrar til samfunnet med å tilby en enklere måte for fargesynsforskere å få tak i sentrale utregninger nødvendig for deres forskning. Dette kan videre tilknyttes med FNs bærekraftmål [50], hvor dette passer godt inn hos delmål 9.5[1]: Vi styrker forskning i sektorer hvor fargematchfunksjoner er relevant med å tjenestegjøre med vår applikasjon.

Et annet felt som vi styrker til den sosiale bærekraften, er gjennom den enkle tilgjengeligheten av applikasjonen. Fordi den finnes over internettet, kan alle slags brukere enklere finne frem til utregningene. Dette styrkes videre av det responsive og universelle utseendet programmet har, slik vist frem tidligere i 7.3.

Miljømessig bærekraft

Vår applikasjon bidrar også i stor grad til miljømessig bærekraft.

Først og fremst, bruker programvaren sofistikerte teknikker hos alle komponenter for å forsikre seg mindre unødvendige utregninger. Gjennom React sin optimaliserte 'rendering' (5.2.1), TypeScript sin type-sikkerhet (5.2.2), backend sine feilmeldinger (7.2.2) og mulighet for cache (3), så bidrar vi til at tjenesten unngår å utføre unødvendige utregninger som tar opp tid og energi.

Programmet er også pakket inni kontainere fra Docker, som bidrar til bruken av mindre ressurser for å tjenestegjøre programmet. Til sist, gjenbrukes det eksisterende kode fra kodelager med åpen kildekode, blant annet basisprogrammet og tilgjengelige biblioteker. Gjennom deres bruk, reduserer vi ellers mengden vi ville ha trengt for å lage det selv, sammen med at vi gjenbruker noe som allerede eksisterer.

Økonomisk bærekraft

Til sist, dekker programmet også en liten del innenfor økonomisk bærekraft. Koden vi har laget for dette prosjektet skal være fri kildekode med åpen lisens, slik

[1] `https://fn.no/om-fn/fns-baerekraftsmaal/industri-innovasjon-og-infrastruktur`, hentet 20.05.2024.

tidligere sagt i kap. 8.3. Dette bidrar til at flere målgrupper kan bruke programvaren, irrelevant av deres økonomiske status.

10.1.4 Bruk av kunstig intelligens

Bruk av KI i akademia byr på en del etiske og moralske spørsmål. Overbruk eller avhengighet av KI kan undergrave studentenes egen læring og kreativitet. Det er viktig å benytte KI på en måte som fremmer og ikke hindrer kritisk tenkning.

På grunn av dette, har det blitt satt fokus på å bruke KI minimalt for dette prosjektet. Vi ble tidlig enige om at vi ikke ønsket å benytte det i stor grad, og heller vise frem hva vi selv klarer å produsere. Når det er sagt, er kunstig intelligens et veldig brukbart verktøy som kan gi store fordeler når det brukes moralt og teknisk riktig. For oss var dette først og fremst feilsøking. Kodetolkeren til mange tilgjengelige kunstige intelligens finner ofte fort slurvefeil eller forglemmelser man har gjort i løpet av koding. For dette, har KI da blitt brukt som et verktøy for å oppdage disse feilene. I tillegg til dette, har også KI blitt brukt som en læringsassistent. Konsepter vi støtte på under utvikling og rapportskriving som vi slet med å forstå kan KI lett gi oss essensen av uten at man må lese gjennom mange artikler. Dette gir oss en overordnet forståelse som gjør videre læring betydelig lettere.

KI ble benyttet til å generere små deler av kode for oss. Dette er kun småting som ikke faller innenfor kodeforståelse, og kunne ha blitt gjort selv. Et eksempel er fargekoder for elementer i CSS-filene og importeringer for eksterne biblioteker. Et unntak til dette var når vi skulle implementere Docker for tjenestegjørelse. Utrulling er en veldig intrikat prosess hvor mye kan gå galt. Da ble KI benyttet for å få et førsteutkast av dockerfilene. Det er viktig å merke at dette virket ikke 'rett ut av boksen' og vi måtte gjøre betydelige endringer for å få det til å fungere som det skulle.

Når det kom til rapportskriving, ble kunstig intelligens benyttet kun i renskriving, og ikke til noe generering. Når vi leste igjennom og oppdaget at setningsbyggingen var litt merkelig eller for lang, ble det av og til benyttet KI for å strukturere setninger anderledes for å øke lesbarheten for rapporten. Vi forsikret oss om at det ikke forekom noe tap av innhold eller mening. Det var også anbefalt et verktøy kalt LanguageTool[2] av vår veileder for renskriving, men dette fant vi lite nytte i. Verktøyet støttet ikke norsk helt, og ble kun brukt for å sjekke feilstavelser av ord.

10.2 Kritikk av oppgaven

10.2.1 Frontend

Frontend til applikasjonen er bygger med React og TypeScript. Siden react er komponertbasert har har vi bygget en modulær og gjenbrukbar kodebase. Dette gjør koden lettere å vedlikeholde og videreutvikle. React har et stort økosystem av og

[2] `https://languagetool.org/`, hentet 21.05.2024

tredjepartisbiblioterker som har vært til stor hjelp og akselerert utviklingsprosessen.

Det har også byd på noen utfordringer ved å benytte React. Læringskruven har vært ganske bratt og det var ganske innviklet og vanskelig å forstå i oppstartsfasen. På grunn av dette bygget vi inn en del grunnleggende hindringer for oss selv som var vanskelig å nøste opp i senere. Dette var spesielt et problem når det kom til design av layout og flytte rundt på komponenter senere i utviklingsfasen. React krevde også mye oppstartsarbeid for å få det til å fungere i det lokale utviklingsmiljøet.

Vi hadde alle litt erfaring med JavaScript fra før så overgangen var ikke veldig utfordrene.

Selv om vi har hatt en fokusert og effektiv utviklingsprosess, må vi innrømme at vi dessverre ikke har utført noen formelle tester på frontend-komponentene. Dette er et område som krever oppmerksomhet i fremtidige iterasjoner av prosjektet for å sikre at brukergrensesnittet fungerer feilfritt og gir en optimal brukeropplevelse. Til tross for dette har den praktiske brukstesten vist at løsningen fungerer tilfredsstillende i de fleste scenarioer.

10.2.2 Backend

Når det gjelder backend, ville vi ha sagt at vi er fornøyd med flere aspekter, men også at det finnes ulemper med vår implementasjon.

Den største delen vi er fornøyde med, er dets optimalisering gjennom prosjektutviklingen og ytelsen den tilbyr som en tjeneste. Utover hele prosjektperioden, har det hele tiden vært et stort fokus på å utlevere noe raskt og effektivt. Det som startet som en enkel prototype med lite initiell funksjonalitet, ble eventuelt om til en komplett tjeneste med forbedringer og optimaliseringer i flertall. Det ble gjort flere typer av optimaliseringer fra det vi startet med; fra å modulisere utregningsfunksjoner, til et komplett skifte av rammeverket vi bruker. Alt av dette ble gjort for å minske på responstiden og gjøre programvaren vi utleverer så tilgjengelig og brukbar som mulig. Vi tar mye stolthet i dette, og er derfor aldeles fornøyd med ytelsen tjenesten tilbyr. Et annet aspekt verdt å ta med, er at det ble ikke brukt noe kunstig intelligens for kodeutviklingen av denne komponenten. Alt av kode som finnes i API for dette prosjektet ble produsert enten manuelt eller med gjenbruk tydelig deklarert i både kode og dokumentasjon. Det ble heller ikke brukt for forslag for løsninger, heller annen form av hjelp der. Vi mener at dette demonstrerer god beherskelse av programmering og oppgaven i sin helhet, og videre styrker på noen av våre læringsmål diskutert tidligere. Til sist, så vi er fornøyd med at det har blitt utført ordentlig enhets- og integrasjonstesting gjennom dets utvikling for å forsikre seg tilgjengelighet og kodekvalitet. Vi ser god kvalitet i det vi har laget, og det er hele tiden noe å være stolt av.

Et stort aspekt vi ikke er fornøyd med, er bruken av diagram som en ressurs. Dersom vi ser på oversikten over alle forespørsler hos vedlegg I, ser vi en klar trend at endepunktene med `/plot` tar lengre responstid enn andre endepunkter. Dette

forårsakes av at tjenesten selv utfører flere utregninger på sin side for å utregne alle verdier. Vi mener også at det skal være vanlig for frontend å takle data fra backend for å skape disse diagrammer; ikke at backend lager dem selv. Noe likt kan sies om sidemeny som en ressurs, hvor klientsiden skal også ha ansvaret for tegningen av dette. Men vi ser at responstiden for dette er ikke så høy som de er hos diagrammer. Om dette skal skiftes, er det opp til potensielle fremtidige utviklere av programvaren.

10.3 Videre opplegg

10.3.1 Manglende funksjonalitet

Selv om programmet vårt oppfyller mange av de grunnleggende kravene for å visualisere data er det noen få funksjonaliteter som ikke ble implementert. Det første, og mest kritiske, er at det ikke mulig å lagre tabellen som vises. Dette begrenser muligheten for videre bruk og deling av utregnede resultater. Videre mangler programmet funksjonalitet for å vise noen av grafene renormalisert. Dette skulle vært implementert gjennom en avkryssningsboks og ville gitt brukerene større fleksibilitet og kontroll over datavisualiseringen. Symbolene på toppen av grafen er heller ikke formatert riktig. Dette kan i verste fall resultere i missforståelser eller feil tolkning av dataene. Til slutt mangler programmet en 'About' seksjon. Denne er viktig for å gi brukerene informasjon om programmets formål, utviklere og versjonshistorikk. Disse manglene bør adresseres i fremtidige iterasjoner av applikasjonen for å forbedre funksjonalitet.

10.3.2 Bedre design

Vi kunne ha lagt mer vekt på dette estetiske designet av applikasjonen. På grunn av prosjektets natur som et vitenskapelig verktøy valgte vi imidlertid å prioritere funksjonalitet og presisjon fremfor estetikk. Vårt hovedfokus var å skikre at applikasjonen kunne utføre de nødvendige beregningene, og presentere dataene på en klar og forståelig måte. Det estetiske aspektet ble derfor anset som sekundært.

En annen ting vi kunne tatt mer i betraktning var universelt design. Vi vurderte å inkludere funksjonalitet for å bedre brukervennligheten og tilgjengeligheten av applikasjonen for mennesker med fysiske nedsettelser. Vi anså ikke dette som veldig kritisk da vi skulle utvikle en applikasjon som kun benyttes av en håndfull forskere verden over. Vi tenkte dette kunne være noe ekstra vi kunne inkludere dersom det var tid til det.

10.3.3 Nyere diagramfremvisning

Måten diagrammene ble implementert på ansees for å være litt gammeldags. Sidemenyen og plottene bygges og leveres direkte av python APIet istedenfor å bli laget av frontenden bassert på data levert av APIet. Dette kommer med noen fordeler og ulemper. Den største fordelen er at plotbyggingen får direkte tilgant til

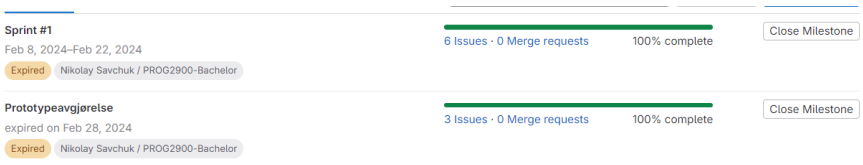
plotpunktene fra compute filen i backend. Dette forhindrer inviklet databehandling som ellers måtte ha blit gjort frontend. En ulempe er at dette gjør forntend avhengig av backend for layout og visning av elementer. I tilleg til at det strider med prinsippet 'løs kobling og høy sammenheng' reduserer det fleksibiliteten og responsiviteten i designet. Diagrammene er ikke responsive i den forstand at de opererer med faste størrelser for å sikre en 1:1 skala. Applikasjonens sidemeny plasserer seg under graf/tabellen hvis vinduet til nettleseren forminskers, men tabellen/grafen blir ikke mindre. Dette betyr at diagrammene ikke tilpasser seg ulike skjermstørrelser optimatlt. Dette kan være problematisk for brukere som ønsker å bruke applikasjonen på forskjellige enheter. Manglen på responsivt design kan redusere den generelle opplevelsen av applikasjonen.

10.4 Evaluering av arbeidet

10.4.1 Utviklingsprosessen

Som skrevet tidligere, valgte vi å bruke scrum for vår utviklingsmetodikk. Vi hadde jobbet med det før med forskjellige grader av suksess, og mente at vi skulle bruke dette prosjektet for å lære oss mer om hvordan man skulle bruke det i profesjonell sammenheng. Nå som vi tenker tilbake på vår prosess, mener vi at det kunne ha gått bedre - men vi er fortsatt fornøyd med hvordan metodikken gikk.

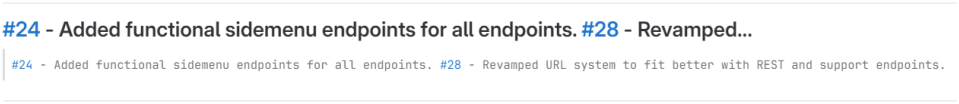
Vi mener at bruken av et scrumboard sammen med sprintperioder gjorde det enklere å organisere hvordan arbeidet skulle fordeles oss i mellom. Det var enkelt å organisere og huske hva vi måtte gjøre til enhver tid, samtidig som det var direkte mulig å endre på de registrerte oppgavene. Sprintperiodene var også satte opp med gode lengder. Vi følte at perioder med lengder av en uke var for korte, og ville føre til mye stress. Det ble også brukt milepæler for å holde kontroll på sprintperiodene, slik vist i figur 10.1.



Figur 10.1: Utdrag av milepæler fra prosjektet.
Milepælene i bildet dekker både den første sprinten, og en del av den andre sprinten hvor avgjørelsen for prototypen måtte utføres.

For å holde styr på prosjektet, benyttet vi oss av GitLab[3] for versjonskontroll som kodelager. Dette har hele tiden vært en standard for oss gjennom våre utdannelser. I løpet av kodeutviklingen, benyttet vi oss bruk av unike IDer tilknyttet våre gitte oppgaver på scrumboardet, slik vist på figur 10.2.

[3] `https://about.gitlab.com/`, hentet 20.05.2024



Figur 10.2: Eksempel på 'commit' med bruk av 'issue' ID for god dokumentasjon av arbeid.

Dette systemet ble brukt i varierende grad, men vi mener allikevel at det er noe vi burde ta med oss til fremtiden. Dette er bedre profesjonell etikette, for det dokumenterer eksakt hvilke oppgaver ble jobbet på til enhver tid.

Noe annet positivt var de ukentlige møtene med både veileder og oppdragsgiver. Det er viktig for oss å lage et prosjekt som holder seg innenfor både hva som er forventet fra en veileder, og hva som er ønsket fra en oppdragsgiver. Det er takket være dem at vårt prosjekt har blitt som det er.

Men, noen ulemper med bruken av scrum var nok våre tilnærminger av arbeidsoppgaver i produktkøen, og vedlikehold av det.

Tidlig i prosjektet lagde vi produktkø uten noe kjennskap til hvor stor eller liten en gitt oppgave skulle være - heller, baserte vi det på våre tidlige estimeringer fra prosjektets start. På grunn av dette, fant vi ut at enkelte arbeidsoppgaver tok lengre tid enn forventet. Dette konkluderte i at flere av sprintene måtte bli utvidet for oppgaver slik presentert i kap. 6. Vi mener at dette var forårsaket av mangel på erfaring innenfor mange nye teknologier vi aldri hadde brukt før. Dette førte til en følelse av at vi ikke gjorde nok, og tidene vi hadde satt av til utvikling gikk rask ut. Mangelen på realistiske tidsrammer og uforutsette tekniske utfordringer bidro til at vi ofte måtte justere tidsplanen vår. Dette skape en del stress og bekymring for om vi ville klare å fullføre i tide.

Scrumsystemet krevde også mye vedlikehold for å fungere ordentlig. Fordi det var ikke mye utvikling på starten av prosjektet, var alt opprettholdt godt i systemet - men etterhvert som utviklingen foregikk og arbeidsmengden begynte å øke, begynte vi å ha lite tid for opprettholding. Dette forårsakte at enkelte deler av systemet som milepæler ble eventuelt ikke brukt videre i prosjektet; kun scrumboard og dets issues var de eneste aspektene igjen for vedlikehold.

Allikevel, lærte vi fortsatt mye om arbeidsmetodikk og utviklingsprosessen gjennom dette. Vi lærte mye om hvor viktig det var å ha fleksibilitet og god kommunikasjon gjennom utviklingsperioden, sammen med importansen av å ha en solid alternativ plan for 'når ikke går som planlagt'. I korte ord, dersom vi skulle evaluere oss på utviklingsprosessen, ville vi ha sagt at vi er 'god, men trenger justeringer'.

10.4.2 Rapport

Vi mener at rapporten kunne ha vært bedre og mer omfattende dersom den hadde blitt skrevet på mer gradvis over en lengre periode. Istedenfor å dokumentere

arbeidet vi gjorde grundig underveis, hadde gruppa en tendens til å fokusere mest på utviklingen av applikasjonen. Dette førte til at rapporten ble hovedsakelig skrevet mot slutten av prosjektet, som videre betydde et stort tidspress på oss helt opp til leveringen av oppgaven.

Å fokusere mer på utviklingen tidlig i prosjektet var nødvendig for å både finne arkitekturen som oppgaven skulle basere seg på, samtidig som å få en god start på eventuell videreutvikling. Vi må allikevel innrømme at en bedre balanse mellom utvikling og skriving av rapport ville ha styrket dokumentasjonen vår betydeligere, og vi ville ha unngått mye stress senere i prosjektperioden. En bedre tilnærming til rapportskrivingen kunne ha inkludert:

- Oppdateringer om fremdrift etter hver sprint.
- Oppdateringer om utfordringer vi møtte på underveis.
- Løpende dokumentasjon av beslutninger vi gjorde.
- Identifisering kritiske områder for diskusjon og analyse.

10.4.3 Annet arbeid

Noe vi har lagt merke til for seint i prosjektperioden, er at vi ikke har ført timer over arbeidet vi hadde gjort for prosjektet. Selv om det fantes et system for det tidligere i prosjektet, så ble dette også glemt når prosjektet begynte å kreve mer arbeid - noe som kan knyttes til vedlikeholdsproblemet vi hadde beskrevet tidligere. Selv om det var ikke et så stort problem utover prosjektperioden, erkjenner vi at det ville ha vært nyttig både for rapportskriving og dokumentasjon. Dette ville ha gitt våre lesere og andre utviklere som kan jobbe videre med programvaren en kvantitativ mening om hvor mye innsats ble satt inni prosjektet. Selvfølgelig, skal vi ta dette med oss til fremtiden ved profesjonelle sammenheng og prosjekter.

Det samme kan sies om våre møtereferater. De ble laget tidlig i prosjektet, men når utviklingen tok av - så gikk det også bort. Møtereferater ville ha vært svært nyttig både for rapporten, men også for utviklingsprosessen. Med å skrive dem, ville vi ha hatt bedre kontroll over eksakt hvilket arbeid måtte gjøres, sammen med offentlig dokumentasjon på sitater fra møtene til å bruke i rapporten. For oss var det ikke et stort problem under utviklingsprosessen gjennom vår scrumboard og produktkø, men nå som vi skriver dette, anerkjenner vi her også at det ville ha vært fint å inkludere dem.

10.5 Konklusjon

Gjennom denne bacheloroppgaven har vi fått god innsikt og erfaring innenfor webutvikling, som vil være uvurderlig når vi nå går inn i arbeidslivet. Vi har forsøkt å integrere prinsipper for bærekraft gjennom hele utviklingsprosessen, fra valg av teknologier til optimalisering av ressursbruk. Vårt arbeid har bidratt til å stryke vitenskapelig forskning gjennom å forenkle tilgangen til avanserte verktøy for visualisering av fargematchfunksjoner.

Noe av det viktigste vi har lært er gjennom dette prosjektet hvor viktig det er med fleksibilitet og å kunne tilpasse seg nye utfordringer. Selv om vi hadde en solid plan på plass møtte vi stadig på tekniske utfordringer. Dette har lært oss at å kunne juster tidsrammer og arbeidsmetoder ofte er avgjørende for å sikre at prosjektmålene oppnås.

Vi har også fått kjenne på hvor viktig det er med tidsstyring og kontinuerlig dokumentasjon. Mangelen på et system for timeføring gjorde det litt utfordrende å identifisere tidskrevende arbeidsoppgaver.

Vi har samarbeidet godt og det har vært en avgjørende faktor for prosjektet og har vært en viktig faktor for vår suksess. Selv om vi møtte på noen kommunikasjonsutfordringer, karte vi å løse disse gjennom regelmessige møter og klar fordeling av oppgaver. Dette har lært oss hvor viktig det er med effektiv kommunikasjon ar lagarbeid.

Samlet sett har vi prøvd å integrere prinsipper for bærekraft gjennom hele utviklingsprosessen, fra valg av teknologier til optimalisering av ressursbruk. Ved å styrke vitenskapelig forskning og forbedre tjenligheten til enkle verktøy for visualisering av fargematchfunksjoner, bidrar vi til en mer bærekraftig fremtid.

Bibliografi

[1] I. Farup, J. H. Wold, T.-H. Yang, G. D. de La Riva og Bartek, *CIE Functions*, versjon 1.0.2, Basisprogrammet for oppgave. adresse: https://github.com/ifarup/ciefunctions.

[2] K. Hofstad, «måling (naturvitenskap),» *Store norske leksikon*, 2021, Hentet 19. mai 2024.

[3] T. Holtebekk, «kolorimeter - fargebestemmelse,» *Store norske leksikon*, 2023, Hentet 19. mai 2024 fra https://snl.no/kolorimeter_-_fargebestemmelse. adresse: https://snl.no/kolorimeter_-_fargebestemmelse.

[4] J. Setchell, «8 - Colour description and communication,» i *Colour Design*, ser. Woodhead Publishing Series in Textiles, J. Best, red., Woodhead Publishing, 2012, s. 219–253. adresse: https://www.sciencedirect.com/science/article/pii/B9781845699727500084.

[5] M. Polo, *CIE1931 RGBCMF.svg*, Normalized RGB functions for monochromatic beams of light of specific wavelength in en: CIE 1931 color space., nov. 2007. adresse: https://commons.wikimedia.org/wiki/File:CIE1931_RGBCMF.svg.

[6] S. Sulyman, «Client-Server Model,» *IOSR Journal of Computer Engineering*, årg. 16, s. 57–71, jan. 2014. DOI: 10.9790/0661-16195771.

[7] M. Team, *Microsoft® Application Architecture Guide*. Microsoft Press, 2009, ISBN: 9780735642799. adresse: https://books.google.no/books?id=D9on897Ep7AC.

[8] H. Mili, A. Elkharraz og H. Mcheick, «Understanding separation of concerns,» jan. 2004. adresse: https://www.researchgate.net/publication/244446574_Understanding_separation_of_concerns.

[9] IBM, «Architectural characteristics of web-based applications,» jan. 2024, Hentet 18.05.2024. adresse: https://www.ibm.com/docs/en/db2-for-zos/13?topic=environment-architectural-characteristics-web-based-applications.

[10] Bart, *Overview of a three-tier application vectorVersion.svg*. adresse: https://commons.wikimedia.org/wiki/File:Overview_of_a_three-tier_application_vectorVersion.svg.

[11] J. Duckett, *HTML CSS - Design and Build Websites*. Wiley Press, 2011.

[12] J. Duckett, *JavaScript JQuery - Interactive front-end web development*. Wiley Press, 2014.

[13] D. Singh, «Understanding Browser Rendering: The Critical Rendering Path,» adresse: https://www.linkedin.com/pulse/understanding-browser-rendering-critical-path-divyansh-singh.

[14] Z. Gollwitzer. «How your browser loads, parses, and renders a webpage.» (2024), adresse: https://www.fullstackfoundations.com/blog/how-browser-loads-parses-renders-webpage (sjekket 20.05.2024).

[15] «Populating the page: how browsers work.» (), adresse: https://developer.mozilla.org/en-US/docs/Web/Performance/How_browsers_work (sjekket 20.05.2024).

[16] P. Irish og T. Garsiel. «How your browser loads, parses, and renders a webpage.» (2011), adresse: https://web.dev/articles/howbrowserswork (sjekket 20.05.2024).

[17] M. Kosaka. «Inside look at modern web browser (part 3).» (2018), adresse: https://developer.chrome.com/blog/inside-browser-part3/ (sjekket 20.05.2024).

[18] W. Preiser og E. Ostroff, *Universal Design Handbook* (M-H handbooks). McGraw-Hill, 2001, ISBN: 9780071376051. adresse: https://archive.org/details/universaldesignh0000unse/page/n27/mode/2up.

[19] E. Marcotte, «Responsive Web Design,» *A List Apart*, mai 2010. adresse: https://alistapart.com/article/responsive-web-design/.

[20] H. Kang, G. Liu, W. Quan, L. Meng og J. Liu, «Theory and Application of Zero Trust Security: A Brief Survey,» *Entropy*, årg. 25, nov. 2023. DOI: 10.3390/e25121595.

[21] I. C. Instructor, «The Five Languages or Dimensions of Interaction Design,» *Interaction Design*, 2016. adresse: https://www.interaction-design.org/literature/article/the-five-languages-or-dimensions-of-interaction-design.

[22] *Interaction Design: Beyond Human-Computer Interaction*. Wiley, 2002.

[23] IBM og M. Goodwin, «What is an API?» *IBM Explainers*, 2024. adresse: https://www.ibm.com/topics/api.

[24] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, Doctoral dissertation, University of California., 2000. adresse: https://ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf.

[25] F. Belqasmi, R. Glitho og C. Fu, «RESTful Web Services for Service Provisioning in Next-Generation Networks: A Survey,» *IEEE Communications Magazine*, årg. 49, s. 66–73, des. 2011. DOI: 10.1109/MCOM.2011.6094008.

[26] D. K. Wagh og R. Thool, «A Comparative study of SOAP vs REST web services provisioning techniques for mobile host,» *Journal of Information Engineering and Applications*, årg. 2, s. 12–16, jul. 2012.

[27] W3, *WebAssembly Core Specification*, W3C Recommendation, des. 2019. adresse: https://www.w3.org/TR/wasm-core-1/.

[28] D. Pockstaller, S. Huber og L. Demetz, «Comparing the Energy Consumption of WebAssembly and JavaScript in Mobile Browsers,» nov. 2023, s. 121–127. DOI: 10.5220/0012205600003584.

[29] GeeksForGeeks, «Floating point error in Python,» des. 2023, Hentet 19. Mai 2024. adresse: https://www.geeksforgeeks.org/floating-point-error-in-python/.

[30] P. J. Eby, «PEP 3333 - Python Web Server Gateway Interface v.1.0.1,» PEP 3333, 2010. adresse: https://peps.python.org/pep-3333/.

[31] Wikipedia contributors, *Human–computer interaction — Wikipedia, The Free Encyclopedia*, [Online; accessed 19-May-2024], 2024. adresse: https://en.wikipedia.org/w/index.php?title=Human%E2%80%93computer_interaction&oldid=1222972301.

[32] J. Nielsen. «Usability 101: Introduction to Usability.» (2012), adresse: https://www.nngroup.com/articles/usability-101-introduction-to-usability/ (sjekket 10.04.2024).

[33] J. Ighalo. «Using react-responsive to implement responsive design.» (2021), adresse: https://blog.logrocket.com/using-react-responsive-to-implement-responsive-design/ (sjekket 20.05.2024).

[34] GeeksForGeeks. «React Introduction.» (2024), adresse: https://www.geeksforgeeks.org/reactjs-introduction/ (sjekket 17.05.2024).

[35] I. Akinyemi. «Build Responsive Web Pages With React-Responsive And TypeScript.» (2021), adresse: https://blog.openreplay.com/build-responsive-web-pages-with-react-responsive-and-typescript/ (sjekket 20.05.2024).

[36] J. Lanctot. «The key ingredients of RESTful APIs: resources, representations, and statelessness.» (2023), adresse: https://www.torocloud.com/blog/the-key-ingredients-of-restful-apis-resources-representations-and-statelessness (sjekket 20.05.2024).

[37] Wikipedia contributors, *Flask (web framework) — Wikipedia, The Free Encyclopedia*, [Online; accessed 19-May-2024], 2024. adresse: https://en.wikipedia.org/w/index.php?title=Flask_(web_framework)&oldid=1222790474.

[38] Wikipedia contributors, *Model–view–controller — Wikipedia, The Free Encyclopedia*, [Online; accessed 19-May-2024], 2024. adresse: https://en.wikipedia.org/w/index.php?title=Model%E2%80%93view%E2%80%93controller&oldid=1222042229.

[39] C. Gackenheimer, *Introduction to React*. Springer Science, 2015.

[40] A. Banks og E. Porcello, *Learning React: Functional Web Development with React and Redux*. O'Reilly Media, 2017, ISBN: 9781491954577. adresse: https://books.google.no/books?id=pMTADgAAQBAJ.

[41] D. Banerjee. «Using strongly typed vs. statically typed code.» (2023), adresse: https://blog.logrocket.com/using-strongly-typed-vs-statically-typed-code/ (sjekket 20.05.2024).

[42] N. Raval. «TypeScript vs JavaScript: Know The Difference.» (2023), adresse: https://radixweb.com/blog/typescript-vs-javascript (sjekket 18.05.2024).

[43] Wikipedia contributors, *Visual Studio Code — Wikipedia, The Free Encyclopedia*, [Online; accessed 21-May-2024], 2024. adresse: https://en.wikipedia.org/w/index.php?title=Visual_Studio_Code&oldid=1223317495.

[44] Wikipedia contributors, *JetBrains — Wikipedia, The Free Encyclopedia*, [Online; accessed 19-May-2024], 2024. adresse: https://en.wikipedia.org/w/index.php?title=JetBrains&oldid=1221276587.

[45] R. H. Inc. «What is Docker?» (2018), adresse: https://www.redhat.com/en/topics/containers/what-is-docker (sjekket 17.05.2024).

[46] Docker. «Use containers to Build, Share and Run your applications.» Offisiell dokumentasjon fra Docker. (), adresse: https://www.docker.com/resources/what-container/ (sjekket 17.05.2024).

[47] Wikipedia contributors, *Monad (functional programming) — Wikipedia, The Free Encyclopedia*, [Online; accessed 20-May-2024], 2024. adresse: https://en.wikipedia.org/w/index.php?title=Monad_(functional_programming)&oldid=1216198666.

[48] GeeksForGeeks. «What is StrictMode in React ?» (2023), adresse: https://www.geeksforgeeks.org/what-is-strictmode-in-react/ (sjekket 20.05.2024).

[49] G. van Rossum, B. Warsaw og A. Coghlan, «PEP 8 – Style Guide for Python Code,» PEP 0008, 2001. adresse: https://peps.python.org/pep-0008/.

[50] J. E. Ravndal og J. H. Halleraker. «FNs bærekraftsmål i Store Norske Leksikon på snl.no.» Hentet 20.05.2024. (mai 2023), adresse: https://snl.no/FNs_b%C3%A6rekraftsm%C3%A5l.

Vedlegg A

Definisjoner

Dette vedlegget skal inneholde definisjoner og forklaringer på ord og uttrykk brukt i denne rapporten. Denne lista skal ikke inkludere tekniske begrep og deres definisjoner, men heller uttrykk brukt spesifikt i rapporten.

- **'Alt-i-nettleser':** Et uttrykk som brukes for den klient-baserte arkitekturen, hvor utregninger kjøres hos klientens egen maskin istedenfor en server.
- **Hvitpunktet:** Uttrykk i fargematchfunksjoner. Spesifikk verdi for de tre stimulus i øyner for å oppnå fargen hvit.
- **Purpurlinja:** En purpur linje som oppstår i kromatisitetsdiagram.
- **'Dockerisering'**: Et uttrykk brukt for å omgjøre et program om til en Docker kontainer.
- **'Klient-tjener':** Både et uttrykk som brukes for arkitekturmodellen med samme navn, men også for den tjener-baserte arkitekturen, hvor utregninger kjøres hos server gjennom forespørsler fra klient.
- **Søkestrenger:** Addisjonelle parametere i URL, også kjent som 'query string' på engelsk.

Vedlegg B

Standardavtale

NTNU

Norges teknisk-naturvitenskapelige universitet

Fastsatt av prorektor for utdanning 10.12.2020

STANDARDAVTALE

om utføring av studentoppgave i samarbeid med ekstern virksomhet

Avtalen er ufravikelig for studentoppgaver (heretter oppgave) ved NTNU som utføres i samarbeid med ekstern virksomhet.

Forklaring av begrep

Opphavsrett
Er den rett som den som skaper et åndsverk har til å fremstille eksemplar av åndsverket og gjøre det tilgjengelig for allmennheten. Et åndsverk kan være et litterært, vitenskapelig eller kunstnerisk verk. En studentoppgave vil være et åndsverk.

Eiendomsrett til resultater
Betyr at den som eier resultatene bestemmer over disse. Utgangspunktet er at studenten eier resultatene fra sitt studentarbeid. Studenten kan også overføre eiendomsretten til den eksterne virksomheten.

Bruksrett til resultater
Den som eier resultatene kan gi andre en rett til å bruke resultatene, f.eks. at studenten gir NTNU og den eksterne virksomheten rett til å bruke resultatene fra studentoppgaven i deres virksomhet.

Prosjektbakgrunn
Det partene i avtalen har med seg inn i prosjektet, dvs. som vedkommende eier eller har rettigheter til fra før og som brukes i det videre arbeidet med studentoppgaven. Dette kan også være materiale som tredjepersoner (som ikke er part i avtalen) har rettigheter til.

Utsatt offentliggjøring
Betyr at oppgaven ikke blir tilgjengelig for allmennheten før etter en viss tid, f.eks. før etter tre år. Da vil det kun være veileder ved NTNU, sensorene og den eksterne virksomheten som har tilgang til studentarbeidet de tre første årene etter at studentarbeidet er innlevert.

1. Avtaleparter

Norges teknisk-naturvitenskapelige universitet (NTNU) Institutt: Institutt for datateknologi og informatikk
Veileder ved NTNU: Ivar Farup e-post og tlf. Ivar.farup@ntnu.no, 61 13 52 27
Ekstern virksomhet: CIE TC1-97 Ekstern virksomhet sin kontaktperson og e-post; Jan Henrik Wold Jan.h.wold@ntnu.no
Student: Lars Edvin Jonsson Hoff Fødselsdato: 18.12.1996
Student: Nikolay Savchuk Fødselsdato: 13.11.2002
Student: Anders Mariendal Brunsberg Fødselsdato: 28.09.1996

Partene har ansvar for å klarere eventuelle immaterielle rettigheter som studenten, NTNU, den eksterne eller tredjeperson (som ikke er part i avtalen) har til prosjektbakgrunn før bruk i forbindelse med utførelse av oppgaven. Eierskap til prosjektbakgrunn skal fremgå av eget vedlegg til avtalen der dette kan ha betydning for utførelse av oppgaven.

2. Utførelse av oppgave

Studenten skal utføre: (sett kryss)

Masteroppgave	
Bacheloroppgave	X
Prosjektoppgave	
Annen oppgave	

Startdato: 8.1.2024
Sluttdato: 21.5.2024

Oppgavens arbeidstittel er: Webapp CIE

Ansvarlig veileder ved NTNU har det overordnede faglige ansvaret for utforming og godkjenning av prosjektbeskrivelse og studentens læring.

3. Ekstern virksomhet sine plikter

Ekstern virksomhet skal stille med en kontaktperson som har nødvendig faglig kompetanse til å gi studenten tilstrekkelig veiledning i samarbeid med veileder ved NTNU. Ekstern kontaktperson fremgår i punkt 1.

Formålet med oppgaven er studentarbeid. Oppgaven utføres som ledd i studiet. Studenten skal ikke motta lønn eller lignende godtgjørelse fra den eksterne for studentarbeidet. Utgifter knyttet til gjennomføring av oppgaven skal dekkes av den eksterne. Aktuelle utgifter kan for eksempel være reiser, materialer for bygging av prototyp, innkjøp av prøver, tester på lab, kjemikalier. Studenten skal klarere dekning av utgifter med ekstern virksomhet på forhånd.

Ekstern virksomhet skal dekke følgende utgifter til utførelse av oppgaven:

Dekning av utgifter til annet enn det som er oppført her avgjøres av den eksterne underveis i arbeidet.

4. Studentens rettigheter

Studenten har opphavsrett til oppgaven[1]. Alle resultater av oppgaven, skapt av studenten alene gjennom arbeidet med oppgaven, eies av studenten med de begrensninger som følger av punkt 5, 6 og 7 nedenfor. Eiendomsretten til resultatene overføres til ekstern virksomhet hvis punkt 5 b er avkrysset eller for tilfelle som i punkt 6 (overføring ved patenterbare oppfinnelser).

[1] Jf. Lov om opphavsrett til åndsverk mv. av 15.06.2018 § 1

I henhold til lov om opphavsrett til åndsverk beholder alltid studenten de ideelle rettigheter til eget åndsverk, dvs. retten til navngivelse og vern mot krenkende bruk.

Studenten har rett til å inngå egen avtale med NTNU om publisering av sin oppgave i NTNUs institusjonelle arkiv på Internett (NTNU Open). Studenten har også rett til å publisere oppgaven eller deler av den i andre sammenhenger dersom det ikke i denne avtalen er avtalt begrensninger i adgangen til å publisere, jf. punkt 8.

5. Den eksterne virksomheten sine rettigheter

Der oppgaven bygger på, eller videreutvikler materiale og/eller metoder (prosjektbakgrunn) som eies av den eksterne, eies prosjektbakgrunnen fortsatt av den eksterne. Hvis studenten

---

skal utnytte resultater som inkluderer den eksterne sin prosjektbakgrunn, forutsetter dette at det er inngått egen avtale om dette mellom studenten og den eksterne virksomheten.

Alternativ a) (sett kryss) Hovedregel

X	Ekstern virksomhet skal ha bruksrett til resultatene av oppgaven

Dette innebærer at ekstern virksomhet skal ha rett til å benytte resultatene av oppgaven i egen virksomhet. Retten er ikke-eksklusiv.

Alternativ b) (sett kryss) Unntak

	Ekstern virksomhet skal ha eiendomsretten til resultatene av oppgaven og studentens bidrag i ekstern virksomhet sitt prosjekt

Begrunnelse for at ekstern virksomhet har behov for å få overført eiendomsrett til resultatene:

6. Godtgjøring ved patenterbare oppfinnelser

Dersom studenten i forbindelse med utførelsen av oppgaven har nådd frem til en patenterbar oppfinnelse, enten alene eller sammen med andre, kan den eksterne kreve retten til oppfinnelsen overført til seg. Dette forutsetter at utnyttelsen av oppfinnelsen faller inn under den eksterne sitt virksomhetsområde. I så fall har studenten krav på rimelig

godtgjøring. Godtgjøringen skal fastsettes i samsvar med arbeidstakeroppfinnelsesloven § 7. Fristbestemmelsene i § 7 gis tilsvarende anvendelse.

7. NTNU sine rettigheter

De innleverte filer av oppgaven med vedlegg, som er nødvendig for sensur og arkivering ved NTNU, tilhører NTNU. NTNU får en vederlagsfri bruksrett til resultatene av oppgaven, inkludert vedlegg til denne, og kan benytte dette til undervisnings- og forskningsformål med de eventuelle begrensninger som fremgår i punkt 8.

8. Utsatt offentliggjøring

Hovedregelen er at studentoppgaver skal være offentlige.

Sett kryss

X	Oppgaven skal være offentlig

I særlige tilfeller kan partene bli enige om at hele eller deler av oppgaven skal være undergitt utsatt offentliggjøring i maksimalt tre år. Hvis oppgaven unntas fra offentliggjøring, vil den kun være tilgjengelig for student, ekstern virksomhet og veileder i denne perioden. Sensurkomiteen vil ha tilgang til oppgaven i forbindelse med sensur. Student, veileder og sensorer har taushetsplikt om innhold som er unntatt offentliggjøring.

Oppgaven skal være underlagt utsatt offentliggjøring i (sett kryss hvis dette er aktuelt):

Sett kryss		Sett dato
	ett år	
	to år	
	tre år	

Behovet for utsatt offentliggjøring er begrunnet ut fra følgende:

Dersom partene, etter at oppgaven er ferdig, blir enig om at det ikke er behov for utsatt offentliggjøring, kan dette endres. I så fall skal dette avtales skriftlig.

Vedlegg til oppgaven kan unntas ut over tre år etter forespørsel fra ekstern virksomhet. NTNU (ved instituttet) og student skal godta dette hvis den eksterne har saklig grunn for å

be om at et eller flere vedlegg unntas. Ekstern virksomhet må sende forespørsel før oppgaven leveres.

De delene av oppgaven som ikke er undergitt utsatt offentliggjøring, kan publiseres i NTNUs institusjonelle arkiv, jf. punkt 4, siste avsnitt. Selv om oppgaven er undergitt utsatt offentliggjøring, skal ekstern virksomhet legge til rette for at studenten kan benytte hele eller deler av oppgaven i forbindelse med jobbsøknader samt videreføring i et master- eller doktorgradsarbeid.

9. Generelt

Denne avtalen skal ha gyldighet foran andre avtaler som er eller blir opprettet mellom to av partene som er nevnt ovenfor. Dersom student og ekstern virksomhet skal inngå avtale om konfidensialitet om det som studenten får kjennskap til i eller gjennom den eksterne virksomheten, kan NTNUs standardmal for konfidensialitetsavtale benyttes.

Den eksterne sin egen konfidensialitetsavtale, eventuell konfidensialitetsavtale den eksterne har inngått i samarbeidprosjekter, kan også brukes forutsatt at den ikke inneholder punkter i motstrid med denne avtalen (om rettigheter, offentliggjøring mm). Dersom det likevel viser seg at det er motstrid, skal NTNUs standardavtale om utføring av studentoppgave gå foran. Eventuell avtale om konfidensialitet skal vedlegges denne avtalen.

Eventuell uenighet som følge av denne avtalen skal søkes løst ved forhandlinger. Hvis dette ikke fører frem, er partene enige om at tvisten avgjøres ved voldgift i henhold til norsk lov. Tvisten avgjøres av sorenskriveren ved Sør-Trøndelag tingrett eller den han/hun oppnevner.

Denne avtale er signert i fire eksemplarer hvor partene skal ha hvert sitt eksemplar. Avtalen er gyldig når den er underskrevet av NTNU v/instituttleder.

Signaturer:

Instituttleder: v Emneansvarlig Dato: 5/2-24
Veileder ved NTNU: Ivar Farup

Dato: 31.01-2024

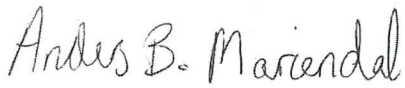
Ekstern virksomhet:
Jan H. Wold

Dato: 01.02-2024

Student: Lars Edvin Jonsson Hoff

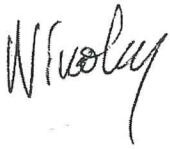
Dato: 31.01.24

Student: Anders Brunsberg Mariendal



Dato: 31.01.24

Student: Nikolay Savchuk
Dato: 31.01.2024



Vedlegg C

Prosjektplan

NTNU
Kunnskap for en bedre verden

Department of Computer Science

IDATG2900 - Bachelor thesis

Web app for visualization of CIE-functions - Project plan

Authors:
Lars Edvin Jonsson Hoff (470525)
Anders Brunsberg Mariendal (472499)
Nikolay Savchuk (562942)

01.02.2024

Table of Contents

1 Goals and restrictions

1.1 Background

Each and every being on Earth perceives the world in a different way. Some insects are able to see UV light usually invisible to us humans, while others are completely blind. However, most of us (especially humans) perceive the world through the visible light spectrum - though not all of us see it in the same way.
How someone sees the world depends on a great amount of factors; so, in an attempt to create representations on how one sees colours, the International Commission on Illumination (CIE) created standards and mathematical functions that do exactly this.

With this, a referential application was made in collaboration within the CIE TC1-97 Farup et al. n.d., in which one can use these functions. Our task is to use this application, and design a web application which houses all of the same functionalities as it - so that scientists who use it can easily adapt and adopt it as an alternative choice of colour functions.

1.2 Project Goals

1.2.1 The Effect

We wish our application has the effect of giving more comfort and usability to the scientists within CIE all around the world. While it is possible for most scientists to use the desktop application version, for some it may seem more attractive to rather use a web application version of it - less to download, available on most (if not all) devices that support the web, and so on.
We wish to make this with them in mind, so that we know for sure that the effect is oriented towards them.

1.2.2 The End-Result

The ultimate end result of our project is to create an web application that has all of the same functionalities as the desktop application this thesis is based on. It should be using an user interface that should be no stranger to users of the original application, while still being adapted for the web browser through responsive design.

1.2.3 The Learning Outcome

We can learn a great deal of matters through this project, but some that we wish to put emphasis on would be:

- Testinging, experimenting and discussing different approaches on how to tackle a project:
  - The tried and true backend approach through subjects such as websockets or APIs.
  - Newer and experimental approaches within popular open source alternatives.
- Communication and teamwork in an actual project, both between team members as well as the team and product owner.
- The usage of scrum as a development model in a more professional environment than before.
- How to combine our strengths in different fields to create a product our product owner would be satisfied with.

1.3 Framework

- The web application should support HTML5, and be accessible through most mainstream web browsers.
- The final version of the application - this implies a chosen prototype that we deemed fit for the project, and thus developed completely - should be completely developed by 16th of April, one month before the thesis's deadline.

2 Scope

2.1 Problem

How a person perceives colour depends on a great set of parameters, and so to make it easier to represent this, the CIE declared a set of standards and mathematical functions regarding colour vision. Most scientists however, do not wish to do the math by hand on these highly sophisticated functions (as well as represent them through plots and tables by extension), so they naturally resort to computers, using applications such as the referential application this project is going to be based on.

While the desktop application is sufficient and gets the job done for most, desktop applications has some inherent flaws such as accessibility and convenience. Through this project we wish to expand on the accessibility and usability of the core application. This will be done by developing a web-based version. The final project will be accessible on most devices. We will achieve this by developing a few different prototypes to fine the optimal framework for our project.

2.2 Task Description

The objective is to create a web application which houses the very same functionalities as the desktop application. This includes and implies:

- The application has to be able to compute various CIE functions, with their own unique and modifiable inputs.
- For each function, the application has to be able to display them as either plots and/or tables.
  - Adapt the user interface from the original application, into a faithful recreation within the web browser.
  - Plots are interactive; they can be moved around, zoomed in/out on, labeled, and gridded, among other features. Additionally, the plot can be saved as an image and further adjusted.
  - Tables have to support a given precision of values, unique to a function - sometimes ranging between 5-7 decimal figures.
  - For each function, the program has to be able to display an informational screen that includes basic information regarding the function. This includes parameters, symbols in the function, wavelengths, normalizations, etc.

In addition to this we have to make some underlying decisions with regards to the framework of the web application. This is primarily:

- Use various frameworks to create working prototypes with the given functionalities above.
- Test the prototypes, and discuss positives/negatives with each one to then choose one that fits ideally for the project.

2.3 Limitations

- While our application has to have the same functionalities as the one it is based on, it shouldn't have any extra large visible functionalities that does not exist in the original application. If we find some aspects that can clearly be improved upon without compromising the integrity the the program potential changes can be discussed with the product owner.
- Featuees and functionalities that are "behind the scenes" and invisible to the user, such as caching, can be developed freely.

3 Project organising

3.1 Routines and group rules

Work Methodology
For our development process we chose to utilise Scrum. This is because of its flexibility and efficiency in managing complex projects. Scrum's iterative approach allows for regular reassessment and adaptation, which is crucial in a dynamic project environment. It facilitates team collaboration ensuring that everyone is aligned and contributing effectively.

Attendance
All team meetings are mandatory, unless otherwise agreed. Project supervisor and product owner may request to move time and place of the meeting. All group members may call for extraordinary meetings if needed.

Absence
If one is prevented from attending at the agreed times and places, a message should be sent to the group members in advance. Planned absences should be discussed with the team members collectively. UIO n.d.

Consequences
In case of breach of the group's rules, a verbal warning will be given first. In case of repeated breaches, the offender will receive a written warning. If a group member receives two written warnings, a meeting with the group advisor will be arranged. Read conflict management.

Meetings
Meeting invitations should be sent out at least 2 days before the meeting. The minutes of the meeting should be ready no later than the next day after a meeting has been held. At meetings, we all agree that time should be used efficiently. We want everyone to come prepared to each meeting.

Decisions
All decisions regarding structure and technology used should be made collectively where all group members must be present. All group members have one vote, and the majority decides.

Conflict Management
All irregularities should initially be discussed at group meetings. All group members can bring up things they believe deviate from the agreements that have been made. If agreement or a solution is not reached, it should be brought to the group supervisor who is responsible for whether the concerned problem child becomes a resource for the team again or the group is split in two. If the group is split, both parties will gain access to all the work previously done.

3.2 Responsibilities and roles

- Product owner
  - This person represents the wants and needs of the CIE organisation.
  - He is also responsible for communicating the requirements of the web application.
- Supervisor
  - The supervisor is responsible for providing academic guidance and helping the students understand and apply different development concepts and theories.
  - He may also assist with administrative aspects. This includes procedures for submission and assessment criteria.
  - The most crucial responsibility of the supervisor is critical assessment and providing constructive feedback to the students.
- Group leader
  - Coordinates the overall project, ensuring that tasks are distributed, deadlines are met, and everyone is on track.
  - Facilitate effective communication among group members, ensuring that everyone is informed about project updates, meetings, and tasks.
  - Maintain open communication with the thesis advisor, seeking guidance when needed and sharing progress updates.
  - Keep track of the project's progress, monitor individual and group achievements, and provide updates to the group and the advisor.
  - Mediate any conflicts or issues that may arise among group members, promoting a harmonious working environment.
  -
- Scrum master
  - Make sure that all Scrum events occur while adhering to the designated time frames.
  - Exert their influence to eliminate any obstacles hindering the Scrum Team's progress.
  - Prioritise the creation of valuable increments that align with the Definition of Done.
  - Ensure that the work produced by group members meets the required standards and aligns with the project's objectives. scrum.org 2024
- Developers
  - Responsible for creating clean efficient code and design.
  - Recognise, rank, and execute tasks throughout the software development life cycle.
  - Conduct validation and verification tests.
  - Document different phases of development.
  - Execute their tasks in accordance with fixed routines and code practices.
- Editor
  - Oversee the documentation of the project, including meeting minutes, progress reports, and any necessary documentation for the thesis itself.
  - Responsible for planing, coordination, and review of material before delivery.
  - Has the final say on wording and phrasing.
  - Collaborate with group members in preparing for the presentation of the bachelor thesis, ensuring that everyone is well-prepared and confident.

4 Planning, follow-up and reporting

The project description defines the wanted functionality of the end product very well, as the end product essentially is a translation from desktop software to web application. A key aspect to the project however is the research and deciding on different technologies that will yield the best implementation of the web application. As such there is quite a bit of uncertainty surrounding the specific details of the application at the start of the project. Therefore flexibility and adaptability are key aspects.

4.1 Main division of the project

After carefully reviewing different frameworks for project management in correspondence with the project description, it was decided that Scrum would be a beneficial approach for the project.

The reasoning for this decision is that scrum is an agile method with key principles such as flexibility, teamwork, iterative progress and continuous improvement. Atlassian n.d. This ensures that the group is able to adapt based on feedback from product owner, supervisor or internal discussion. Even if the project is well defined, it is beneficial to be able to adapt and make changes to the software if more suitable solutions or technology is discovered. The frequency of meetings also ensures that everyone involved stays updated on the progress of the project, and facilitates better collaboration. Working by the scrum principles also gives a good overview of the project's overall progression which can help with identifying concerns and issues early and thereby make the necessary adjustments.

Scrum will be implemented in the following way:

- **Roles**
  - **Scrum master**: Facilitates the process and ensures that Scrum practices are followed.
  - **Product owner**: Represents the stakeholder, defines requirements.
  - **Development team**: Cross-functional team members who develops the product.
- **Artifacts**
  - **Product backlog**: A prioritized list of features describing functionality of end product.
  - **Sprint backlog**: A subset of the product backlog that is to be completed during the sprint.
  - **Increment**: The sum of the product backlog items completed during previous sprints.
- **Events**
  - **Sprint**: The time period allocated for a given sprint backlog to be worked on.
  - **Sprint planning**: Meeting between team and product owner at the start of each sprint where the sprint backlog is decided.
  - **Daily Stand-Up**: Short meeting between team members for status updates.
  - **Sprint Review**: Meeting between team and product owner at the end of the sprint to inspect the outcome of the sprint. Feedback from product owner, updating the product backlog.
  - **Sprint retrospective**: Meeting between team members. Takes place between the end of one sprint and the start of the next sprint. Evaluate the previous sprint and make plans for improvements for the coming sprint.

The group decided on a sprint duration of 2 weeks, as this seems like a good balance between frequency for feedback and discussion with the product owner and allocated time to implement features from the sprint backlog.

Sprint planning will take place at the start of each sprint. In this meeting the team and product owner will discuss and decide upon the sprint backlog that is to be worked on for the coming sprint.

Sprint review will take place at the end of each sprint. In this meeting the team and product owner will discuss the outcome of the sprint. The increment is inspected and the product backlog is updated. This gives a clear overview of the progress made up to this point and the remaining work for future sprints.

Sprint Retrospective will take place after the sprint review. The team members will discuss the sprint itself and evaluate what went well, and what changes to implement to improve future sprints. This ensures continuous improvement throughout the course of the project.

Following the scrum framework there will be held short stand-up meetings between the team members where each member briefly informs what they have been working on since the last stand up. Traditionally these stand-up meetings are held daily when following the scrum framework. Due to the scope of the project/thesis we decided that having stand-up meetings every other day will be more expedient.

In addition to following the Scrum framework as described, the team plans to utilize a Kanban board or Scrum board to keep track of the sprint backlog. This will be created digitally so that every team member can update the board and get a clear overview of how the sprint is going at all times. In this board every item from the sprint backlog will be added and given status labels such as "to-do", "in progress", "code review" and "done", and the visualisation will make it clear what has been done and is left to do.

4.2 Plan for status meetings and decision points during the period

As the sprint duration is two weeks, the sprint planning and sprint review will be held every other week. These two events will be done over one meeting between the team, supervisor and product owner where the meeting starts with a sprint review before moving over to sprint planning. After the Sprint planning and sprint review meeting, the team will do the sprint retrospective.

Every other Wednesday, where there are no sprint events planned, there will be a status meeting between the team and the supervisor.

Every Monday, Wednesday and Friday the team will have stand-up meetings.

Schedule for meetings:

- Stand-up meetings three times every week: Monday, Wednesday and Friday from 11:00 – 11:15
- Sprint planning and sprint review every two weeks on a Wednesday 11:15 - 12:30
- Sprint retrospective every two weeks on a Wednesday 12:30 - 13:00
- Status meeting with supervisor on Wednesdays with no Sprint events from 11:15-12:00

5 Organisation of quality assurance

5.1 Tools

Here is a list of the tools (libraries, software, services) we plan to use within the project. Keep in mind that this is the project planning phase - and thus, the future project may not be limited to these as we start work on the project.

NAME	USAGE
Docker	Docker gets used in the backend to containerize our software to the server.
PyScript	PyScript gets used as one of the possible approaches to the task, by introducing Python to the front-end. Anaconda n.d.
Django	Django gets used as one of the possible approaches to the project, by being a popular backend framework within Python.
Flask	Flask is also one of the possible approaches within backend, also being a popular Python framework.
PyCharm	PyCharm gets used as an IDE for Python development.
Visual Studio Code	Visual Studio Code gets used as a code editor for front-end development, as well as an alternative for Python development.
GitLab	GitLab gets used as the main repository for our project, as well as potentially the main area for us to plan sprints and sprint boards.
Diagrams.net	Diagrams.net gets used to create UML diagrams (as well as general diagrams) in case those are needed.
Figma	Figma gets used to design the application with wireframes and prototypes in case it is needed.
Overleaf	Overleaf gets used to create documentation for reports through LaTeX.

5.2 Quality Assurance of Documentation

To ensure proper quality of the source code, we wish to stick with practices we've learnt over the cource of our degrees. Naturally, we will comment our code with explanations for bits and pieces that is harder to read, while still keeping the shorter comments for bits that are easier to read, but still useful. All comments will be in English due to the project's background of being potentially used for CIE, an international organistation, later. This will make fruther potential development easier.

To ensure correct usage of the application for users, we also wish to include a README.md within the project repository, containing proper documentation on how to run, update and use the project.

5.3 Quality Assurance of Source Code

Here, we also wish to follow standard practices we learnt over the course of our degrees. Each developer gets their own branch in the repository that they're solely responsible for, alongside one main branch that hosts the newest changes. Any developer may work on an issue within the

current sprint backlog, in which they themselves are naturally responsible for updating that issue in addition to developing, testing, and merging their work to main (thus, they themselves are responsible for any merge issues that may arise).
Due to the nature of our project - in which we have to create prototypes of several frameworks - each prototype of different framework may get their own "main" branch that primarily contains its development. For each sprint, current prototypes are assigned a version to manage version controls easier.

We also wish to perform code review of each other's work before we classify something as "completely done". We wish to do this for most changes to the source code, and that any changes to the repository should be generally accepted by the entire group.

In addition, it is important to note that testing of each approach would be different, due to their differences. For example, testing of the PyScript approach requires insight into the performance of the application on a given web browser - while testing of a backend approach requires network testing to see if the client got the right calculations from the server. We intend to carefully document these as well.

5.4 Fixed development routines

We wish to follow the following routines in regards to the development of the project:

- Every developer is to meet up on a given meeting (or, if they cannot, document this to the rest of the group in advance).
- Every developer is responsible for logging their own hours used for the project through the usage of a shared Excel document.
- Every developer is responsible for updating issues they have taken upon themselves to work with in regards to a current sprint, as well as discussing their progress in the scrum meetings.

5.5 Risk analysis

Our risk analysis will consist of a series of potential risks linked to the project. We've decided that every risk will have an ultimate risk level, decided by the risk's potential total danger if realised (in "loss"), as well as the general chance of the risk happening (in "probability").
For the sake of simplicity, all of these have been standardized to "high", "some", or "minimal"; with the final risk level being a culmination of the loss and probability.

NAME	DESCRIPTION	LOSS	PROBABILITY	RISK LEVEL
Asset Loss	There may be some documentation and/or source code that gets lost over the course of development.	Some	Minimum	Some
Unfinished Aspect	There may be a part of the project that is left unfinished until the deadline of our project.	High	Minimum	Some
Obsolete Technology	There may arise a situation where our project uses technology that is either obsolete, or not recieving updates any longer.	Some	Minimum	Minimum
Unexpected Development Problem	There may be an abstract problem within development that causes serious delays and/or unexpected changes in the project over the course of development.	Some	Some	Some

5.6 Risk management plan

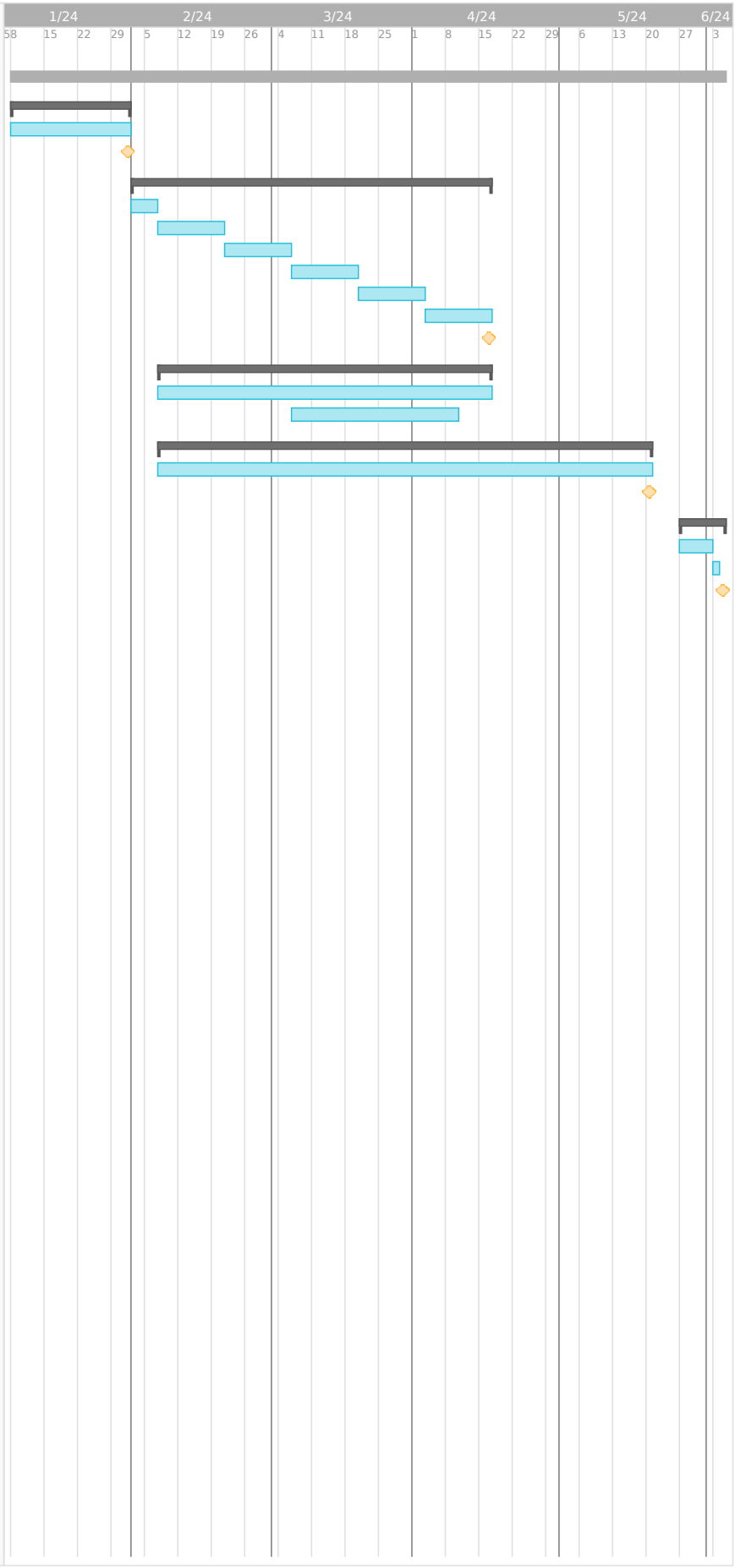
In the risk management above, we discussed risks in general - and while some are in need of critical management plans in case they are realised, some are not so deserving. Thus, we have the table below which references to risks above, and then details a management plan on how to proceed.

NAME	MEASURE?	DESCRIPTION
Asset Loss	Yes	Considering the fact that our project should be fairly easy to recover with the usage of GitLab and commits, there is little need for a measure for it - but to be on the safe side, we wish to store backups based on their version.
Unfiunished Aspect	None	There's no measure you can take to an unfinished aspect except for finish it, or make it at least somewhat viable for common usage.
Obsolete Technology	None	We'll accept this risk, given the nature of the project. In the case where the approach regarding Python in frontend is eventually made obsolete, we at least have other prototypes of other approaches that we can continue development from.
Unexpected Development Problem	None	Problems arise out of the blue, and there is no real measure one can take.

6 Plan for execution

6.1 Gantt chart

teamgantt
Created with Free Edition



Web App	**0h**	**0%**
Project plan	**0h**	**0%**
Project plan	0	0%
Project plan delivery	0	0%
Project	**0h**	**0%**
Research	0	0%
Sprint 1	0	0%
Sprint 2	0	0%
Sprint 3	0	0%
Sprint 4	0	0%
Sprint 5	0	0%
Deadline coding	0	0%
Testing	**0h**	**0%**
Code testing	0	0%
User testing	0	0%
Dissertation writing	**0h**	**0%**
Thesis writing	0	0%
Bachelor thesis delivery	0	0%
Presentation	**0h**	**0%**
Planing	0	0%
Rehearsal	0	0%
Presentation	0	0%

6.2 Milestones, activities and decisions

- **Research Phase:** This initial stage involves gathering necessary information and insights that will inform the rest of the project. It's crucial for understanding the problem space and identifying viable solutions.
- **Sprint Development:** The project is divided into several sprints, each focusing on different aspects of the web app. These sprints involve iterative development, with each sprint building upon the learnings from the previous one.
- **Deadline for Coding:** This milestone marks the completion of the main coding phase. It's a crucial point where the core functionalities of the web app should be implemented and operational.
- **Testing Phase:** This stage is divided into code testing and user testing. Code testing ensures the technical robustness of the app, while user testing focuses on the app's usability and user experience.
- **Dissertation Writing:** Alongside the development, writing the thesis is a continuous process. This phase involves documenting the project's progress, methodologies, findings, and conclusions.
- **Thesis Submission:** This is the final milestone where the completed thesis is submitted. It signifies the culmination of research, development, and writing efforts.
- **Presentation Preparation:** Planning and rehearsing for the final presentation of the project. This is where we showcase your work, highlighting the key aspects and learnings from the project.

Early on in the development process we will decide on which framework we will utilise for version control. We believe we have two realistic options. GitHub or GitLab. This is because of their robust version control capabilities and their wide adaptation in the developer scene. GitHub and GitLab facilitates collaborative coding, allowing multiple contributors to work simultaneously without conflict. It also provides a system for tracking changes, reviewing code, and managing updates efficiently. The choice we make will come down to which of the two frameworks is more compatible with Scrum, our work methodology.

By the conclusion of the initial Scrum sprint, we expect to have developed basic yet operational prototypes that demonstrates appropriate framework's capabilities. These prototypes serve as a proof of concept, illustrating how the framework can be utilised in our specific project context. This approach fosters a sense of achievement and momentum early in the project, setting a positive tone for subsequent development phases.

Bibliografi

Anaconda (n.d.). *PyScript*. URL: https://github.com/pyscript/pyscript.

Atlassian (n.d.). *What is scrum?* URL: https://www.atlassian.com/agile/scrum.

Farup, Ivar et al. (n.d.). *CIE Functions*. URL: https://github.com/ifarup/ciefunctions.

scrum.org (2024). *What is a Scrum Master?* URL: https://www.scrum.org/resources/what-is-a-scrum-master (visited on 16th Jan. 2024).

UIO (n.d.). *Eksempel på gruppeavtale*. URL: https://www.uio.no/studier/emner/matnat/ifi/IN1060/v18/timeplan/gruppe-3/eksempel-pa-arbeidskontrakt.pdf.

Vedlegg D

Oppgavebeskrivelse

Oppgave 8 BPROG, BIDATA 3 stk

Oppgavetittel: Webapp for visualisering av CIE-funksjoner

Bedrift: Fargelabben, IDI, NTNU

Kontaktperson: Ivar Farup og Jan H. Wold

E-post: Ivar.farup@ntnu.no
Telefon: 61135227
Lokasjon: Gjøvik

Oppgaven passer til 3 studenter fra BIDATA og/eller BPROG

Beskrivelse av oppgaven:

Bakgrunn

En persons fargesyn varierer med flere parametre, f.eks. alder og feltstørrelse. Standardiseringsorganet CIE har laget standarder og anbefalinger for hvordan funksjoner som representerer fargesynet, kalt fargematchfunksjoner, skal beregnes og representeres. Gjennom bl.a. prosjektet «Individual Colour Vision-based Image Optimisation» støttet av Norges forskningsråd, er det ved NTNU og USN utviklet en referanseimplementasjon programvare for for å kunne beregne disse funksjonene for en såkalt standardobservatør. Programvaren er utviklet i Python med en backend og et enkelt brukergrensesnitt i PyQt.

Oppgave

Vi ønsker gjennom dette bachelorprosjektet å få utviklet en Webapp med i hovedsak samme funksjonalitet som desktopapplikasjonen. Siden den eksisterende beregningskoden er implementert i Python, vil det være hensiktsmessig at backend-delen av løsningen forblir i Python. En viktig del av oppgaven blir å finne ut hva som er et egnet rammeverk for resten av applikasjonen gitt denne føringen. Løsninger som Dash, Flask, og Django kan synes naturlige å sjekke ut, men det er også åpent for andre løsninger.

Vedlegg E

Produktkø

Frontend - functionality for field size for CIE XYZ standard colour-matching functions and CIE xy standard chromaticity diagram
#32 · created 1 week ago by Anders Brunsberg Mariendal
Backend Approach
0 updated 1 week ago

Backend - Deployment
#31 · created 1 month ago by Nikolay Savchuk
Backend Approach In-Progress Produktkø
0 updated 1 month ago

Backend - Error Handling
#30 · created 1 month ago by Nikolay Savchuk
Backend Approach In-Progress Produktkø
0 updated 1 week ago

Backend - Asynkronisitet
#29 · created 1 month ago by Nikolay Savchuk
Backend Approach Godkjenning Sprintkø
Closed 0 closed 1 month ago

Backend - Revamp av API
#28 · created 1 month ago by Nikolay Savchuk
Backend Approach In-Progress
0 updated 1 week ago

Backend - Base precision
#27 · created 1 month ago by Nikolay Savchuk
Backend Approach Godkjenning Produktkø
Closed 0 closed 1 month ago

Backend - Wavelength floating point
#26 · created 1 month ago by Nikolay Savchuk Sprint #5
Backend Approach Godkjenning Produktkø
Closed 0 closed 1 month ago

Backend - Søke om server
#25 · created 2 months ago by Lars Edvin Jonsson Hoff
Backend Approach
0

Backend - Html sidemeny endpoint
#24 · created 2 months ago by Lars Edvin Jonsson Hoff
Backend Approach
Closed 0 closed 1 week ago

Frontend - Lage flere diagrammer
#23 · created 2 months ago by Lars Edvin Jonsson Hoff
Backend Approach Godkjenning
0 updated 1 week ago

Backend - Compute.py modularisering
#22 · created 2 months ago by Nikolay Savchuk Sprint #4
Backend Approach Godkjenning
0 updated 1 week ago

Backend - Endpoints
#21 · created 2 months ago by Nikolay Savchuk Sprint #4
Backend Approach Godkjenning
Closed 0 closed 1 month ago

Backend - Testing
#20 · created 2 months ago by Nikolay Savchuk Sprint #4
Backend Approach Godkjenning Produktkø Sprintkø
Closed 0 closed 1 month ago

Frontend - Redux
#19 · created 2 months ago by Nikolay Savchuk Sprint #3
Backend Approach In-Progress Produktkø
0 updated 2 months ago

Frontend - Checkbox (labels, log10, renormaliserte verdier)
#18 · created 2 months ago by Nikolay Savchuk Sprint #4
Backend Approach Godkjenning
0 updated 1 week ago

Frontend - Modulisering
#17 · created 2 months ago by Nikolay Savchuk Sprint #3
Backend Approach In-Progress Produktkø
0 updated 2 months ago

Frontend - Nedtrekksmeny
#16 · created 2 months ago by Nikolay Savchuk Sprint #3
Backend Approach Godkjenning Produktkø
0 updated 2 months ago

Frontend - Tabell
#15 · created 2 months ago by Nikolay Savchuk Sprint #3
Backend Approach Godkjenning Produktkø
0 updated 1 month ago

Frontend - Sidemeny
#14 · created 2 months ago by Nikolay Savchuk Sprint #3
Backend Approach
Closed 0 closed 1 week ago

Testing av prototyper
#13 · created 3 months ago by Nikolay Savchuk Prototypeavgjørelse
Dokumentasjon Produktkø
Closed 0 closed 2 months ago

Rapportskriving
#12 · created 3 months ago by Nikolay Savchuk Rapportskriving
Dokumentasjon In-Progress
0 updated 1 week ago

Backend Prototype, v3
#11 · created 3 months ago by Nikolay Savchuk Sprint #3
Backend Approach Produktkø
0 updated 2 months ago

Frontend Prototype, v3
#10 · created 3 months ago by Nikolay Savchuk Sprint #3
Frontend Approach Produktkø
Closed 0 closed 2 months ago

Backend Prototype, v2
#9 · created 3 months ago by Nikolay Savchuk Prototypeavgjørelse
Backend Approach Produktkø
Closed 0 closed 2 months ago

Frontend Prototype, v2
#8 · created 3 months ago by Nikolay Savchuk Prototypeavgjørelse
Frontend Approach Produktkø
Closed 0 closed 2 months ago

Backend Prototype, v1
#7 · created 3 months ago by Nikolay Savchuk Sprint #1
Backend Approach In-Progress Produktkø
Closed 0 closed 2 months ago

Frontend Prototype, v1
#6 · created 3 months ago by Nikolay Savchuk Sprint #1
Frontend Approach Produktkø Sprintkø
Closed 0 closed 2 months ago

Arkitektur - Komponentdiagrammer
#5 · created 3 months ago by Nikolay Savchuk Sprint #1
Dokumentasjon Produktkø
Closed 0 closed 2 months ago

Kravspesifikasjoner - Sekvensdiagrammer
#4 · created 3 months ago by Nikolay Savchuk Sprint #1
Dokumentasjon Produktkø
Closed 0 closed 2 months ago

Kravspesifikasjoner - Use-cases og use-case diagram
#3 · created 3 months ago by Nikolay Savchuk Sprint #1
Dokumentasjon Produktkø
Closed 0 closed 2 months ago

Arkitektur - Wireframe
#1 · created 3 months ago by Nikolay Savchuk Sprint #1
Dokumentasjon Produktkø Sprintkø
Closed 0 closed 3 months ago

Vedlegg F

Klient-tjener prototype stresstest

Performance test report - May 19, 2024 (#1)

Open in Postman

Postman collection: performance tst

Report exported on: May 19, 2024, 20:56:40 (GMT+2)

Test setup

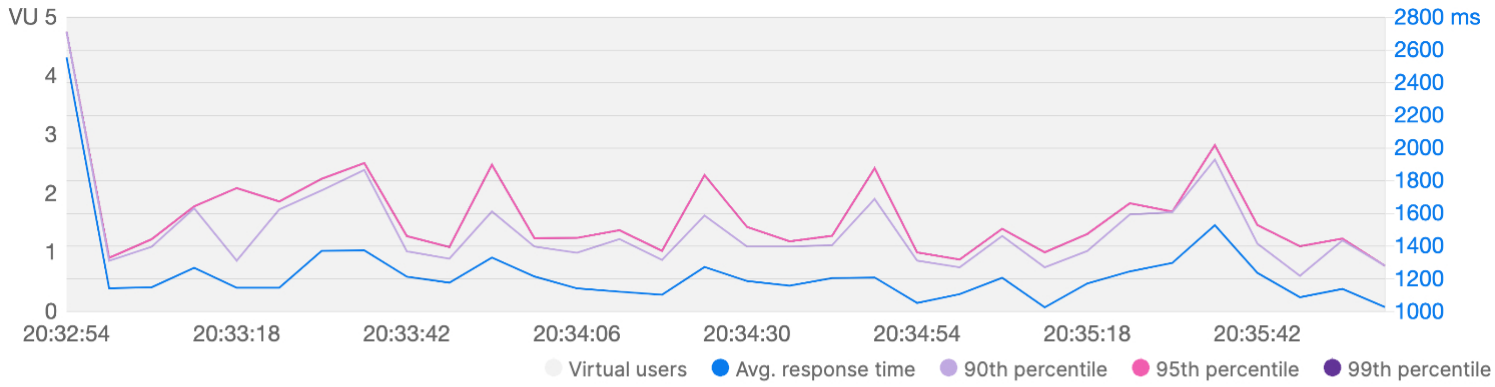
Virtual users	Start time	Load profile
5 VU	May 19, 20:32:56 (GMT+2)	Fixed
Duration	**End time**	**Environment**
3 minutes	May 19, 20:36:02 (GMT+2)	New Environment

1. Summary

Total requests sent	Throughput	Average response time	Error rate
421	2.26 requests/second	1,211 ms	0.00 %

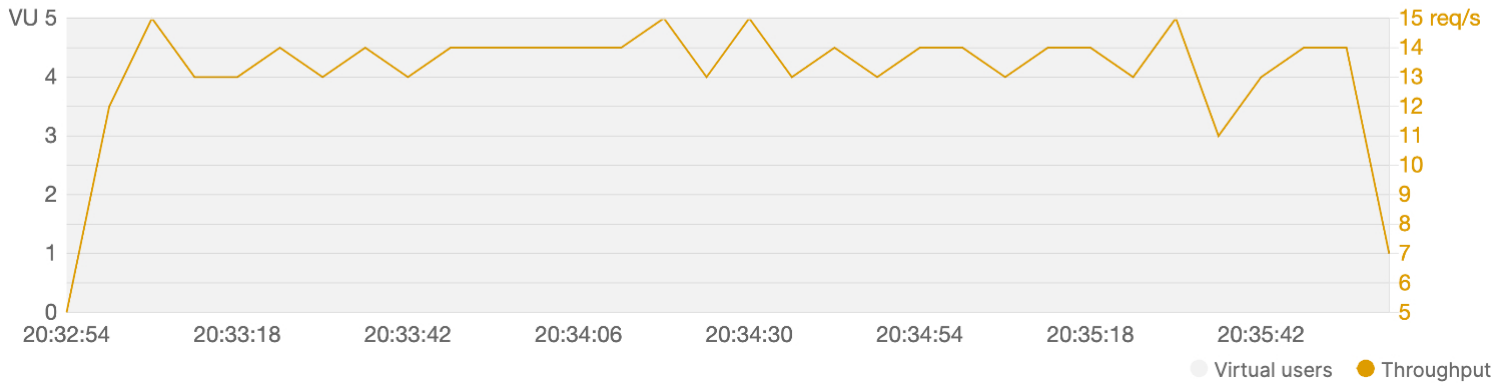
1.1 Response time

Response time trends during the test duration.



1.2 Throughput

Rate of requests sent per second during the test duration.



1.3 Requests with slowest response times

Top 5 slowest requests based on their average response times.

Request	Resp. time (Avg ms)	90th (ms)	95th (ms)	99th (ms)	Min (ms)	Max (ms)
POST New Request http://127.0.0.1:5000/compute_all_specific_data	1,211	1,518	1,663	2,360	841	2,713

2. Metrics for each request

The requests are shown in the order they were sent by virtual users.

Request	Total requests	Requests/s	Min (ms)	Avg (ms)	90th (ms)	Max (ms)	Error %
POST New Request http://127.0.0.1:5000/compute_all_specific_data	421	2.26	841	1,211	1,518	2,713	0

3. Errors

This run has no errors

All requests were sent successfully and returned a 2xx response code.

Testing API performance on Postman

Postman enables you to simulate user traffic and observe how your API behaves under load. It also helps you identify any issues or bottlenecks that affect performance.

Learn more about testing API performance.

Vedlegg G

Kodestykker av parametersystem

Kodeliste G.1: Backend - Utsnitt av parameterfunksjon, del 1.

```
# parameterfunksjon tatt fra prosjektet,
        # kommentarer i dette vedlegget er annerledes enn de i programmet
        # for hensikten av enklere formidling til leser
def createAndCheckParameters(disabled, calculation, request):
    # funksjon som forsøker å lese til gitt type,
        # lik til Rust sin <Option> type (enten Some(...) or None)
    def string_to_type_else(string, type, other):
        try:
            return type(string)
        except ValueError:
            return other
        except TypeError:
            return None
    # forskjellig behandling av parametere for standardfunksjoner
    if disabled:
        # henter inn *obligatoriske* parametere enten som None eller som float med
                # bruk av det over
        parameters = {
            "field_size": string_to_type_else(request.args.get('field_size'),
                float, None),
            "age": string_to_type_else(request.args.get('age'), float, None)
        }
        # sjekker om de finnes
        for (name, value) in parameters.items():
            if value is None:
                raise SanicException(
                    ("Value error.",
                    "Invalid input for '{}' either due to absence or invalid
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣type".format(name),
                     "Control that '{}' is present, and is of the 'float'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣type.".format(name)), status_code=422)
                # runder av i tilfellet det ble gitt et desimaltall
        parameters['age'] = round(parameters['age'])
```

Kodeliste G.2: Backend - Utsnitt av parameterfunksjon, del 2.

```
                # deretter, forsøker å finne de valgfrie parametere som gjelder
                # blir enten til:
                        # float dersom oppgitt verdi *og* det er et tall
```

```
                        # -1 dersom verdi er oppgitt, men *ikke* et tall
                        # None dersom ingenting er oppgitt
        optionals = {
            "min": string_to_type_else(request.args.get('min'), float, -1),
            "max": string_to_type_else(request.args.get('max'), float, -1),
            "step_size": string_to_type_else(request.args.get('step_size'), float, -1),
        }
                # utfører takling av valgfri parameter over;
        for (name, value) in optionals.items():
            # om feil type, send feilsvar tilbake til brukeren
            if value is -1:
                raise SanicException((
                "Type␣error.",
                "Invalid␣input␣for␣'{}'␣due␣to␣invalid␣type".format(name),
                "Control␣that␣the␣value␣is␣of␣a␣'float'␣type,␣or␣remove␣it␣to
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣use␣default␣settings."), status_code=422)
            # om ingenting er oppgitt, gi det en standardverdi og oppdater dict
            if value is None:
                given = 0
                if name == "min": given = 390.0
                if name == "max": given = 830.0
                if name == "step_size": given = 1.0
                parameters.update({name: given})
            # ellers, oppdater parameter dict til å inneholde verdien som den er
            else:
                parameters.update({name: float(value)})

        # starten av feilhåndtering
        if parameters['field_size'] > 10 or parameters['field_size'] < 1:
            raise SanicException(("Value␣error.", "Invalid␣value␣for␣'field_size'.",
                                  "Control␣that␣the␣value␣is␣between␣1.0␣and␣10.0."),
                                  status_code=422)
                # evt. flere andre feilhåndteringer ...
```

Kodeliste G.3: Backend - Utsnitt av LMS utregningsfunksjon

```
def compute_LMS_Modular(parameters):
    def inner_LMS(variation):
        # compute.py line 1565
         _all  = my_round(np.arange(390., 830. + .01, .1), 1)
        # compute.py line 1575
        LMS_base_all = LMS_energy(parameters['field_size'], parameters['age'])[0]
        if parameters['base']:
            # compute.py line 1572
            LMS_base_all = LMS_energy(parameters['field_size'],
            parameters['age'], base=True)[0]
        # compute.py line 1585
        ( _all , L, M, S) = LMS_base_all.T
        # compute.py lines 1586-1588
        L_spline = scipy.interpolate.InterpolatedUnivariateSpline( _all , L)
        M_spline = scipy.interpolate.InterpolatedUnivariateSpline( _all , M)
        S_spline = scipy.interpolate.InterpolatedUnivariateSpline( _all , S)
        # compute.py lines 763-768
        LMS_sf = 6
        logLMS_dp = 5
        if parameters['base']:
            LMS_sf = 9
            logLMS_dp = 8
        # compute.py lines 770-772
        (La, Ma, Sa) = np.array([sign_figs(L_spline(variation), LMS_sf),
                                 sign_figs(M_spline(variation), LMS_sf),
                                 sign_figs(S_spline(variation), LMS_sf)])
        # compute.py line 773
        LMS = chop(np.array([variation, La, Ma, Sa]).T)
        if parameters['log']:
            LMS[:, 1:][LMS[:, 1:] == 0] = -np.inf
            LMS[:, 1:][LMS[:, 1:] > 0] = my_round(
                np.log10(LMS[:, 1:][LMS[:, 1:] > 0]), logLMS_dp)
            LMS[:, 0] = my_round(LMS[:, 0], 1)
            return chop(LMS)
        else:
            LMS[:, 0] = my_round(LMS[:, 0], 1)
            # compute.py line 779
            return chop(LMS)
        return LMS
    dict = {
        "result": inner_LMS(np.arange(parameters['min'], parameters['max'] + .01,
        parameters['step_size'])),
        "plot": inner_LMS(my_round(np.arange(parameters['min'], parameters['max'] + .01,
        .1), 1))
    }
    return dict
```

Vedlegg H

Kodestykker for diagramressurs

Følgende kode dekker `macleod_graph()` fra `graph.py` modulen i prosjektet.

H.1 Python

Kodeliste H.1: Backend - Utsnitt av Pythonkode for MacLeod-Boynton diagrammet.

```
def macleod_graph(parameters):
    temp = parameters.copy()
    plots_json = cieapi.new_calculation_JSON(compute_MacLeod_modular, temp)
    temp['info'] = True
    info_json = cieapi.new_calculation_JSON(compute_MacLeod_modular, temp)
    # retrieves relevant datapoints from a range
    points = retrievePoints(macleod_graph, parameters)
    title = (
        "MacLeod-Boynton␣ls␣chromaticity␣diagram<br>␣Field␣size:
␣␣␣␣␣␣␣␣␣␣␣␣{}  ,␣Age:␣{}␣yr,␣Domain:␣{}␣nm␣-␣{}␣nm,␣Step:␣{}␣nm".
        format(parameters['field_size'], parameters['age'],
                parameters['min'], parameters['max'],
                parameters['step_size']))
    xaxis = "l<sub>MB</sub>.␣{},␣{}".format(
        parameters['field_size'], parameters['age'])
    yaxis = "S<sub>MB</sub>.␣{},␣{}".format(
        parameters['field_size'], parameters['age'])
    # creates the raw html
    raw = head(macleod_graph) + """ ...
```

Kodeliste H.2: Backend - Utsnitt av Pythonkode for head() fra graph.py

```
def head(function):
    return """
    <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src='https://cdn.plot.ly/plotly-2.32.0.min.js'></script>
    <style>
        #plot {
            border: 1px solid black;
        }
```

```
        #plot * {
            margin-left: auto;
            margin-right: auto;
        }
        #inputrow {
            display: flex;
            flex-direction: row;
            justify-content: space-around;
        }
    </style>
</head>
<body>
    <div id='plot'></div>
    """ + checkboxes(function) + """
    <script>
    """
```

H.2 JavaScript

N.B: Bruken av `plots_json`, `info_json`, `title`, `xaxis`, `yaxis` og `points` er fra Pythonkoden, og er elementer som settes inn for å dynamisk generere diagrammet basert på parametere.

Kodeliste H.3: Backend - Utsnitt av JavaScriptkode for MacLeod-Boynton diagrammet.

```
    const results = '""" + plots_json + """';
    const info = '""" + info_json + """';
    const plot = JSON.parse(results)['plot'];
    const information = JSON.parse(info);

    // from StackOverflow, see module description.
    output = plot[0].map((_, colIndex) => plot.map(row => row[colIndex]));

    const config = {responsive: true}
    var layout = {
        showlegend: false,
        autosize: true,
        title: '""" + title + """',
        margin: { l: 50, r: 10, b: 50, t: 50, pad: 4 },
        height: 800,
        width: 800,
            yaxis: {
                scaleanchor: "x",
                zeroline: false,
                title: '""" + yaxis + """'
            },
            xaxis: {
                zeroline: false,
                nticks: 10,
                title: '""" + xaxis + """',
            }
        }

    // macleod curve
    var curve = {
        x: output[1],
```

```
        y: output[3],
        xaxis: "x-aspect",
        yaxis: "y-aspect",
        mode: 'lines',
        line: {
                color: 'rgb(0, 0, 0)'
        }
    }

    const y_points = []
    const x_points = []
    const pointes = """ + str(points) + """;
    for (let i = 0; i < pointes.length; i++) {
        let index = output[0].indexOf(pointes[i]);
        x_points.push(output[1][index]);
        y_points.push(output[3][index]);
    }

    var points = {
        x: x_points,
        y: y_points,
        mode: 'markers',
        type: 'scatter',
        textposition: 'top right',
        marker: {
            color: 'rgb(255, 255, 255)',
            size: 10,
            line: {
                color: 'rgb(0, 0, 0)',
                width: 2
            }
        }
    }
    var illuminantE = {
        x: [information['white'][0]],
        y: [information['white'][2]],
        mode: 'markers',
        type: 'scatter',
        textposition: 'top right',
        marker: {
            color: 'rgb(255, 255, 255)',
            size: 10,
            line: {
                color: 'rgb(0, 0, 0)',
                width: 2
            }
        }
    }
    var purpleline = {
        x: [information['tg_purple'][0][1], information['tg_purple'][1][1]],
        y: [information['tg_purple'][0][2], information['tg_purple'][1][2]],
        mode: 'lines',
        line: {
            color: 'rgb(150, 32, 240)',
            width: 3
        }
    }

    // adds event listeners for grid and label button as done earlier
```

```
    grid.addEventListener('change', (event) => {
        var grid = document.getElementById('grid').checked;
        layout['yaxis']['showgrid'] = grid;
        layout['xaxis']['showgrid'] = grid;

        Plotly.react('plot', [curve, points, illuminantE, purpleline], layout, config);
    })

    label.addEventListener('change', (event) => {
    var label = document.getElementById('label').checked;
        if (label) {
            points['mode'] = 'markers+text';
            points['text'] = """ + str(list(map(str, points))) + """;
            illuminantE['mode'] = 'markers+text';
            illuminantE['text'] = ['E'];
        } else {
            points['mode'] = 'markers';
            points['text'] = [];
            illuminantE['mode'] = 'markers';
            illuminantE['text'] = [];
        }
        Plotly.react('plot', [curve, points, illuminantE, purpleline], layout,
            config);
    })

    Plotly.react('plot', [curve, points, illuminantE, purpleline], layout, config);
```

Vedlegg I

Stresstest

Performance test report - May 17, 2024 (#5)

Open in Postman

Postman collection: deployed-tests

Report exported on: May 17, 2024, 21:42:50 (GMT+2)

Test setup

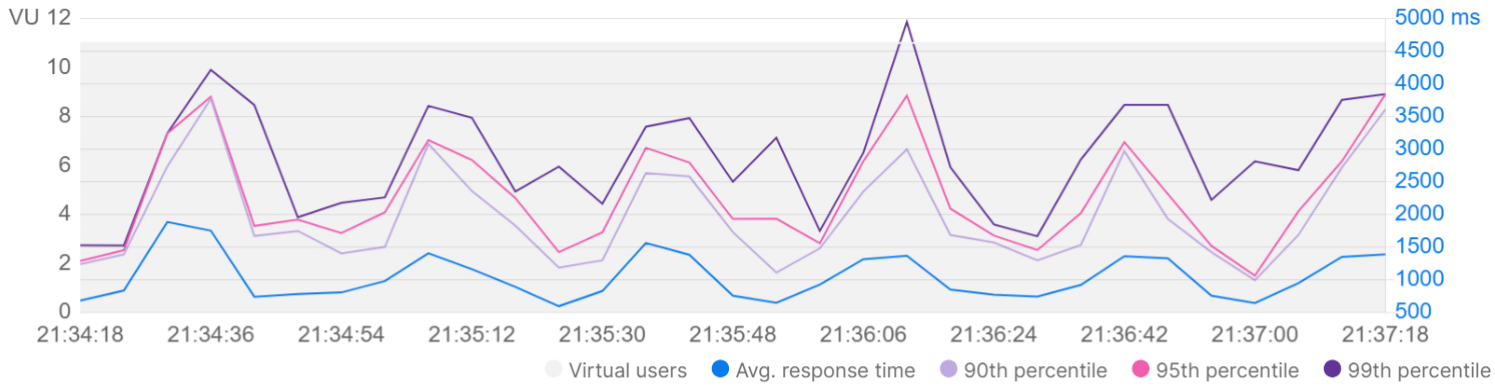
Virtual users	Start time	Load profile
11 VU	May 17, 21:34:11 (GMT+2)	Fixed
Duration	**End time**	**Environment**
3 minutes	May 17, 21:37:19 (GMT+2)	-

1. Summary

Total requests sent	Throughput	Average response time	Error rate
1,836	9.80 requests/second	958 ms	0.00 %

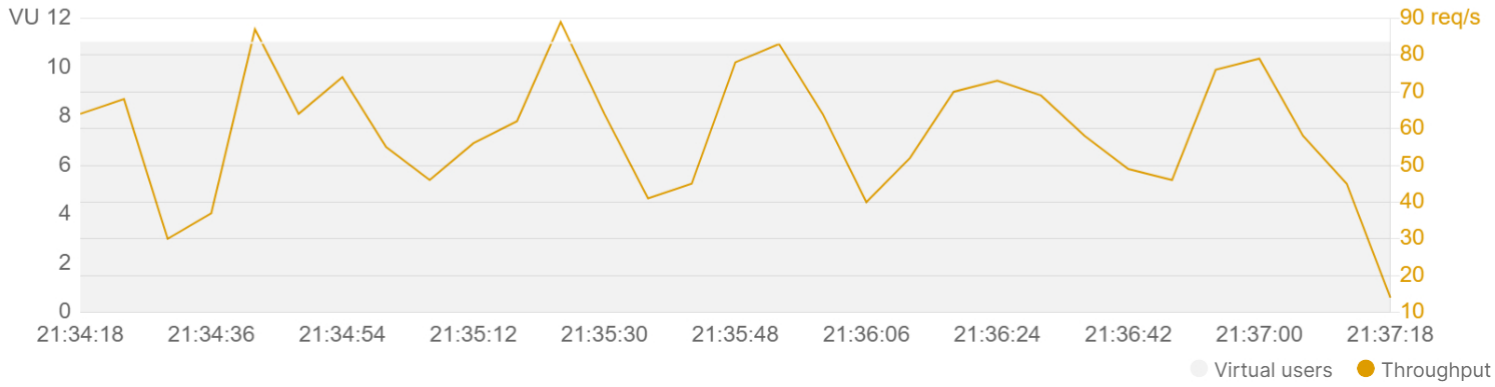
1.1 Response time

Response time trends during the test duration.



1.2 Throughput

Rate of requests sent per second during the test duration.



1.3 Requests with slowest response times

Top 5 slowest requests based on their average response times.

Request	Resp. time (Avg ms)	90th (ms)	95th (ms)	99th (ms)	Min (ms)	Max (ms)
GET xy-plot http://10.212.136.66:8000/api/v2/xy/plot?field_size=2.0&age=23&max=700&min=400	2,370	3,663	3,804	4,946	1,029	4,946
GET xyz-plot http://10.212.136.66:8000/api/v2/xyz/plot?field_size=2.0&age=23&log10&max=700&min=400	1,969	2,998	3,113	3,607	979	3,607
GET xyp-plot http://10.212.136.66:8000/api/v2/xy-p/plot?field_size=2.0&age=23&max=700&min=400	1,882	2,503	2,836	3,771	1,118	3,771
GET xyz-std-plot http://10.212.136.66:8000/api/v2/xyz-std/plot?field_size=2.0	1,538	2,833	3,552	3,841	454	3,841
GET xy-std-plot http://10.212.136.66:8000/api/v2/xy-std/plot?field_size=10	1,439	2,302	2,500	3,343	607	3,343

2. Metrics for each request

The requests are shown in the order they were sent by virtual users.

Request	Total requests	Requests/s	Min (ms)	Avg (ms)	90th (ms)	Max (ms)	Error %
GET lms http://10.212.136.66:8000/api/v2/lms/calculation?field_size=1.0&age=64&min=390.0&max=830.0&step_size=0.5&optional=log,base	70	0.37	242	624	1,179	1,458	0
GET maxwellian http://10.212.136.66:8000/api/v2/lms-mw/calculation?field_size=2.0&age=23&log10&max=700&min=400	70	0.37	112	433	888	1,036	0

GET macleod http://10.212.136.66:8000/api/v2/lms-mb/calculation?field_size=2.0&age=23&log10&max=700&min=400	70	0.37	193	446	760	1,141	0
GET xyz http://10.212.136.66:8000/api/v2/xyz/calculation?field_size=2.0&age=23&log10&max=700&min=400	70	0.37	321	735	1,058	1,530	0
GET xy http://10.212.136.66:8000/api/v2/xy/calculation?field_size=2.0&age=23&max=700&min=400	70	0.37	302	762	1,285	1,633	0
GET xy-std http://10.212.136.66:8000/api/v2/xy-std/calculation?field_size=10	70	0.37	167	590	1,102	1,291	0
GET xyp-calculation http://10.212.136.66:8000/api/v2/xy-p/calculation?field_size=2.0&age=23&max=700&min=400	70	0.37	355	956	1,404	1,877	0
GET xyzp-calculation http://10.212.136.66:8000/api/v2/xyz-p/calculation?field_size=2.0&age=23&max=700&min=400	70	0.37	526	987	1,461	2,050	0
GET xyz-std http://10.212.136.66:8000/api/v2/xyz-std/calculation?field_size=2.0	70	0.37	268	705	1,161	2,319	0
GET lms-plot http://10.212.136.66:8000/api/v2/lms/plot?field_size=2.0&age=23&log10&max=700&min=400	70	0.37	322	857	1,450	2,815	0
GET maxwell-plot http://10.212.136.66:8000/api/v2/lms-mw/plot?field_size=2.0&age=23&log10&max=700&min=400	70	0.37	291	854	1,496	2,720	0
GET macleod-plot http://10.212.136.66:8000/api/v2/lms-mb/plot?field_size=2.0&age=23&log10&max=700&min=400	69	0.37	422	970	1,504	2,970	0
GET xyz-std-plot http://10.212.136.66:8000/api/v2/xyz-std/plot?field_size=2.0	69	0.37	454	1,538	2,833	3,841	0

GET xyz-plot http://10.212.136.66:8000/api/v2/xyz/plot?field_size=2.0&age=23&log10&max=700&min=400	67	0.36	979	1,969	2,998	3,607	0
GET xy-plot http://10.212.136.66:8000/api/v2/xy/plot?field_size=2.0&age=23&max=700&min=400	66	0.35	1,029	2,370	3,663	4,946	0
GET xyp-plot http://10.212.136.66:8000/api/v2/xy-p/plot?field_size=2.0&age=23&max=700&min=400	65	0.35	1,118	1,882	2,503	3,771	0
GET xyzp-plot http://10.212.136.66:8000/api/v2/xyz-p/plot?field_size=2.0&age=23&max=700&min=400	64	0.34	558	1,408	2,795	3,820	0
GET xy-std-plot http://10.212.136.66:8000/api/v2/xy-std/plot?field_size=10	63	0.34	607	1,439	2,302	3,343	0
GET lms-sidemenu http://10.212.136.66:8000/api/v2/lms/sidemenu?field_size=2.0&age=23&log10&max=700&min=400	63	0.34	24	487	1,382	2,313	0
GET xy-std-sidemenu http://10.212.136.66:8000/api/v2/xy-std/sidemenu?field_size=10	63	0.34	73	366	872	1,313	0
GET macleod-sidemenu http://10.212.136.66:8000/api/v2/lms-mb/sidemenu?field_size=2.0&age=23&log10&max=700&min=400	61	0.33	162	680	1,435	2,620	0
GET maxwell-sidemenu http://10.212.136.66:8000/api/v2/lms-mw/sidemenu?field_size=2.0&age=23&log10&max=700&min=400	60	0.32	146	585	1,195	1,841	0
GET xyz-sidemenu http://10.212.136.66:8000/api/v2/xyz/sidemenu?field_size=2.0&age=23&max=700&min=400	60	0.32	307	797	1,523	1,786	0
GET xy-sidemenu http://10.212.136.66:8000/api/v2/xy/sidemenu?field_size=2.0&age=23&max=700&min=400	60	0.32	314	988	1,578	2,448	0

GET xyp-sidemenu http://10.212.136.66:8000/api/v2/xy-p/sidemenu?field_size=2.0&age=23&max=700&min=400	59	0.31	583	1,298	1,933	2,252	0
GET xyzp-sidemenu http://10.212.136.66:8000/api/v2/xyz-p/sidemenu?field_size=2.0&age=23&max=700&min=400	59	0.31	560	1,196	1,952	2,251	0
GET xyz-std-sidemenu http://10.212.136.66:8000/api/v2/xyz-std/sidemenu?field_size=2.0	59	0.31	22	396	977	1,684	0
GET api-main http://10.212.136.66:8000/api/v2	59	0.31	46	511	1,675	3,174	0

3. Errors

This run has no errors

All requests were sent successfully and returned a 2xx response code.

Testing API performance on Postman

Postman enables you to simulate user traffic and observe how your API behaves under load. It also helps you identify any issues or bottlenecks that affect performance.

Learn more about testing API performance.

Vedlegg J

Belastningstest

Performance test report - May 17, 2024 (#7)

Open in Postman

Postman collection: deployed-tests

Report exported on: May 17, 2024, 22:04:26 (GMT+2)

Test setup

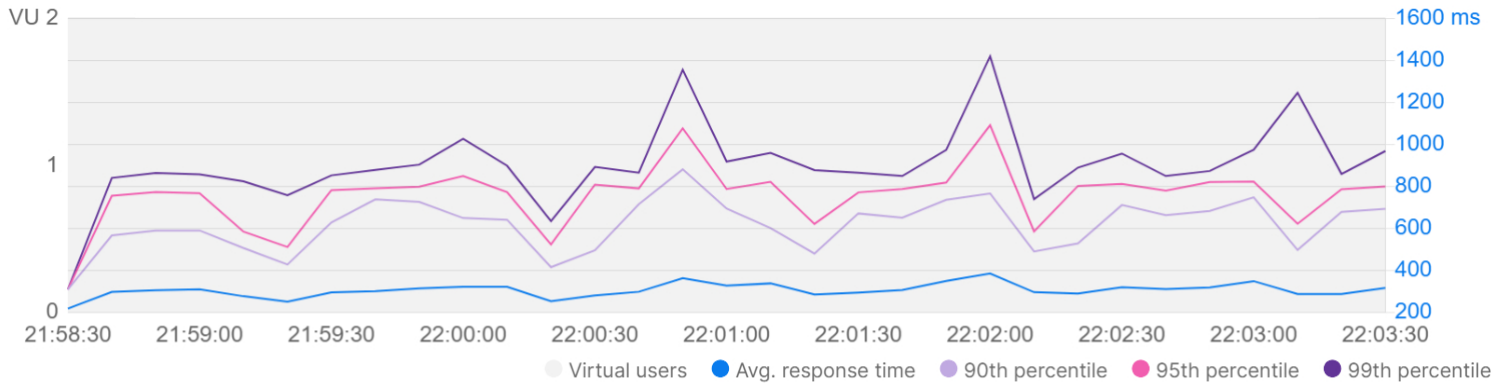
Virtual users	Start time	Load profile
2 VU	May 17, 21:58:31 (GMT+2)	Fixed
Duration	**End time**	**Environment**
5 minutes	May 17, 22:03:39 (GMT+2)	-

1. Summary

Total requests sent	Throughput	Average response time	Error rate
1,403	4.56 requests/second	307 ms	0.00 %

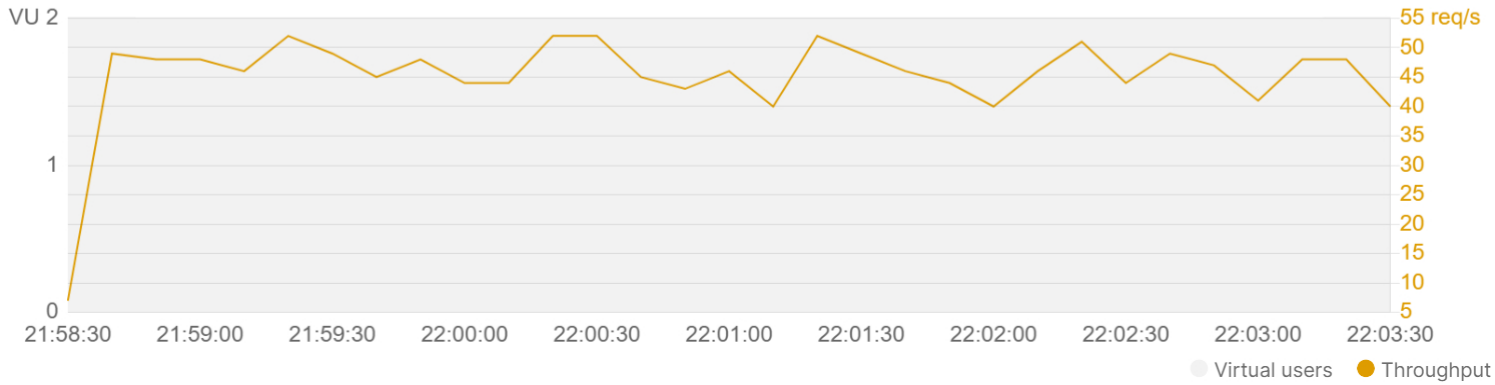
1.1 Response time

Response time trends during the test duration.



1.2 Throughput

Rate of requests sent per second during the test duration.



1.3 Requests with slowest response times

Top 5 slowest requests based on their average response times.

Request	Resp. time (Avg ms)	90th (ms)	95th (ms)	99th (ms)	Min (ms)	Max (ms)
GET xy-plot http://10.212.136.66:8000/api/v2/xy/plot?field_size=2.0&age=23&max=700&min=400	884	974	1,092	1,356	767	1,356
GET xyp-plot http://10.212.136.66:8000/api/v2/xy-p/plot?field_size=2.0&age=23&max=700&min=400	808	878	983	1,340	711	1,340
GET xyz-plot http://10.212.136.66:8000/api/v2/xyz/plot?field_size=2.0&age=23&log10&max=700&min=400	657	737	749	946	539	946
GET xyp-sidemenu http://10.212.136.66:8000/api/v2/xy-p/sidemenu?field_size=2.0&age=23&max=700&min=400	509	533	561	629	475	629
GET xy-std-plot http://10.212.136.66:8000/api/v2/xy-std/plot?field_size=10	429	463	661	1,420	336	1,420

2. Metrics for each request

The requests are shown in the order they were sent by virtual users.

Request	Total requests	Requests/s	Min (ms)	Avg (ms)	90th (ms)	Max (ms)	Error %
GET lms http://10.212.136.66:8000/api/v2/lms/calculation?field_size=1.0&age=64&min=390.0&max=830.0&step_size=0.5&optional=log,base	51	0.17	239	278	313	377	0
GET maxwellian http://10.212.136.66:8000/api/v2/lms-mw/calculation?field_size=2.0&age=23&log10&max=700&min=400	51	0.17	140	167	198	268	0

GET macleod http://10.212.136.66:8000/api/v2/lms-mb/calculation?field_size=2.0&age=23&log10&max=700&min=400	51	0.17	117	173	198	228	0
GET xyz http://10.212.136.66:8000/api/v2/xyz/calculation?field_size=2.0&age=23&log10&max=700&min=400	50	0.16	272	314	360	373	0
GET xy http://10.212.136.66:8000/api/v2/xy/calculation?field_size=2.0&age=23&max=700&min=400	50	0.16	249	287	311	393	0
GET xy-std http://10.212.136.66:8000/api/v2/xy-std/calculation?field_size=10	50	0.16	123	187	221	312	0
GET xyp-calculation http://10.212.136.66:8000/api/v2/xy-p/calculation?field_size=2.0&age=23&max=700&min=400	50	0.16	287	316	336	360	0
GET xyzp-calculation http://10.212.136.66:8000/api/v2/xyz-p/calculation?field_size=2.0&age=23&max=700&min=400	50	0.16	300	352	373	1,008	0
GET xyz-std http://10.212.136.66:8000/api/v2/xyz-std/calculation?field_size=2.0	50	0.16	186	239	284	500	0
GET lms-plot http://10.212.136.66:8000/api/v2/lms/plot?field_size=2.0&age=23&log10&max=700&min=400	50	0.16	224	275	303	435	0
GET maxwell-plot http://10.212.136.66:8000/api/v2/lms-mw/plot?field_size=2.0&age=23&log10&max=700&min=400	50	0.16	223	277	292	1,246	0
GET macleod-plot http://10.212.136.66:8000/api/v2/lms-mb/plot?field_size=2.0&age=23&log10&max=700&min=400	50	0.16	236	278	304	527	0
GET xyz-std-plot http://10.212.136.66:8000/api/v2/xyz-std/plot?field_size=2.0	50	0.16	318	406	491	622	0

GET xyz-plot http://10.212.136.66:8000/api/v2/xyz/plot?field_size=2.0&age=23&log10&max=700&min=400	50	0.16	539	657	737	946	0
GET xy-plot http://10.212.136.66:8000/api/v2/xy/plot?field_size=2.0&age=23&max=700&min=400	50	0.16	767	884	974	1,356	0
GET xyp-plot http://10.212.136.66:8000/api/v2/xy-p/plot?field_size=2.0&age=23&max=700&min=400	50	0.16	711	808	878	1,340	0
GET xyzp-plot http://10.212.136.66:8000/api/v2/xyz-p/plot?field_size=2.0&age=23&max=700&min=400	50	0.16	303	341	355	644	0
GET xy-std-plot http://10.212.136.66:8000/api/v2/xy-std/plot?field_size=10	50	0.16	336	429	463	1,420	0
GET lms-sidemenu http://10.212.136.66:8000/api/v2/lms/sidemenu?field_size=2.0&age=23&log10&max=700&min=400	50	0.16	22	32	42	156	0
GET xy-std-sidemenu http://10.212.136.66:8000/api/v2/xy-std/sidemenu?field_size=10	50	0.16	100	113	129	237	0
GET macleod-sidemenu http://10.212.136.66:8000/api/v2/lms-mb/sidemenu?field_size=2.0&age=23&log10&max=700&min=400	50	0.16	122	156	174	199	0
GET maxwell-sidemenu http://10.212.136.66:8000/api/v2/lms-mw/sidemenu?field_size=2.0&age=23&log10&max=700&min=400	50	0.16	139	153	177	220	0
GET xyz-sidemenu http://10.212.136.66:8000/api/v2/xyz/sidemenu?field_size=2.0&age=23&max=700&min=400	50	0.16	189	202	211	252	0
GET xy-sidemenu http://10.212.136.66:8000/api/v2/xy/sidemenu?field_size=2.0&age=23&max=700&min=400	50	0.16	226	264	278	294	0

GET xyp-sidemenu http://10.212.136.66:8000/api/v2/xy-p/sidemenu?field_size=2.0&age=23&max=700&min=400	50	0.16	475	509	533	629	0
GET xyzp-sidemenu http://10.212.136.66:8000/api/v2/xyz-p/sidemenu?field_size=2.0&age=23&max=700&min=400	50	0.16	381	422	438	612	0
GET xyz-std-sidemenu http://10.212.136.66:8000/api/v2/xyz-std/sidemenu?field_size=2.0	50	0.16	22	29	33	97	0
GET api-main http://10.212.136.66:8000/api/v2	50	0.16	32	59	71	166	0

3. Errors

This run has no errors

All requests were sent successfully and returned a 2xx response code.

Testing API performance on Postman

Postman enables you to simulate user traffic and observe how your API behaves under load. It also helps you identify any issues or bottlenecks that affect performance.

Learn more about testing API performance.

Abstract

This bachelor's thesis focuses on the development of a web application for visualizing CIE functions. CIE functions are standardized mathematical functions that describe how an average person perceives colors based on various parameters such as age. The application is being developed as an extension for an existing application "CIE Functions,"created in collaboration between NTNU and the University of South-Eastern Norway.

The main goal of the project is to develop a web application that offers the same functionalities as the original application. The application should support calculations of various color matching functions for standard observers and represent the results in an easily understandable way. The project includes testing different frameworks to find the most suitable solution, as well as developing prototypes and discussion of the choice of technology.

The development has been carried out using Python for backend calculations and API design, and modern web technologies such as React and TypeScript for the frontend architecture. The project has also had a strong focus on user-centered design and a responsive interface to ensure accessibility and user-friendliness.

Through this work, we have gained a deeper understanding of both old and new forms of web technologies, as well as the importance of good development practices such as documentation and testing. The web application we have developed will facilitate research in color vision and colorimetry, making it more accessible.

Sammendrag

Denne bacheloroppgaven fokuserer på utviklingen av en webapplikasjon for visualisering av CIE-funksjoner. CIE-funksjoner er standardiserte matematiske funksjoner som beskriver hvordan et gjennomsnittlig menneske oppfatter farger basert på forskjellige parametere som for eksempel alder. Applikasjonen lages som en utvidelse for den eksisterende programvaren 'CIE Functions', utviklet i samarbeid mellom NTNU og Universitetet i Sørøst-Norge.

Hovedmålet med prosjektet er å utvikle en webapplikasjon som tilbyr de samme funksjonalitetene som den opprinnelige applikasjonen. Applikasjonen skal støtte beregninger av forskjellige fargematchfunksjoner for standardobservatører og representere resultatene på en lett forståelig måte. Prosjektet inkluderer testing av forskjellige rammeverk for å finne den mest egnede løsningen, samt utvikling av prototyper og diskusjon rundt valg av teknologi.

Utviklingen har blitt gjennomført ved bruk av Python for backend-beregninger og API-design, og moderne webteknologier som React og TypeScript for frontend-arkitektur. Prosjektet har også hatt et sterkt fokus på brukersentrert design og et responsivt grensesnitt for å sikre tilgjengelighet og brukervennlighet.

Gjennom dette arbeidet har vi fått en dypere forståelse av både gamle og nye webteknologier, samt viktigheten av god utviklingspraksis som dokumentasjon og testing. Webapplikasjonen vi har utviklet vil lette forskningen innen fargesyn og kolorimetri og gjøre den mer tilgjengelig.

Forord

Bacheloppgaven er utarbedid av to dataingeniører studenter og en programmerings student ved NTNU i Gjøvik. Denne bacheloroppgaven har blitt gjennomført i samarbeid med Fargelabben ved Institutt for Datateknologi og Informatikk (IDI) ved NTNU, med veiledning av Ivar Farup og Jan Henrik Wold. Prosjektet har hatt som mål å utvikle en webapplikasjon for visualisering av fargematchfunksjoner. Prosjektet tar utgangspunkt i et allerede eksisterende Python-basert programvare.

Vi ønsker å takke våre veiledere, Ivar Farup og Jan H. Wold, for deres uvurderlige støtte og veiledning gjennom hele prosjektet. Deres innsikt og råd om programmering og kolorimetri har vært avgjørende for prosjektet, og gjort det mulig for oss å navigere de betydelige utfordringene vi har møtt.

Gjøvik, Mai 2024
Nikolay Savchuk, Lars Edvin Jonsson Hoff og Anders Brunsberg Mariendal.

Innhold

Figurer

Tabeller

Kodelister

Akronymer

API Application Programming Interface. 16–18, 36, 41, 47, 54, 62, 65, 67, 68, 74, 92–94, 98, 101

ASGI Asynchronous Server Gateway Interface. 19, 39, 57, 58, 77

CIE Commission internationale de l'eclairage. 2, 10

CSS Cascading Style Sheets. 13, 14, 16, 66, 84, 85

CSSOM CSS Object Model. 14

DOM Document Object Model. 14, 36, 37, 46, 47

FFI Foreign Function Interface. 47

HATEOAS Hypermedia As The Engine Of Application State. 17

HCI Human Computer Interaction. 20

HMR Hot Module Replacement. 44

HTML HyperText Markup Language. 12–14, 16, 59, 61, 76, 85

HTTP HyperText Transfer Protocol. 17, 18, 53, 87

IDE Integrated Development Environment. 39

IDI Institutt for datateknologi og informatikk. 1, 5

JS JavaScript. 13, 14, 16, 18, 32, 36, 46–48, 71, 74, 85

REST Representation State Transfer. 17, 29, 53, 54, 59, 75, 76

SOAP Simple Object Access Protocol. 17, 18

SPA Single Page Application. 16

URL Uniform Resource Locator. xiv, 17, 52–54, 59, 67, 68, 73, 74, 76–79, 94

USN Universitetet i Sørøst-Norge. 1

Wasm WebAssembly. 18, 32, 35, 36, 46, 47, 49

WSGI Web Server Gateway Interface. 19, 35, 55–57

XML Extensible Markup Language. 18

Kapittel 1

Introduksjon

1.1 Bakgrunn

Fargelabben ved IDI (Institutt for datateknologi og informatikk)[1] hos NTNU Gjøvik driver med en rekke forskningsprosjekter innenfor fargelære og -styring, med spesiell engasjement innenfor flere felt som datasyn, medisin og bevaring av bilder. Blant annet har de også forskning innenfor fargesyn og hvordan ulike mennesker tar opp farger forskjellig - hvor det er da satt spesiell fokus på å bruke standardiserte matematiske funksjoner (også kalt 'fargematchfunksjoner') for å uttrykke noen sitt fargesyn, gitt relevante parametere som deres alder og feltstørrelse av eksperimenter.

Til denne hensikt ble det produsert en refereanseprogramvare kalt 'CIE Functions' [1] (se figur 1.1) i et bredt samarbeid mellom NTNU og USN (Universitetet i Sørøst-Norge), med bakgrunn i den tekniske komiteen 'CIE TC1-97'[2]. Applikasjonen klarer å beregne en rekke fargefunksjoner for et gjennomsnittlig individ med gitte parametere (også kalt en 'standardobservatør'), og representere resultatene i ulike former som gjør det enkelt å referere til. Applikasjonen er utviklet gjennom Python med åpen kildekode, og et brukergrensesnitt innenfor PyQt[3] biblioteket.

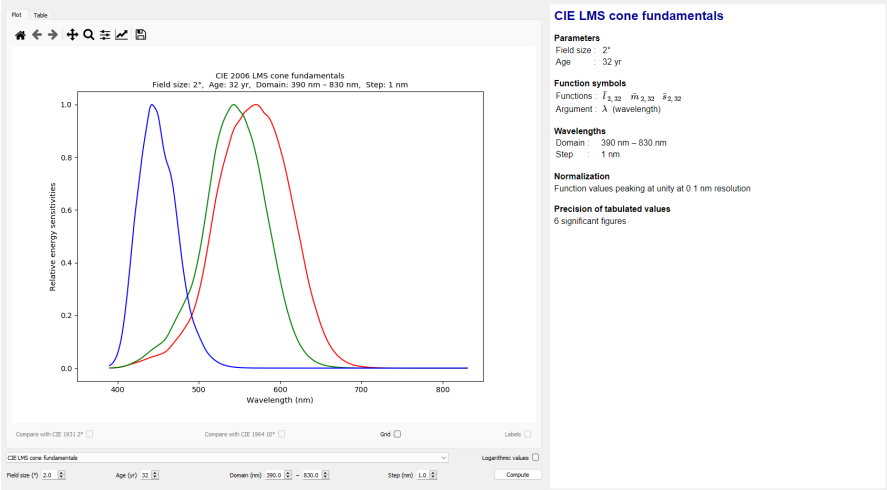
1.1.1 Oppgavedefinisjon

Vår oppgave er å ta for oss denne programvaren, og utvikle en webapplikasjon med de samme funksjonalitene som eksisterer i basisprogrammet. Dette inkluderer visuelle funksjonaliteter, hvor den skal inneholde ett brukergrensesnitt som virker likt som det i programmet. I og med at koden for utregning av fargematchfunksjonene allerede eksisterer i form av Python kode, så er det et viktig punkt å lage løsningen med basis i Python.

[1] `https://www.ntnu.edu/colourlab/view/about`, hentet 19.05.2024

[2] https://cie.co.at/technicalcommittees/age-and-field-size-parameterised-calculation-cone-fundamental-based-spectral, hentet 19.05.2024

[3] http://www.riverbankcomputing.com/software/pyqt/, hentet 19.05.2024



Figur 1.1: Skjermbilde av basisapplikasjonen.

Det ønskes også at det testes forskjellige rammeverk for programmet for å finne det som er best egnet til applikasjonen. Dette innebærer tidlig utvikling av prototyper for ulike rammeverk, og deretter testing med diskusjon for å velge den som passer best. Løsningen vi anser som best utvikles videre på for å oppnå alle funksjonaliteter påkrevd i prosjektet.

1.1.2 Avgrensning

Prosjektets rammeverktesting skal fokusere på rammeverket selv, og ikke inkludere testing eller diskusjon av hvordan maskinvare og annet programvare brukt ville ha påvirket applikasjonens ytelse. Alle tester gjøres derfor på samme maskinvare for å forsikre seg konsise forskningsresultater. Maskinvare er også utenfor rammet av oppgaven, og er avgrenset for oss.

Prosjektet skal ikke inkludere nye funksjoner som ikke allerede finnes i basisapplikasjonen. Hvis det oppstår behov for ekstra funksjonalitet, må dette avtales direkte mellom oppdragsgiver og gruppen. I slike tilfeller skal den nye funksjonaliteten også implementeres i basisapplikasjonen, slik at den oppdateres til å inkludere den samme funksjonaliteten.

1.2 Målgrupper & Mål

1.2.1 Målgrupper

Målgruppen for dette prosjektet blir fargesynsforskere rundt i hele verdenen. I og med at basisapplikasjonen ble laget med basis i en teknisk komité for den internasjonale kommisjon for belysning (CIE)[4], så kan det antas at dette prosjektets

[4]https://cie.co.at/

applikasjon vil gjelde den samme målgruppa - og derfor være laget spesifikt med dem i tankene. Denne gruppa vil også inkludere Fargelabben ved NTNU, for de inngår i gruppen 'fargesynsforskere'.

Rapporten, derimot, er spesifikt skrevet for Fargelabben og vår oppdragsgiver. Gjennom denne rapporten ønsker vi å formidle alt arbeidet som har gått inn i applikasjonen på en måte som både forteller om og forklarer prosjektet til lekfolk. Vi erkjenner at programmering og fargematematikk ikke nødvendigvis tilhører samme domene, så rapporten er skrevet med dette i tankene.

1.2.2 Mål

Vi ønsker å oppnå tre spesifikke mål med denne oppgaven:

Resultatsmål

Resultatet av dette prosjektet blir en ferdigutviklet webapplikasjon som kan hostes på en egen webserver. Webapplikasjonen skal inneholde alle av de samme funksjonalitetene som eksisterer i basisapplikasjonen, med et likt brukergrensesnitt men med videre adapsjon for responsivt design.

Effektsmål

Effekten prosjektet utgjør vil gjøre det enklere for fargesynsforskere å utføre beregninger relatert til fargesyn, og forhåpentligvis skape mer interesse innenfor forskningsemnet med å tilby enkel tilgang til beregningsfunksjonene. Den vil også gjøre det mer komfortabelt og brukbart for forskere som ikke kan bruke basisapplikasjonen med å tilby et alternativ.

Læringsmål

Vi har erklært følgende læringsmål som punkter av spesiell fokus med tanke på utdypelse og å lære mer om dem:

- Utvikling, testing og vurdering av både tradisjonelle nettrammeverk - men også nyere teknologier.
- Utvikling og diskusjon rundt relevante klientrammeverk med tanke på brukersentrert design.
- Profesjonell etikette angående programvareutvikling gjennom flere relevante metoder (praktisk bruk av arbeidsmetodikker, dokumentasjon, osv.).
- Mer erfaring innen gruppearbeid, hvordan vi skal kunne samarbeide for å skape et større prosjekt.
- Hvordan vi skal formidle våre funn i en rapport, og presentasjon, gjennom drøftinger og utdypelser.

1.3 Gruppe

Vår gruppe består av tre studenter; Nikolay Savchuk (BPROG[5]), Lars Edvin Jonsson Hoff (BIDATA[6]), Anders Brunsberg Mariendal (BIDATA) - og er derfor en blandet gruppe med folk fra forskjellige grader. Allikevel, har vi en rekke unike kompetanser som gjør oppgaven godt egnet for oss.

For våre dataingeniørstudenter, så tilbyr de stor kompetanse og innsikt i optimalisering av dataflyt og integritet av systemer, samtidig som de bringer frem en dypere forståelse av underliggende systemer, nettverk og databehandling fra deres fag.

Programmeringsstudenten har også ekstra erfaring innenfor Python. Inkludert fagene han har hatt over graden, så har han også jobbet som en læringsassistent for et fag innenfor kunstig intelligens for Python - og hatt ett større prosjekt som brukte Python for trening og testing av maskinlæringsmodeller.

Dette gjør at gruppen er godt rustet for å kunne designe og implementere en fullstendig løsning som vil oppfylle oppdragsgivers krav.

1.4 Rammer

1.4.1 Tidsrammer

- Kodeutviklingen, det vil si utviklingen av alle krav og funksjonaliteter, skal være ferdig innen 18. april 2024. Denne fristen gjelder kun for de essensielle funksjonalitetene. Endringer som følge av videre testing og/eller finpussing kan fortsatt utføres frem til den endelige tidsfristen.
- Den endelige tidsfristen for hele prosjektet, som inkluderer en ferdigutviklet programvare og en tilhørende prosjektrapport, er satt til 21. mai 2024.

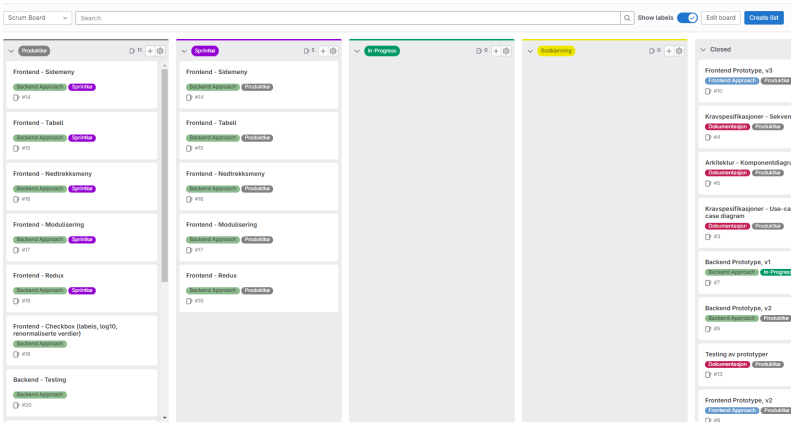
1.4.2 Teknologiske rammer

- Webapplikasjonen skal støtte `HTML5`, og være tilgjengelig gjennom de fleste populære nettlesere.
- I likhet med basisapplikasjonen skal webapplikasjonen ha åpen kildekode og falle under samme type lisens.

1.4.3 Andre rammer

- Det er bestemt at gruppen skal følge arbeidsmetodikken 'scrum', med en sprintperiode på to uker. Det skal brukes et scrumboard for å holde styr av oppgavene for en sprint og eventuelle oppdateringer til produktkø for prosjektet. Scrumboardet lages gjennom GitLab (1.2), hvor alle medlemene av gruppa er ansvarlig for å oppdatere oppgaver de selv jobber med.

[5]https://www.ntnu.no/studier/bprog, hentet 19.05.2024
[6]https://www.ntnu.no/studier/bidata/cybersikkerhet, hentet 19.05.2024



Figur 1.2: Utdrag av relevant scrumboard rett etter at en sprint hadde blitt planlagt.
Inneholder tabell for produktkø (1. fra venstre), sprintkø (2. fra venstre) og progresjon av oppgave (enten "in-progress", venter godkjenning", eller fullført/"closed").

- Det skal holdes et møte hver onsdag, ved klokken 11:15 til 12:00. Møtet skal holdes mellom alle gruppemedlemmer, sammen med produkteier og veileder. Møtet holdes fysisk på et angitt kontor ved NTNU Gjøvik, med mindre noe annet er bestemt.
    - Møter med produkteier og veileder kan avlyses dersom det ikke er noe nytt å rapportere. Dersom møter avlyses flere uker på rad har gruppen ansvar for å rapportere status for prosjekt til både produkteier og veileder gjennom epost.
- Møter mellom gruppemedlemmer skjer ved avtaler, og kan holdes både digitalt og fysisk avhengig av hva som avtales.
- Alle medlemmer av gruppa er utviklere for programvare. Dette betyr at de er selv ansvarlig for å lage effektiv og korrektkode, samt alle ansvar tilknyttet deres arbeid (testing, dokumentasjon, osv.).

1.5 Øvrige roller

- Vår oppdragsgiver er Jan Henrik Wold, som representerer Fargelabbens interesser ved IDI. Oppdragsgiver er ansvarlig for å representere bedriften, og å kommunisere deres tanker angående utviklingen av programvaren.
- Vår veileder er Ivar Farup. Han er ansvarlig for å gi akademisk veiledning til programvare og rapport, sammen med støtte angående administrative aspekter og kritisk tilbakemelding.
- Rollen som Scrum master har rullert fra sprint til sprint. Siden Scrum masterens og gruppeledreens ansvarsområder har en del overlapping fant vi det

hensiktsmessig å slå sammen disse to rollene. Scrum masteren er ansvarlig for å at alle Scrum-hendelser skjer innenfor de angitte tidsrammene, samt samarbeid innad i teamet.

- Utviklerrollen har blitt fylt av alle gruppemedlemene gjennom alle sprintene. Utviklerene er ansvarlige for å implementere oppgavene satt i sprintloggen uke til uke. Dette skal gjøres med ren og effektiv koding. Utviklerene er også ansvarlige for dokumentasjon og testing av skrevet kode.
- Redaktørrollen er fylt av Nikolay Savchuk. Redaktøren er ansvarlig for å overse dokumentasjonen av prosjektet. Dette inkluderer møtereferater, fremgangsrapporter og dokumentasjon nødvendig for rapporten selv. Han har også så ansvaret for levering av oppgaven og etterfølgende fremføring.

1.6 Rapportoppsett

1.6.1 Oppsett av rapport

Rapporten er oppsatt inni følgende kapitler:

1. **Kapittel 1 - Introduksjon** dekker bakgrunnen for prosjektet, hva oppgaven for prosjektet er, og eventuelle avgrensninger, mål og rammer. Den skal også introdusere gruppen og øvrige roller for prosjektet.
2. **Kapittel 2 - Teori** dekker den teoretiske bakgrunnen som er relevant for prosjektet. Dette handler kort om fargematchfunksjoner og den faglige teorien.
3. **Kapittel 3 - Kravspesifikasjoner** dokumenterer alle kravene vi har laget for prosjektet. Dette inkluderer funksjonelle, ikke-funksjonelle, operasjonelle og produktkø. Det har også blitt laget diagrammer.
4. **Kapittel 4 - Design & Arkitektur** tar for seg designet av det grafiske brukergrensesnittet, og introduksjon av arkitekturer. Det skrives om to arkitekturer, begge som potensielle løsninger å utføre oppgaven på.
5. **Kapittel 5 - Teknologier** omhandler teknologier som skal brukes i prosjektet, både de som var planlagt og de som ble tatt i bruk underveis. Disse nevnes etter at arkitekturen relevant for oppgaven er beskrevet.
6. **Kapittel 6 - Prosess** dekker hele utviklingsprosessen, organisert i sprinter. Det første delkapitlet omhandler diskusjonen og beslutningen mellom de to arkitekturene som tidligere er introdusert i rapporten. De påfølgende delkapitlene beskriver videreutviklingen av den valgte arkitekturen.
7. **Kapittel 7 - Implementasjon** fokuserer på det siste produktet etter utviklingsprosessen. Her utdypes ulike implementasjonsdetaljer. Spesifikt skal det diskuteres koden som ble utviklet og tankegangen bak hvorfor løsninger ble implementert på den måten de ble.
8. **Kapittel 8 - Testing & kvalitetssikring** beskriver hvordan programvaren sikrer kodekvalitet, gjennomfører testing, og bruker lisensiering.
9. **Kapittel 9 - Utrulling & Installasjon** beskriver hvordan programvaren ble iverksatt for tjenestegjøring på en webserver, og kort om hvordan den skal installeres.

10. **Kapittel 10 - Diskusjon & Konklusjon** inneholder en helhetlig diskusjon og drøfting av prosjektet. Det skal spesifikt utdypes om våre mål, bruken av KI, kritikk om det vi har laget, hvilket videre opplegg finnes det, og deretter en evaluering av vårt arbeid. Kapittelet slutter med en konklusjon.

1.6.2 Terminologi og praksis

Denne seksjonen inkluderer klargjøringer av terminologi brukt i rapporten, sammen med avklaringer angående bruk av font, styles og annet i rapporten.

- Alle fremtidge referanser til 'basisapplikasjon', 'basisprogrammet' (og lignende) refererer til 'CIE Functions' [1] programmet, som vårt prosjekt er basert på. Vårt prosjekt er en utvidelse av dette programmet. For denne grunn, gjenbruk av kode fra dette programmet er lov for å oppbevare konsistenthet med oppgaven.
- Det skal benyttes seg av `kodefont` for ting som typer og klasser. I tillegg kan det også benyttes for filnavn og variabelnavn dersom det passer.

Sammen med dette, anbefaler vi å konsultere vedlegg A for definisjoner om ord og uttrykk brukt i rapporten.

Kapittel 2

Teori

2.1 Fargematchfunksjoner

Når vi ønsker å måle noe, betyr det å fastlegge det på en kvantitativ måte slik at målingen kan senere brukes i sammenligninger og/eller utregninger [2]. Vi har flere typer målinger; meter for lengde, sekund for tid, og gram for vekt er kun noen allmennkjente eksempler. Et eksempel på et område hvor måling blir mer komplisert derimot, er innen fargevitenskap. Det er mulig å måle den kvantitative siden av lysbølger gjennom et spektrofotometer[1] for å se bølgelengdens frekvens, men denne målingen tar ikke hensyn til hvordan vi mennesker oppfatter fargen. Om vi skal måle denne siden av farger, bruker vi heller et visuell kolorimeter med standardiserte lyskilder, hvor brukeren justerer på en gitt farge inntil den er lik en annen referansefarge i verktøyet [3].

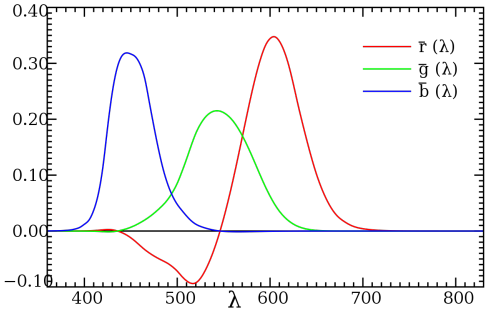
Denne typen tester danner grunnlaget for 'fargematchfunksjoner'. Disse funksjonene er basert på tappene i øynene våre og hvordan hjernen tolker signalene de sender. Tappene er reseptorer som gjør det mulig for oss å oppfatte kombinasjoner av rødt, grønt og blått lys, og dermed se hele spekteret av synlig lys. For å oppfatte farge, stimuleres disse tappene i varierende grad av forskjellige lys. Hvis det fantes en metode for å måle intensiteten for alle tre tappene for bestemte bølgelengder, ville det være mulig å lage en modell av hvordan et individs tapper reagerer under ulike lysforhold.

Dette ble oppnådd på 1930-tallet av forskeren William D. Wright, som utførte et eksperiment med en gruppe standardobservatører. Individer ble utsatt for et delt synsfelt adskilt av en rett linje. I det ene feltet var det en ukjent referansefarge, og i det andre feltet en kombinasjon av justerbare primærfarger. Deltakerne fikk justere intensiteten på primærfargene for å oppnå en matchende farge i begge feltene gjennom additiv fargeblanding. Målet var å justere fargene til begge feltene viste identiske farger ([4], 8.12).

Ved å gjennomføre dette eksperimentet med de samme parameterne på flere

[1] https://snl.no/spektrofotometer, hentet 19.05.2024

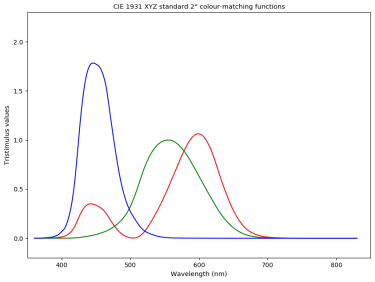
individer, ble det mulig å skape en fargematchfunksjon ved hjelp av tre vektingsfunksjoner for hver referansestimulus som ble testet i eksperimentet. Denne undersøkelsen dannet grunnlaget for den kjente 'CIE 1931' fargematchfunksjonen, som er utgangspunktet for all fargemåling og kolorimetrisk matematikk.



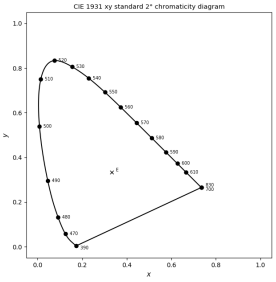
Figur 2.1: CIE 1931 (RGB) fargematchfunksjon, normalisert [5] .

Denne modellen (slik vist i figur 2.1), hadde klart å uttrykke monokromatiske fargestimuli som punkter bestående av r, g, b verdier korresponderende til deres respektive farger. Modellen ble snart videre utviklet til å ikke inkludere negative tall, og ble kjent som standardiseringsfunskjonen CIE 1931 (XYZ) (figur 2.2a) ([4], 8.12) når det ble snart innført av den internasjonale belysningskommisjonen (CIE). De tre verdiene av X, Y, Z ble virtuelle referansestimuli for fargekoordinater i deres fargevektorrom.

Denne modellen ble videre utviklet til å separere luminansen fra fargen, til å kun sitte igjen med kromasiteten - hvor sluttresultatet ble to punkter kjent som (x, y) - videre dannet det vi kaller et kromatisitetsdiagram for (x, y), karakterisert av dets hestesko form (figur 2.2b) ([4], 8.12).



(a) CIE 1931 (XYZ) fargematchfunksjon.



(b) CIE 1931 (x, y) kromatisitetsdiagram.

Figur 2.2: CIE 1931
Begge figurer er hentet fra basisprogrammet.

Bruken av disse modellene gjorde det mulig å representere farger som koordinatverdier i matematikk, noe som videre dannet grunnlaget for bedre represen-

tasjon av fargestimuli i teknologi. Et eksempel på deres betydning er ICC-profiler; spesielle profiler for enheter med kameraer eller skjermer som sikrer korrekt fargegjengivelse mellom dem[2]

2.2 Faglig teori

2.2.1 Webapplikasjon

En webapplikasjon kan beskrives som 'et program for nettleseren'. Mens vanlige applikasjoner er spesifikt laget for å kjøre individuelt på en enkelt datamaskin, er webapplikasjoner utviklet for å kjøre i nettlesere. De er avhengige av både en klientmaskin for å kjøre programmet og servere for å levere funksjonaliteter. Med andre ord, en webapplikasjon krever ikke bare brukerens maskin, men også minst en servermaskin for å håndtere funksjonaliteten og all interaksjon fra brukeren. Denne modellen er så vanlig at den har fått det tekniske navnet 'klient-tjener modellen' ([6] s. 67, kap. 2). Modellen består av to lag:

- **'Klient'/'Frontend'** - utviklingen av brukerinteraksjon *gjennom* det visuelle.
- **'Tjener'/'Backend'** - utviklingen av arkitektur og funksjonalitet *for* det visuelle.

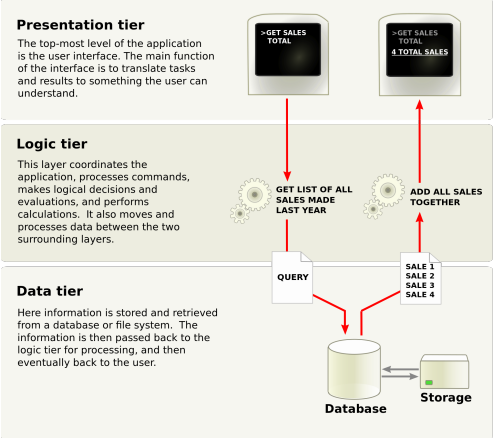
Disse skal dekkes mer i deres respektive kapitler nedenfor.

I tillegg til dette, tas det også opp ulike arkitektursmodeller for applikasjonen. Forskjellige applikasjoner krever forskjellige krav basert på deres påkrevde funksjonaliteter; enkelte må kanskje kunne lagre informasjon over lang tid, mens andre bare må vise frem et enkelt dokument. Gjennom dette ble det utviklet en serie av ulike modeller for hvordan utviklere skulle kunne strukturere deres webapplikasjoner, hvor noen relevante eksempler er:

- **'Tre-lag' arkitekturen (som vist på figur 2.3):** Et tradisjonelt og populært valg hvor webapplikasjonen sin struktur består av tre individuelle lag ([7], kap. 5).
  1. 'Presentasjonslaget' hos brukerens enhet, korresponderende til 'frontend' laget diskutert tidligere.
  2. 'Logikklaget' hos en webserver, korrespondere til hoveddelen av 'backend'.
  3. 'Datalaget' hos en separat database, også ofte omtalt som en del av 'backend'.

  Denne oppdelingen av lag (også kjent som prinsippet 'separasjonen av lag' ([8], s. 2)) er det som gjør arkitekturen et populært valg hos mange utviklere. Med å ha hvert lag uavhengig fra de andre, er det enklere å øke/redusere dets skalerbarhet i et system. Lagene er også strukturert på en måte hvor

[2] `https://www.color.org/faqs.xalter#wh2`, hentet 21.05.2024.



Figur 2.3: Et eksempel av en elementær '3-tier' arkitektur. [10]

brukere aldri snakker direkte med datalaget. De må passere logikklaget, noe som resulterer i en større grad av sikkerhet.

- **'To-lag' arkitekturen:** Et mindre brukt men allikevel populært valg for arkitektur [9]. Denne strukturen er veldig sammenlignbar med arkitekturen forklart over, men bruker kun to lag:

  1. 'Presentasjonslaget' slik beskrevet tidligere.
  2. 'Backend', hvor det kan bestå av en webserver, en database eller begge på samme maskin.

Denne arkitekturen består av færre komponenter, men gir til gjengjeld fordeler som at det er enklere å opprettholde, billigere, og dekker krav hos de fleste tjenester hvor en database og/eller funksjonalitet er ikke viktige aspekter.

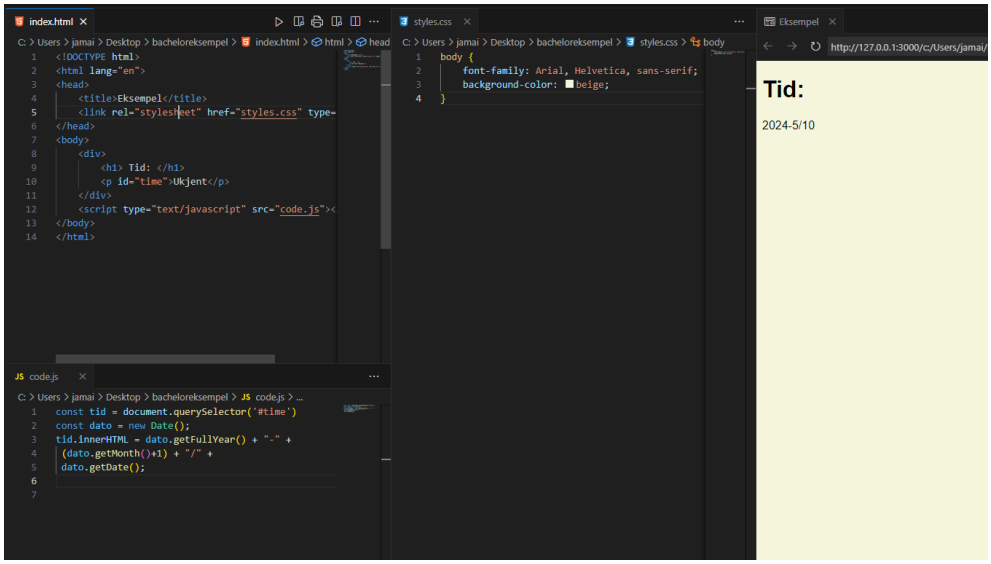
2.2.2 Frontend

Frontendutvikling er aspektet av webutvikling hvor det fokuseres på klienten, primært gjennom 'det visuelle' og 'det interaktive'. Dette gjøres gjennom utviklingen av et brukergrensesnitt som skal tillate en bruker å interagere med logikken til en applikasjon gjennom dets design. For hensikten av å formidle teorien bedre, deles dette kapittelet i to deler - en som forteller de tekniske detaljene, og en annen som går innom design.

Teknisk

Når det gjelder den tekniske siden hos frontendutvikling, snakker vi hovedsakelig om tre tekniske lag som bygger opp hele strukturen av hva en bruker ser og gjør på en webapplikasjon:

1. **Innholdslaget (HTML)**: Den essensielle strukturen og innholdet av en gitt nettside, med alt av tekst og semantisk struktur ([11], s. 20). Hver eneste



Figur 2.4: Enkel eksempel av frontend gjennom alle tre lag.

tittel, bilde og tabell som finnes på en nettside er der takket være bruken av HTML, også kjent som 'HyperText Markup Language' - ett dokumentasjonsspråk som bruker semantiske 'tagger' for å skille elementer fra hverandre.

2. **Presentasjonslaget (CSS)**: Presentasjon av innholdslaget utføres av presentasjonslaget, hvor regler for stil og design bestemmes ([11], s.229-230). Dette inkluderer også visuell struktur - annerledes fra den semantiske strukturen tilgjengelig gjennom HTML. Dette laget kontrolleres av stilspråket CSS ('Cascading Style Sheets').
3. **Interaksjonslaget (JS)**: Hvordan en bruker interagerer med innholdet blir behandlet gjennom interaksjonslaget, som både tilsetter indirekte respons (som dynamiske oppdateringer), og direkte (som umiddelbar tilbakemelding for input) til et ellers statisk innhold ([12], s. 4-5). Ulik de to foregående lagene, dette laget bruker faktisk et programmeringsspråk kalt JavaScript (JS).

Disse tre lagene utgjør grunnlaget for frontend. Innholdslaget er det eneste obligatoriske, mens de to andre kan betraktes som tillegg for å forbedre det visuelle aspektet eller legge til rette for brukerinteraksjon i et ellers statisk dokument. En demonstrasjon av disse lagene kan sees i figur2.4.

- Innholdslaget forteller dokumentet den semantiske strukturen av nettsiden. Det beskriver hvor det for eksempel skal finnes en tittel og et paragraf med innholdet 'ukjent'. Den lenker også opp presentasjons- og interaksjonslaget.
- Presentasjonslaget gir dokumentet reglene at bakgrunnsfargen for `body` taggen skal være beige, og at teksttypen skal være enten Arial, Helvetica, eller sans-serif.
- Interaksjonslaget tilbyr indirekte respons/interaksjon med å dynamisk opp-

datere innholdet av paragrafen fra innhold til å inneholde dagens dato.

En nettleser vil kunne tolke disse tre lagene i en serie av prosesser [13]:

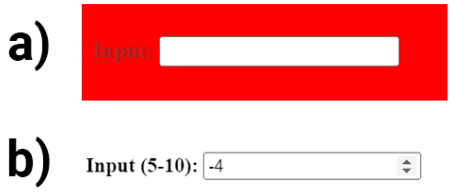
1. **Parsing:**
    - **Lasting av HTML:** Når en nettside, eller URL besøkes,henter nettleseren HTML-dokumenter fra serveren. Dette er startpunktet for å bygge en nettside[14].
    - **Analyse av HTML:** Parsetrinnet begynner med at nettleseren leser den rå HTML-teksten. Her brytes dokumentet opp i ulike elementer som for eksempel `<div>`, `<span>` og `<img>`. Formålet med dette er å forstå strukturen i dokumentet[15].
2. **Document Object Model(DOM):**
    - **Bygging av DOM-treet:** Etter parsing, bygger nettleseren DOM-treet. Dette er en strukturert representasjon av HTML-dokumenter. DOM representerer alle elementene som objekter. Dette gjør det mulig for JavaScript å sammhandle dynamisk med DOM[15].
    - **Scripting:** Når DOM-treet er ferdigstilt, kan JavaScript kjøre for å manipulere DOM ved å legge til, fjerne eller endre elementer og deres egenskaper[16].
3. **Rendering:**
    - **CSS Object Model:** Samtitid som DOM-treet bygges analyserer nettleseren også alle CSS filene tilknyttet HTML-dokumentet for å bygge CSS Object Model(CSSOM). CSSOM representerer alle stilene[15].
    - **Visuell rendering:** Når både DOM og CSSOM er på plass begynner nettleseren å tegne innholdet på skjermen. Dette innebærer beregning av layout får hvert element basert på dens dimensjoner og posisjon, etterfulgt av den faktiske tegningen av elementene[15].
4. **CSS og JavaScript:**
    - **Styling:** CSS-koden brukes for å definere hvordan HTML-elementene skal se ut på skjermen. Dette bidrar til å transformere det strukturerte innholdet for å gjøre det mer estetisk[17].
    - **Interaktivitet:** JavaScript bruker DOM for å legge til interaktivitet i nettsiden. Dette kan inkludere å håndtere brukerinteraksjoner som klikk og tastetrykk, gjøre API-calls og oppdatere brukergrensesnitt dynamisk[14].

Design

Hvordan en frontend er designet spiller vanligvis en stor rolle for brukere; dersom de skal interagere med applikasjonen på en fornuftig måte, så må applikasjonen først være laget fornuftig. På grunn av dette er design et veldig viktig felt i frontendutvikling. Dette er et stort felt med mye teori av alle former, men for hensikten

av vår rapport, så forteller vi heller om tre nøkkelord sentralt for designet: 'universell', 'validert' og 'brukersentrert':

- Det er viktig at designet er passet til å være '**universell**'. Det er viktig at applikasjonen er inklusiv og kan bli brukt av mest mulig folk ([18], s. 1.5), uavhengig av deres livssituasjoner. Det er naturlig at man ønsker at en tjeneste skal nå ut til flest mulig folk, og ikke kun til den gjennomsnittlige personen.
  Noen folk har dårlig syn som gjør det vanskeligere å se figurer med dårlig kontrast og store mengder av tekst - mens andre har vansker med interaksjon, og krever enklere tilgang til ressursene de trenger. Slik inklusivt design inkluderer også 'responsivt' design; prinsippet unikt til webutvikling hvor designet må passe brukerens enhet, uansett størrelse [19].
  Et eksempel på det motsatte av universell design vises frem på punkt a av figur 2.5; dårlig fargekontrast hos elementer gjør det vanskelig for folk med nedsatt synsevne til å se kanter (og derfor hva som er hva på bildet).
- Noe nesten like viktig er at designet er '**validert**'; trygg for både oss og brukeren. Det er ett kjent prinsipp innenfor programvareutvikling at '*en skal aldri stole på en bruker*' ([20], s. 1), for de kan enten ha ondsinnede tanker om å angripe programmet - eller angripe det med et uhell gjennom ukontrollert input. Derfor, har frontend også ansvaret for å forsørge seg å kun gjøre det mulig å ta inn validerte og spesifikke input fra brukeren, samt gi dem kun det nødvendige for å minimalisere fare for uhell.
  Et eksempel på det motsatte av slik kontrollert design er på punkt b) av figur 2.5; ugyldig input (enten med uhell eller med vilje) kan forårsake programfeil hos en applikasjon dersom de blir sendt videre uten validering.
- Til sist, må designet også være naturligvis **sentrert for brukeren**. Det finnes flere prinsipper på hvordan dette oppnås, men her tar vi for oss 'de fem dimensjonene av design', sammen med Don Normans prinsipper for interaksjonsdesign.
  - De fem dimensjonene sier at det skal være enkelt å forstå informasjon gjennom ord, at det brukes visuelle representasjoner for informasjonsformidling, at det finnes fysiske objekter som kan brukes (mus, telefon, tastatur), at det tar akseptabel tid å få informasjon, og sist at brukerens reaksjon på applikasjonen tas i betraktning mens de bruker den. [21].
  - Don Normans prinsipper mener at god synlighet, passende tilbakemeldinger, begrensinger (likt konseptet som 'kontrollert input' definert over), tydelig kartlegging, konsistenthet og tydelig markering av bruk er alle gode prinsipper å ha med seg når man ser for seg mulige interaksjoner innenfor design [22], s. 21.



Figur 2.5: Eksempel på dårlig design; a) viser frem 'eksklusiv' design, og b) viser frem ikke-validert input.

Rammeverk

Oppgjennom årene med webutvikling, har det blitt langt mer vanlig å bruke spesielle rammeverk som tar seg av frontend (istedenfor å bruke en kombinasjon av ren HTML, CSS og JS). Disse rammeverkene gjør det enklere å bygge kompliserte webapplikasjoner med å lage komponentmodeller som kan uttrykkes som nettsider - i tillegg til å lage effektive 'SPA' webapplikasjoner gjennom oppdateringer av kun en side. De tilbyr også en rekke fordeler over vanlig JS, som enklere modulisering og gjenbruk av kode, og greiere tilgang til flere verktøy skapt av arbeidsmiljøet rundt rammeverket. Kjente eksempler er Angular[3], Vue[4] og Ember[5] - som har hver av sine fordeler og ulemper relativ til hverandre.

2.2.3 Backend

Backendutvikling er det andre aspektet av webutvikling. Her fokuseres på alt av applikasjonslogikk, vanligvis med tanke på det som skjer hos frontenden av webapplikasjonen. Dette går langt mer inn på arkitekturen av hele webapplikasjonen, sammen med alle former av funksjonalitet som finnes.

I motsetning til frontend er det ingen fast og obligatorisk teknisk struktur for en backend; den kan skrives i flere ulike programmeringsspråk, hver på sine egne metoder og standarder. Vi tenker heller at dets tekniske struktur avhenger av funksjonalitene den må kunne dekke avhengig av strukturen til applikasjonen. I noen tilfeller kreves det bruk av en database, mens i andre situasjoner kreves det bruk av flere instanser av backend for lastbalansering.

Allikevel, finnes det en vanlig struktur for backend (som applikasjonslogikk) i form av det som kalles en 'API'.

Application Programming Interface

Som diskutert tidligere ved 2.2.1, ble backend beskrevet som '*utviklingen av funksjonalitet for det funksjonelle*' - noe som kan betyr at backendprogrammet vil utføre gitte aksjoner fra frontendprogrammet gjennom en brukers instruksjoner. Denne

[3]https://angular.io/, besøkt 19.05.2024
[4]https://vuejs.org/, besøkt 19.05.2024
[5]https://emberjs.com/, besøkt 19.05.2024

formen for kommunikasjon mellom to datasystemer danner grunnlaget for det vi kaller for en 'API'.

Et programmeringsgrensesnitt (kjent som 'Application Programming Interface' på engelsk) er i hovedsak et grensesnitt for kommunikasjon mellom to applikasjoner ([23]). På samme måte som at vi sier at brukergrensesnittet er hvordan vi som mennesker kommuniserer med data gjennom visuelle representasjoner - så er en API hvordan datamaskiner snakker med hverandre gjennom visse representasjoner av data.

Det finnes flere måter å konstruere et 'API' på. Man kan selvfølgelig lage en intern representasjon spesifikt for egne oppgaver, men det er også mulig å følge visse arkitekturer som kan være mer gunstige basert på kravene man har. Noen eksempler på slike arkitekturer er:

- **REST ('Representation State Transfer')**: Arkitekturstil som baserer seg på seks designprinsipper av Roy Fielding ([24], kap. 5). Stilen er kjent for å være enkel å implementere, har god bruk for klient gjennom `HTTP` metoder, og for fordelene hvert prinsipp tilbyr. For at en server skal følge arkitekturen (også kalt å 'være RESTful'), så må disse seks prinsippene følges:
  1. Separasjon av klient og server gjennom forskjellig logikk. Det skal implementeres en separasjon av bekymringsområder, hvor det som gis til klienten er kun det klienten skal bekymre seg for - og server har kun det *den* må bekymre seg for ([24], 5.1.2).
  2. Server lagrer ikke tidligere laget informasjon fra klientforespørsel (verken resultat av forespørsel eller informasjon om forespørsel).Det er tilstandsløs for å forbedre skalerbarhet og sikkerhet ([24], 5.1.3).
  3. Data hentet fra server må markeres enten implisitt eller eksplitt om det kan settes i klientens egen cache for å øke ytelse og minimalisere mengde av forespørsler fra en gitt bruker ([24], 5.1.4).
  4. Hele applikasjonen må være utviklet på en måte hvor komponenter ikke vet hvem de skal kommunisere med konkret - så derfor, må de utføre deres logikk uavhengig av hvem som skal bruke det ([24], 5.1.6).
  5. Dersom serveren sender ut kjørbar kode med hensikt om å øke funksjonalitet for tjenesten, så må koden kunne kjøre hos klienten ved mottakelse av ressurs. Dette prinsippet er valgfritt ([24], 5.1.7).
  6. Alle unike ressurser har en representasjon tilgjengelig på en unik URL ('Uniform Resource Locator') som kan brukes til å identifisere kun den gitte ressursen. Gjennom denne URL'en, så må en bruker kunne manipulere ressursen gjennom bruk av HTTP med riktig semantisk metode. En respons må også inneholde metainformasjon om forespørselen og dets status. Til slutt, må informasjonen en klient får fra ressursen være nok til å traversere og finne alle ressurser uten tidligere kjennskap til dets arkitektur ('HATEOAS') ([24], kap. 5.1.5).
- **SOAP (Simple Object Access Protocol):** SOAP er en annen arkitekturstil

som baserer seg på bruken av en meldingsprotokoll av samme navn[6]. Stilen baserer seg på tre komponenter; en server med SOAP, et register av alle servere, og en klient som bruker registeret til å finne en ønsket server. Klienten binderseg til serveren for en dialog hvor eventuelle kommunikasjoner foregår gjennom XML og HTTP ([25], kap. 2.1). SOAP har fordelen med at den har god sikkerhet med bedre støtte fra webstandarder ([26], s. 14).

- **GraphQL**: En egen arkitektur basert på sitt eget språk og kjøring[7]. Ideen er at brukere kan lage svært spesifikke forespørsler på eksakt hva de ønsker fra en tjeneste (gjennom en melding skrevet i språket). En API som kjører på GraphQL skal da kunne ta inn forespørselen, og gi kun det brukeren ønsket. Den har fordelene av å være enkel å bruke med å kun trenge en forespørsel fra en klient for å få alt av informasjon en ønsker, og har god bakgrunn som et prosjekt utviklet av Facebook i 2012.

2.2.4 WebAssembly

Det ble tidligere skrevet i rapporten at det er tradisjonelt å ha JavaScript som "interaksjonslagetfor en frontend. Dette skyldes en enkel grunn: Det har lenge vært det eneste programmeringsspråket en kunne ha i frontend. Dette er på grunn av nettstandarder og hva mesteparten av nettlesere støttet. For de fleste så var dette heller ikke nødvendigvis et problem; JavaScript var laget spesifikt for webutvikling, og tok nytte av mange innovasjoner som Chrome V8[8]. Det var kun i 2019, hvor Wasm (WebAssembly) ble introdusert som en offisiell standard av W3C [27].

WebAssembly er inspirert av programmeringsspråket Assembly, og er selv også et 'lavt nivå' språk spesifikt utviklet med tanke på frontend for webapplikasjoner[9].

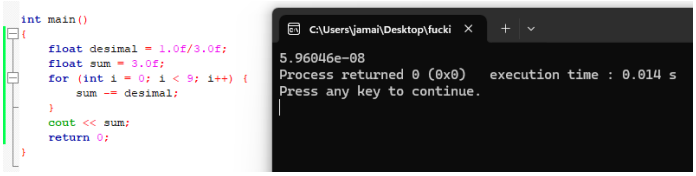
Wasm er ikke et språk man skriver man vanligvis skriver direkte, men heller et språk andre 'høynivå' språk blir kompilert til. Det åpner teoretisk sett for muligheten å kjøre flere språk gjennom nettleseren sammen med deres eventuelle fordeler og ulemper. Selv om teknologien er nyere og mindre brukt enn JS, så finnes det forskning som indikerer at Wasm er mer energieffektiv [28] og raskere enn det tradisjonelle valget for visse applikasjoner. Mange mener allikevel at begge burde brukes sammen for å oppnå full nytelse av hva en nettleser har å tilby. Et kjent eksempel på en kompilator for Wasm er Emscripten[10], som har evnen til å kompilere C/C++ med støtte for flere APIer som `SDL` og `pthreads`.

2.2.5 Beregningspresisjon

Noe viktig å vite om til denne oppgaven er det besynderlige fenomenet som oppstår hos datamaskiner når de regner med desimaltall. Ta eksemplet under, ved

[6] https://www.w3.org/TR/soap/, besøkt 19.05.2024
[7] https://graphql.org/, hentet 21.05.2024
[8] https://v8.dev/, besøkt 19.05.2024
[9] `https://webassembly.org/`, hentet 21.05.2024
[10] https://emscripten.org/

figur 2.6:



Figur 2.6: Eksempel på kalkulasjonspresisjonsfeil i C++.
Resultatet skulle bli 0, for $3-(10*1/3)=0$ - men programmet mener heller at det er lik $5.9604*10^{-}8$.

Datamaskiner utfører matematiske beregninger ved å representere tall binært. Dette er ikke er et problem for heltall fordi de kan defineres komplett i en binær form.Desimaltall er derimot vanskelige å uttrykke i binær form (for eksempel, 0.1 i binær form er lik en uendelig sekvens av 0.000110011001100...) [29]. De kan ikke uttrykkes nøyaktig som de faktisk er, så datamaskiner tilnærmer seg ved å kutteddenne uendelige sekvensen til en gitt presisjon. Dette resulterer vanligvis i et tall som er *omtrent* likt det opprinnelige desimaltallet. Om vi tar 0.0001100110011 med å kutte av eksemplet over, så ville vi ha fått desimaltallet 0.09997558593 når vi konverterer det tilbake - noe som er nært men ikke likt 0.1.

I mange tilfeller, ender datamaskinen opp med et tall som er så nært det opprinnelige tallet at det spiller lite rolle - men i felt som økonomi og fysikk, så kan dette fenomenet være katastrofalt dersom det oppstår. Dette problemet løses vanligvis med opp/nedrunding av tallet.

2.2.6 WSGI & ASGI

I et forsøk på å skape bedre grunnlag for nettrammeverk i Python, ble det skapt en konvensjon i 2010 kalt WSGI (kort for 'Web Server Gateway Interface') med detaljer for en slags abstraksjon mellom en webserver og applikasjon bygget opp i Python [30]. Målet var å standardisere deres kommunikasjon mellom hverandre, for å tillate mer allsidighet og uavhengighet fra servere.

Denne innsatsen hadde flere suksesser, og grunnla basis for flere nettrammeverk tilgjengelig i Python - inkludert kjente eksempler som Flask og tidligere versjoner av Django[11].

Med introduksjonen av asynkronitet i Python gjennom `await` og `async` ble det etterhvert laget en ny konvensjon i 2019 for å støtte dette for nettforespørseler. Dette ble gjort fordi WSGI kun støttet synkron takling av forespørsler. Denne konvensjonen kalles for ASGI (kort for 'Asynchronous Server Gateway Interface'), og introduserte bruken av asynkron kommunikasjon mellom applikasjon og serverlag tidligere diskutert hos WSGI for å kunne støtte asynkron takling av forespørsler

[11] `https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface`, besøkt 19.05.2024

[12].

Denne etterkommeren hadde stor suksess, og resulterte også i opprettelsen av flere kjente rammeverk, som FastAPI[13], nyere versjoner av Django[14], og Sanic[15].

2.2.7 Human Computer Interacton

Human Computer Interaction (HCI) er studiet av hvordan mennesker som brukere samhandler med datamaskiner. Gjennom studiet er det mulig for utviklere å lære hvordan de kan fjerne feil i design, og forbedre brukeropplevelsen. Den letteste måten oppnå dette på er å integrere HCI prinsippene i utviklingsprosessen. [31]

Brukervennlighet er en vesentlig egenskap ved HCI. Det handler om metoder for å gjøre applikasjonen mer intuitiv under designprosessen. Lærbarhet, effektivitet, minneverdighet, brukerfeil og tilfredshet er alle viktige faktorer. Lærbarehet defineres som hvor enkelt en bruker kan utføre sine oppgaver første gang de bruker programmet. Når en bruker har mestret grensesnitter, referer effektivitet til hvor enkelt det er for brukeren å utføre oppgaven brukeren har satt seg. Minneverdighet handler om hvor lett det er for brukeren å reetablere seg ferdigheter når brukeren kommer tilbake til applikasjonen etter å ikke ha brukt den på en stund. Feil henviser til hvor mange feil brukeren gjør og hvor enkelt det er for dem å korrigere dem. Tilfredshet betyr hvor behagelig designet er å bruke. [32]

[12]https://asgi.readthedocs.io/en/latest/specs/main.html, besøkt 19.05.2024
[13]https://fastapi.tiangolo.com/, besøkt 19.05.2024
[14]https://docs.djangoproject.com/en/5.0/howto/deployment/asgi/, hentet 19.05.2024
[15]https://sanic.dev/en/, hentet 19.05.2024

Kapittel 3

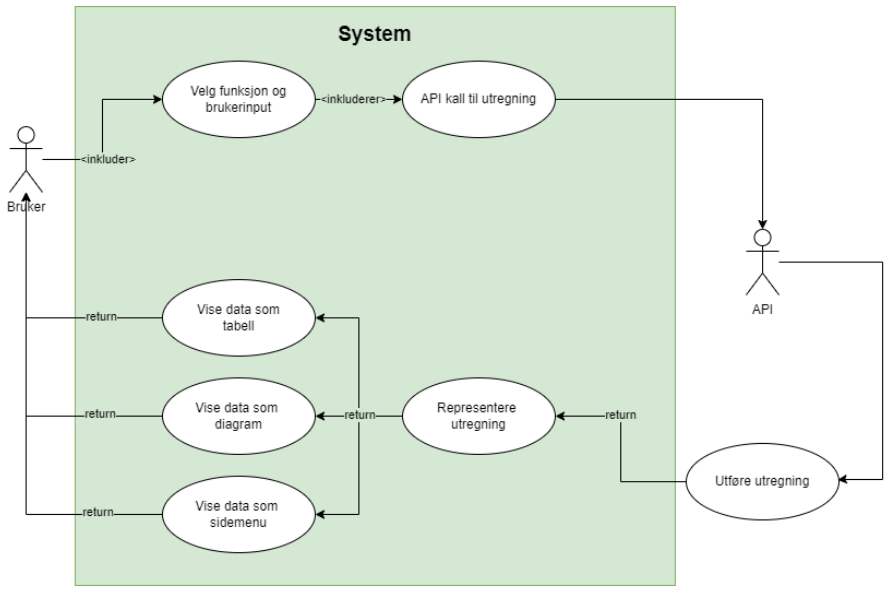
Kravspesifikasjon

For å takle denne oppgaven, har vi laget et sett med krav for hvordan resultatsmålet skal oppnås. Dette inkluderer funksjonelle krav, ikke-funksjonelle krav, og operasjonelle krav med innblandelse fra domenemodell og produktkø.

3.1 Funksjonelle krav

Fordi vårt prosjekt handler om å lage en derivert kopi av basisapplikasjonen, har følgende funksjonelle krav blitt satt med tanke på dette programmet. Dette er detaljert i følgende use-case diagram og use-cases:

3.1.1 Use-Case Diagram



Figur 3.1: Use Case-diagram

3.1.2 Use-cases:

I og med at de funksjonelle kravene skal være lik mellom basisapplikasjon og webapplikasjon, så er disse use-case scenarioene også direkte utførbare på basisprogrammet.

Use Case 1

- Navn: Interagere med graf for valgt funksjon.
- Aktør: Bruker.
- Mål: Bruke webapplikasjonen til å se på og interagere med graf tilhørend en spesifikk fargeromsfunksjon for et gitt sett med parametere.
- Beskrivelse: En bruker velger en av de tilgjengelige fargeromsfunksjonene i programmet. Deretter endrer de på input verdier hos webapplikasjonen, og trykker på en knapp som skal regne ut grafen. Etter det skal grafen vises i webapplikasjonen, hvor brukeren kan interagere med den.

Use Case 2

- Navn: Aktivere logaritmiske verdier for LMS.
- Aktør: Bruker.
- Mål: Bruke webapplikasjonen til å finne logaritmiske verdier for LMS fargeromsfunksjon.
- Beskrivelse: En bruker velger enten 'LMS' eller 'LMS base' fra programmet. Deretter, justerer de på input verdiene for valgt funksjon i webapplikasjon, før de trykker en knapp for utregning. Etter dette finner brukeren en knapp som er markert med 'logarithmic values', som de trykker på for å se at det aktivereres eller deaktivereres logaritmiske verdier for LMS.

Use Case 3

- Navn: Finne en spesifikk verdi i tabell for valgt funksjon.
- Aktør: Bruker.
- Mål: Bruke webapplikasjonen til å finne verdier korresponderende til en spesifikk verdi for en funksjonstabell.
- Beskrivelse: En bruker velger en av de akutelle fargeromsfunksjonene i webapplikasjonen. Deretter justerer de på input, og trykker på utregningsknappen for å få utregningene. Til slutt, trykker de på en knapp som bytter ut grafen med en tabell. I denne tabellen kan brukeren bla og lete etter en spesifikk verdi for funksjonens utregninger.

3.2 Ikke-funksjonelle krav

Vi har også laget følgende ikke-funksjonelle krav til applikasjonen vår:

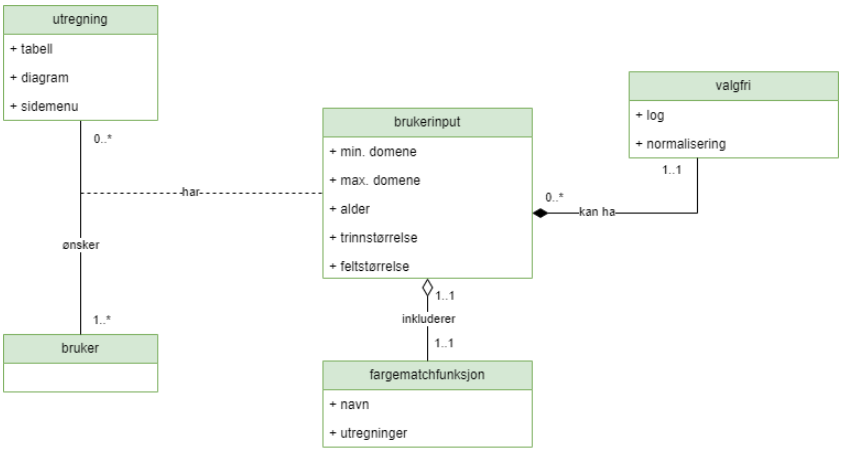
- Webapplikasjonen burde ha et brukergrensesnitt som er likt som det som finnes i basisprogrammet. Brukergrensesnittet burde være tilgjengelig for alle brukere, uavhengig av teknisk erfaring og/eller livssituasjon.
- Webapplikasjonen burde designes med tanke på ytelse. Det skal sørges for at beregninger og/eller visualiseringer utføres med så lite forsinkelser som mulig, og med raskest responstid mulig.
- Webapplikasjonen burde også designes med tanke på skalerbarhet, dersom det må skaleres opp i fremtiden.
- Webapplikasjonens kildekode skal være under åpen lisensiering. Den må også ha ordentlig dokumentasjon for planer om utrulling og gjenbruk av kode.

3.3 Operasjonelle krav

I og med at dette prosjektet skal tjenestegjøres og brukes som et produkt av oppdragsgiver etter den endelige tidsrammen, er det viktig for oss å detaljere operasjonelle krav. Men, fordi vi har avgrensningen at vi ikke skal fokusere på maskinvare, har vi laget følgende operasjonelle krav:

- Svartid på alle mulige forespørsler til system fra bruker må ligge under 1000ms.
- Systemet må kunne takle forespørsler fra flere brukere samtidig, med forslag som asynkronitet eller andre muligheter som kan være relevante for valgt arkitektur.

3.4 Domenemodell



Figur 3.2: Domenemodell

3.5 Produktkø

Produktkøet ble etablert ved starten av prosjektet, og oppdatert jevnlig gjennom dets bruk på Scrumboardet. Det er inkludert i vedlegg E.

Kapittel 4

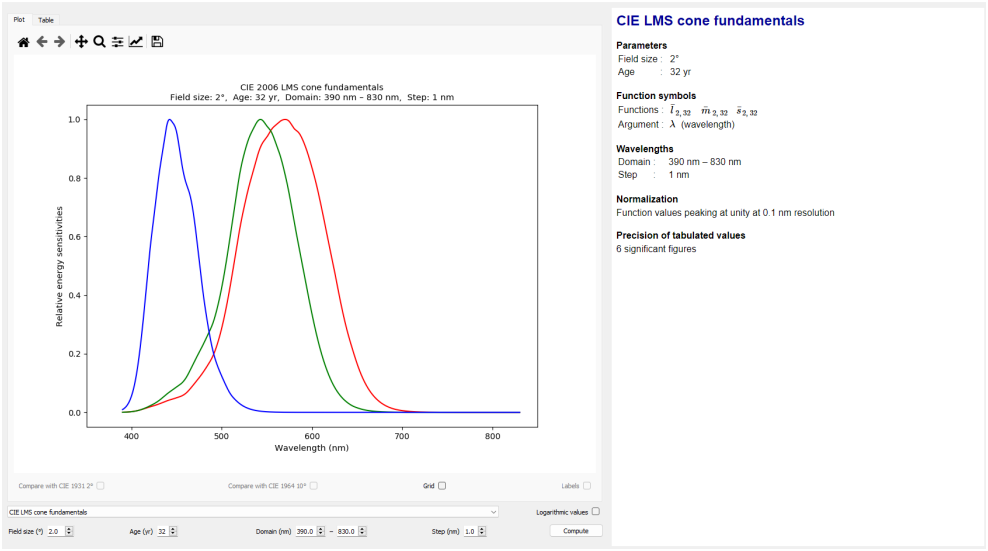
Design og Arkitektur

4.1 Grafisk grensesnitt

Oppdragsgiver uttrykket tidlig et ønske om at webapplikasjonen skulle holdes så lik som mulig til basis applikasjonen (vist i vedlegg D). For å oppnå dette, fokuserte vi på å opprettholde et uniformt grensesnittdesign og interaksjonslogikk mellom basis- og webapplikasjonen. Vi sørget for at alle elementene i brukergrensesnittet var like i både utseende og funksjonalitet. Dette vil bidra til å gjøre overgangen sømløs for brukeren.

I applikasjonen kan brukeren velge mellom ti forskjellige fargematchfunksjoner i en nedtrekksmeny. Under nedtrekksmenyen kan brukeren spesifisere ulike parametere, som feltstørrelse, alder, domene og skritt, dersom det er ønskelig. Etter dette kan brukeren trykke på knappen "compute" for å oppdatere en grafisk og tabulær fremstilling av utregningene, sammen med en informativ sidemeny med formler og annen nyttig informasjon benyttet for utregningene.

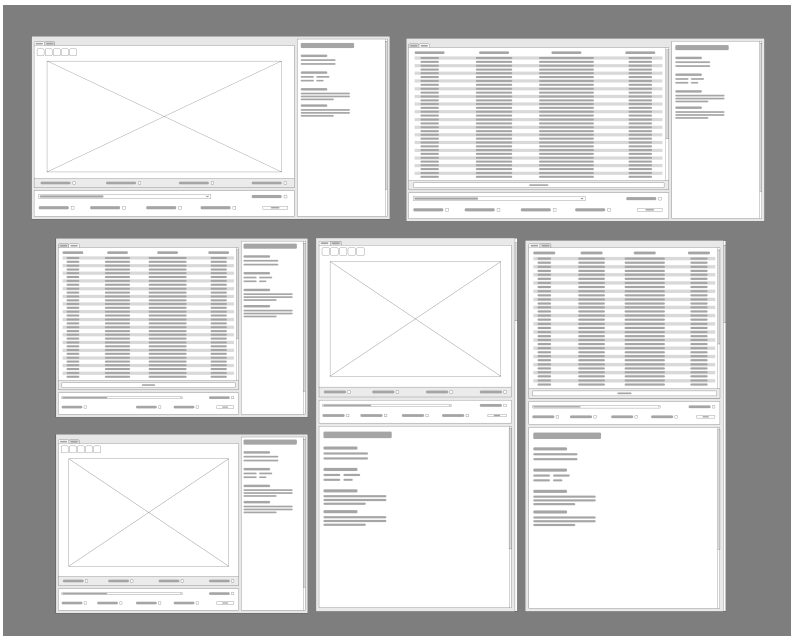
Øverst i applikasjonen finnes det en navigasjonslinje, hvor brukeren kan velge mellom tabulær eller grafisk visning. Under dette, er det plassert to iframes ved siden av hverandre. Den venstre iframen oppdateres ettersom brukeren bytter mellom tabulær eller grafisk fremvisning. Den høyre iframen viser formler og annen informasjon benyttet for for å regne ut de dataene som vises.



Figur 4.1: Basisapplikasjonen 'CIE Functions'

4.1.1 Wireframe

For å oppfylle de ikke-funksjonelle kravene til brukergrensesnittet ble det laget flere detaljerte wireframes. Dette vil hjelpe oss med å sikre intuitiv navigasjon. Brukere skal enkelt kunne finne og benytte alle funksjoner uten behov for omfattende opplæring.



Figur 4.2: Wireframe for brukergrensesnittet

Wireframen, vist på figur 4.2, tar hensyn til responsivt design, samtidig som det forsøker å forholde seg til utseendet til basisprogrammet. Wireframen viser applikasjonen ved fullskjerm (to øverste elementer), applikasjonen ved mindre skjerm (to elementer fra venstre bunn), og til sist, applikasjonen ved enda mindre skjerm (to elementer fra midten bunn og høyre bunn). Den minste skjermen skal ligne til dimensjonene av en mobilenhet.

Utseendet ble laget som en kopi av det som allerede finnes i basisprogrammet, med tanken at en god replikasjon vil gjøre det enklere for brukere å tilvende seg den nye applikasjonen. En ny funksjonallitet, som vi tilføyer med responsivt design i tankene, er å flytte sidemenyen til bunnen av grensesnittet når skjermen blir for smal. Dette finnes ikke i basisprogrammet, men vi mente at det var nødvendig å inkludere dette for å støtte flere enheter og dimensjoner for applikasjonen.

4.2 Systemarkitektur

Som skrevet tidligere i rapporten, så er en stor del av vår oppgave å prøve forskjellige rammeverk for å finne hvilket som passer vårt prosjekt best. For å lage en interessant oppgave med mye diskusjon, har vi valgt å gå for to helt unike arkitekturer med hver av deres fordeler og ulemper:

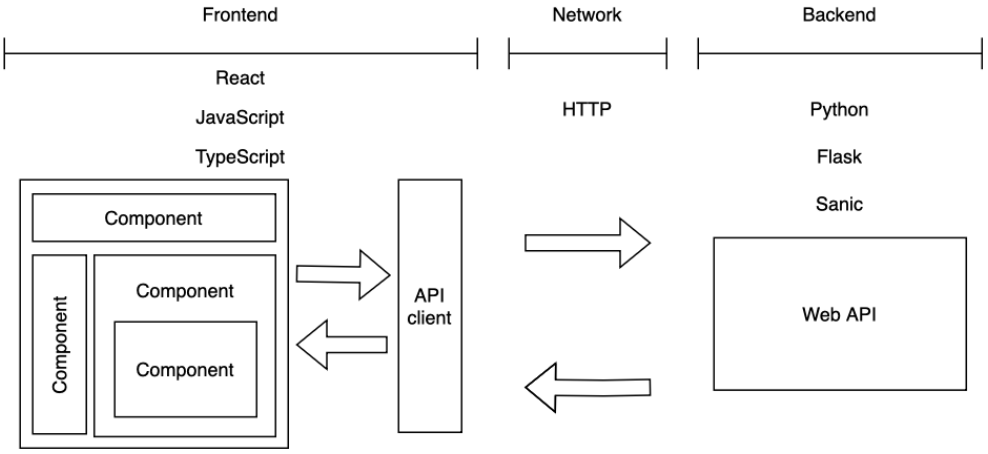
- En 'klient-tjener-basert' arkitektur, hvor utregninger utføres på backend(tjeneren), og klient refererer til brukergrensesnitt eller frontend.
- En 'frontend-basert' arkitektur hvor utregninger utføres heller på frontend, hos klienten. For videre referanser, skal denne arkitekturen bli referert til som 'alt-i-nettleseren'.

Planen er å se på disse arkitekturene, finne passende teknologier og eventuelt lage prototyper hvor vi ser på fordeler og ulemper. Deretter beslutter vi oss for en av dem og utvikler vårt produkt basert på den valgte arkitekturen.

4.2.1 Arkitektur 1: 'Klient-tjener'

I en klient-tjener basert arkitektur, mener vi at applikasjon blir laget med en to-lagsarkitektur, bestående av følgende lag:

1. Presentasjonslaget(frontend) hos klienten: Dette er det visuelle laget, og består av et grafisk brukergrensesnitt som brukerene kan samhandle med direkte.
2. Logisk applikasjonslag hos serveren(backend): Dette er hvor beregninger vil bli utført og APIen lages.



Figur 4.3: Arkitekturdiagram for frontend-backend løsning

Presentasjonslaget

Presentasjonslaget, ofte referert til som brukergrensesnittet, er der brukerene samhandler med applikasjonen. I vårt tilfelle syntes vi det var hensiktsmessig å benytte React og TypeScript. Dette gir flere fordeler:

1. Responsiv Design: Bruk av React gjør det mulig å lage en responsiv og dynamisk UI som kan tilpasse seg forskjellige skjermstørrelser og enheter[33].
2. Komponentbasert Arkitektur: Reacts komponentbaserte tilnærming tillater gjenbrukbare og isolerte komponenter. Dette gjør applikasjonen letter å vedlikeholde og skalere[34].
3. Typesikkerhet: TypeScript gir statisk typesjekking. Dette bidrar til å redusere feil under utviklingen, og gjør koden lettere å forstå og vedlikeholde[35].
4. Responsiv Ytelse: Brukergrensesnittet skal reagere raskt på brukerens handlinger med minimal ventetid.

Applikasjonslaget

Applikasjonslaget, også kjent som det logiske laget, er der kjernelogikken til applikasjonen blir implementert. I vårt system er dette laget utviklet i Python. Python ble valgt fordi basisapplikasjonen er laget på Python, samt at programmeringsspråket er spesielt egnet for beregninger og databehandling. Applikasjons laget er ansvarlig for å oppfylle de følgende funksjonelle kravene til applikasjonen:

1. Beregininger: Dette laget håndterer komplekse beregninger baser på parameter spesifisert av brukeren.
2. API Funksjonalitet: Applikasjonslaget eksponerer APIer som gjør det mulig for forntend å kommunisere med backend og hente nødvendige data.

API Design

Vi valgte å bruke REST-prinsipper for å strukturere vårt API. REST er et sett med retningslinje som sikrer skalerbarhet, enkelhet og effektivitet i kommunikasjonen mellom klient og server. Noen av fordelene med REST:

- Skalerbarhet: RESTful APIer kan enkelt skaleres over flere servere[36].
- Modularitet: Hver ressurs er klart definert. Dette gjør det enkelt å oppdatere og vedlikeholde systemet[36].
- Statelessness: Hver forespørsel fra klient til servern må inneholde all informasjonen som kreves for å forstå og behandle forespørselen. Dette bidrar til å forenkle serverens logikk[36].

Domenemodell

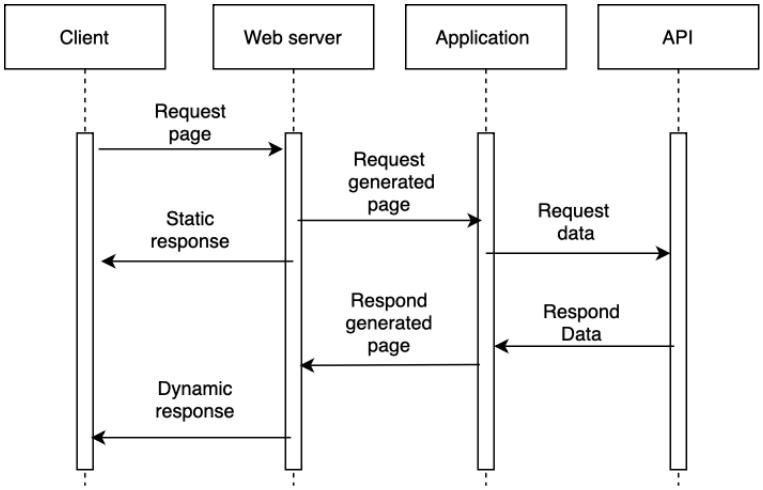
Diagrammet viser relasjonene mellom ulike elementer i systemt 3.2:

1. Bruker: En bruker kan ønske å utføre flere beregninger. Hver bruker kan initier en eller flere beregningsforespørsler.
2. Utregning: Etter representere selve beregningsprosessen. Hver utregning er koblet til brukerinput via en relasjon som indikerer at en utregning har en eller flere brukerinput.
3. Brukerinput: Dette inneholder flere forskjellige parametre. brukerinput inkluderer også en fargematchfunksjon som kan være enten en standardiseringsfunksjon eller en parameterbasert funksjon.
4. Valgfri Funksjonalitet: Brukerinput kan også ha valgfrie spesifikasjoner. For eksempel om de ønsker logoritmisk visning av data eller renormaliserte verdier.

Domenemodellen spiller en kritisk rolle i å opprettholde systemets integritet og funksjonalitet. Ved å implementere nødvendige regler og prosesser for datamanipulering legger domenemodellen grunnlaget for en robust og pålitelig applikasjon. Godt samarbeid mellom domenemodellen og REST APIet er essensielt for å sikre nøyaktig output fra applikasjonen basert på de spesifiserte parameterne.

Sekvensdiagram

Sekvensdiagrammet illustrerer hvordan en klientforespørsel håndteres sekvensielt gjennom ulike lag i denne webappliasjonen, fra forespørsel til respons. Diagrammet viser samspillet mellom klienten, webserveren, webapplikasjonen og APIet.



Figur 4.4: Sekvensdiagram for backend-basert løsning

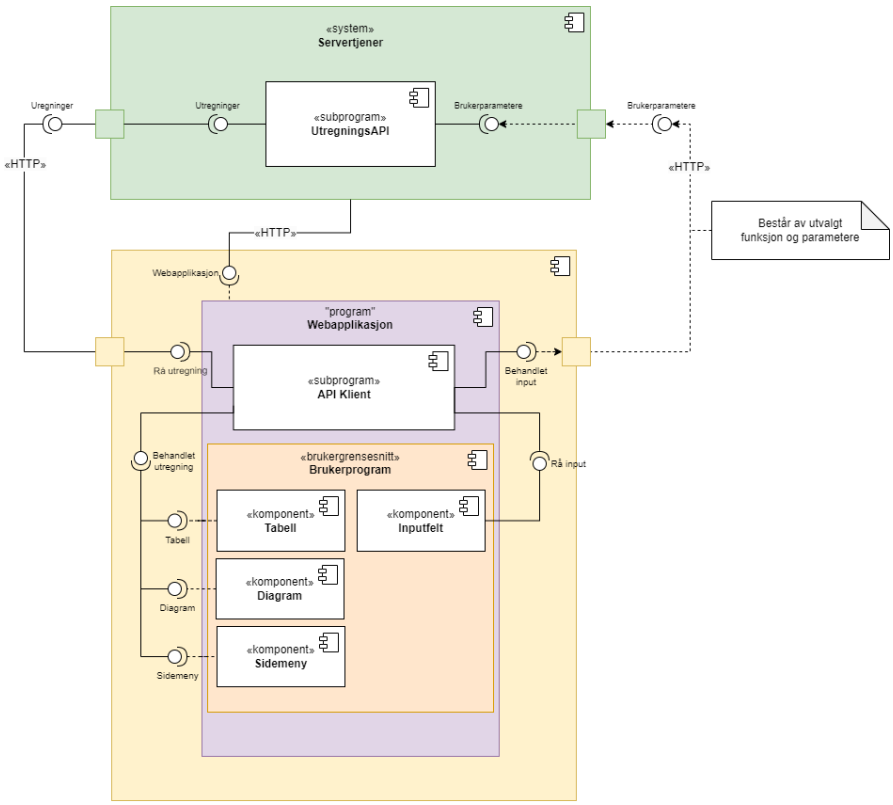
Når en klient, som typtisk er en nettleser, sender en forespørsel om en side til webserveren vurderer webserveren om innholdet er statisk eller dynamisk. statisk innhold, som HTML-filer, bilder eller CSS blir umiddelbart sendt tilbake til klienten som en statisk respons. Dette er rask og krever ingen videre behandling.

For dynamisk innhold må webserveren be applikasjonen om å generere innhold på siden.

APIet fungerer som et grensesnitt til å utførte beregninger. Når applikasjonen gjør en forespørsel til APIet, gjør APIet de nødvendige beregningene og returnerer dataen til applikasjonen. Applikasjonen bruker deretter disse dataene til å lage den dynamiske siden.

Komponentdiagram

Komponentdiagrammet viser hvordan de ulike delene av systemet henger sammen. Hovedkomponentene er API, nettleser, webapplikasjon og brukergrensesnitt. Hver komponent har definerte ansvarsområder:



Figur 4.5: Komponentdiagram for backend-basert løsning

1. **API:** Håndterer alle beregninger og brukerparametere. Dette inkluderer å utføre beregninger basert på brukerinput og returnere resultatene til webapplikasjonen.
2. **Nettleser:** 'Hoster' webapplikasjonen og presenterer den til brukeren. Den mottar også data fra APIet for å oppdatere brukergrensesnittet.
3. **Webapplikasjon:** Inkluderer API-klienten som kommuniserer med backend for å hente data og behandle input. Dette laget fungerer som en mellommann mellom frontend og backend.
4. **Brukergrensesnitt:** Består av ulike komponenter som tabell, inputfelt og diagram. Disse komponentene gir brukeren mulighet til å samhandle med applikasjonen og viser de nødvendige dataene på en strukturert måte.

4.2.2 Arkitektur 2: 'Alt-i-nettleseren'

Diskusjon

I en frontend-basert arkitektur flyttes all nødvendig logikk for beregninger til brukerens nettleser, i motsetning til den mer tradisjonelle arkitekturen beskrevet tidligere. Selv om dette ikke er like tradisjonelt, så er det heller ikke en uvanlig arkitektur for webapplikasjoner hvor umiddelbar respons er det viktigste; et godt

eksempel er nettleserspill[1] hvor spillet kjøres i selve nettleseren uten å måtte kommunisere med en tjener for å kjøre.

Det er viktig å merke at tjeneren eksisterer fortsatt i en slik arkitektur - for den er ansvarlig for å utlevere webapplikasjonen. På grunn av dette, kan vi fortsatt omtale denne arkitekturen som 'to-lag' - men det er riktigere å omtale den som en variant.

Vi kan se hvordan med å se på dets komponent- og sekvensdiagram, som vist i figur 4.6 og 4.7. Tjeneren vil fremdeles eksistere for å gi fra seg webapplikasjonen - men istedenfor at den skal selv utføre applikasjonslogikken, flyttes det over nativt inni nettleseren. Om vi skulle sammenligne det med den forrige arkitekturen, kan man egentlig si at de er identiske - kun at applikasjonslaget har blitt flyttet over til nettleseren.

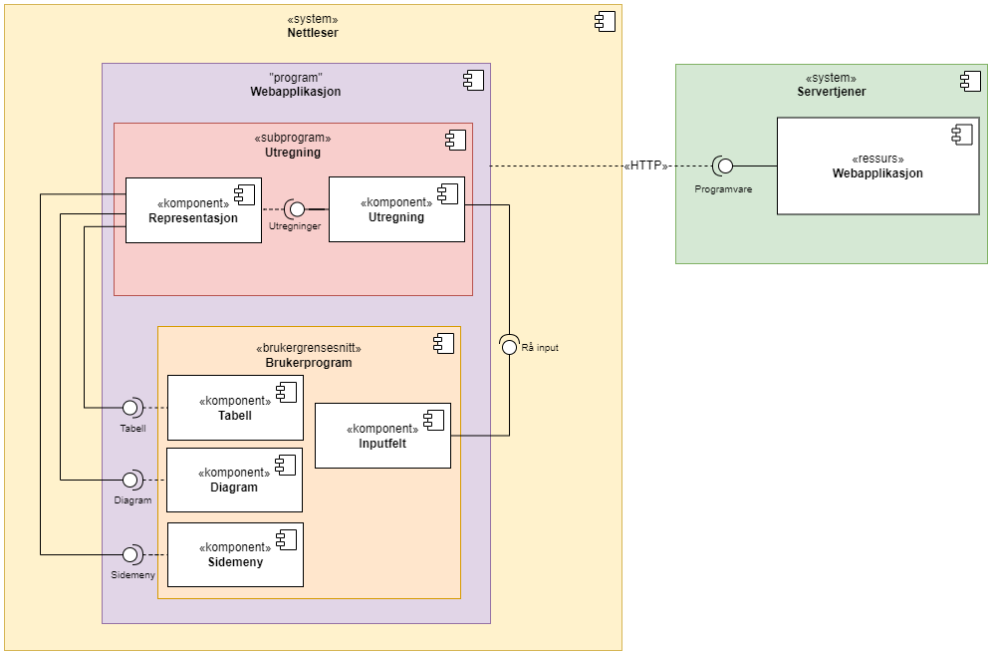
For noen år siden, ville en slik arkitektur ha vært vanskelig å utføre. Dette skyldes at det ville ha innebåret å oversette utregningskoden til JS - noe som kan være tidsintensiv, gi forskjellige resultater fra basiskoden[2], og muligens ha vært umulig på grunn av bruken av språkekslusive biblioteker. Men med ankomsten av WebAssembly, har denne barrier blitt brutt, og mulige funksjonaliteter som man kan utføre gjennom en webapplikasjon har økt drastisk. En arkitektur der beregninger utføres av et Wasm-basert rammeverk for Python, har blitt mulig, og er derfor et reelt alternativ for denne typen oppgaver.

Fordelene med en slik arkitektur er flere. Den viktigste er at alt flyttes til klienten. Serverens eneste oppgave blir å levere det statiske dokumentet med applikasjonen, noe som krever minimal ytelse fra serveren og dermed gjør det billig å opprettholde. Alt av utregninger vil avhenge på klienten sin enhet. Vi antar ikke at det skal være et problem for klientens enhet, da WebAssembly skal være rask og gi 'nesten-nativ' ytelse[3]. Med tanke på selve ytelsen til basisprogrammet, antar vi ikke at dette skal være et bekymringsområde for oss.

Komponentmodell

Vises på figur 4.6.

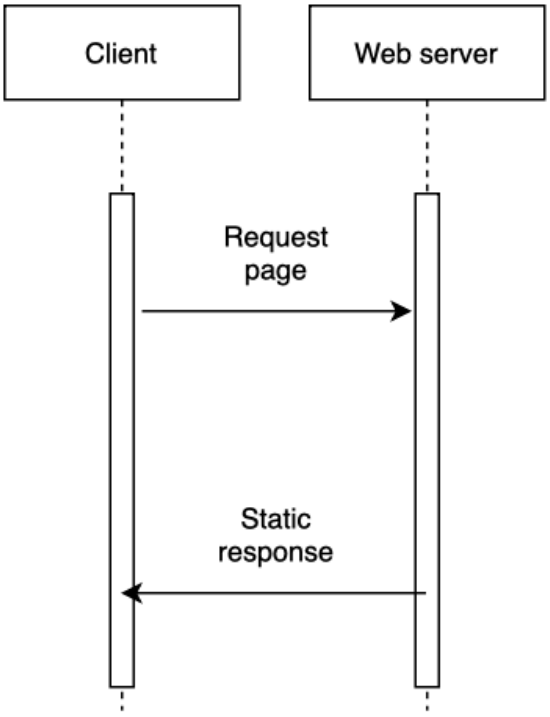
[1] https://no.wikipedia.org/wiki/Nettleserspill, besøkt 19.05.2024
[2] Med tanke på forskjellige språk sine typer; oversettelsen mellom dem.
[3] `https://developer.mozilla.org/en-US/docs/WebAssembly`, hentet 19.05.2024



Figur 4.6: Komponentdiagram for frontend-basert løsning.

Sekvensdiagram

Vises på figur 4.7.



Figur 4.7: Sekvensdiagram for frontend-basert løsning

Kapittel 5

Teknologier

5.1 Oppgavearkitekturer

Som diskutert tidligere, ønsker vi først å eksperimentere med to typer arkitekturer som er radikalt forskjellige fra hverandre, i håp på at vi kan finne den beste av dem gjennom testing og diskusjon senere gjennom utviklingsprosessen.

En av dem skal da være basert på 'alt-i-nettleser' arkitektruren, mens den andre skal være basert på et 'klient-tjener' forhold. For dem, har vi valgt følgende teknologier:

5.1.1 'Klient-tjener'- Flask

For den backend-baserte arkitekturen, gikk vi med rammeverket kalt Flask. Flask er et WSGI-basert rammeverk for å skape webapplikasjoner gjennom Python, utviklet av Armin Ronacher i 2004[1]. Senere ble det overlatt til Pallets i 2016 [37]. Rammeverket har åpen kildekode sammen med en BSD-3-Clause lisens på Github[2].

Vi valgte Flask over andre kjente rammeverk fordi vi hadde allerede jobbet med det tidligere i flere prosjekter gjennom våre grader, og vi syntes at det var godt å jobbe med. Rammeverket er også populært, enkelt tilgjengelig og godt dokumentert. Dette gjør det enklere for oppdragsgiver å oppdatere senere etter tidsrammen for prosjektet.

5.1.2 'Alt-i-nettleser' - PyScript

For den frontend-baserte arkitekturen, valgte vi det nye og moderne rammeverket kalt PyScript. PyScript ble skapt av 'PyScript Development Team' hos Anaconda, Inc.[3], og er et WebAssembly-basert rammeverk som gjør det mulig å kjøre Pyt-

[1] `https://flask.palletsprojects.com/en/2.0.x/`, hentet 21.05.2024
[2] https://github.com/pallets/flask, hentet 19.05.2024.
[3] https://pyscript.net/, hentet 19.05.2024

hon hos en klient sin nettleser. Rammeverket har åpen kildekode[4] med Apache 2.0-lisensen, og oppdateres ofte av flere utviklere med ønske om å forbedre rammeverket over tid.

PyScript i seg selv er ikke en Wasm-basert kompilator, men er heller et lag av abstraksjoner, bygget opp og API basert på Pyodide; ett rammeverk utviklet hos Mozilla med åpen kildekode som videre fungerer som en port av CPython til WebAssembly [5]

5.2 Klient-tjener frontend

5.2.1 React

React[6] er et lettvekts JavaScript bibliotek hvor hovedfokuset er å gi raske, skalerbare og enkle web applikasjoner. React ble utviklet av en liten gruppe utviklere hos Facebook for å løse utfordringene rundt å utvikle komplekse brukergrensesnitt med oppdatering av datasett [34].

I motsetning til de eksisterende MVC (Model-View-Controller)[38] rammeverkene som ofte brukes til webutvikling, så fokuserer React på 'view' delen av MVC. Det bruker modulære komponenter og virtuell DOM, som gjør at visningen kan endres individuelt kun for de komponentene som trenger å oppdateres. Med dette begrenses antallet DOM oppdateringer som trengs. Dette optimaliserer «rendering» og gjør at en React app oppfattes som rask og responsiv.

Som nevnt er React et lettvekts bibliotek, noe som betyr at det ikke inneholder alt av verktøy man gjerne forventer av et rammeverk for webutvikling[39]. Til gjengjeld kan eksisterende JavaScript biblioteker sømløst integreres. Et eksempel på dette er at React kan benytte seg av `Redux` dersom det trengs en mer kompleks håndtering av programtilstand i webapplikasjonen[40].

5.2.2 TypeScript

TypeScript er en utvidelse av JavaScript utviklet av Microsoft[7]. Det er designet slik at all kode skrevet i JavaScript kan benyttes med nøyaktig samme syntaks i TypeScript. TypeScript legger til en rekke funksjonaliteter til JavaScript for å forenkle utviklingen av større applikasjoner mer håndterbar.

Ett av disse funksjonalitetene er statisk sjekking av variabeltype. Dette lar utviklere sjekke typen til en variabel, parameter eller returverdi under utvikling. Dette gjør det betydelig lettere å fange opp feil før koden kjører og reduserer bugs og kjørtidsfeil.[41]

En annen element er et større fokus på objektorientert programmering gjennom bruk av klasser. Mens JS støtter en tidlig prototype av dette, så utvider Type-

[4] https://github.com/pyscript/pyscript, hentet 19.05.2024
[5] `https://pyodide.org/en/stable/project/about.html`, hentet 19.05.2024
[6] https://react.dev/, hentet 19.05.2024
[7] `https://www.typescriptlang.org/`, hentet 19.05.2024

Script dette til mer klassisk og kjent form. Dette tillater utviklere som kommer fra for eksempel Java eller C å benytte teknikker som klasser, interfaces og arv[42].

5.2.3 Visual Studio Code

Visual Studio Code (også kjent som VS Code) er et populært koderedigeringsprogram utviklet av Microsoft med støtte for Windows, macOS, Linux og nettlesere[8]. Programmet har støtte for utallige forskjellige kodespråk og har utviklingsverktøy som «intelligent code completion», «debugging», «syntax highlighting» og «code refactoring» innebygd[43].

I tillegg til dette, har VS Code et stort utvalg av forskjellige verktøy skapt både av Microsoft og av andre brukere, hvor brukeren kan laste ned ulike utvidelser for alle slags funksjonalitet. Dette gjør at utviklere kan tilpasse VS Code i større grader gjennom preferanser og funksjonalitet som vil hjelpe med det gjeldende prosjektet.

5.2.4 Plotly & Recharts

Plotly[9] og Recharts[10] er to ulike biblioteker for grafisk visualisering av data.

Plotly er et allsidig og sterkt bibliotek som legger til rette for å lage interaktive visualiseringer av data, som for eksempel grafer, av høy kvalitet. Det støtter mange ulike typer grafer og tilbyr en rekke innebygde funksjoner som hjelper med interaktiviteten og funksjonaliteten til grafen. Eksempler på dette er innebygd funksjonalitet for forstørrelse og panorering.

Plotly er originalt et JavaScript bibliotek for webutvikling, men støtter nå en rekke andre programmeringsspråk som Python[11] og MATLAB[12]. Plotly har støtte for både enkle og særdeles komplekse grafiske visualiseringer av data. Plotly er også et sterkt bibliotek med tanke på skalerbarhet, da det er bygd for å effektivt håndtere veldig store datasett.

Recharts er et bibliotek bygd på React komponenter og utnytter D3.js[13], et JavaScript bibliotek for å produsere dynamiske og interaktive datavisualiseringer i nettlesere. Dets komponenter er rene React komponenter som sømløst passer inn i rammeverkets økosystem. Dette betyr at det kan utnytte Reacts sine funksjonaliteter som `props` for tilpasning og `state` for dynamiske oppdateringer. Det er også designet for effektivitet innen rammeverkets virtuelle DOM, som gjør at det kan hurtig og effektivt håndtere oppdateringer og oppdatere 'rendering' hos webapplikasjonen.

[8] `https://code.visualstudio.com/`, hentet 19.05.2024
[9] `https://plotly.com/`, hentet 19.05.2024
[10] `https://recharts.org/en-US/`, hentet 19.05.2024
[11] `https://plotly.com/python/`, hentet 19.05.2024
[12] `https://plotly.com/matlab/`, hentet 19.05.2024
[13] `https://d3js.org/`, hentet 19.05.2024

Recharts har god støtte for enkle graftyper som linjegraf, stolpediagram med mer, men kommer gjerne for kort når det er krav om mer komplekse visualiseringer av data. Den har også en begrensning når det kommer til håndtering av veldig store datasett, da dette vil påvirke ytelsen i en større grad.

5.3 Klient-tjener backend

5.3.1 Python

Som ønsket av oppgaven, så skal Python brukes som programmeringsspråket for utregninger relevant til prosjektet. Språket er et populært høyt-nivå objekt-orientert språk [14] med åpen kildekode [15], kjent for dets enkle syntax og populære fagmiljø. Det har lenge vært et av de mest populærte språkene innenfor programmering (med noen undersøkelser som sier at det er det tredhe mest populære språket[16], hvor det har spesiell erkjennelse innenfor maskinlæring.

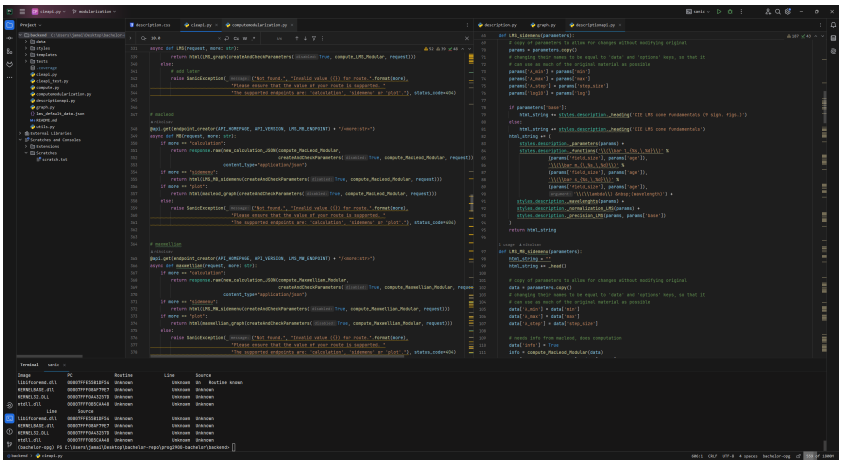
Den tilbyr mange funksjonaliteter, som dynamisk variabeltypesetting, enkel og rask debugging, kan kjøre uten kompilering, og er veldig universell blant forskjellige systemer. Den har også et veldig rikt økosystem av mange biblioteker og rammeverk for alle mulige oppgaver.

For dette prosjektet, fant vi følgende biblioteker relevant:

- **Pandas**[17]: Et populært bibliotek for manipulering og analyse av datastrukturer. Dets gode ytelse med diverse datastrukturer, sammen med fleksible og enkle metoder gjør det anerkjent i språkets økosystem. Biblioteket har åpen kildekode med BSD-3-Clause lisens[18].
- **NumPy**[19]: Et annet populært bibliotek for datastrukturer, kun at det fokuserer mer på multidimensjonelle `array` og matematikk. Biblioteket er også anerkjent godt i økosystemet, og er integrert med flere andre biblioteker tilgjengelig i språket. Denne har åpen kildekode med egen lisens[20]
- **Unittest & Pytest**: Dette er to populære testingbibliotek i applikasjonen, hvor Unittest er tilgjengelig i standardbiblioteket[21] mens PyTest[22] er enkel og separat med åpen kildekode og MIT-lisens[23].

[14] `https://www.python.org/doc/essays/blurb/`, hentet 20.05.2024
[15] `https://docs.python.org/3/license.html`, hentet 20.05.2024
[16] `https://survey.stackoverflow.co/2023/#technology`, hentet 20.05.2024
[17] `https://pandas.pydata.org/`, hentet 19.05.2024
[18] `https://github.com/pandas-dev/pandas`, hentet 19.05.2024
[19] `https://numpy.org/`, hentet 19.05.2024
[20] `https://github.com/numpy/numpy`, hentet 19.05.2024.
[21] `https://docs.python.org/3/library/unittest.html`, hentet 19.05.2024.
[22] `https://docs.pytest.org/en/8.2.x/`, hentet 19.05.2024.
[23] `https://github.com/pytest-dev/pytest`, hentet 19.05.2024

5.3.2 PyCharm



Figur 5.1: Skjermbilde av PyCharm som utviklingsmiljø.

For utvikling av backend, blir det brukt PyCharm[24] som hovedutviklingsmiljø. PyCharm er en høykvalitets IDE laget av JetBrains, et firma anerkjent for både utviklingen av flere sofistikerte IDE, i tillegg til å lage programmeringsspråket Kotlin for Androidenheter [44].

Miljøet tilbyr flere funksjonaliteter som vi mener gjør det overlegent over andre utviklingsmiljøer. I tillegg til at den tilbyr et enkel installering, et sofistikert debuggingsystem med brekkepunkter og omfattende deteksjon av programproblemer. Den har også innebygd støtte for flere biblioteker ved å tilby spesielle funksjonaliteter for dem.

5.3.3 Sanic

Et annet webapplikasjonrammeverk for Python (som skal tas opp senere i rapporten) heter Sanic[25]. Sanic er et ASGI-basert rammeverk, skapt spesifikt for hensikten om å 'være rask' gjennom bruk av spesiell asynkron kode i Python. Den var først laget i 2018 med åpen kildekode og MIT lisens[26]. Ved dets lansering, virket det rask som en stor inspirasjon til andre rammeverk for dets gode hastighet (som for eksempel FastAPI[27] som selv anerkjente rammeverket som en inspirasjonskilde[28]).

Rammeverket er kjent for å være lett, fleksibel og å ha et 'Flaskaktig syntaks'[29], samtidig som det har et stort økosystem av biblioteker dedikert til å tilsette flere

[24] `https://www.jetbrains.com/pycharm/`, hentet 19.05.2024
[25] `https://sanic.dev/en/`, hentet 19.05.2024.
[26] `https://github.com/sanic-org/sanic`, hentet 19.05.2024.
[27] `https://fastapi.tiangolo.com/`, hentet 19.05.2024
[28] `https://fastapi.tiangolo.com/alternatives/#sanic`, hentet 19.05.2024.
[29] `https://sanic.readthedocs.io/en/18.12.0/`, hentet 19.05.2024

funksjonaliteter til rammeverket. Selv som et eget rammeverk, tilbyr den funksjonaliteter spesifisert til å 'få jobben ferdig'; dette inkluderer god debugging, automatisk TLS for sikkerhet og mye mer. En annen god side ved Sanic er dets skalerbarhet, hvor det kan tillate flere arbeidsprosesser til å kjøre og tjenestegjøre for en webapplikasjon.

5.3.4 Docker

For utrullingen av prosjektet, planlegges det å bruke Docker[30]. Docker er en plattform som gjør det mulig å automatisere utrulling av applikasjoner som selvstendige konteinere [45]. Plattformene det består av har åpen kildekode med Apache-2.0 lisensen[31]

Som sagt, så benytter plattformen seg av konteinere, som er enkle og isolerte miljøer med alt av data nødvendig for at en gitt applikasjon skal kjøre. Disse konteinere er selvstendig og kan kjøres likt uavhengig av operativsystem, dette er en fordel som gjør Docker et godt valgt til portabilitet av programvare mellom systemer [**docker.docs**]. De kan også kjøres i plenum for å øke skalerbarhet for en tjeneste.

Vi ser at Docker inneholder en rekke viktige komponenter, som:

- **Docker Engine**: Komponenten som står for å bygge og kjøre konteinere.
- **Docker Image**: Et uforanderlig miljø som har alt man trenger for å kjøre en applikasjon[46].
- **Docker Kontainer**: Kjører instanser av en 'image' fortalt over.
- **Docker Compose**: Verktøy for å definere og kjøre applikasjoner bestående av flere konteinere ved hjelp av YAML-filer.

5.4 Annet

5.4.1 Maskinvare

For dette prosjektet, ble det brukt to typer av maskinvarer for ytelsestester og tjenelsegjørelse av applikasjonen. Deres spesifikasjoner er vist frem i tabell 5.1, og er:

1. Maskinvare 1: En Acer Swift SFX14-41G brukt av en student i gruppa for utvikling og kjøring av umiddelbare ytelsestester kun for utviklingsperioden.
2. Maskinvare 2: En dedikert virtuell maskin hos SkyHiGh[32] for siste ytelsestester diskutert ved kap. 8.1.2, i tillegg til videre tjenestegjørelse av applikasjonen.

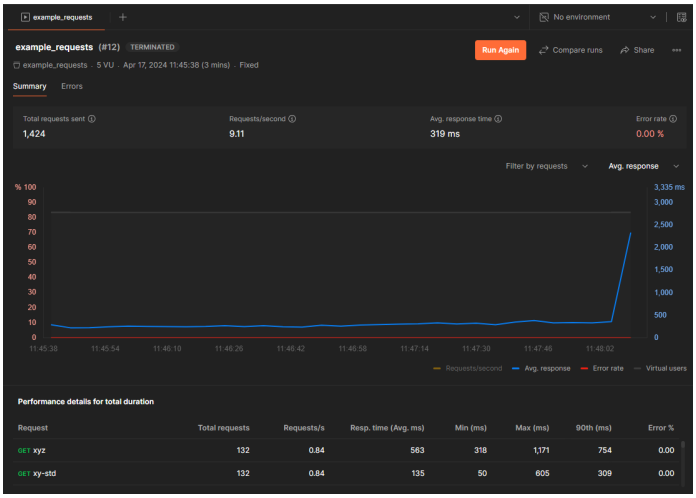
[30] `https://www.docker.com/`, hentet 19.05.2024.
[31] `https://github.com/docker`, hentet 19.05.2024.
[32] `https://www.ntnu.no/wiki/display/skyhigh`, besøkt 19.05.2024

	CPU	RAM	Minne
Maskinvare 1	8 kjerner	16 GB	50 GB+
Maskinvare 2	2 kjerner	2 GB	5-10 GB

Tabell 5.1: Maskinvarespesifikasjoner

5.4.2 Postman

Postman[33] er en plattform for avansert testing av API. Verktøyet tillater for flere metoder av testing og overvåking av tjenester, som elementære enhetstester av respons - til sofistikerte integrasjons- og ytelsestester. For dette prosjektet, ble det brukt til å teste responsinnhold fra backend, i tillegg til diverse ytelsestester utført over helheten av prosjektet.



Figur 5.2: Skjermbilde av Postman brukt i prosjektet for trafikktesting.

5.4.3 Overleaf

Overleaf[34] er en webapplikasjon for skriving av dokumenter gjennom LaTeX(LaTex). Applikasjonen tillater for flere medlemmer å jobbe sammen på et dokument, og har vært en standard for oss studenter i flere år. For dette prosjektet, brukes det til å skrive rapporten for oppgaven, samt andre tekster relevant for oppgaven.

5.4.4 Figma

Figma[35] er en webapplikasjon for designing og prototyping av diagrammer og wireframe. Programmet har blitt brukt av oss studenter jevnlig gjennom vår grad. I dette prosjektet, skal det brukes for å generere wireframe for prosjektet.

[33] `https://www.postman.com/`, hentet 19.05.2024
[34] `https://www.overleaf.com/`, hentet 19.05.2024
[35] `https://www.figma.com/`, hentet 19.05.2024.

5.4.5 Drawio

Drawio[36] er en webapplikasjon for designing av UML-diagrammer. Likt Figma, så er dette programmet også godt kjent med oss, og derfor bruker vi det med stor troverdighet. I dette prosjektet, var det brukt til å lage use-case diagram, domenemodell, og flere andre diagrammer for rapporten.

[36] `https://www.drawio.com/`, hentet 19.05.2024.

Kapittel 6

Prosess

Dette kapittelet skal gjennomgå utviklingsprosessen vi har hatt gjennom prosjektperioden. Fordi prosjektet vårt gjennomgikk flere drastiske endringer i løpet av våre sprinter, mener vi at det er best for rapportens skyld å først formidle prosjektet gjennom utviklingsprosessen med bruken av sprinter. Dette vil følges opp av et eventuelt kapittel som tar det endelige produktet fra alle sprinter, og går mer innom detalj på implementasjon.

6.1 Sprint 1-2

De første to sprintperioder gikk til valget vi måtte gjøre mellom våre arkitekturer; 'alt-i-nettleser' og 'klient-tjener'. Selv om begge er reelle måter å løse oppgaven på, kunne vi velge kun en. Derfor, bestemte vi oss at vi skulle finne ut hvilken var best for oppgaven med å lage prototyper som skulle demonstrere spesifikke funksjonaliteter.

Disse funksjonalitene ble bestemt til å være:

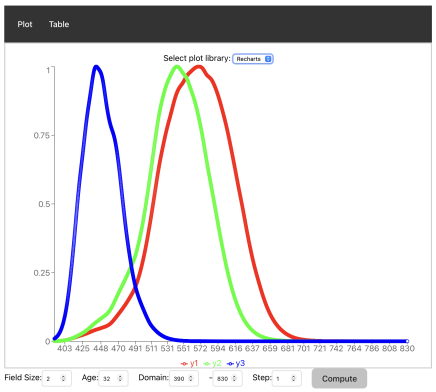
- Utføre utregning fra basisprogrammet.
- Vise frem et diagram som finnes i basisprogrammet.
- Få input fra bruker gjennom parametere i prototype.
- Vise frem og/eller ha en form av en tabell.

Etter deres utvikling, var planen å ta dem opp til diskusjon med å fokusere på deres fordeler og ulemper innenfor deres utviklingsprosess og prototypen selv. Kun etter det, mente vi at vi var klar til å gjøre et endelig valg.

6.1.1 Prototype for 'klient-tjener':

For utvikling av 'klient-tjener' prototypen, valgte vi å benytte Flask som vårt nettrammeverk slik tidligere skrevet. I tillegg til dette, bestemte vi oss å også bruke React, Vite og Typescript for å utvikle brukergrensesnittet. React gjør det mulig for oss å lage det dynamisk og responsiv gjennom Typescript, mens Vite sørger

for rask byggetid og bruk av HMR[1]. Med bruken av dette, utviklet vi en prototype med de nødvendige funksjonalitetene.



Figur 6.1: Diagram gjennom Recharts

Figur 6.2: Diagram gjennom Plotly

Prototypen ble utviklet med tanke på dets tidligere diskuterte arkitektur, hvor brukeren sender en forespørsel til serveren via et endepunkt, serveren behandler forespørselen og utfører nødvendige beregninger, før det sender resultatene tilbake til klienten. Dette ble gjort gjennom bruk av en forespørsel med metoden `POST`, og innmat av JSON for å håndtere innkommende data og input fra brukeren. Innmaten ble sendt som en `Map` av både `result` og `plot` verdier fra utregningene direkte.

Vi testet også to biblioteker for diagramvisualisering, som er da Plotly og Recharts slik tidligere introdusert. Hensikten var å finne hvilken var best for å visualisere diagrammet.

Under utviklingen av backend prototypen møtte vi også på en del utfordringer. Den største av disse var 'beregningsfeil' slik tidligere fortalt om i løpet av kapittel 2.2.5. Under testing oppdaget vi at ikke alle beregningene ga de forventede resultater, og at flere viste den klassiske karakteristikken av tall preget med feilen (slik vist i figur 6.3). I basisapplikasjonen var det implementert egendefinerte avrunding og 'kutte' funksjoner som sørget for at tallene ble korrekte, men allikevel med deres bruk, hadde vi fortsatt disse resultatene.

X	Y 1	Y 2	Y 3
390	0.000415003	0.00036834899999999997	0.00954729
391	0.0005026500000000001	0.000448015	0.0114794
392	0.000607367	0.0005439650000000001	0.013798600000000001

Figur 6.3: Tabell med beregningsfeil.

[1]https://webpack.js.org/concepts/hot-module-replacement/, hentet 19.05.2024

Det ble også utført ytelsestester av denne tjenesten for å evaluere ytelsen under moderat belastning med 5 virtuelle brukere i Postman, slik dokumentert i mer detalj hos vedlegg F. For hensikten av enklere lesing, har relevant informasjon også blitt tilsatt her:

	Verdi
Forespørsler per sekund	2.26
Gjennomsnittlig respons tid	1211 ms
Totalt antall forespørsler	421

Tabell 6.1: Ytelsestest av API

Vi ser at modellen bruker 1211 ms for å få svar fra en forespørsel, noe som kan virke ekstremt; men det må klargjøres at endepunktet som ble testet utregnet alle utregningsfunksjoner fra basisprogrammet. Med dette i tanke, er denne responstiden faktisk en fordel for prototypen - for det viser til at denne tjenesten bruker kun 1.2 s til å gi en klient utregningene (og kan reduseres gjennom modulisering av utregningene).

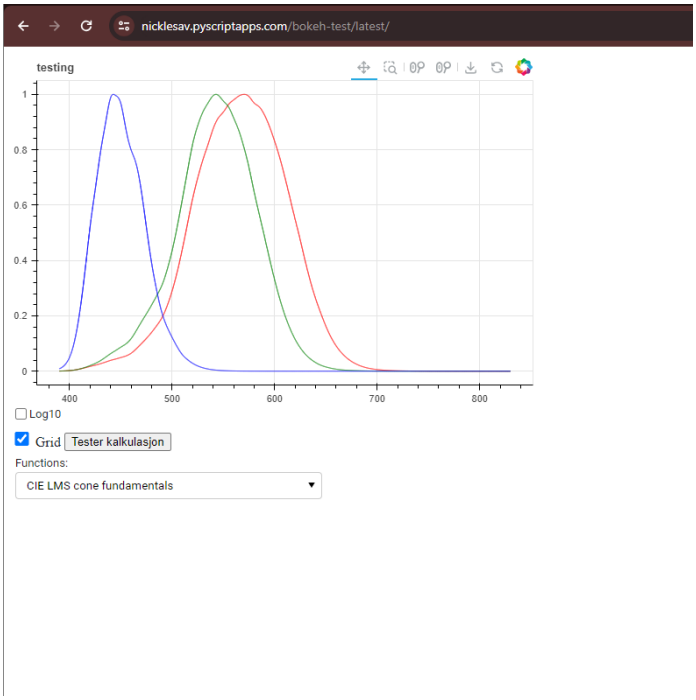
For å videre få innsikt i dette, brukte vi Postman igjen for å sende en enkel forespørsel for å se mer på den, hvor vi tok med viktige punkt i tabell 6.2.

	Verdi
Respons tid	883 ms
Nedlastet data	3.44 MB
Nedlastningshastighet	3.90 MB/s

Tabell 6.2: Enkel test av API

Responstiden uten stress var satt på 883 ms, et tall vi mener kan igjen reduseres gjennom modulisering. Vi ser også at den nedlastede dataen er høy - igjen en sideeffekt av at prototypen regner ut alle på en gang.

6.1.2 Prototype for 'alt-i-nettleser':



Figur 6.4: Skjermbilde av frontend-basert prototype med flere funksjonaliteter.

Utviklingen av 'alt-i-nettleser' prototypen foregikk gjennom en dedikert utviklingstjeneste av PyScript selv, hvor registrerte brukere fikk tillatelse til å enklere lage demoapplikasjoner[2]. Selv om det er mulig å lage en lokal prototype, så ble heller dette valgt for fordelen av å teste om prototypen fungerte andre enheter. Det var jo teknisk sett 'nytt', og kunne ha problemer med forskjellige nettlesere.

Prototypen selv (som vist på figur 6.4) har funksjonalitene for ett funksjonelt diagram, og brukerinteraksjon gjennom avmerkningsbokser og nedtrekksmeny for flere funksjoner. En tabell ble ekskludert fra utviklingen, for vi fant raskt ut at dette rammeverket hadde en god del ulemper, og kun noen få fordeler.

Det første problemet vi fant, var en direkte forskjell i evner mellom den tradisjonelle JS, og alle Wasm-baserte rammeverk. Mens JavaScript hadde direkte evne til å manipulere DOM (og derfor direkte generere dynamisk innhold), så har det ikke blitt implementert innenfor WebAssembly ennå, med fremtidige planer til det[3]. Selv om dette ble ikke ett direkte problem for utregningskoden nødvendig for figur og tabell for Python kunne fremdeles kjøre, så ble det heller to problemer med tanke på kompatibilitet:

[2]https://pyscript.com/, hentet 19.05.2024
[3]https://developer.mozilla.org/en-US/docs/WebAssembly/Concepts, hentet 19.05.2024

1. For å kunne vise resultatene det utregner, må Python kommunisere gjennom JS som proxy til dokumentet. Selv om rammeverket tilbyr en FFI med tilgang til standard webAPI som DOM[4], så kommer problemet inn når man må kombinere data mellom begge språkene. Et eksempel er at unike typer kreve rundveier for å være kompatibel, som bruk av JSON for å passere på informasjon mellom språkene. Det er også spesielt vanskelig å finne ut hvor det oppstår programfeil.
2. I det tilfellet hvor denne arkitekturen velges, så er det sannsynlig at den må gjennomgå en radikal endring i fremtiden. I det tilfelle hvor Wasm oppdateres til å bruke DOM innebygd, da finnes det grunnlag for å måtte skifte koden til å tilpasse dette. Dette kan være komplisert for oppdragsgiveren i fremtiden, noe vi ønsker å unngå helst.

Et annet problem ble ytelsen. Fordi arkitekturens ytelse avhengte av enheten til brukeren, så mente vi først at dette var noe som avhengte av brukerne selv. Så lenge det falt innen våre forventninger vi hadde for våre maskiner, så ville vi ha ansett det som godt nok til å fortsette. Denne tanken var dessverre endret når vi så at prototypen selv tok ca. 20 sekund på å laste inn i nettleseren.

Dette var klart et bekymringsområde for oss nå, så vi sjekket både hastighet og minnebruk av vår prototype og to andre demoapplikasjoner (tre-på-rad[5], K-means[6]) for PyScript, hvor vi fant følgende resultater:

	Prototype	Tre-på-rad	K-means
Gjennomsnitt	582 MB	127 MB	1012 MB
Median	517 MB	110 MB	1200 MB

Tabell 6.3: PyScript - Minnebruk

	Prototype	Tre-på-rad	K-means
Gjennomsnitt	21.946s	3.094s	20.947s
Median	22.71s	3.06s	20.79s

Tabell 6.4: PyScript - Lastehastighet

Testene sjekker for minnebruk og hvor lang tid det tar for nettsiden å laste inn. Den sjekker også hvor mye av dette er rammeverket selv, med å ta i faktor en demo uten mye kode ([5]), og en annen med flere biblioteker og utregninger ([6]). Minnebruket ble målt gjennom funksjonalitet i Google Chrome, og lastehatighet med en stoppeklokke. Det ble utført 10 tester av hver kandidat, hvor gjennomsnitt og median ble tatt.

[4] `https://docs.pyscript.net/2024.5.2/user-guide/dom/#FFI`, hentet 19.05.2024
[5] `https://examples.pyscriptapps.com/tic-tac-toe/latest/`, hentet 21.05.2024
[6] `https://examples.pyscriptapps.com/kmeans-in-panel/latest/`, hentet 21.05.2024

Gjennom bruk av disse tester, bekrefter vi våre bekymringer - og ser at PyScript *kan* være ressursintensiv for klienten, hvor det avhenger av typen av applikasjon. For enkle applikasjoner brukes det relativt vanlig tid og minne sammenlignet med vanlige nettsider - men for avanserte situasjoner (som vårt prosjekt), så drar både minnebruk og lastehatighet høyt opp - med nesten 600 MB og 20 sekund for å laste inn. Dette gikk langt over våre forventninger av en slik arkitektur, og vil virke negativt mot det senere i vår beslutning.

Det er også viktig å nevne at det ble funnet at dette rammeverket hadde problemer med noen nettlesere (som Safari[7]), noe som kan peke til at denne løsningen støttes ikke av alle aktuelle nettlesere - en annen negativ ulempe til dette.

Til tross av disse ulemper, fantes det fortsatt en rekke fordeler med denne prototypen:

- **Klient-basert Python:** Som tidligere skrevet, så er fortsatt en stor fordel av en 'alt-i-nettleser' løsning at det kjøres hos klienten - til tross for våre resultater diskutert ovenfor. Selv om det tar lang tid å laste inn, så kan det være fordelsaktig dersom 'klient-tjener' modellen har også høy responstid for forespørsler.
- **Pythons økosystem i nettleser:** Fordi vi kan kjøre Python i nettleseren, betyr det at vi kan kjøre alle av dets biblioteker og rammeverk direkte hos en klient - noe som er en stor fordel når det kan kombineres med JavaScript i nettleseren for å oppnå utrolig mye funksjonalitet.

6.1.3 Diskusjon & utvalg av arkitektur

La oss starte diskusjonen med å først se på tre punkter av relevans for diskusjonen; testene brukt, hvordan utviklingen føltes, og fordeler/ulemper mellom dem:

1. **Testene:** Vi ser at testene indikerer høyere tilgjengelighet og mindre ressursbruk hos 'klient-tjener' enn 'alt-i-nettleser'. Mens en av dem bruker omlag 20s på å laste inn og samtidig bruker litt over 500 MB av midlertidig minne, så ser vi at den andre prototypen utgir resultatene relativt øyeblikkelig og med mindre data brukt - sammen med evnen at dette kan optimaliseres gjennom modulisering. Med dette i tanke, mente vi at 'klient-tjener' vant enkelt når det gjaldt testene.
2. **Utvikling:** Vi mener også at det føltes tryggere å programmere gjennom 'klient-tjener' prototypen enn den andre. Vi erkjenner at 'alt-i-nettleser' er fortsatt 'ny', men det føltes rat å programmere både JS og Python gjennom bibliotekene tilgjengelig i rammeverket. Det var også vanskeligere med tanke på programfeil. Imens hos den kjente 'klient-tjener' prototypen med Flask og React, følte vi oss langt mer sikker med rammeverkets anerkjennelser og innebygd støtte.

[7] `https://www.apple.com/safari/`, hentet 21.05.2024

3. **Fordelene og ulempene:** Med å sammenligne dem, ser vi allerede en stor kontrast. 'Alt-i-nettleser' har flere ulemper utenom de diskutert tidligere, som tydeligvis ustøttet bruk i noen nettlesere. Hos vår andre arkitektur, ser vi kun beregningsfeilen, men dette er noe vi antar kan fikses raskt når det undersøkes nærmere på. Med dette, vant 'klient-tjener' igjen.
   Det er også mulig å si at 'klient-tjener' som en arkitektur har bedre bakgrunn enn den andre arkitekturen. Selv om 'alt-i-nettleser' er laget av svært anerkjente utviklere, så er det fortsatt ikke sikkert at Wasm blir den nye normen i den snare fremtiden. Vi vet ikke om klientbaserte webapplikasjoner gjennom dette blir noe stort i fremtiden. Dette er ikke et problem for 'klient-tjener', for å bruke rammeverk som Flask og React har dannet flere tilgjengelige nettsider på internettet allerede. De er svært populære, og er essensielt normen for flere utviklere. Vi kan si oss mer sikre for oppdragsgiver å bruke denne over noe som er mer 'eksperimentelt' som 'alt-i-nettleser'.

En av de største fordelene med 'klient-tjener' prototypen var at beregningene ble utført på serveren. Med dette unngår vi at brukerens datamaskin blir en flaskehals for ytelse. Dette betydde at vi kunne håndtere tyngre beregninger og mer komplekse oppgaver uten å bekymre oss for brukerens maskinvare. Dette ville være spesielt viktig for vårt prosjekt, som krevde omfattende beregninger.

Vi mener allikevel at det finnes noe potensiale for 'alt-i-nettleser' prototypen om fremtiden. Som sagt, så er PyScript og WebAssembly relativt nye teknologier som fortsatt venter på komplette implementasjoner hos nettlesere. Når dette er komplett og fullført med flere optimaliseringer som reduserer innlastingstiden og minnebruk, antar vi at rammeverket kan da være mer ideell for oppgaven.

Med alt dette i tanke og de andre ulempene diskutert, måtte vi si at 'klient-tjener' prototypen var derfor den beste arkitekturen for oss å velge, og med dette, fant vi vår arkitektur for oppgaven.

6.2 Sprint 3-5

Dette delkapittelet representerer de neste tre sprintperioder etter vår prototypefase av mulige løsninger - hvor vi videreutvikler på 'klient-tjener' prototypen vi hadde laget tidligere for å virkelig begynne prosjektets utvikling til resultatmål.

6.2.1 Utregningsfunskjoner

Det første steget var implementasjon av moduliserte utregningsfunksjoner basert på de allerede eksisterende hos `compute.py`[8]. I deres opprinnelige former hos basisprogrammet (og slik prototypen hadde gjort det), så hentes alt av relevante utregninger fra den kollektive `compute_tabulated()`, ansvarlig for å utføre alle

[8] `https://github.com/ifarup/ciefunctions/blob/master/tc1_97/compute.py`, hentet 21.05.2024

	Modulariserte funksjoner			Direkte fra compute.py		
	P1	P2	P3	P1	P2	P3
LMS	29 ms	27 ms	24 ms	27 ms	29 ms	31 ms
MacLeod	49 ms	39 ms	41 ms	226 ms	265 ms	232 ms
Maxwellian	31 ms	36 ms	30 ms	214 ms	216 ms	223 ms
XYZ	130 ms	374 ms	379 ms	180 ms	489 ms	487 ms

Tabell 6.5: Vurdering av modulariserte utregningsfunksjoner mot de tilgjengelig fra compute.py

`compute_x` tilbehørene til deres korresponderende fargematchfunksjoner. Vi mente at dette var en dårlig løsning for oss, for utregningen av alle ville ha kun gjort applikasjonen saktere. Derfor, en moduliseringen av funksjonene ville ha gjort den raskere. Det er også ikke lov for oss å lagre tidligere resultater (med tanke på REST sitt prinsipp om statsløshet), så vi kunne enten bruke `compute_x` funksjonene individuelt, eller lage våre egne moduliserte.

Vi mente at det fantes noe god praksis i å lage våre egne moduliserte versjoner med tanke på ytelsen. Noen av de opprinnelige funksjonene vil ta mer tid for å produsere utregninger fordi de måtte utføre det både for en `result` og en `plot`. Men vår løsning på dette punktet kunne kun gi en av dem om gangen, så det var lite mening i å regne begge ut. For å teste om denne tanken hadde mening, laget vi moduliserte versjoner av et par fra `compute.py`. De ble sammenlignet med deres opprinnelige motparter med tanke på hastighet.

Dette ble gjort for endepunktene tilbehørene LMS, Macleod-Boynton, Maxwellian og XYZ, hvor det ble testet med tre parametere for hver funksjon. Resultatene av dette vises frem i tabell 6.5.

Vi ser i tabellen at for det spilte ingen rolle for LMS, og at tiden forble den samme. Men for MacLeod og Maxwellian spesielt, undersøkte vi en stor reduksjon i responstid. Det fantes også en merkbar reduksjon hos XYZ, en utregningsfunksjon kjent til oppdragsgiver å ha tatt relativ 'lang' tid. Dette var nok for oss å bestemme at vi ønsket moduliserte funksjoner, så en del av sprinten gikk til videreutviklingen av dem.

For å gjøre det enkelt å finne feil i våre moduliserte utregninger og/eller programfeil, laget vi også en improvisert plattform for testing i form av et endepunkt (slik vist frem i figur (6.5)). Dette ble utført med å bruke basisprogrammet til å eksportere forskjellige utregninger til `csv` filer - hvor filene ble etterpå sammenlignet til utregningene vårt program laget med hjelp av Pandas igjen.

```
{
  "LMS": true,
  "LMS-LOG10": true,
  "LMS-BASE": true,
  "LMS-PLOT": true,
  "MB": false,
  "MB-PLOT": false,
  "MW": true,
  "MW-PLOT": false
}
```


Figur 6.5: Skjermbilde av resultat fra plattformen for testing.

6.2.2 Utregningspresisjon

Som skrevet tidligere, var et av problemene som fantes i prototypen feil med beregningspresisjon. Fordi vår målgruppe er forskere som krever nøyaktige tall fra utregninger, så er dette et problem som må takles seriøst og riktig. Derfor, ble det utført en stor innsats til dette over perioden.

Vi visste at det var et problem med selve eksportingen av utregningene. Fordi vi brukte de opprinnelige funksjonene i prototypen, kunne dette umuligvis ha vært forårsaket av egen kode. Derfor, så måtte det ha vært noe i hvordan den blir til JSON som forårsaker dette til tallene. Med dette, var det vurdert flere løsninger, men ingen av dem virket ideell for oss:

- Direkte bruk av json[9] fra standardbiblioteket ga programfeil, for den hadde ikke evnen til å eksportere en `numpy.ndarray`[10] (multidimensjonell array) til JSON.
- En egen JSON-enkoder var implementert og testet, men presisjonsproblemet fortsatte, for kodingen skrev tallene som direkte som de fantes i programmet.
- Å lagre tallene i en mini database som SQLite[11] momentant var en vurdert ide, men det var dømt for ressursintensiv å lage for denne hensikt.

Kun etter at vi fant Pandas, fant vi en mulig løsning for dette problemet. Vi kunne først konvertere utregningene om til en dedikert `pandas.DataFrame` type tilgjengelig i biblioteket. Gjennom datastrukturen, fikk vi tilgang til mange sterke metoder. En av dem var `.to_json()`[12] hvor spesifikke verdier av parameteren `double_precision` forandret på mengden av desimaltall hos tallene i utregningene. Med denne metoden, la vi merke til at flere tall ble uttrykt riktigere i den resulterende JSON innmaten hos responsen.

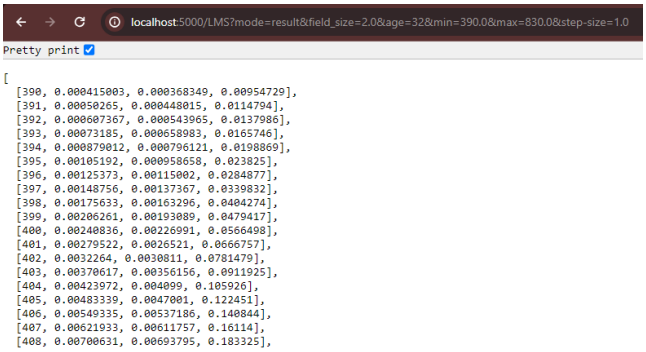
[9] `https://docs.python.org/3/library/json.html`, hentet 20.05.2024

[10] `https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html`, hentet 20.05.2024

[11] `https://docs.python.org/3/library/sqlite3.html`, hentet 20.05.2024

[12] `https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_json.html`, hentet 20.04.2024

Denne løsningen hadde initielt kun en dårlig side, og det var at vi måtte forlate eksporteringen med `dict`. Pandas støtter kun spesifikk eksportering, og kan ikke utføre metoden på en `dict` med flere datastrukturer i seg. Datastrukturen vi hadde fra prototypen måtte derfor forlates for denne nye metoden. Som et følge, måtte en eventuell bruker ha sendt flere forespørsler for å få all utregningsinformasjon, noe vi ser på som et negativt problem.



Figur 6.6: Skjermbilde av utregningsresultater etter Pandas løsning.

Allikevel, godtok vi risikoen og mente at dette måtte gjøres for å få de riktig tallene. Dette var inntil vi fant ut at det allikevel fantes problemer med denne metoden. Fordi metoden kun øker mengden av desimaltall (og indirekte presisjonen av et tall) så betyr det at tall med mange desimaler ble påvirket positivt, men tall med mange signifikante sifre og heltall ble derimot påvirket negativt.

Alle tall med en mengde av signifikante sifre som var over terskelen av desimaltall tillatt i en `pandas.DataFrame` eksportering ble rundet av, og derfor ikke utlevert i riktig form. I tillegg, enkle heltall i utregningene begynte å selv få de samme presisjonsproblemene man finner i utregning med desimaltall, noe vi fant ut oppstod når parameteren var satt for høyt. Begge av disse er alvorlige problemer som måtte tas opp videre i utviklingen.

Det ble også funnet et seriøst problem med testplattformen vi hadde brukt inntil nå for å forsikre oss at utregninger var riktige. Fordi `pandas.DataFrame` kan kun støtte en viss mengde av desimaler slik fortalt over, så ble det snart klart at den kunne ikke muligens ha de riktige tallene for de utregningene. Vi fant flere problem når vi undersøkte dette videre. Med bruk av debuggingsverktøy i PyCharm, fant vi dessverre ut at selve innlastingen av filene for testing var preget av kalkulasjonsfeil. Dette betydde at vi måtte også revurdere og restrukturere dette i fremtiden.

6.2.3 URL & parametere

En annen ting utviklet i denne perioden, var skiftet fra å bruke `POST` forespørsler med JSON innmat, til å heller bruke URL med betydningsfulle søkestrenger.

Dersom vi skulle lage en RESTful tjeneste, var det viktig å følge prinsippene til det.

Vår prototype følgte ikke det sjette prinsippet; dersom vi tenker på alle utregninger som unike ressurser, så ville deres fellesbruk av kun en URL ha brutt prinsippet. I tillegg, så er bruken av `POST` forespørsler for 'å hente informasjon' ikke en norm [13], for det er designert at den metoden er spesifikt for når brukere skal tilsette informasjon til en tjeneste med hensikt på å oppdatere den.

```
.../api/v1/funksjon?parameter1=x&parameter2=x&...
```

Figur 6.7: Struktur av URL ved første sprint.

Derfor, forandret vi først tjenesten til å heller kun ta `GET` forespørsler, for det var den riktige metoden spesifisert i HTTP for å hente informasjon[14]. For å passere videre informasjon om brukerparametere, valgte vi å gå for søkestrengene som en del av URL for å følge det sjette prinsippet, og gi hver unike beregningsressurs en unik URL. Vi endte opp eventuelt med et system som likner det funnet i figur 6.7, og var fornøyde med dette systemet for nå.

6.2.4 Stil & utforming

For frontend, ble fokusert en stor del på av perioden på stil og utforming for å sikre at applikasjon ikke bare fungerte som den skulle, men også at den var estetisk og brukervennlig.

Sammen med bruken av React og Typescript, brukte vi også og for å designe komponentene både visuelt estetisk og visuelt strukturelt for å holde oss innenfor kravene vi hadde laget tidligere i prosjektet.

For å gjøre dette, eksperimenterte vi med flere teknologier for å både plassere komponentene på riktige steder hos nettsiden - samtidig som at det skulle være enkelt å oppdatere for fremtiden. Derfor, gikk en stor del av sprinten til dette - hvor gjennom flere iterasjoner, landet vi med `Flex`[15] hos CSS. Vi hadde vurdert og testet `Grid`[16] og Bootstrap[17], men `Flex` ble utvalgt fordi det var enkelt å sette opp og bruke når man ble mer vant til det.

6.2.5 Modulisering

En annen stor del av denne sprinten ble brukt på å dele opp koden hos frontend inni moduler. Ved å modulisere prosjektet, kunne vi gjøre koden mer organisert

[13] https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST, hentet 19.05.2024

[14] https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET, hentet 19.05.2024

[15] `https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Flexbox`, hentet 20.05.2024

[16] `https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Grids`, hentet 20.05.2024

[17] `https://getbootstrap.com/`, hentet 20.05.2024

og lettere å vedlikeholde for fremtidige utviklere av dette prosjektet. Å modulisere koden ville også ha skapt et godt grunnlag for resten av applikasjonen. Dette var en kontinuerlig prosess som vedvarte gjennom hele utviklingsfasen av brukergrensesnittet, men som vi startet med allerede nå.

6.2.6 Konstruksjon av URL i frontend

Etter som tjeneren ble mer og mer RESTful, måtte vi utvikle et nytt og bedre system for å kommunisere med den. Dette ble gjort med et samarbeid mellom en `StringBuilder`, nedtrekksmeny og brukerspesifiserte parametre. Når brukeren velger en ny funksjon i nedtrekksmenyen, vil `StringBuilder` hente informasjonen sammen med eventuelle parametere til å bygge en ny URL for å passe det nye systemet.

6.3 Sprint 6

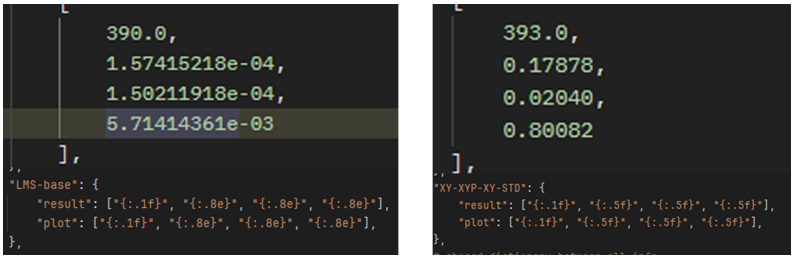
6.3.1 Stringformatering for JSON

For backend i denne sprinten, ble det ettersøkt videre hvilke metoder kunne fikse presisjonsproblemet fra den forrige sprinten. Selv om det var funksjonelt til en viss grad, så var det ikke tillatt av oss å la det være slik, spesielt med tanke på at vår målgruppe er forskere som trenger presise tall. Derfor, fortsatte søket inntil vi så nærmere på problemfeltets teori, og fant et interessant funn for prosjektet.

Denne formen av feil er overalt hos datamaskiner som tar i bruk desimaltall - og i felt hvor det er viktig å ha de riktige tallene (som økonomi), så har det blitt utviklet en rekke triks mot det som vi hadde ikke sett nærmere på tidligere. Et av dem er å uttrykke tallene som `string` slik vi hadde forsøkt tidligere, men i tillegg også spesifisere presisjon gjennom formatering. Et relevant eksempel er PayPal sin API[18]. Vi mente først at dette var en blindvei, for vi hadde ingen slike formater tilgjengelig direkte. Å forsøkte å gjette de var heller ikke et godt tegn. Det var kun når vi så nærmere på koden til basisprogrammet, at vi fant at dette var eksakt hvordan de opprinnelige utviklerne hadde taklet oppgaven, sammen med spesifikke formater oppstilt i koden[19]. Kun ved dette funnet, fant vi ut at dette var løsningen son både kunne brukes for å oppnå riktige tall, men også måtte for å oppnå komplett konsistenthet med basisprogrammet.

[18] `https://developer.paypal.com/docs/api/payments/v2/`, legg merke til `currency|medlemmet`, hentet 20.05.2024

[19] `https://github.com/ifarup/ciefunctions/blob/master/tc1_97/table.py#L155`, eksempel her, hentet 21.05.2024.



Figur 6.8: Eksempler av utregninger fra backend sammen med deres formateringsstruktur under.
I det første bildet, er det deklarert bruken av standardform med de første 8 desimaltall. I den andre, er det deklarert bruken av kun de første 5 desimaltall.

Det var opprinnelig forsøkt å se etter en tilgjengelig enkoder som tillate spesifikk formatering. Dette søket kom med ingen resultat, for det var et nisje ønske og ble ikke funnet. Dette preget oss til å igjen ta opp ideen om en egen JSON-enkoder, kun med evnen til å bruke en viss format når ønsket. Denne teknikken endte opp med å fungere så godt at den er fortsatt implementert i programmet. Sammen med dette, tillot dette oss å ekspandere ressursene vi utga fra kun en `numpy.ndarray`, tilbake til `dict` slik vi hadde det i prototypen. Dette er fordi vi kunne videre utvikle på enkoderen til å inkludere dette enkelt. Gjennom dette, var presisjonsproblemet løst på en måte som indirekte gjenbruker og oppnår konsistenthet med basisprogrammet.

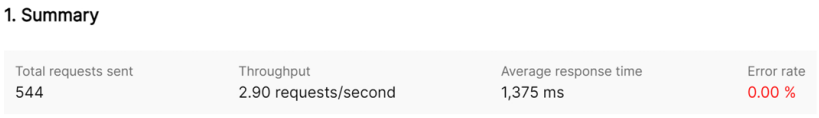
6.3.2 Vurdering & skifte av rammeverk

Etter at vi fikk de riktige resultatene hos utregningene i kapitlet over, antok vi at vi hadde kommet til et punkt hvor backend var nesten helt fullført for prosjektet. Derifra, mente vi at det eneste som gjenstod var ordentlig implementasjon med det for frontend (og eventuelle endringer det forårsaker). Med basis i denne antagelsen, ble det utviklet tester for å teste applikasjonen; først enkle enhetstester for kodekvalitet, men deretter stresstesting for å teste dets skalerbarhet. Dette var hvor vi innså at vi hadde gjort en kritisk feil i utviklingsprosessen.

Ett av våre krav var at applikasjonen måtte være asynkron - men fordi Flask er et WSGI, støttet det ikke innebygd asynkronisme. Dette kravet kunne ha blitt argumentert mot dersom applikasjonen oppfylte kravet om '*maksimalt 1000 ms responstid*' i stresstestet. Om programmet hadde klart å passere kravet for stresstesten, kunne vi ha argumentert at det fantes ikke lenger noe grunnlag for optimaliseringer som det. Dette betydde at vi kunne se vekk fra asynkronismen, men vi fant snart at det var nødvendig allikevel, som vist i resultatene hos figur 6.9.

I løpet av testen, fant vi ut at den gjennomsnittlige responstiden var 1.4 sekund - hvor noen utregningsfunksjoner[20] hadde gjennomsnittlig responstid på 3.5—4.3 sekund. Grunnen til dette var klar når man tar i faktor at en stresstest sender ut

[20] Spesifikt hos endepunkter /xyz og /xy.

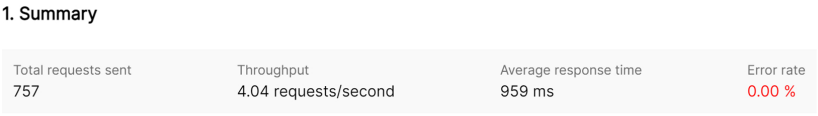


Figur 6.9: Resultater fra stresstest hos vanlig Flask rammeverk.

flere forespørsler på samme tid: Flask sine synkrone evner kan kun takle en forespørsel om gangen, så flere ender opp med å vente i en kø. Selv om dette var en stresstest med større mengder forespørsler enn det som er forventet, mente vi at dette var ikke akseptabelt.

For denne grunn, gikk resten av sprinten til å undersøke mulige løsninger for å gjøre rammeverket raskere. Vi mente at det var best å se på andre teknologier og utvikle ufullstendige demovarer av dem, kun med utregningene vi hadde nå. Gjennom det, kunne vi teste dem på lik grunnlag som vanlig Flask, finne ut hvilken ga best ytelse, og fortsette med det. Vi valgte tre mulige løsninger, og undersøkte dem videre:

Flask Aysnc



Figur 6.10: Stresstestresultater for flask async.

Problemet vi diskuterte over, var at ventetiden var for lang og at rammeverket tok kun vare av en forespørsel om gangen. Selv om rammeverket kan ikke endres på hvordan det takler forespørsler (fordi det er ikke en del av WSGI sin struktur, og alle forespørsler vil ta en lik mengde tid), noe som *kan* endres på er den direkte ventetiden applikasjonen bruker. Med å ta vare av en annen forespørsel mens systemet venter på svar fra seg selv om utregningene, så er det mulig å minimalisere ventetiden. Dette er prinsippet bak `flask async`[21]. Med å implementere `async` og `await` inni rammeverket, åpner vi opp til at den kan takle flere forespørsler på samme tid gjennom metoden over.

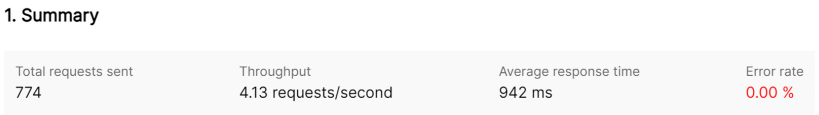
Gjennom dette, kan vi bekrefte at ventetiden var hovedproblemet vi hadde hos det opprinnelige rammeverket gjennom stresstester utført her (vist på figur 6.10). På testene, ser vi en umiddelbar reduksjon i responstid ned til 959 ms, sammen med en 139.3% raskere prosessering av forespørsler i sekundet. Dette er

[21] `https://docs.python.org/3/library/asyncio-task.html`, hentet 20.05.2024

i tillegg til at det var enkelt å implementere, og har sikkert videre potensiale for mer optimalisering gjennom bruk av flere `await` for spesielt intensiv kode.

Allikevel, er det viktig å peke ut at de samme utregningsfunksjonene som tok lang tid hos det opprinnelige rammeverket, tok allikevel over 1000 ms her også. De hadde gjennomsnittlig responstid innenfor 2.4 — 2.9 sekund. Dette var ikke kravsoppfyllende, men allikevel en stor forbedring og god start.

Flask med Gevent



Figur 6.11: Stresstestresultatet for Gevent.

En annen løsning innenfor Flask for asynkron takling av oppgaver, er bruken av biblioteket Gevent[22], med åpen kildekode og MIT lisense [23] . Dette biblioteket kombinerer bruken av korutiner sammen med spesielle 'grønne tråder' fra Greenlet[24] for å oppnå asynkron takling av forespørsler - lik hvordan `async` og `await` takler det oppe. Fordelen denne har over dem, er at det kreves *enda* mindre kode å implementere dette inni en allerede eksisterende Flask applikasjon.

Allikevel, med basis på figur 6.11, ser vi at ventetiden forblir hovedsakelig den samme. Dette forårsakes av ingen stor endring i rammeverket fra den tidligere sett `async` løsningen, i og med at begge er kun veier man kan takle ventetiden mellom utregninger på. I tillegg, ser vi en liten minimal reduksjon i responstid for de utfordrende utregningsfunksjonene, hvor deres responstid tok nå 2.2 — 2.8 sekund i gjennomsnittet.

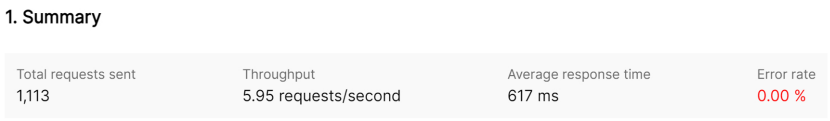
Sanic

Vi ser i de to tidligere løsninger at de *hjelper*, men ikke så mye. Dette er fordi uansett hva man gjør, så er Flask fortsatt ett WSGI. Dets hele struktur og basis er spesifikt for synkrone forespørsler. Om vi skal takle asynkrone forespørsler, er det derfor naturlig å velge et asynkront rammeverk.

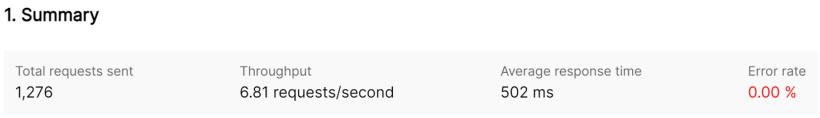
Det finnes et stort utvalg av disse (som FastAPI og nyere versjoner av Django), men gjennom en anbefaling fra en venn og videre undersøkelse i det, ble rammeverket kalt Sanic valgt. Vi ønsket ikke å undersøke hvert eneste slikt rammeverk, så vi valgte dette som vår eneste representant av ASGI-slaget. Rammeverket ble også valgt fordi den hadde lik syntaks som tjenesten vår hadde allerede, så det var enkelt å implementere.

[22] `https://flask.palletsprojects.com/en/3.0.x/deploying/gevent/`, hentet 20.05.2024
[23] `https://github.com/gevent/gevent`, hentet 20.05.2024
[24] `https://greenlet.readthedocs.io/en/latest/`, hentet 20.05.2024

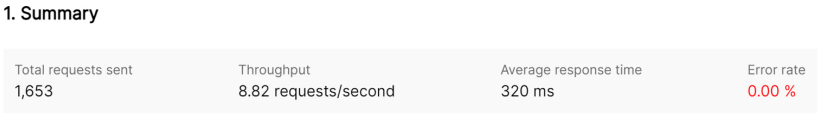
For testene, tok vi bruk av dets evne til å øke skalerbarhet, og kjørte tre tester med forskjellige variasjoner i arbeidsprosesser. Ett av testene hadde kun en arbeidsprosess for å rettferdig sammenligne det med tidligere tester.



Figur 6.12: Stresstestresultat for Sanic (1 arbeidsprosess).



Figur 6.13: Stresstestresultat for Sanic (2 arbeidsprosesser).



Figur 6.14: Stresstestresultat for Sanic (4 arbeidsprosesser).

Vi ser at alle tester gjør det merkbar bedre i hastighet enn demoene fortalt tidligere. Vi mener at dette forårsakes av den innebygde asynkroniteten i rammeverket, og dets bruk istedenfor kun enkle korutiner. Dette kan bevises gjennom den første testen, hvor det brukes kun en arbeidsprosess, og allikevel er den raskere på responstid enn alle implementasjoner med Flask. På den spesifikt, ser vi at dets gjennomsnittlig responstid på de utfordrende utregningsfunksjonene sank til $800 - 912$ ms; godt innenfor våre krav.

Denne responstiden kan forbedres ved økning av arbeidsprosesser for økt horisontal skalerbarhet (gitt at maskinvare har nok ressurser). Gjennom dette, ser vi enda mer reduksjon i tid, først fra 502 ms (2 arbeidsprosesser), og deretter 320 ms (4 arbeidsprosesser). For den siste, reduseres responstiden hos de utfordrende funksjonene helt ned til $619 - 837$ ms, en storartet forbedring i ytelse sammenlignet med de tidligere undersøkt biblioteker.

Skifte av rammeverk

Det er klart å se (selv med videre implementasjoner fra biblioteker med korutiner), at Flask kan ikke forbedres til å takle kravet holdt av denne oppgaven. Det ble utkonkurrert komplett av ASGI-baserte rammeverk. Selv om det finnes basis i å dra videre med sammenligninger av flere slike rammeverk for å se hvilken er

raskest, så mener vi at vi har sammenlignet nok rammeverk for helheten av oppgaven. Vi kutter det der, og bare fortsetter med Sanic som vårt nye rammeverk.

Hele følgende skifte og implementasjonen til Sanic gikk veldig raskt, takket være dets fordel av et Flaskaktig syntax. For sikkerhetens skyld for å se om våre resultater var nøyaktige, kjørte vi en siste stresstest med alle funksjonaliteter, hvor vi igjen fikk like resultater.

6.3.3 Annet hos backend

Sidemeny & URL

Det var også i løpet av denne perioden at det ble bestemt at det er best å la sidemenyet være en addisjonell ressurs som avgis av backend. Dette ble gjort med tanke på å redusere mengden av forespørsler nødvendig til serveren for å lage det gjennom frontend selv. Vi ser at det kan kreve alt fra 1-3 innkallinger til forskjellige funksjoner for frontend å gjøre det selv. Derfor, for å redusere dette til kun en, utfører vi heller ressursutdelingen hos backend selv som en optimalisering i prosjektet.

Vi visste at dette var mulig gjennom bruken av `<iframe>` elementet i HTML semantikken. Dersom vi fikk tjenesten til å avgi et HTML dokument som var dynamisk generert til en gitt URL, så ville det ha vært mulig for frontend av applikasjonen kalle til det gjennom URL. Denne implementasjonen ble hvordan vi endte opp å gjøre det i den resulterende programvaren, og dekkes derfor mer i kap. 7.2.2.

Sammen med dette, måtte vi endre på strukturen av våre URL. Inntil nå, har alle av våre URL til backend vært i form av figur 6.7, hvor den avgir kun utregningsresultater som den eneste ressurs. Men med introduksjonen av et nytt ressurs, måtte vi restrukturere på dette. Vi valgte å gå for å bruke funksjonen som en del av stien, hvor man deretter spesifiserer formen av ressurs (nå enten `/calculation` eller `/sidemenu`). Med dette i tanke, restrukturerte vi URL'ene våre til se slik ut (figur 6.15):

```
.../api/v2/funksjon/ressurs?parameter1=x&parameter2=x&...
```

Figur 6.15: Struktur av ny URL for ny ressursutgivelse.

Dette gjør det mer riktig med tanke på REST, samtidig som det ikke forandrer alt for mye på systemet vi har på plass allerede. Etter at dette ble laget, var implementasjonen av sidemeny startet og avsluttet på samme dag.

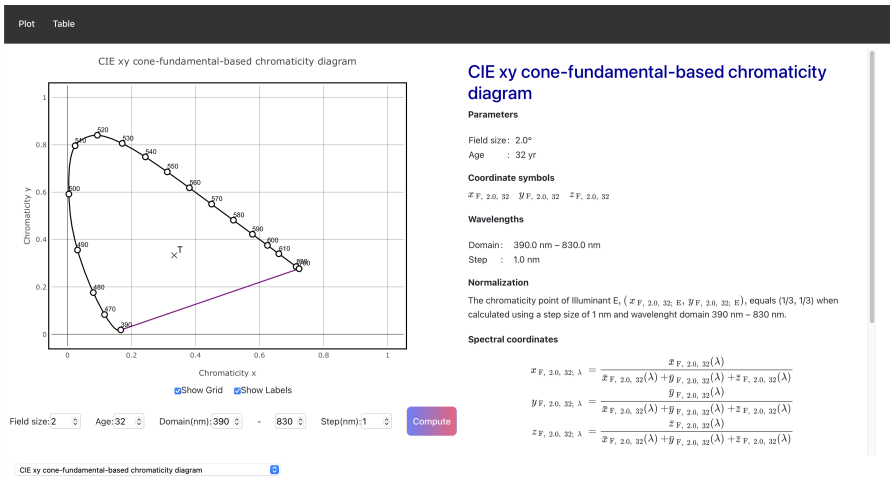
Plattform for testing

Testplattformen vi hadde laget tidligere (selv om den var preget av problemer) var allikevel svært viktig å ha med oss. Å kunne sammenligne utregninger fra vårt program og basisprogrammet underveis i utviklingen er et nødvendig steg vi mente var absolutt kritisk å inkludere. Derfor, så var det også fokusert noe innsats her for å finne en løsning til dette.

Vi flyttet først testplattformen til en egen enhetstestingmodul gjennom Unittest slik at de kunne både sjekke resultatene og øke kodekvaliteten. Etter det, forsøkte vi et par mulige løsninger. Vi bestemte oss om å fortsette bruken av Pandas, men å umiddelbart konvertere det til et `numpy.ndarray` fra Numpy. Dette ga oss ikke de samme feilene til tross fortsettelsen av Pandas bruken, så vi antok at den oppbevarer kvaliteten av tallene til en nok grad til at det ikke påvirkes av kalkulasjonsfeil. Dette ble vår endelige løsning for testplattformen, og den vi fortsetter å bruke i våre tester videre i utviklingsperioden, med mer detaljer tilgjengelig i kapittel 8.1.1.

6.3.4 Plotting av diagrammer

Mye av utviklingen i frontend for denne sprinten gikk til å utvikle brukergrensesnittet, sammen med visualiseringen av diagrammer og data gjennom Plotly. Målet var å lage og forbedre diagrammene slik at de ble mer informative. De eksisterende funksjonene for datahåndtering ble oppdatert slik at de kunne hente viktige plotpunkter fra vår tjeneste. Dette inkluderte utregninger for elementer som hvitpunktet til grafene, plotpunkter for spesielle bølgelengder og endepunktene til purpurlinja hos diagrammet.



Figur 6.16: Brukergrensesnittet etter sprint 6
Purpurlinja er den lilja-fargede linja på bunnen av hesteskokurven. Hvitpunktet er punktet i midten av diagrammet. Plotpunktene for bølgelengdene er punktene på selve kurven.

Disse dataene forberedes og manipuleres med funksjoner som `AddSpecificWavelengthPoints` for punktene på buen i grafen og `prepareChartData` for hvitpunktet og purpurlinjen. Disse funksjonene sikrer at alle nødvendige punkter blir plottet riktig på grafen.

6.4 Sprint 7 - Slutt

6.4.1 Nytt ressursystem

Det var nå antatt at vår backend kunne avgi alle utregninger nødvendig for diagrammene å illustreres hos frontend, slik vist på figur 6.16. Allikevel fantes det fortsatt et stort problem vi igjen ikke innså.

Mens noen av våre funksjoner krevde kun en forespørsel for å få alt av utregninger nødvendig (som `/lms`), så krevde andre flere utregninger fra andre funksjoner (for eksempel `xy-p`, trenger `xy` for hesteskokurven). I basisprogrammet, ble alt for ett gitt sett av parametere utregnet på samme tid og lagret til et globalt `dict`, så man kunne enkelt referere til utregningene tidligere laget hos datastrukturen. Dette var ikke mulig for oss fordi vi moduliserte funksjonene for å unngå eksakt dette. På grunn av dette, krevde frontend å sende flere forespørsler. Vi mente at dette var ikke en intuitiv løsning, og ønsket gjerne å redusere denne mengden til et minimalt.

Som en rask løsning, ble derfor ressurssystemet for utregningene forandret fra å kun inneholde direkte resultater fra `computemodularized.py`, til å også inneholde utregningene inkludert de for valgfri parametere. Det ble også iverksatt at alle nødvendige komponenter fra andre funksjoner (som hesteskoen nevnt over) var også utregnet og inkludert i ressursene. Vi essensielt gjorde det mindre modulisert, og mer likt basisprogrammet. Dette skiftet gjorde det mulig for frontend å kun sende *en* forespørsel for alt av informasjon, noe vi anså som positivt.

Men, denne endringen forårsaket et nytt problem i form av store forandringer; en stor del av parametersystemet måtte endres til å stoppe bruken av valgfri parametere. Endringen betydde også at flere av de moduliserte funksjonene mistet deres mening i å være modulisert. Dette påvirket ytelsen og tjenestens evne til å kjøre. Dette ga oss stress, sammen med tanken at frontend var ikke ferdig med alle diagramfunksjonaliteter ennå, og tidsrammen for utvikling var nærme.

6.4.2 Tilbakerulling & diagramendepunkt

Som både en løsning på systemskiftet oppe *og* for å få flere funksjonaliteter for diagrammer enkelt, så ble det oppdaget at backend kunne selv ta vare av den oppgaven. Med tanke på hvordan sidemeny var en ressurs i form av representasjonen HTML som ble tilsatt til en nettside gjennom `<iframe>`, så kom vi til ideen at diagrammet selv kan uttrykkes også som en slik ressurs.

En slik løsning ville ikke kun ha gjort det raskere enn før å starte diagrammer (med å direkte generere koden for dem gjennom serveren), men også ville ha

gjort skiftet av ressurser unødvendig. Vi kunne dra tilbake til det gamle systemet med de moduliserte utregningsfunksjonene. Gjennom bruken av diagramkoden allerede laget, ble det også veldig enkelt og raskt å implementere grafene med de manglende funksjonaliteter.

Det negative med dette, er at dette var trolig 'unødvendig'. Selv om det er mulig for tjeneren å avgi diagrammet som en ressurs, så er det best å la frontend gjøre det. Dette ville ha vært enklere og mer ytelseseffektiv for alle. Vi gjorde dette allikevel for å tilsette flere funksjonaliteter som hadde tidligere manglet.

Gjennom mye innsats over kort tid, ble diagramendepunktet snart utviklet inn i applikasjonen. Dette endte opp med å være implementasjonen vi brukte i det siste produktet, og er derfor vi ikke drar i mye detalj om det. Helle skal dets implementasjonsdetaljer dekkes i det neste kapittel.

6.4.3 Integrering av <iframe>, sidemeny & design

Ettersom det nå var utviklet en ferdigstilt løsning for grafene gjennom `<iframe>` levert av backend, måtte dette implementeres hos frontend. Det ble først gjort en opprydding i frontend delen av prosjektet, hvor alt med tilknytning til gammel implementasjon av plotting ble slettet. Det ble deretter opprettet nye React komponenter for å fremstille både plot og sidemeny gjennom `<iframe>` som hentes fra deres respektive endepunktene i APIet.

Denne løsningen er ressurseffektiv med tanke på antall API kall og håndtering av de ulike dataene tilknyttet tilleggsfunksjonene som logaritmiske verdier, sammenligning med standarder. Disse tilleggsvalgene ble tidligere returnert separat gjennom egne API kall til sine respektive endepunkter. Slik det er nå slipper man håndtering og lagring av disse relativt store utregningene hver gang man endrer fargematchfunksjon eller gjør en ny beregning med nye parametere.

Men selv om dette er en fordel med tanke på antall API kall og simplifisering av logikk for frontend, har det også noen negative sider for applikasjonen i en helhet. Det første er at det begrenser hvor tilpasselig `<iframe>` er til komponentene sammenlignet med om det ble plottet direkte som React komponent. Det andre er at siden `<iframe>` er ferdig bygd, er det også forhåndsbestemt hvilke tilleggsvalg som er aktive eller inaktive. For eksempel, om man slår av grid for så å endre en parameter og gjøre en ny kalkulasjon vil grid automatisk aktiveres igjen. Man vil da måtte bli nødt til å skru av eller på et gitt tilleggsvalg hver gang man gjør en endring som gjør at `<iframe>` blir lastet inn på nytt.

Det ble også implementert funksjonalitet for at sidemenyen blir vist under plot/tabell og parameterskjema dersom bredden på nettleservinduet beveger seg under en gitt størrelse. Dette gjør at applikasjonen er brukbar for flere typer skjermer og konfigurasjoner av nettleservinduet. Gjennom dette, oppnår vi det opprinnelige designet vi hadde planlagt gjennom vår wireframe, og implementert god responsiv design. Vi skal ta dette mer opp i neste kapittel gjennom figurer.

6.4.4 Utrulling

Etter dette, var det på tide å starte med utrulling. Hensikten med dette, er at det ikke skal være nødvendig å kjøre testmiljøene og starte de individuelle lokale serverne på lokal maskin hver gang man ønsker å kjøre applikasjonen. Vi ønsket å effektivisere prosessen for å starte applikasjonen, og å gjøre det mulig å få tilgang til den uten noen forberedelser på lokal maskin.

`requirements.txt` ble opprettet i backend og inneholder alt av avhengigheter som trengs å at apiet skal kjøre og utregning skal skje. Tilsvarende eksisterer det en fil: package.json i front en som brukes til det samme.

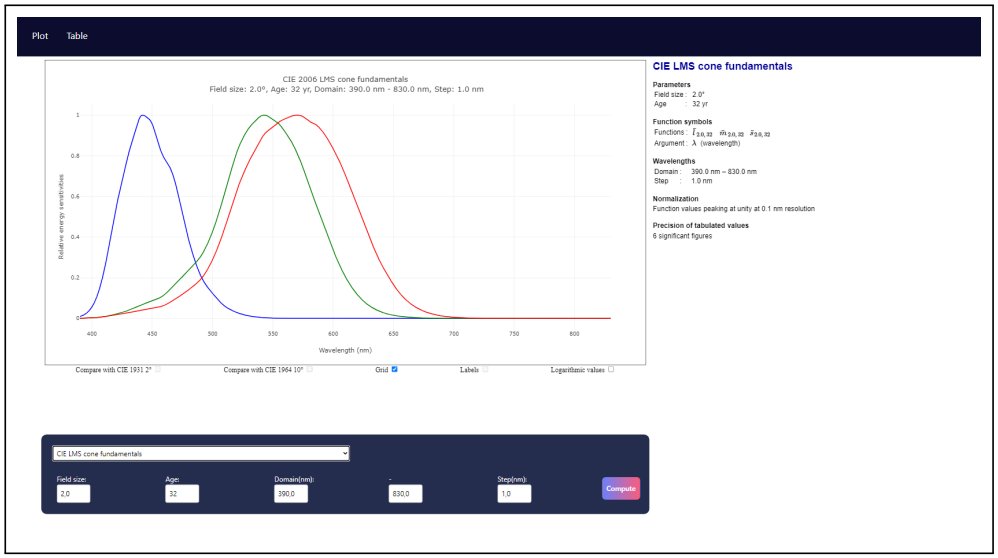
Dette ble gjort når vi ble ferdig med alle andre funksjonaliteter, og alt var klart til utrulling. Vi spesifiserer mer i detalj hvordan utrulling foregikk hos kap. 9.1.

Kapittel 7

Implementasjon

7.1 Frontend

Applikasjonens frontend har som oppgave å ta imot parametere og valg av fargematchfunksjon fra brukeren. Disse benyttes for hente data fra backends API, for så å grafisk fremstille dette for brukeren i form av plot, tabell og en sidemeny. Frontend er bygd opp av ulike React komponenter som alle tjener et spesifikt formål for en ønsket funksjonalitet. Disse utnytter seg av React sin virtuelle DOM for å raskt og effektivt oppdatere visning og tilstand. De ulike komponentene er designet for å være tilspissede og avgrenset til å kun håndtere en spesifikk utfordring. På denne måten blir koden lettere leselig, får en løsere kobling og høyere samhørighet. Ved å splitte funksjonalitet som ikke er direkte knyttet til en komponent til en egen klasse blir koden også langt mer modulær ved at ulike komponenter som alle avhenger av felles funksjonalitet kan importere og benytte seg av denne funksjonen. Man trenger da ikke å duplisere kode for hver komponent som avhenger av denne funksjonaliteten.
De ulike komponentene benytter seg av et sett hjelpefunksjoner og tredjepartisbiblioteker for å kunne oppnå ønsket funksjonalitet på mest mulig effektiv og ryddig måte. I App.tsx settes routing til graf og tabell. Her blir de ulike kompeonentene pakket inn i ParameterLayout, ParametersProvider og UseContentControllerProvider for å tilrette legge for manipulering av parametere og dynamisk rendering.

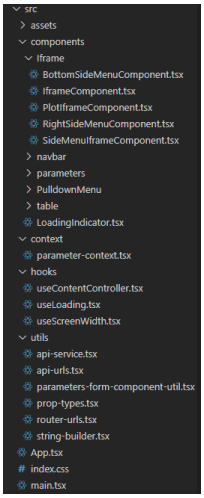


Figur 7.1: Slutt resultatet av frontend.

7.1.1 Kodestruktur

Kodestruktur for frontend kan ses i figur 7.2. Det er forsøkt å strukturere koden på en slik måte at det er intuitivt for utenforstående å navigere seg i applikasjonens filer. Alle filer faller under kildekode mappen `src`. Her har man en rekke underliggende mapper:

- `assets`: Alle ressurser som er knyttet til applikasjonen. Her finner man applikasjonens logo.
- `components`: Her finnes alle React komponentene. Disse er igjen inndelt i underliggende mapper for å gruppere hver komponent sammen med sin tilhørende CSS-fil.
- `context`: Filer som har ansvar for å håndtere kontekst og tilstand i applikasjonen.
- `hooks`: Egenkomponerte hjelpefunksjoner som har integrasjon av Reacts innebygde hooks. Logikken for disse er separert fra komponentene slik at de lett kan benyttes av alle komponentene som måtte trenge dem.
- `utils`: Akkurat som for funksjonene i `hooks`, er funksjonene man finner i `utils` hjelpefunksjoner med logikk som er adskilt fra den enkelte komponent for å oppnå en løsere kobling og høyere modularitet. Forskjellen mellom dem koker ned til at `utils` funksjonene er tilstandsløse.



Figur 7.2: Filstruktur for frontend.

I tillegg til et bevisst oppsett av filstruktur er det benyttet navnkonvensjon har til hensikt å gjøre det enkelt å forstå hvordan applikasjonen virker:

- React komponenter: PascalCase -> `MyComponent`
- Css- og typescript filer: kebab-case -> `my-component.css`
- Hooks og typescript funksjoner: camelCase -> `myComponent`.
  Hooks har i tillegg prefiks 'use', `useMyHook`.

Hensikten med dette oppsettet er som nevnt å gjøre koden så enkel og intuitiv som mulig for utenforstående, men også å få en applikasjon med høy modularitet, løs kobling og høy samhørighet.

7.1.2 Integrasjon med tjenesten

All integrasjon av interaksjoner med API er sentralisert i en egen fil kalt `api-service`, som man igjen finner i `utils` mappen. Funksjonen `fetchApiData()` tar inn parametere for endepunkt, som reffererer til valgt fargematchfunksjon, og brukerspesifiserte parametere. Den bruker så hjelpefunksjonen `stringBuilder()` for å bygge URL strengen for å hente ønsket data fra backend. Den bruker så en `fetch` funksjon med den bygde URL'en og spesifiserer at det er en GET metode som ønsker å hente et JSON svar. Dette svaret blir så lagret i en variabel kalt `response`. Det er også bygget inn feilhåndtering hvor det blir gjort en sjekk på `response`; om den er av riktig type, altså JSON. Om formatet er riktig blir dette returnert som data for `result`, som så kan brukes for å fylle inn data i tabellen. Om formatet er feil eller det skjedde en feil ved henting av data fra backend, så vil API returneres henholdsvis en beskrivende feilmelding på feil struktur på respons eller en HTTP status kode slik beskrevet senere i kap. 7.2.2.

7.1.3 React komponenter

Applikasjonen består av en rekke react komponenter som til sammen utgjør det grafiske brukergrensesnittet.

Navbar

Navigasjonsbaren er festet til toppen av nettleservinduet. Denne benytter seg av React Router DOM for å håndtere navigasjon mellom de forskjellige rutene i applikasjonen. Disse er plot og table.

Kodeliste 7.1: TypeScript kode for navigasjonsbaren

```
const Navbar = () => {
    return (
        <nav className="navbar">
            <ul className="navbar-list">
                <li className="navbar-item"><Link to={PLOT_ROUTE}>Plot</Link></li>
                <li className="navbar-item"><Link to={TABLE_ROUTE}>Table</Link></li>
            </ul>
        </nav>
    );
}

export default Navbar;
```

Som man kan se fra kodeliste 7.1, vil man ved å trykke på den ønskede grafiske fremstillingen blir man rutet til det respektive endepunktet; å trykke på plot ruter brukeren til URL/plot.

Plot- og tabell komponent

Komponentene for plot og tabell er plassert på samme sted i applikasjonen. Det er gjennom Navigasjonbaren og den resulterende rutingen det blir bestemt hvem av de to komponentene som vises. Plot komponenten er en React komponent som returnerer en iframe med en ferdig plottet graf basert på hvilken URL-streng den mottar.

Tabell komponenten mottar data fra et kalkulasjonskall til API. Denne dataen blir så puttet inn i en tabell for en oversiktlig representasjon av dataene.

Sidemeny

I likhet med plot komponenten, mottar sidemeny komponenten en iframe med ferdig stylet html kode basert på valgt fargematchfunksjon og parametere. Plasseringen av denne komponenten avhenger av bredden på nettleservinduet. Hvis den er under 910 piklser vil sidemenyen bli plasser under plot/tabell og parameterskjema. Dette gjør det lettere å operere applikasajonen på enheter med smalere bredde på skjerm. Implementeringen av dette skjer gjennom hjelpefunksjonen useScreenWidth(). Bredde verdien for skjermen definerer to ulike kompo-

nenter for sidemenyen RightGridinformationIframe og BottomGridInformation-Iframe. Hvordan dette er implementers vises under i kodeliste 7.2

Kodeliste 7.2: Funksjon som endrer plassering av sidemenyen basser på vindustørelse

```
const BottomSideMenuComponent: React.FC = () => {
  const screenWidth = useScreenWidth();
  return (
    <>
      {screenWidth <= 1200 && (
        <div className="sid">
          <SideMenuIframe />
        </div>
      )}
    </>
  );
};
export default BottomSideMenuComponent;
```

Parameterskjema og tilstandshåndtering

Parameterskjemaet er den mest avanserte komponenten i applikasjonen. Den krever tilstandshåndtering og må utnytte React virtual DOM for å oppfatte ulike endringer. Denne komponenten er ansvarlig for å vise og håndtere endring av parametere basert på brukerens input for ulike fargematchfunksjoner. Koden viser hvordan komponenten tar imot parametere fra brukeren og oppdaterer tilstanden basert på disse.

Tilstandshåndteringen skjer ved hjelp av Reacts innebygde hooks som 'useState' og 'useEffect' kombinert med egenkomponerte funksjoner. Siden applikasjonens kompleksitet og krav til tilstandshåndtering ikke var veldig omfattende, ble det besluttet å bruke Reacts innebygde hooks fremfor tredjepartsbiblioteker som Redux.

Alle fargematchfunksjonene benytter i stor grad de samme parameterne: feltstørrelse, alder, minimums- og maksimumsverdi for domenet av bølgelengder, og trinnstørrelse. For å sikre at applikasjonen husker disse parameterne når brukeren bytter mellom visninger, er tilstandshåndteringen implementert slik at parametrene bevares ved visningsbytte. Dette gjør det mulig for brukeren å utføre en beregning og sømløst bytte mellom tabell og plot uten å måtte utføre beregningen på nytt.

To variabler, 'generalFieldSize' og 'dropDownFieldSize', er implementert for å håndtere forskjellene mellom de generelle fargematchfunksjonene og standardfunksjonene som kun tar hensyn til feltstørrelser på 2 eller 10 grader. 'dropDown-FieldSize' lagrer den bruker-valgte feltstørrelsen fra nedtrekksmenyen for standardfunksjonene, mens 'generalFieldSize' lagrer den brukerspesifiserte feltstørrelsen for de øvrige fargematchfunksjonene. Gjennom funsksjonene som kan ses i kodeliste 7.3, settes feltstørrelsen i henhold til valgt funksjon, slik at en endring i parameter for en funksjon ikke påvirker andre funksjoner unødig.

Kodeliste 7.3: Håndtering av ulik field$_s$*izetypeogtilleggsparameter'base'*

```
useEffect(() => {
    const methodNumber = parseInt(selectedOption.replace('method', ''));
    setParameters(prev => {
      const updatedParams = { ...prev };

      // Handle optional parameter for method2
      if (selectedOption === 'method2') {
        updatedParams.optional = 'base';
      } else {
        delete updatedParams.optional;
      }
      // Handle field size based on method number
      if (methodNumber >= 1 && methodNumber <= 8) {
        updatedParams.field_size = generalFieldSize;
      } else if (methodNumber >= 9 && methodNumber <= 10) {
        updatedParams.field_size = dropdownFieldSize;
      }
      return updatedParams;
    });
    setParamsUpdated(true);
  }, [selectedOption, dropdownFieldSize, setParameters]);

// Handles parameter change for every function except xyz
const handleParameterChange = (event: ChangeEvent<HTMLInputElement>): void => {
    const { name, value } = event.target;
    const numericValue = parseFloat(value);

    // Validates the user specified input
    parameterSchema.validateAt(name, { [name]: numericValue })
      .then(() => {
        if (name === 'field_size') {
          setGeneralFieldSize(numericValue);
        }
        setParameters(prev => ({ ...prev, [name]: numericValue }));
      })
      .catch(err => {
        console.error(err.errors);
      });
};

// Handles parameter change for xyz functions,
// where field size of either 2 or 10 degrees are the parameters
const handleDropdownChange = (event: ChangeEvent<HTMLSelectElement>): void => {
    const selectedDegree = parseFloat(event.target.value);
    setDropdownFieldSize(selectedDegree);
    setParameters(prev => ({ ...prev, field_size: selectedDegree }));
    setParamsUpdated(true);
};
```

Denne fleksibiliteten og dynamikken gjør parameterskjemaet til en kritisk del av applikasjonens frontend, som effektivt håndterer brukerinput og tilstandsoppdateringer basert på valgte parametere og funksjoner.

7.1.4 Validering av brukerinput

Validering av brukerspesifisert inndata for parametere skjer ved hjelp av JS biblioteket 'Yup'. Dette implementeres ved at man først definerer et skjema som beskriver reglene som gjelder for valideringen.

Kodeliste 7.4: Yup skjema som definerer regler for de ulike parameterne

```
export const parameterSchema = yup.object().shape({
    field_size: yup.number()
      .required('Field size is required')
      .min(1.0, 'Field size must be greater than 1.0')
      .max(10.0, 'Field size must be less than 10.0'),
    age: yup.number()
      .required('Age is required')
      .min(20, 'Age must be greater than 20')
      .max(80,  'Age must be less than 80'),
    min: yup.number()
      .required('Min domain is required')
      .min(390.0, 'Min domain must be greater than 390.0')
      .max(400.0, 'Min domain must be less than 400.0'),
    max: yup.number()
      .required('Max domain is required')
      .min(700.0, 'Max domain must be greater than 700.0')
      .max(830.0, 'Max domain must be less than 830.0'),
    step_size: yup.number()
      .required('Step size is required')
      .min(1.0, 'Step size must be greater than 1.0')
      .max(5.0, 'Step size must be less than 5.0')
  });
```

I skjemaet som er tatt i bruk(kodeliste 7.4) for dette prosjektet brukes en .required(), .min() og .max() regel for alle feltene for parameter inndata. Navnene på disse regelfunksjonene er nokså selvforklarende; .required() spesifiserer og påser at det må finnes en verdi mens .min() og .max() definerer en minimums- og maksimumsverdi og definerer med det et tillat spenn for verdier for den gjeldende parameteren.

Applikasjonen initialiseres med standardparametere, så disse trenger ikke å valideres. Derfor implementeres sjekken i funksjonen som håndterer parameterendringer. Her brukes en innebygd funksjon fra Yup biblioteket som utfører sjekken på om de definerte reglene er fulgt. Hvis den nye verdien oppfyller kravene stilt av reglene vil verdien oppdatere seg, men om den nye verdien ikke oppfyller kravene vil endringen stoppes i sanntid.

Man har valget for hvordan man vil håndtere en feilsituasjon. Feilhåndteringen er at verdiendring blokkeres dersom den faller utenfor det tillatte spennet, og at det skrives en beskrivende feilmelding til konsollen. Det kan ofte være god praksis å gi tilbakemelding til brukeren, hvor det forklares hva som er feil. Da sluttbrukerne av denne applikasjonen er godt inneforstått med disse begrensningene i parameterverdi, kan det anses som intuitivt at man er utenfor det tillatte verdiområdet dersom man ikke får økt eller senket en verdi ytterligere.

7.1.5 Responsivt design

Som det kom av wireframes var det tenkt at applikasjonen skulle være utviklet med responsivt design til grunne. Dette for å sørge for at applikasjonen kan brukes på en rekke ulike enheter med forskjellige skjermstørrelser. Dette ble lagt litt på hyllene gjennom de fleste av sprintene, da det var utviklingen av funksjonalitet som var hovedfokus. Graden av hvorvidt responsivt design er implementert kan derfor diskuteres. Det kommer også en begrensning gjennom at grafene og tabellen må være stor nok til at det er mulig å interagere med, for ikke å snakke om å i det hele tatt se dem.

Med dette tatt i betraktning ble det bestemt at det måtte være en minste verdi på bredde på komponentene som viser graf og tabell. Det ble også bestemt at dersom skjemen er over en gitt høyde og bredde, skal scrollbar'en gjemmes. Dette anses da som å kjøre applikasjonen i fullscreen, og hele applikasjonen skal være innenfor visningen. Dersom enten høyden eller bredden blir mindre enn den definerte verdien dukker scrollbar opp igjen slik at man har mulighet til å scrolle i lengde eller bredde retning for å allikevel kunne bruke applikasjonen. Hvordan dette er integrert kan ses i kodeliste 7.5, som viser et hvordan dette implementeres med 'media queries'.

Kodeliste 7.5: Media Queries som brukes for å bestemme oppførsel ved gitte dimensjoner

```
.plo{
  width: 66.33%;
  min-width: 900px;

....

@media (max-height: 910px) {
  :root{
    overflow: auto;
  }
}
@media (max-width: 1200px) {
  :root{
    overflow: auto;
  }
  .plo,
  .tab {
    width: 100%;
    margin-right: 5%;
  }
  .sid {
    width: 100%;
    margin-top: 20px;
  }
}
```

Både som det fremkommer i kodeliste 7.5 og som nevnt i 7.1.3, blir også sidemenyen plassert under parameter skjemaet når bredden på skjermen blir under 910 piksler. Dette kan ses i figur 7.3.

Figur 7.3: Sidemeny skyves ned.

7.2 Backend

Dette kapitlet skal fokusere på implementasjonsdetaljer for backenden hos applikasjonen. Vi ønsker først å deklarere at det blir gjenbrukt kode fra basisprogrammet [1]. Dette var for å opprettholde likheten med programmet, og forsørge seg for de samme resultatene. For å forsikre seg korrekt dokumentasjon, har all gjenbruk av kode blitt dokumentert tydelig og klart i dokumentasjonen hos koden.

Det skal nå dras inn på strukturen av tjenesten backend tilbyr. før strukturen av koden og hvorfor den er laget slik den er. Dette er noe vi mener skal først, fordi det er hensiktsmessig for rapportens struktur og formidling.

7.2.1 Tjenestestruktur

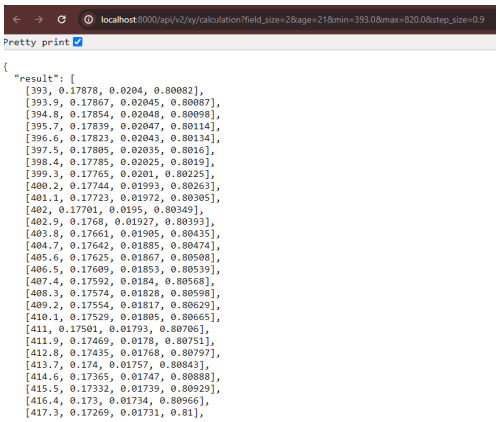
Backend tilbyr en rekke tjenester på dets flere URL for å tilby komplett funksjonalitet sammen med frontend til enhver bruker av applikasjonen. Dets primære tjeneste er å avgi utregninger i form av diverse representasjoner, tilgjengelige på URL som er logisk strukturert for å gi enkel mening til brukere - slik de er vist frem på figur 7.4.

```
- ../api/v2/xy/calculation?field_size=2&age=21max=820.0&step_size=0.9
- ../api/v2/lms/sidemenu?field_size=2.0&age=23&log10&max=700&min=400
- ../api/v2/lms-mb/plot?field_size=1.5&age=51&max=700
```

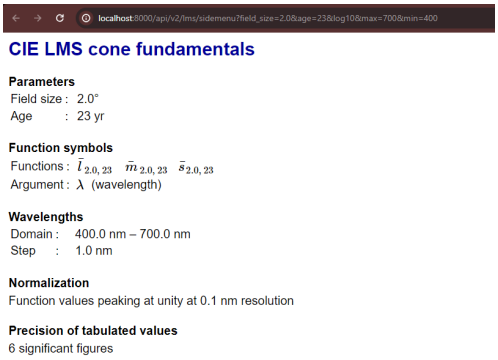
Figur 7.4: Eksempeler av URL for finalisert webapplikasjon.

Alle ressurser som avgis av APIet, trenger tre ting:

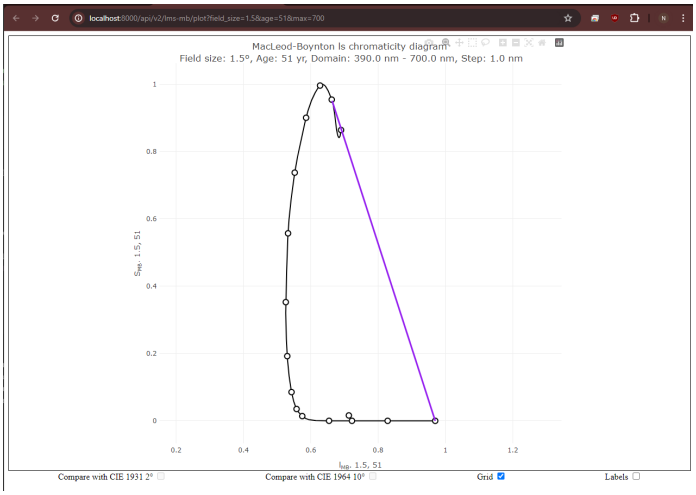
- En funksjon tilgjengelig i programmet representert i stien til lenka, hvor eksempler over er `/xy`, `/lms-mb` og `/lms`.
- En valgt representasjon av utregningsressursen, noe som kan ta tre former:
    - `/calculation` (som vist i figur 7.5) vil utgi utregningene for funksjon og gitte parametere direkte til brukeren i form av `application/json`.
    - `/sidemenu` (figur 7.6) vil representere en informasjonsside for funksjonen, lik til sidemenyet som finnes i basisprogrammet. Denne er i form av `text/html`, hvor det genereres dynamisk fra applikasjonen.
    - `/plot` (figur 7.7) vil representere diagrammet for funksjonen, sammen med funksjonaliteter tilgjengelig hos funksjonen. Denne er også i form av `text/html`, hvor det består av innebygd JS kode for å implementere det funksjonelle diagrammet.
- Et sett av URL baserte parametere for å representere brukerinput gjennom bruk av URL queries.



Figur 7.5: Eksempel på resultat fra /calculation endepunkt.



Figur 7.6: Eksempel på resultat fra /sidemenu endepunkt.



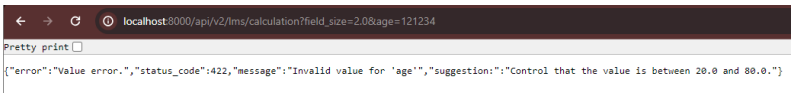
Figur 7.7: Eksempel på resultat fra /plot endepunkt.

Implementasjonsdetaljer og nærmere diskusjon til alle tre former av ressurser over vil tas opp videre i 7.2.2. Men, i tillegg til alt dette, finnes det en addisjonelle endepunkter for å øke kvaliteten av tjenesten, i form av et statusendepunkt. `/status` vil utgi informasjon om selve tjenesten, inkluderende hvor lenge serveren har vært oppe, og hvilken versjon er den nyeste.

RESTful

Et sentralt punkt for oss er å forsikre at vår tjeneste følger REST for å være RESTful slik vi hadde tenkt i kap. 4.2.1 - noe vi skal nå deklarere med tanke på prinsippene deklarert tidligere i kap. 2.2.3. Rekkefølgen de deklareres i følger samme rekkefølge som de ble introdusert.

1. Vi mener at tjenesten oppfyller prinsippet om 'separasjon av klient og server', men ønsker å utdype angående noe spesifikt. Som vi vet, så gir tjenesten også vekk ressurser i form av representasjon `text/html`, hvor den er da dynamisk generert hos backend. Vi erkjenner at dette kan ses ut som at tjenesten drar inni denne bekymring som skal opprinnelig ligge hos frontend - men vi mener at vi følger prinsipper allikevel, for vi kun avgir en ressurs i form av et HTML dokument - og selve visningen av det er opp til frontend.
2. Tjenesten lagrer ingen informasjon fra tidligere klientforespørsler, og derfor enkelt oppfyller kravet om statsløshet.
3. Alle av svarene som sendes fra server markeres eksplisitt om kan lagres gjennom bruk av `cache-control header` i svarene tjenesten genererer. Alle utregningsresultater har fått verdien `private` (som gjør at de kan lagres i privat cache hos nettlesere), mens statusendepunkt har fått en `no-store` - aldri cachet.
4. Det finnes ingen kode i applikasjonen som spesifiserer at det krever noe fra et spesifikk arkitekturkomponent. Arkitekturen vi har brukt krever kun to komponenter som kjører separat fra hverandre slik det er implementert nå.
5. Tjenesten implementerer bruken av dette prinsippet med å levere kjørbar kode hos noen av ressursene den utgir (gjennom endepunktet `/plot`). Koden kjøres på nettleseren til brukeren for å tilsette ekstra funksjonaliteter (funksjonelle avmerkningsbokser, diagrammer).
6. Alle ressurser som tilbys av tjenesten finnes på unike URL, hvor de kan kun hentes gjennom bruken av `GET`, som er da semantisk riktig. Det brukes også statuskoder for å inneholde informasjon om respons, i tillegg til at det finnes en hjemmeside med nok informasjon for en bruker å kunne lære å traversere seg rundt i tjenesten - sammen med detaljerte feilmeldinger som forteller mulige forslag til å fikse en gitt feil (7.8). Med dette, kan vi si at tjenesten oppfyller dette kravet.



Figur 7.8: Eksempel på feilmelding fra tjeneste.

Fordi vi mener at applikasjonen vår faller innom disse kravene, så kan vi kalle det for en RESTful applikasjon - og være fornøyd med dets evner.

7.2.2 Kodestruktur

Før vi drar inn på spesifikke endepunkter, ønsker vi først å belyse programmets generelle struktur. Dette er fordi rammeverkets syntaks er veldig likt andre rammeverk i andre språk fra vår egen erfaring, hvor vi ser følgende fellestrekk:

- Det lages en instans av en `app` for å representere en webapplikasjon.
- Til instansen, lages det kodefunksjoner for å representere endepunkter.

- Etter deklarasjonen av alle endepunkter, kjøres applikasjonen slik at den kan tjene til endepunktene.

Sanic passer inn her, om vi referer til kodelisten under. Vår hovedinstans er en `sanic.app.Sanic` gjennom `Sanic(__name__)`. Det er mulig å tilsette kontekster til denne applikasjonen (blant annet databaser eller tilgjengelige andre Sanic applikasjoner)[1], men for vår oppgave, tilsatte vi kun CORS[2] beskyttelse for å tillate andre maskiner å hente data fra tjenesten. CORS er ikke tilgjengelig innebygd i Sanic, men rammeverket har et stort økosystem med biblioteker for flere funksjonaliteter, hvor dette er kun en av dem[3].

Etter at applikasjonsinstansen er deklarert, tilsetter vi endepunkter gjennom funksjonen `@api.get()` sammen med en påkrevd `string` parameter for å representere URL for endepunktet. En slik tilkalling *må* alltid følges opp enten av en annen `@api.get()` for å definere flere URL, eller en funksjonsdeklarasjon med kode for å takle forespørsler sendt til de gitte URL - hvor det må kunne sende en respons. Fordi rammeverket støtter ASGI, kan vi tilsette `async` foran endepunktsfunksjonene for å gjøre dem asynkrone i rammeverket.

Kodeliste 7.6: Backend - Generell struktur

```
# hovedinnstans
api = Sanic(__name__)
CORS(api)

...
# utsnitt av endepunkt
@api.get(endpoint_creator(API_HOMEPAGE, API_VERSION, LMS_ENDPOINT) + "/<more:str>")
async def LMS(request, more: str):
    # behandling av endepunkt kommer her ...

...
# oppstart av applikasjon
if __name__ == '__main__':
    api.run(debug=True, port=8080)
```

Oppbygningen av URL hos alle endepunkter foregår gjennom en dedikert funksjon kalt `endpoint_creator(...)`, som tar inn `string`, og konstruerer et URL-sti for ett endepunkt basert på parametere. For dette, følger vi god praksis med å bruke konstanter med navn i store bokstaver deklarert øverst i filen. Det brukes også en `/<more:str>` for å gjøre det mulig å hente alle videre stier relatert til endepunktet fra denne URL.

I kodelista under, ser vi et eksempel på innmatet hos et endepunkt for utregning. Alle slike endepunkt deler på denne strukturen, hvor deres eneste forskjeller er deres URL og hvilke fargematchfunksjon de utregner for. I dette eksemplet, brukes det `API_HOMEPAGE, ("/api")` og `API_VERSION, ("v2")`, sammen med en `LMS_ENDPOINT, ("lms")` for å definere endepunktet for fargematchfunksjonen 'LMS'.

[1] `https://sanic.dev/en/guide/basics/app.html#application-context`, hentet 20.05.2024
[2] `https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS`, hentet 20.05.2024
[3] `https://sanic.dev/en/plugins/sanic-ext/http/cors.html`, hentet 20.05.2024

Sammen med bruken av `more`, kan sjekke en eventuell klient sin URL for å se hva de forsøker å hente. Verdien av denne variabelen sjekkes og rutes videre til ett av fire mulige valg.

Kodeliste 7.7: Backend - Eksempel av utregningsendepunkt.

```
@api.get(endpoint_creator(API_HOMEPAGE, API_VERSION, LMS_ENDPOINT) + "/<more:str>")
async def lms(request, more: str):
    if more == "calculation":
        return response.raw(new_calculation_JSON(compute_LMS_modular,
                                                 create_and_check_parameters(
                                                     True,
                                                     compute_LMS_modular,
                                                     request)),
                            content_type="application/json",
                            headers={"cache-control": "private"})
    if more == "sidemenu":
        return html(LMS_sidemenu(create_and_check_parameters(
            True,
            compute_LMS_modular,
            request)),
            headers={"cache-control": "private"})
    if more == "plot":
        return html(LMS_graph(create_and_check_parameters(
            True,
            compute_LMS_modular,
            request)),
            headers={"cache-control": "private"})
    else:
        not_found(more)
```

Brukeren kan enten sendes til å hente de tre tilgjengelige representasjoner av ressurser introdusert tidligere, eller så får de en feilhåndteringsmelding. Alle av disse er videre modulisert og delt inni passende filer. Mer informasjon om disse skal ses på i deres egne kapitler eventuelt. Før vi gjør det, ønsker vi å introdusere det underliggende systemet delt gjennom hele systemet med evnen til å behandle brukerinput og lagre det i en praktisk form; parametersystemet.

Parametersystemet

Rett etter selve webapplikasjon, ligger parametersystemet vi hadde laget som det nest viktigste systemet i hele programmet. For programvaren, så er det en `dict`-variabel med all brukerinformasjon hentet fra en gitt URL. Men, i hele systemet, er denne variabelen ansvarlig for all inntak av brukerinformasjon, validering av det (og tilstrekkelig feilhåndtering), og til sist dets universelle anvendelse i alle utregningsfunksjoner videre i programmet.

For enklere formidlingen, skal systemet diskuteres med tanke til kodelistene inkludert hos vedlegg G. Om vi ser på kodeliste G.1, så ser vi at vi starter opp hele parametersystemet gjennom funksjonen `createAndCheckParameters(...)`. Denne funksjonen eksisterer hos alle endepunkter, for alle ressurser. Funksjonen tar inn selve utregningsfunksjonen, sammen med et objekt som representerer klientforespørselen kalt `request` og en `bool` for sjekking om dette er for en standardise-

ringsfunksjon (hvor kun en nødvendig parameter sjekkes). Funksjonen vil enten returnere en `dict` av validerte parametere, eller så vil den utføre en `raise` og sende et feilsvar til klienten umiddelbart uten å kjøre resten av koden.

G.1 introduserer bruken av en indre hjelpefunksjon kalt `string_to_type_else` basert på `<Option>`[4] og `Maybe`[5] monadtypene fra programmeringsspråkene Rust og Haskell. På samme måte disse typene kan takle potensielle feil hos programmet på en grasiøs måte [47] (med å ha gitte verdier være abstrakte enten 'noe av type' eller 'ingenting') - så tar denne funksjonen og returnerer enten en verdi, `None` eller en spesifisert verdi (som vi skal se på i del 2/kodeliste G.2). Den oppnår dette med bruken av `try` og `except`; kode som gjør det mulig å forsøke potensielt feilaktig kode som innmat, hvor den vil fange programfeil i koden og kjøre spesifikk kode for en gitt type feil. Funksjonen tar inn en `string` (kalt 'string'), en datatype kalt 'type' som 'string' skal forsøkes å konverter til, og noe som 'other'. Resten av del 1 bruker funksjonen for å finne ut om de obligatoriske parameterne `field_size` og `age` er tilgjengelig i URL hos forespørselen. Den lager en `dict` med representanter for dem, hvor deres verdier er da resultatene fra hjelpefunksjonen. Om de er innenfor URL, så hentes dem og omgjøres til den ønskede `float` typen - ellers, vil hjelpefunksjonen utføre en feil, og feilen blir fanget av funksjonen - hvor den vil da returnere en `None`. Etter dette, vil alle parametere sjekkes om de har en `None` verdi, der om en har det, så vil programmet umiddelbart utføre en `raise` av typen `SanicException` med parametere for en egen feilhåndteringsfunksjon diskutert senere.

G.2 viser en videre bruk av hjelpefunksjonen, hvor den brukes sammen med en alternativ verdi kalt 'other'. Dette brukes for at hjelpefunksjonen skal kunne forskjelliggjøre mellom feil av *ingen* verdi og med *feil* verdi slik tidligere sagt. Når vår hjelpefunksjon forsøker å lese inn noe, så kan det oppstå kun tre utfall: Den vil enten klare å lese det korrekt, ikke kunne lese det fordi det er ikke av riktig form - eller ikke lese det for ingenting er oppgitt. For alle tre situasjoner, takler den det forskjellig for å gjøre ruting mulig til hver scenario mulig, og derfor unike behandlinger av dem - hvor ett eksempel er i den oppgitte kodelista, hvor den blir brukt til å gi standardverdier til parametere som ikke er oppgitt, men også kunne gi en feilmelding om de er oppgitt med feil verdi.

Om parameterne passerer disse sjekkene og drar videre, så begynner den andre fasen av feilvalidasjon - hvor de sjekkes om de er av riktige verdier (med ett eksempel i de siste 5 linjer av kode i del 2 for sjekking av `field_size`). Dersom alle parametere klarer dette sjekket, så drar de videre til en ekstra vurdering for `optional` parameteren, en egen dedikert parameter for valgfri input (som `log` hos `/lms`). Etter alt av dette, burde funksjonen gi et brukbart `dict` med kompatibilitet i alle endepunktfunksjoner.

[4] `https://doc.rust-lang.org/std/option/`, hentet 20.05.2024
[5] `https://wiki.haskell.org/Maybe`, hentet 20.05.2024

Endepunkt for utregning

Hovedressursen som skal avgis av alle utregningsendepunkter, er selvfølgelig utregningene selv. Prosessen for dette er abstrakt delt mellom alle endepunkter, og innebærer to steg:

- **Utregningen:** Det utføres et kall til en modulisert versjon av en `compute_x()` funksjon fra `compute.py`[6] (hvor `x` er en fargematchfunksjon fra basisapplikasjonen). `compute.py` er fra kodelageret til basisprogrammet, hvor utregningskoden for de moduliserte versjonene er direkte basert på dem. Den moduliserte funksjonen vil da gi utregningene som et resultat.
- **Formatering:** Utregningene deretter pakkes inn i en format de ikke kan bli påvirket videre i, i form av en JSON-basert enkoder med tilgang til stringformatering for spesifisering av presisjon.

Vi skal ikke dra i detalj på alle utregninger for de er unik til hver funksjon i applikasjonen. Allikevel, mener vi at det er best å nevne et eksempel og hva vi har gjort. Om vi ser på kodebiten E.3 tilgjengelig hos vedlegg G, ser vi et utsnitt av den moduliserte utregningsfunksjonen for `/lms` endepunktet. Den deler struktur med alle andre utregningsfunksjoner, hvor:

- Den tar inn kun en parameter kalt `parameters`, som representerer resultatet fra parametersystemet forklart tidligere. Denne variabelen vil inneholde alt av nødvendig brukerinput for å utføre utregninger basert på dem - og er veldig brukbar når den brukes universelt mellom alle endepunktfunksjoner.
- Alle slike utregningsfunksjoner vil returnere en `dict`, bestående enten av `result` & `plot` medlemmer - eller av andre medlemmer dersom bruken av `info` i brukerparametere aktiveres. Noen funksjoner støtter ikke bruken av `info` (slik dette eksemplet), men dette tas vare av i parametervalidasjon.
- Alle linjer i funksjonen er direkte basert på linjer hos `compute.py`. Som vi har sett tidligere hos 6.5, så har vi bevist at det gir det raskere resultater å modularisere istedenfor å bruke de opprinnelige funksjonene. Selv om den løsningen er gammel og gjelder ikke her lenger, så kan den fortsatt gjelde for visse situasjoner (som bruken av `'info'`, hvor den må utlevere forskjellige utregningsresultater enn de vanlige `result` og `plot`).
    - For å direkte vise at dette er gjenbruk, har modulen til koden selv deklarert dette klart - sammen med at vi har tilsatt direkte kommentarer til hvilke linjer korresponderer til hva i det opprinnelige.

Det som er heller mer interessant, er formatering av resultatene. Som vi husker fra kap. 6, så var det et stort fokus av oss å utlevere de riktige tallene uten påvirkning fra feil i presisjon - inntil vi fant en løsning basert på en egen JSON-enkoder med bruk av spesifikke formateringer for gitte utregninger.

Dette systemet starter opp med en hardkodet `dict` i selve programmet, av slik

[6] `https://github.com/ifarup/ciefunctions/blob/master/tc1_97/compute.py`, hentet 21.05.2024

utseende:

Kodeliste 7.8: Backend - Utdrag av stringformateringer

```
calculation_formats = {
    "LMS-base-log": {
        "result": ["{:.1f}", "{:.8f}", "{:.8f}", "{:.8f}"],
        "plot": ["{:.1f}", "{:.8f}", "{:.8f}", "{:.8f}"],
    },
    ...,
    "XYZ-XYZP-XYZ-STD": {
        "result": ["{:.1f}", "{:.6e}", "{:.6e}", "{:.6e}"],
        "plot": ["{:.1f}", "{:.6e}", "{:.6e}", "{:.6e}"],
    },
    # shared dictionary between all info
    "info-1": {
        "norm": ["{:.8f}", "{:.8f}", "{:.8f}"],
        "white": ["{:.6f}", "{:.6f}", "{:.6f}"],
        ...
        "trans_mat": ["{:.8f}", "{:.8f}", "{:.8f}"],
        # "trans_mat_N": ["{:.8f}", "{:.8f}", "{:.8f}"],
    }
}
```

De er hentet direkte fra basisapplikasjonen sin kildekode, og er organisert i `list` av string med forskjellige formateringer i seg. Mengden av formateringer korresponderer direkte til lengden av en rad i utregningene - og derfor, så er en formatering direkte korresponderende til et medlem i raden. For eksempel, i `LMS-base-log`, så er det forventet fire elementer per rad i utregningen - og at det første medlemmet formateres gjennom `{:.1f}`, et desimalltall med kun en desimal. Dette er viktig å huske på når vi fortsetter.

Dette systemet brukes primært i en rutingsfunksjon kalt `new_calculation_JSON(...)`, som både starter utregningen samtidig som det passerer det (og det utvalgte stringformatet) til vår JSON-enkoder gjennom koden under.

Kodeliste 7.9: Backend - Utdrag av rutingsfunksjon for JSON

```
def new_calculation_JSON(calculation, parameters):
    if parameters['info']:
        return write_to_JSON(calculation(parameters),
            calculation_formats['info-1'])
    if calculation is compute_LMS_Modular:
        if parameters['base']:
            if parameters['log']:
                # log10 base LMS
                return write_to_JSON(calculation(parameters),
                    calculation_formats['LMS-base-log'])
            else:
                # base LMS
                return write_to_JSON(calculation(parameters),
                    calculation_formats['LMS-base'])
        ...
```

Videre kalles `write_to_JSON()` for å starte enkodingen, men selv det er kun en hjelpefunksjon for viderebehandling av resultater fra funksjonen `ndarray_to_JSON`; hvor er faktisk der formateringen foregår.

Kodeliste 7.10: Backend - Formateringsfunksjonen

```
def ndarray_to_JSON(body, formatta):
    # an 1D-array, or by an ndarray recursively calling this function for each row)...
    if body.ndim == 1:
        # Double check if the length of the row is equal to the length of the format
        if len(body) == len(formatta):
            # Format each row respectively as they should be, through indexed for-loop:
            all = []
            for index in range(len(body)):
                # Retrieve the format and chopped body.
                tempar = chop(body[index])
                asda = formatta[index]
                # The '-inf' produced by LMS (log10) is something
                    £ that cannot be parsed by JSON;
                # it gets remade into null instead.
                if math.isinf(body[index]):
                    tempar = "null"
                    asda = "{}"
                # Append the formatted string to the 'all' array,
                    # symbolizing 'all' elements of current row.
                all.append(asda.format(tempar))
            # Join them all with commas, and then output it with [ ]
                # on either side, succesfully making it into
                # a JSON array.
            output = ','.join(all)
            return '[' + output + ']'
        else:
            # should only happen if the row format of a calculation does not match
            # the given format (example, [x, y, z] =/= [f1, f2]
            raise SanicException(
                ("PROCESSING␣ERROR",
                 "There␣has␣been␣an␣error␣inside␣of␣the␣server.",
                 "Contact␣system␣administrator␣for␣server,␣or␣try␣again␣later."),
                status_code=500)
    else:
        # create empty array
        temp = []
        # for each array in current array, recursively call to each
            # and append them to array above
        for row in body:
            temp.append(ndarray_to_JSON(row, formatta))
        # join them with commas, and output with [ ]
        output = ','.join(temp)
        return '[' + output + ']'
```

Formateringsfunksjonen tar inn en `numpy.ndarray` for all utregning (tilkalt tidligere), sammen med stringformatene beskrevet tidligere. For alle medlemmer hos den gitte `numpy.ndarray` som er over enn *en* dimensjon, vil den utføre et rekursivt kall til hver av sine barn inntil det møtes på en endimensjonal medlem. Når den kommer til en `body` av kun en dimensjon, så vil den formatere alle medlemmer av dette sammen med formateringen gitt, lenke alle til en singulær string gjennom `','`, og deretter sette `'[' og ']'` hos start og slutt. Når dette kombineres med dette samme for alle tidligere rekursive kall, ender vi opp med en riktig representasjon av `ndarray` til JSON i form av en `string` som ligner strukturen av en JSON fil.

`write_to_JSON()` tar resultatene, og tilsetter en `map` struktur til `string`en produ-

sert, før den sendes tilbake som resultatet for `new_calculation_JSON()`. Fordi dette er en rå string som kun representerer en JSON, så sendes eksplisitt ut som en rå fil gjennom metoden `.raw()`, men med en `header` som forteller at filen er av en `application/json` type.

I utviklingsprosessen, fortalte vi at vi forsikret oss at tallene var riktig med å sammenligne dem med programmet. For å tilsette mer sikkerhet til funksjonen, ble det også utviklet tester som sjekker lint av sluttresultatet - hvor vi kan si at vi har hatt ingen problemer med en feilstrukturert JSON.

Endepunkt for sidemeny

Som skrevet tidligere i 6.3.3, så måtte sidemeny implementeres som en ressurs i håp på å redusere mengden av forespørsler nødvendig for den vanlige brukeren. Men istedenfor å diskutere den direkte implementasjonen, skal vi heller først faktisk fortelle om basisprogrammet sin implementasjon[7]:

Kodeliste 7.11: CIE Functions - Utdrag fra description.py, linje 1161-1196

```
def lms_mb(data, heading, options, include_head=False):
    html_string = ""
    if include_head:
        html_string += _head()
    html_string += (_heading(heading) +
                    _parameters(data) +
                    _coordinates('\\(l_{\,\mathrm{MB},\,%s,\,%d}\\)' %
                                 (data['field_size'], data['age']),
                                 '\\(m_{\,\mathrm{MB},\,%s,\,%d}\\)' %
                                 (data['field_size'], data['age']),
                                 '\\(s_{\,\mathrm{MB},\,%s,\,%d}\\)' %
                                 (data['field_size'], data['age'])) +
                    _wavelenghts(data) +
                    _normalization_lms_mb(data) +
                    _LMS_to_lms_mb(data, options) +
                    _precision_lms_mb() +
                    _illuminant_E_lms_mb(data) +
                    _purpleline_tangentpoints_lms_mb(data))
    return html_string
```

Basisprogrammet er selv utviklet i Python, hvor den bruker `PyQt` for dets brukergrensesnitt - men dette selv bruker HTML for å uttrykke sidemeny. Det var derfor mulig å direkte gjenbruke deres funksjoner for implementasjonen av sidemenyet[8] - i tillegg til å faktisk bruke det allerede eksisterende parametersystemet til å inneholde alt av det `data` skulle i koden over, og enkelt kalle til utregninger når den trenger det gjennom det.

Gjennom dette, var det faktisk veldig enkelt å implementere deres kode inni vår applikasjon. Det var så lik at vi klarte å gjenbruke en del kode direkte fra

[7] `https://github.com/ifarup/ciefunctions/blob/master/tc1_97/description.py` , hentet 21.05.2024

[8] `https://github.com/ifarup/ciefunctions/blob/master/tc1_97/description.py`, hentet 21.05.2024

modulen. Dette er fint for konsistenthet mellom applikasjonene. Nedenfor er en seksjon av vår kode som er da motparten til den over, slik at leseren selv kan se hvor like våre implementasjoner er:

Kodeliste 7.12: Utdrag fra vår kode, direkte basert på koden over.

```
def LMS_MB_sidemenu(parameters):
    html_string = ""
    html_string += _head()
    data[' _min '] = data['min']
    data[' _max '] = data['max']
    data[' _step '] = data['step_size']
    data['info'] = True
    info = compute_MacLeod_Modular(data)
    data['norm_coeffs_lms_mb'] = info['norm']
    data['lms_mb_white'] = info['white']
    data['lms_mb_tg_purple'] = info['tg_purple']

    html_string += styles.description._heading(u'MacLeod\u2013Boynton␣ls␣chromaticity
␣␣␣␣␣␣␣␣diagram')
    html_string += (
        styles.description._parameters(data) +
        styles.description._coordinates('\\(l_{\,\mathrm{MB},\,%s,\,%d}\\)' %
                                (data['field_size'], data['age']),
                                '\\(m_{\,\mathrm{MB},\,%s,\,%d}\\)' %
                                (data['field_size'], data['age']),
                                '\\(s_{\,\mathrm{MB},\,%s,\,%d}\\)' %
                                (data['field_size'], data['age'])) +
        styles.description._wavelenghts(data) +
        styles.description._normalization_lms_mb(data) +
        styles.description._LMS_to_lms_mb(data, data) +
        styles.description._precision_lms_mb() +
        styles.description._illuminant_E_lms_mb(data) +
        styles.description._purpleline_tangentpoints_lms_mb(data) )

    return html_string
```

En liten endring å markere er skiftet fra lokalt til nett. I basisprogrammet var det installert et bibliotek som heter `MathJax`[9] for formatering av matematikk i tekst - men i og med at vår applikasjons struktur gjorde det umulig å passere dette videre med hver ressurs, så ble bruken av biblioteket skiftet til å heller kalle til biblioteket over en lenke gjennom `<script>`, noe som ser ut som en vanlig metode. Vi hadde også implementert filens CSS som en innebygd `<style>` for å minske kompleksitet med å tjenestegjøre for den individuelt.

Endepunkt for diagram

Det var opprinnelig satt opp til at frontend selv skulle utføre koden og logikk for diagrammene i webapplikasjonen, men med strukturen av ressurser fra backend fikk det til å kreve flere forespørsler, så ble det vanskelig å få det rett og i tid til prosjektets slutt. Derfor som en midlertidig løsning til dette (og basert på det tidligere arbeidet hos endepunkt for sidemeny), så ble det avgjort seint at backend

[9] `https://www.mathjax.org/`, hentet 20.05.2024

skulle utgi et HTML av diagrammet som en ressurs, med tanke at den kunne enkelt lastes inn gjennom bruken av et `<iframe>` hos frontend. Fordelen her var at vi kunne enkelt redusere forespørselsmengden helt ned til 1, men på bekostningen at dette var ment til å være den 'gamle' metoden å gjøre det på - og at arbeidet burde heller ha vært opptil frontend.

Implementasjonen av dette innebærer to steg:

- **Utregninger:** Først må alle relevante utregninger utføres slik at de kan bli uttrykt på diagrammet i det første.
- **Innebygd kode:** Etter at alle utregninger er laget, genereres det et HTML dokument i form av en stor string. Det tilsettes relevant CSS og JS innebygd inni dokumentet, sammen med utregningene på en form de kan tolkes av JS.

Vi skal referere til vedlegg H for videre forklaringer med eksempler fra kode. Det anbefales sterkt at resten av dette leses sammen med det, for det skal ikke tilsettes flere kodelister for resten av dette delkapittelet.

Hos den første kodelisten for utregningsdelen av implementasjonen, ser vi at vi utfører utregningene nødvendig for at diagrammet skal vises - og at de lagres i JSON enn i en `dict` denne gangen. Dette er for å gjøre det mulig å overføre utregningene til den innebygde JavaScript koden hos dokumentet uten noe risiko for tap eller feillesing i passeringen. Vi lager også eventuelle titler for diagrammet gjennom Python for å ta bruk av dets formateringer med brukerparametere, men disse lagres kun som `string`. Etter dette, begynner genereringen av den innebygde koden med å først refereres til en `head()` funksjon sammen med utregningsfunksjonen vår.

`head()` spiller en stor rolle i denne modulen, for den bygger både opp strukturen av HTML dokumentet sammen med nødvendige importeringer av biblioteker (som `Plotly`) - men lager også interaktive avmerkningsbokser inni det gjennom `checkboxes()`. Denne funksjonen tar utregningsfunksjonen, og bruker den til å enten aktivere eller deaktivere visse bokser basert på funksjonen den er gitt som parameter (derfor vi inkluderer utregningsfunksjonen inni `head()`). Den tilsetter deretter det inni dokumentet som HTML elementer i en boks med `Flex` med `<span>` for å holde dem horisontal i applikasjonen.

Når dette er ferdig, begynner prosessen av å sette sammen den innebygde JS koden inni dokumentet. Vi starter med å tilsette utregningene inni koden i form av `string` med JSON innhold - slik at de kan lastes inni koden uten noe konflikter fra dets opprinnelige kode. Utregningene må transponeres slik at strukturen endres til kun fire `array` - hvor den første korresponderer til alle bølgelengder, og resten er for relaterte grupper av tall. Med dette, blir våre utregninger klare for bruk i et diagram.

Vi følger opp dette med å initialisere innstillinger for et `Plotly` diagram med bruken av `config` og `layout` variabler. Innen disse, tilsettes det våre formaterte

`string` med bruk av `'` for å plassere det riktig inn, og ikke ha det reagere med `"""` i Python. Det settes også en konstant `width` og `height` for å forsikre riktig sideforhold (1:1) for kromatisitetsdiagram.

Etter dette, gjør vi klar våre diagram medlemmer med å gjøre dem også om til `Map`, hvor vi detaljerer eksakt hvilke deler av utregninger skal vises på diagrammet sammen med eventuelle andre justeringer for dets utseende. Etter deres deklarasjoner, tilsetter vi funksjonaliteter til avmerkningsboksene laget tidligere gjennom bruken av `addEventListener`. De lages sist i koden for å kunne korrespondere med navnene gitt til diagram medlemmer, hvor justeringer til dem endres på avhengig av status hos boksen - og endelig med alt dette ferdig, tegner vi (og oppdaterer diagrammet gjennom `eventHandler` diagrammet gjennom `react()` metoden fra Plotly.

Noe å legge merke til, er at denne koden kunne ha enkelt blitt delt inni biter og modulisert videre til funksjoner. Den kunne ha vært mer optimalisert med tanke på utregningene. Dette ble gjort med vilje, og skal tas opp senere i rapporten ved diskusjon.

Feilhåndtering

Den siste tingen vi ønsker skrive om for backend, er dets vei på å håndtere programfeil. Vi har allerede sett instanser av dette tidligere i teksten, men det er virkelig viktig for nettrammeverk å avgi meningsfulle feilmeldinger. For denne grunn, ble følgende kode skapt:

Kodeliste 7.13: Utdrag fra vår kode, direkte basert på koden over.

```
@api.exception(SanicException)
async def error_handler(request, exception):
    [title, message, suggestion] = exception.args[0]
    json_error = {
        "error": title,
        "status_code": exception.status_code,
        "message": message,
        "suggestion:": suggestion,
    }
    return json(json_error, status=exception.status_code)
```

Dette er en egen feilhåndteringsfunksjon som vil ta å fange alle instanser av `SanicException`. Lesere kan kanskje huske at vi utføre spesifikt `raise` med denne typen eksempsjon, sammen med meldinger som beskrev programfeil og hva kan gjøres bedre. Det er ikke mye å si om dette, men det er fint å nevne med tanke på HATEOAS fra . I tillegg til at brukeren skal kunne vite alt nødvendig fra hjemmesiden, hjelper disse meldingene en potensiell bruker med å navigere seg rundt gjennom riktigere bruk av parametere.

Det er også tilsatt spesielle feilhåndteringsmeldinger for visse situasjoner:

- Dersom brukeren forsøker å dra til den tidligere versjonen av tjenesten, vil de få en feilmelding som forteller at tjenesten er ikke aktiv.

- Dersom brukeren forsøker å utføre en forespørsel med en HTTP metode som er annerledes enn `GET`, vil tjenesten fortelle at metoden er ikke støttet av tjenesten.

Kapittel 8

Testing & kvalitetssikring

Dette kapittelet drar til testing og kvalitetssikringen utført på ...

8.1 Testing

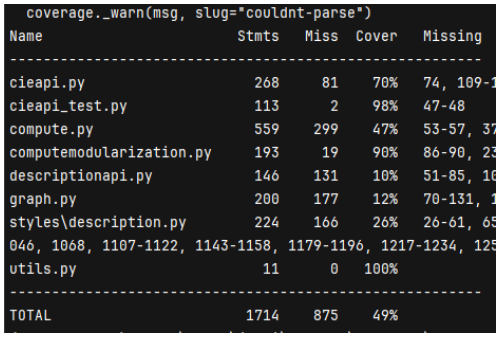
8.1.1 Enhets- & Integrasjonstesting

Over utviklingen av backenden hos tjenesten, har det vært utviklet flere enhets- og integrasjonstester. Vi startet med en testplattform i 6.2.1, før den ble oppgradert til å være en del av enhetstestene vi hadde - og nå er et komprehensiv plattform som dekker både enhets og integrasjonstester tilgjengelig gjennom `cieapi_test.py` filen i repositoriet.

Våre tester i backend forsørger seg at følgende er testet:

- Om vår JSON-enkoder gir data som kan ordentlig tolkes i JSON, gjennom bruken av `json_lint_test()` og korresponderende funksjoner (`test_ndarray_to_JSON()`, `test_advanced_ndarray_to_JSON()`, `test_write_to_JSON()`).
- Om alle av våre endepunkter hos tjenesten gir både de forventede resultater og riktige statuskoder. Dette inkluderer:
    - Nesten alle utregningsendepunkter, hvor det sendes en forespørsel til seg selv for å hente utregninger, laster det inn som JSON - og deretter sammenligner det med en `csv` fra basisprogrammet for de samme brukerparameterne. Koden klarer å laste begge inn uten feil i presisjon, før det sammenligner innholdet og struktur av begge. Denne testen inkluderer ikke `/xyz-p` endepunktet, for basisprogrammet klarer ikke å produsere `csv` filen nødvendig.
    - Alle andre endepunkter testes også, hvor de skal returnere en ønsket statuskode.

Dette ble utført både for det tidligere Flask rammeverket (gjennom Unittest biblioteket) - og en til gang for det nye Sanic rammeverket (gjennom Pytest for å støtte asynkrone forespørsler til seg selv). Med bruk av `coverage` for den aller siste versjonen av programmet, ble det funnet ut at `cieapi.py` har 70% dekning



Figur 8.1: Skjermbilde av testdekning for utrullet versjon av applikasjon.

av testene - mens `computemodularization.py` har 90%. Et skjermbilde av kodedekningen finnes på figur 8.1.

I tillegg til at dette dekker enhetstester, utfører det også integrasjonstester i og med at den sender forespørsler til seg selv. Gjennom denne prosessen, testes integrasjonen av utregninger sammen med alle andre systemer satt på plass (parametervalidering, feilhåndtering, formatering) - noe vi ser fungerer helt greit for applikasjonen.

Naturligvis, så er testene også godt dokumentert. Alle dekninger underveis for applikasjonen er tilsatt i et eget dokument i direktoriet for testing - hvor den inkluderer også brukerparametere for `csv` filene slik at det kan reproduseres.

Kodeliste 8.1: Backend - Eksempel av endepunktstestfunksjon for enhets- og integrasjonstest.

```
# test for macleod
@pytest.mark.asyncio
async def test_lmsmb_endpoint():
    truth_data = load_csv_to_array("./tests/LMS-MB-5-63-1.csv")
    req, response = await cieapi.api.asgi_client.get(
        "/api/v2/lms-mb/calculation?field_size=5&age=63"
        )
    assert np.all(np.array(json.loads(response.body)['result'])
        == truth_data) == True
    assert response.status == 200
```

8.1.2 Belastnings- & Stresstest

Gjennom bruk av belastnings- og stresstester, ønsket vi å undersøke hvor god ytelse tjenesten hadde i realistiske situasjoner. For denne grunn ble Postman brukt til å sende trafikk til den utrullede applikasjonen - hvor mengde av 'virtuelle brukere' (hvor mange sender forespørsler konstant gjennom testen) ble justert for å tilsette variabelen av belastning.

Vi kjørte først en stresstest for å se hvor mange forespørsler tjenesten skal kunne ta hvert sekund, og fortsatt være innenfor vårt operasjonelt krav om 'responstid under 1000 ms'. Denne testen sender forespørsler til alle tre former av ressurser

tilgjengelig hos hvert utregningsendepunkt, så vi ser på den gjennomsnittlige responstiden for å få en sjekke om vi faller fortsatt innenfor kravet. Det vi kom frem med, vises i vedlegg I.

Vi ser at tjenesten kan ta inn 9.73 forespørsler i sekundet og fortsatt operere innenfor våre krav om vi ser kun på gjennomsnittstiden for hver forespørsel. Det er viktig å merke at en del av endepunktene var over dette kravet, men disse er de intensive utregningsfunksjonene vi var allerede kjent med til å være vanskelig å holde innenfor kravet. Allikevel, ser vi at gjennomsnittstiden ligger på 958 ms dersom 11 virtuelle brukere sender forespørsler konstant til tjenesten.

Når det gjelder belastningstester, tok vi basis i dette, og utførte en test med to virtuelle brukere. Denne testen vil ikke se på gjennomsnittstiden slik den forrige, men heller om tjenesten kan sende forespørsel under 1000 ms hver for seg, slik at kravet kan i hvert fall oppholdes for dem. Resultatene av denne testen vises i vedlegg J.

Vi ser at testen klarer å takle mengden av brukere helt fint under de operasjonelle kravene. Den gjennomsnittlige responstiden er 307 ms, hvor den lengste forespørselen (`/xy?plot`) tok 884 ms gjennomsnittlig. Tjenesten behandler 4.55 forespørsler i sekundet, og med tanke på hvordan frontend trenger 3 for hver gang en bruker skal bruke den tjenesten, så betyr det i teorien at de kan enkelt hente inn alle ressurser de trenger innenfor kravet vårt i lett nettverkstrafikk.

Vi mener at vi er fornøyd med resultatene fra ytelsestestene vi har utført her. Det er allikevel viktig å nevne at tjenesten ble utrullet med fire arbeidsprosesser - og at det vil finnes potensiale å gjøre ytelsen bedre med å enten øke denne mengden, eller med å utrulle flere servere med applikasjonen; noe mer relevant for det neste kapittelet.

8.2 Kodekvalitet

I tillegg til testene beskrevet tidligere, har vi forsikret oss god kodekvalitet i en rekke andre aspekter. De ligger beskrevet under:

8.2.1 Frontend

Tiltak for å sikre god kodekvalitet i frontend delen av prosjektet omfatter ESlint, React.Strictmode og statisk typesjekking gjennom TypeScript. ESlint er et fleksibelt verktøy som legger til rette for å inkludere anbefalte regler fra TypeScript, JavaScript og React, samt å definere egne regler for kodepraksis. Ved brudd på de definerte reglene vil man få advarsler eller feil, noe som gjør det lettere å fange opp og rette opp i dårlig kodepraksis og struktur.

React.Strictmode er et verktøy som aktiverer flere sjekker og advarsler når det oppdager potensielle problemer knyttet til livssyklus, bruk av avviklede tredje-

partsbiblioteker og bruk av funksjoner som er frarådet i moderne React applikasjoner [48].

TypeScript har, som nevnt i kapittel 5.2.2, funksjonalitet for statisk sjekking av variabeltype, noe som sikrer at variabler og returverdier er av riktig type under utvikling, og hindrer bugs og kjøretidsfeil.

8.2.2 Backend

- Det ble brukt PEP8 som en kodingsstandard i backend [49]. Denne standarden detaljerer regler angående innrykk av koden, navn, mellomrom og flere andre for å forsikre seg at koden en lager er fin og god. Dette ble forsikret seg gjennom PyCharm sine innebygde funksjonaliteter for det, sammen med bruk av biblioteket autopep8[1] for å automatisk formatere det.
- Det ble brukt versjonskontroll av ferdig utviklede iterasjoner av API, hvor Flaskrammeverket var `v1`, og Sanic er `v2`.
- Det finnes detaljerte kommentarer som beskriver hvorfor noe gjøres, sammen med hva en implementert bit er i tilfeller hvor kode kan være vanskelig å forstå.
    - Alle kommentarer har også med ordentlig henvisning av gjenbrukt kode (mest primært for kode fra basisprogrammet, hvor det oppgis lenke, modul og kodelinjer), men også en instans av noe basert på et svar fra nettsider som StackOverflow.

8.3 Lisensiering

Selv om dette er ikke en del av testing og kvalitetssikring, ønsker vi allikevel kort å nevne bruken av lisensiering. Som en del av dets rolle som en kontinuering av basisapplikasjonen, så vil vårt prosjekt også få en lisensiering. Denne lisensen er av GNU General Public License [2], og forsikrer seg programvare er fri programvare[3].

Gjennom denne lisensieringen, forsikrer vi oss at applikasjonen er fri, noe vi mener kontribuerer til å øke programmets brukbarhet i fremtiden. Dette kan gjøre programmet bedre å jobbe med i fremtiden dersom oppdragsgiver ønsker å ekspandere på prosjektet etter den endelige tidsrammen.

[1] `https://pypi.org/project/autopep8/`, hentet 20.05.2024
[2] `https://www.gnu.org/licenses/gpl-3.0.en.html`, hentet 20.05.2024
[3] `https://no.wikipedia.org/wiki/Fri_programvare`, hentet 20.05.2024

Kapittel 9

Utrulling & Installasjon

9.1 Utrulling

For utrullingen av tjenesten, bestemte vi oss å bruke Docker og dets kontainerbaserte løsninger. Vi ønsket å pakke inn vår applikasjon med alle dets avhengigheter til tredjepartisbiblioteker inni kontainere. Sammen med en `Dockerfile` for å inneholde operasjoner, kunne vi enklere starte opp system for det, og derfor enklere for oppdragsgiver.

Den første ideen var å bruke en kontainer for både tjeneren og frontend, hvor backend skulle tjene frontend gjennom seg selv. Dette var vurdert i møter, men for oppgaven valgte en annen løsning: Å videre dele opp komponentene som skulle tjenestegjøres, med både backend og frontend som egne kontainere. Mens backend skulle kjøres av Sanic sin egne webserver med fire arbeidsprosesser, skulle frontend tjenes av en webserver tilgjengelig i biblioteket Serve[1]. Det kan menes at dette er unødvendig og tilsetter kompleksitet til arkitekturen, men vi mente at denne separasjonen økte tilgjengeligheten av hele tjenesten. Med å ha dem som separate komponenter, kan brukere i hvert fall fortsatt ha nettsiden fremme dersom det er høy trafikk hos backend.

En annen fordel med denne oppdelingen er at det skal være enklere å øke skalerbarheten for backend; med å ha dem separat, så kan en eventuell bruker av programvaren lage flere instanser av kontaineren for backend.

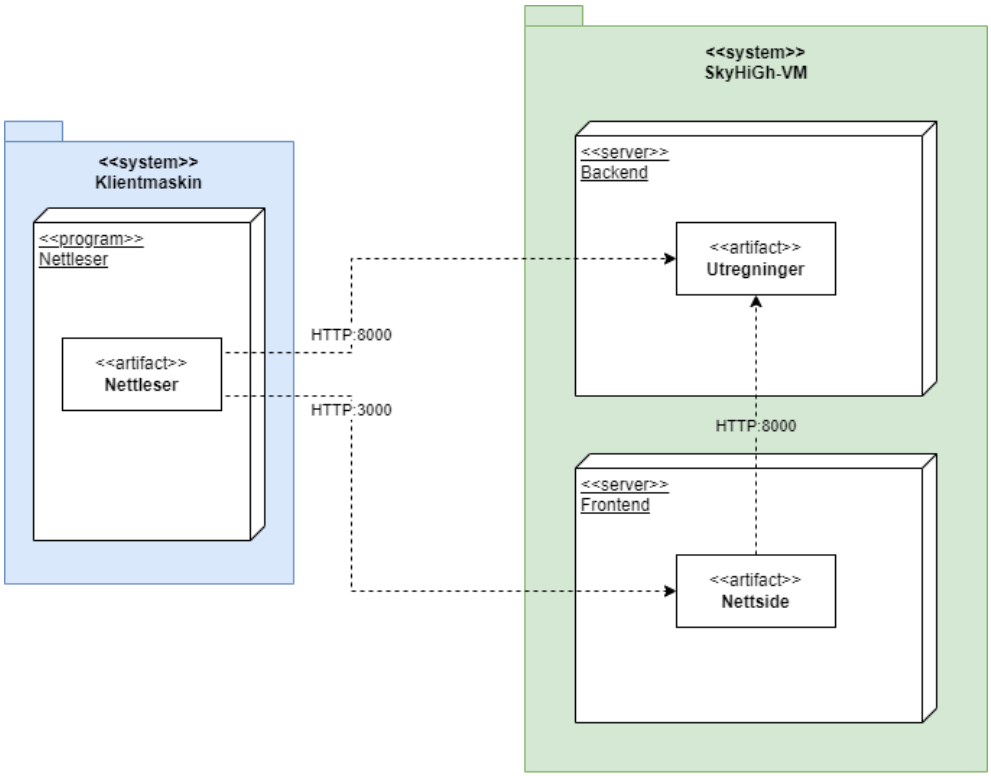
Bruken av dette oppsettet dannet to `images` fra Docker. For å få dem til å begge kjøre, brukte vi `docker-compose` for å kjøre en multi-container Docker applikasjon. På en slik måte, utfører vi at sluttbruker har separat tilgang til både applikasjonen i frontend, og API i backend. Vi benytter oss av 'port mapping'[2] for å justere nettverksportene for hver av komponentene. Backend-tjenesten kjører på port `:8000` inne i containeren og er tilgjengelig på samme port på vertsmaskinen. Frontend-tjenesten kjører på port `:3000` inne i containeren, men er

[1] `https://www.npmjs.com/package/serve`, hentet 20.05.2024

[2] `https://docs.docker.com/network/#published-ports`, hentet 20.05.2024

tilgjengelig på port `:80` på vertsmaskinen. Dette gjør at backend kan nås via `http://<server-ip>:8000`, og frontend kan nås via `http://<server-ip>:80`.

En enklere presentasjon av hele utrullingen finnes i figur 9.1.



Figur 9.1: Utrullingsdiagram for hele tjenesten
Klientmaskinen kommuniserer gjennom HTTP og porter til enten tjeneren til frontend, eller tjeneren til backend. Frontend vil selv kommunisere med tjener til backend.

I sammenheng med denne bacheloroppgaven blir utrullingen et bevis av konsept. Det er produkteier som må ta den offisielle utrulling videre. Det ble i denne sammenheng anskaffet en virtuell maskin gjennom NTNUs SkyHiGh tjeneste for virtuelle maskiner og servere. Hos en utgitt virtuell maskinen, ble vårt GitLab prosjektet klonet og satte på maskinen. Etter at vi hadde skapt våre kontainere på våre lokale maskiner, var nå `docker-` og `docker-compose` filer ferdig satt opp hos maskinen. Den eneste endringen som trengtes var å endre konstanten for APIets URL. Denne bruker nå den virtuelle maskinens IP adresse istedenfor `localhost`, som benyttes når applikasjonen kjøres lokalt. Når dette var gjort var det bare å bygge Docker, og vi fikk det til å starte. Applikasjonen og API er nå tilgjengelig for en hver bruker som er tilkoblet NTNUs nettverk.

Vi hadde skrevet tidligere at maskinvaren brukt var utenfor vår oppgave, med

tanken at oppdragsgiver skulle selv velge det som passet best for dem å tjenestegjøre applikasjonen i fremtiden med - men for å vise kompetanse, ønsker vi også å skrive noen anbefalinger.

Som sagt, benytter Sanic flere arbeidsprosesser for å øke skalerbarheten til tjenesten. Denne prosessen krever flere CPU-kjerner, hvor vi kan anta at to arbeidsprosesser passer per kjerne. I serveren vi benytter finnes det to kjerner, så vi har satt tjenesten til fire arbeidsprosesser, noe som gir gode resultater fra 8.1.2. Dersom oppdragsgiver ønsker å øke skalerbarheten, er det første gode steget å øke antall kjerner på hovedprosessoren til en virtuell maskin, og deretter kjøre containeren med backend med nye innstillinger for arbeidsprosessene.

De andre spesifikasjonene er ikke så viktig, utenom at det burde være minimum 2 GB RAM og nok lagringsplass til tjenesten. Operativsystemet til programmet er heller ikke viktig, for Docker sine konteinere er designet for å kjøre uavhengig av operativsystem. Allikevel, anbefaler vi Linux-baserte distribusjoner for at Docker skal kjøre innebygd i systemet (for det bruker virtuelle maskiner på andre operativsystemer). Linux tilbyr også et sterkt miljø for nettverksadministrasjon og sikkerhet, med enklere oppsetting og vedlikehold av maskinene.

9.2 Installasjon

Programmet har blitt om til kontainere gjennom Docker, og skal derfor være enkelt å installere på flere typer enheter. Sanic støtter også Docker[3], så man kan kjøre følgende prosedyre for å sette opp miljøet:

1. Last ned Docker hvis det ikke allerede er installert.
2. Klone prosjektet fra fra GitHub gjennom disse lenkene:
    - SSH: `git@github.com:group8ntnu2024/ciefunctionsweb.git`
    - HTTPS: `https://github.com/group8ntnu2024/ciefunctionsweb.git`
3. Naviger til rotmappa til prosjektet
4. Kjør:

```
docker-compose up --build
```

For at en klient skal kunne kommunisere med Sanic-konteineren, må portene 8000 og 80 være åpne for trafikk i begge retninger. Dette sikrer at forespørsler og svar kan flyte fritt mellom nettleseren og Sanic-tjeneren og webserveren.

9.3 Demo

Vi har allerede installert programmet på en Ubuntu-server, og den kan besøkes på følgende nettadresse:

[3] `https://sanic.dev/en/guide/deployment/docker.html`, hentet 20.05.2024

http://10.212.136.66/

Linken gir tillgang til tjenesten som er rullet ut på serveren slik at den som ønsker kan navigere og teste funksjonalitetene som har blitt utviklet og implementert. Vær oppmerksom på at denne lenken kun vil fungere innenfor NTNU sitt nettverk. Dette betyr at tjenesten er tilgjengelig for testing og bruk for alle enheter som er koblet til NTNUs interne nettverk, og den gir en mulighet for å se løsningen i aksjon.

Kapittel 10

Diskusjon & Konklusjon

Gjennom prosjektperioden, har vi gjennomgått både utfordringer og suksesser som har vært med på å forme vårt endelige produkt. I dette kapittelet vil vi reflektere over vår hvordan vi har arbeidet, diskutere resultater og trekke konklusjoner basert på det vi har gjort og lært.

10.1 Drøfting

10.1.1 Resultatsmål

Vi mener at vi har klart å oppfylle det resultatsmålet forventet av oss. Gjennom dette prosjektet, mener vi at vi har klart å lage den ferdigutviklede webapplikasjonen slik vi har fortalt om den tidligere i kap. 7.

Programmet dekker alle kravene vi har satt for oss slik skrevet i kap. 3.

- Mesteparten av våre funksjonelle krav er dekket, slik de er beskrevet gjennom use-case og deres diagram. Det er mulig av en bruker å 'velge funksjon og brukerinput', for å da få representasjoner av utregninger basert på det de ga. De kan også interagere med grafer, finne spesifikke verdier, og aktivere valgfri parametere, i tillegg til flere andre funksjonaliteter som finnes i basisprogrammet.
    - Noen ting forblir uferdig, men de tas opp i kap. 10.3.
- Programmet dekker også alle ikke-funksjonelle krav;
    - Applikasjonen har et brukergrensesnitt som er likt til det hos basisprogrammet.
    - Applikasjonen følger ytelseskravet vi har satt her, som bevist i våre tester hos kap. 8.1.2. Vi ser at ved vanlig belastning, så klarer den å gi responstider raskt, gjennom bruken av moduliserte funksjoner slik beskrevet hos kap. 6.2.1 og gode tiltak for skalerbare tjenester slik fortalt i kap. 9.1. Dette punktet gjelder også for det tredje kravet om skalerbarhet.

    - Applikasjonen skal ha åpen lisensiering slik vi hadde dekket det i 8.3. Dette dekker også gjenbruken av kode. Vi har også sørget for dokumentasjon om utrulling hos `README` filen på kodelageret.
- Vi har også sett gjennom kap. 8.1.2 at våre operasjonelle krav dekkes i det siste produktet, mesteparten. Tjenesten vår klarer å takle flere brukere samtidig gjennom Sanic sine asynkrone evner, samtidig som at gjennomsnittstiden for alle forespørsler fra tjenesten ligger under 1000ms.

10.1.2 Effekt- & Læringsmål

Med tanke på lærings- og effektmålene vi hadde deklarert tidligere i prosjektet i kap. 1.2.2, mener vi at vi har klart å utføre det vi hadde ønsket der.

Når vi ser på effektmålene, mener vi at vi har klart å oppfylle dem til en stor grad. Gjennom vår raske og høyt-tilgjengelige (kap. 8.1.2), mener vi at vi har definitivt gjort det enklere for fargesynsforskere å få de beregningene de trenger. Dette gjør også opplevelsen mer komfortabel til forskere - hvor alle beregninger er tilgjengelig på en enkel nettside, istedenfor en egen skrivebordsapplikasjon. Vi har også lagt merke til at vi selv har blitt mer interessert i forskningsemnet. Vi satte ekstra oppmerksomhet i kap. 2.1 med omfattende møter med og tilbakemeldæinger fra oppdragsgiver og veileder - som begge er medlemmer av Fargelabben.

Når det gjelder våre læringsmål, tenker vi også at vi har fått mye ut av dette prosjektet.

Vi har definitivt lært mye om de tradisjonelle nettrammeverk sammen med nyere teknologier, spesielt fra testingen og vurderingen av rammeverk. Vi kan egentlig si at prosjektets essens baserer seg rundt dette: Hvordan vi gikk fra bestemmelsen mellom PyScript og Flask og deretter til å revurdere Flask med andre rammeverk. Dette har gitt oss et stort innsikt i utviklingen av API og webutvikling generelt.

Vi har også lært mer om relevante og aktuelle klientrammeverk innen webutvikling. Gjennom våres grader, lærer vi vanligvis om det elementære - så vi har ikke fått en reell sjanse å se på slike teknologier før. Selv om det var en risiko med tanke på oppgaven, gikk vi fremdeles med å bruke ny teknologi som er relevant i arbeidslivet, slik at vi kunne spesifikt lære mer om det. Når utviklingsprosessen var konkludert innså vi hvor lurt dette var. Vi har alle oppnådd nye høyder innen kodeforståelse og webutvikling .

Det har også blitt lært mye om programvareutvikling og erfaring med å jobbe i grupper. Selv om vi har hatt gruppeprosjekter for store programmer jevnlig gjennom våre grader, så er en bacheloroppgave langt større enn dem. Derfor satte vi ekstra fokus på å utføre profesjonelle etikette og få så mye erfaring fra dette som mulig, noe vi mener vi har lykkes i både gjennom ulemper og fordeler. Dette tas opp mer i diskusjon i evalueringen av gruppearbeidet, hos kap. 10.4.

Til slutt, mener vi at vi har lært flere ting for det siste målet; om hvordan vi skal

formidle dette til en rapport og presentasjon. Etter omfattende tilbakemeldinger fra vår veileder, ble det satte inn mye innsats av alle parter i å søke forbedring. Vi lærte mye fra dem, som blant annet riktigere struktur av diagrammer og bedre ordbruk for å unngå misforståelser i teksten.

10.1.3 Bærekraft

Vi mener også at prosjektet dekker flere aspekter innenfor bærekraft, noe vi skal ta for oss gjennom følgende delkapitler innom det miljømessige, det økonomiske og det sosiale.

Sosial bærekraft

Et felt innen bærekraft som vårt prosjekt falle innunder, er sosial bærekraft. Vi bidrar til samfunnet med å tilby en enklere måte for fargesynsforskere å få tak i sentrale utregninger nødvendig for deres forskning. Dette kan videre tilknyttes med FNs bærekraftmål [50], hvor dette passer godt inn hos delmål 9.5[1]: Vi styrker forskning i sektorer hvor fargematchfunksjoner er relevant med å tjenestegjøre med vår applikasjon.

Et annet felt som vi styrker til den sosiale bærekraften, er gjennom den enkle tilgjengeligheten av applikasjonen. Fordi den finnes over internettet, kan alle slags brukere enklere finne frem til utregningene. Dette styrkes videre av det responsive og universelle utseendet programmet har, slik vist frem tidligere i 7.3.

Miljømessig bærekraft

Vår applikasjon bidrar også i stor grad til miljømessig bærekraft.

Først og fremst, bruker programvaren sofistikerte teknikker hos alle komponenter for å forsikre seg mindre unødvendige utregninger. Gjennom React sin optimaliserte 'rendering' (5.2.1), TypeScript sin type-sikkerhet (5.2.2), backend sine feilmeldinger (7.2.2) og mulighet for cache (3), så bidrar vi til at tjenesten unngår å utføre unødvendige utregninger som tar opp tid og energi.

Programmet er også pakket inni kontainere fra Docker, som bidrar til bruken av mindre ressurser for å tjenestegjøre programmet. Til sist, gjenbrukes det eksisterende kode fra kodelager med åpen kildekode, blant annet basisprogrammet og tilgjengelige biblioteker. Gjennom deres bruk, reduserer vi ellers mengden vi ville ha trengt for å lage det selv, sammen med at vi gjenbruker noe som allerede eksisterer.

Økonomisk bærekraft

Til sist, dekker programmet også en liten del innenfor økonomisk bærekraft. Koden vi har laget for dette prosjektet skal være fri kildekode med åpen lisens, slik

[1] `https://fn.no/om-fn/fns-baerekraftsmaal/industri-innovasjon-og-infrastruktur`, hentet 20.05.2024.

tidligere sagt i kap. 8.3. Dette bidrar til at flere målgrupper kan bruke programvaren, irrelevant av deres økonomiske status.

10.1.4 Bruk av kunstig intelligens

Bruk av KI i akademia byr på en del etiske og moralske spørsmål. Overbruk eller avhengighet av KI kan undergrave studentenes egen læring og kreativitet. Det er viktig å benytte KI på en måte som fremmer og ikke hindrer kritisk tenkning.

På grunn av dette, har det blitt satt fokus på å bruke KI minimalt for dette prosjektet. Vi ble tidlig enige om at vi ikke ønsket å benytte det i stor grad, og heller vise frem hva vi selv klarer å produsere. Når det er sagt, er kunstig intelligens et veldig brukbart verktøy som kan gi store fordeler når det brukes moralt og teknisk riktig. For oss var dette først og fremst feilsøking. Kodetolkeren til mange tilgjengelige kunstige intelligens finner ofte fort slurvefeil eller forglemmelser man har gjort i løpet av koding. For dette, har KI da blitt brukt som et verktøy for å oppdage disse feilene. I tillegg til dette, har også KI blitt brukt som en læringsassistent. Konsepter vi støtte på under utvikling og rapportskriving som vi slet med å forstå kan KI lett gi oss essensen av uten at man må lese gjennom mange artikler. Dette gir oss en overordnet forståelse som gjør videre læring betydelig lettere.

KI ble benyttet til å generere små deler av kode for oss. Dette er kun småting som ikke faller innenfor kodeforståelse, og kunne ha blitt gjort selv. Et eksempel er fargekoder for elementer i CSS-filene og importeringer for eksterne biblioteker. Et unntak til dette var når vi skulle implementere Docker for tjenestegjørelse. Utrulling er en veldig intrikat prosess hvor mye kan gå galt. Da ble KI benyttet for å få et førsteutkast av dockerfilene. Det er viktig å merke at dette virket ikke 'rett ut av boksen' og vi måtte gjøre betydelige endringer for å få det til å fungere som det skulle.

Når det kom til rapportskriving, ble kunstig intelligens benyttet kun i renskriving, og ikke til noe generering. Når vi leste igjennom og oppdaget at setningsbyggingen var litt merkelig eller for lang, ble det av og til benyttet KI for å strukturere setninger anderledes for å øke lesbarheten for rapporten. Vi forsikret oss om at det ikke forekom noe tap av innhold eller mening. Det var også anbefalt et verktøy kalt LanguageTool[2] av vår veileder for renskriving, men dette fant vi lite nytte i. Verktøyet støttet ikke norsk helt, og ble kun brukt for å sjekke feilstavelser av ord.

10.2 Kritikk av oppgaven

10.2.1 Frontend

Frontend til applikasjonen er bygger med React og TypeScript. Siden react er komponertbasert har har vi bygget en modulær og gjenbrukbar kodebase. Dette gjør koden lettere å vedlikeholde og videreutvikle. React har et stort økosystem av og

[2] `https://languagetool.org/`, hentet 21.05.2024

tredjepartisbiblioterker som har vært til stor hjelp og akselerert utviklingsprosessen.

Det har også byd på noen utfordringer ved å benytte React. Læringskruven har vært ganske bratt og det var ganske innviklet og vanskelig å forstå i oppstartsfasen. På grunn av dette bygget vi inn en del grunnleggende hindringer for oss selv som var vanskelig å nøste opp i senere. Dette var spesielt et problem når det kom til design av layout og flytte rundt på komponenter senere i utviklingsfasen. React krevde også mye oppstartsarbeid for å få det til å fungere i det lokale utviklingsmiljøet.

Vi hadde alle litt erfaring med JavaScript fra før så overgangen var ikke veldig utfordrene.

Selv om vi har hatt en fokusert og effektiv utviklingsprosess, må vi innrømme at vi dessverre ikke har utført noen formelle tester på frontend-komponentene. Dette er et område som krever oppmerksomhet i fremtidige iterasjoner av prosjektet for å sikre at brukergrensesnittet fungerer feilfritt og gir en optimal brukeropplevelse. Til tross for dette har den praktiske brukstesten vist at løsningen fungerer tilfredsstillende i de fleste scenarioer.

10.2.2 Backend

Når det gjelder backend, ville vi ha sagt at vi er fornøyd med flere aspekter, men også at det finnes ulemper med vår implementasjon.

Den største delen vi er fornøyde med, er dets optimalisering gjennom prosjektutviklingen og ytelsen den tilbyr som en tjeneste. Utover hele prosjektperioden, har det hele tiden vært et stort fokus på å utlevere noe raskt og effektivt. Det som startet som en enkel prototype med lite initiell funksjonalitet, ble eventuelt om til en komplett tjeneste med forbedringer og optimaliseringer i flertall. Det ble gjort flere typer av optimaliseringer fra det vi startet med; fra å modulisere utregningsfunksjoner, til et komplett skifte av rammeverket vi bruker. Alt av dette ble gjort for å minske på responstiden og gjøre programvaren vi utleverer så tilgjengelig og brukbar som mulig. Vi tar mye stolthet i dette, og er derfor aldeles fornøyd med ytelsen tjenesten tilbyr. Et annet aspekt verdt å ta med, er at det ble ikke brukt noe kunstig intelligens for kodeutviklingen av denne komponenten. Alt av kode som finnes i API for dette prosjektet ble produsert enten manuelt eller med gjenbruk tydelig deklarert i både kode og dokumentasjon. Det ble heller ikke brukt for forslag for løsninger, heller annen form av hjelp der. Vi mener at dette demonstrerer god beherskelse av programmering og oppgaven i sin helhet, og videre styrker på noen av våre læringsmål diskutert tidligere. Til sist, så vi er fornøyd med at det har blitt utført ordentlig enhets- og integrasjonstesting gjennom dets utvikling for å forsikre seg tilgjengelighet og kodekvalitet. Vi ser god kvalitet i det vi har laget, og det er hele tiden noe å være stolt av.

Et stort aspekt vi ikke er fornøyd med, er bruken av diagram som en ressurs. Dersom vi ser på oversikten over alle forespørsler hos vedlegg I, ser vi en klar trend at endepunktene med `/plot` tar lengre responstid enn andre endepunkter. Dette

forårsakes av at tjenesten selv utfører flere utregninger på sin side for å utregne alle verdier. Vi mener også at det skal være vanlig for frontend å takle data fra backend for å skape disse diagrammer; ikke at backend lager dem selv. Noe likt kan sies om sidemeny som en ressurs, hvor klientsiden skal også ha ansvaret for tegningen av dette. Men vi ser at responstiden for dette er ikke så høy som de er hos diagrammer. Om dette skal skiftes, er det opp til potensielle fremtidige utviklere av programvaren.

10.3 Videre opplegg

10.3.1 Manglende funksjonalitet

Selv om programmet vårt oppfyller mange av de grunnleggende kravene for å visualisere data er det noen få funksjonaliteter som ikke ble implementert. Det første, og mest kritiske, er at det ikke mulig å lagre tabellen som vises. Dette begrenser muligheten for videre bruk og deling av utregnede resultater. Videre mangler programmet funksjonalitet for å vise noen av grafene renormalisert. Dette skulle vært implementert gjennom en avkryssningsboks og ville gitt brukerene større fleksibilitet og kontroll over datavisualiseringen. Symbolene på toppen av grafen er heller ikke formatert riktig. Dette kan i verste fall resultere i missforståelser eller feil tolkning av dataene. Til slutt mangler programmet en 'About' seksjon. Denne er viktig for å gi brukerene informasjon om programmets formål, utviklere og versjonshistorikk. Disse manglene bør adresseres i fremtidige iterasjoner av applikasjonen for å forbedre funksjonalitet.

10.3.2 Bedre design

Vi kunne ha lagt mer vekt på dette estetiske designet av applikasjonen. På grunn av prosjektets natur som et vitenskapelig verktøy valgte vi imidlertid å prioritere funksjonalitet og presisjon fremfor estetikk. Vårt hovedfokus var å skikre at applikasjonen kunne utføre de nødvendige beregningene, og presentere dataene på en klar og forståelig måte. Det estetiske aspektet ble derfor anset som sekundært.

En annen ting vi kunne tatt mer i betraktning var universelt design. Vi vurderte å inkludere funksjonalitet for å bedre brukervennligheten og tilgjengeligheten av applikasjonen for mennesker med fysiske nedsettelser. Vi anså ikke dette som veldig kritisk da vi skulle utvikle en applikasjon som kun benyttes av en håndfull forskere verden over. Vi tenkte dette kunne være noe ekstra vi kunne inkludere dersom det var tid til det.

10.3.3 Nyere diagramfremvisning

Måten diagrammene ble implementert på ansees for å være litt gammeldags. Sidemenyen og plottene bygges og leveres direkte av python APIet istedenfor å bli laget av frontenden bassert på data levert av APIet. Dette kommer med noen fordeler og ulemper. Den største fordelen er at plotbyggingen får direkte tilgant til

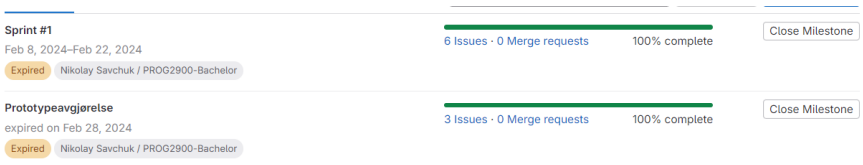
plotpunktene fra compute filen i backend. Dette forhindrer inviklet databehandling som ellers måtte ha blit gjort frontend. En ulempe er at dette gjør forntend avhengig av backend for layout og visning av elementer. I tilleg til at det strider med prinsippet 'løs kobling og høy sammenheng' reduserer det fleksibiliteten og responsiviteten i designet. Diagrammene er ikke responsive i den forstand at de opererer med faste størrelser for å sikre en 1:1 skala. Applikasjonens sidemeny plasserer seg under graf/tabellen hvis vinduet til nettleseren forminskers, men tabellen/grafen blir ikke mindre. Dette betyr at diagrammene ikke tilpasser seg ulike skjermstørrelser optimatlt. Dette kan være problematisk for brukere som ønsker å bruke applikasjonen på forskjellige enheter. Manglen på responsivt design kan redusere den generelle opplevelsen av applikasjonen.

10.4 Evaluering av arbeidet

10.4.1 Utviklingsprosessen

Som skrevet tidligere, valgte vi å bruke scrum for vår utviklingsmetodikk. Vi hadde jobbet med det før med forskjellige grader av suksess, og mente at vi skulle bruke dette prosjektet for å lære oss mer om hvordan man skulle bruke det i profesjonell sammenheng. Nå som vi tenker tilbake på vår prosess, mener vi at det kunne ha gått bedre - men vi er fortsatt fornøyd med hvordan metodikken gikk.

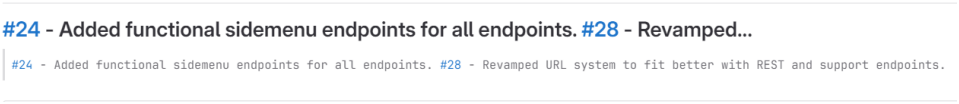
Vi mener at bruken av et scrumboard sammen med sprintperioder gjorde det enklere å organisere hvordan arbeidet skulle fordeles oss i mellom. Det var enkelt å organisere og huske hva vi måtte gjøre til enhver tid, samtidig som det var direkte mulig å endre på de registrerte oppgavene. Sprintperiodene var også satte opp med gode lengder. Vi følte at perioder med lengder av en uke var for korte, og ville føre til mye stress. Det ble også brukt milepæler for å holde kontroll på sprintperiodene, slik vist i figur 10.1.



Figur 10.1: Utdrag av milepæler fra prosjektet.
Milepælene i bildet dekker både den første sprinten, og en del av den andre sprinten hvor avgjørelsen for prototypen måtte utføres.

For å holde styr på prosjektet, benyttet vi oss av GitLab[3] for versjonskontroll som kodelager. Dette har hele tiden vært en standard for oss gjennom våre utdannelser. I løpet av kodeutviklingen, benyttet vi oss bruk av unike IDer tilknyttet våre gitte oppgaver på scrumboardet, slik vist på figur 10.2.

[3] `https://about.gitlab.com/`, hentet 20.05.2024



Figur 10.2: Eksempel på 'commit' med bruk av 'issue' ID for god dokumentasjon av arbeid.

Dette systemet ble brukt i varierende grad, men vi mener allikevel at det er noe vi burde ta med oss til fremtiden. Dette er bedre profesjonell etikette, for det dokumenterer eksakt hvilke oppgaver ble jobbet på til enhver tid.

Noe annet positivt var de ukentlige møtene med både veileder og oppdragsgiver. Det er viktig for oss å lage et prosjekt som holder seg innenfor både hva som er forventet fra en veileder, og hva som er ønsket fra en oppdragsgiver. Det er takket være dem at vårt prosjekt har blitt som det er.

Men, noen ulemper med bruken av scrum var nok våre tilnærminger av arbeidsoppgaver i produktkøen, og vedlikehold av det.

Tidlig i prosjektet lagde vi produktkø uten noe kjennskap til hvor stor eller liten en gitt oppgave skulle være - heller, baserte vi det på våre tidlige estimeringer fra prosjektets start. På grunn av dette, fant vi ut at enkelte arbeidsoppgaver tok lengre tid enn forventet. Dette konkluderte i at flere av sprintene måtte bli utvidet for oppgaver slik presentert i kap. 6. Vi mener at dette var forårsaket av mangel på erfaring innenfor mange nye teknologier vi aldri hadde brukt før. Dette førte til en følelse av at vi ikke gjorde nok, og tidene vi hadde satt av til utvikling gikk rask ut. Mangelen på realistiske tidsrammer og uforutsette tekniske utfordringer bidro til at vi ofte måtte justere tidsplanen vår. Dette skape en del stress og bekymring for om vi ville klare å fullføre i tide.

Scrumsystemet krevde også mye vedlikehold for å fungere ordentlig. Fordi det var ikke mye utvikling på starten av prosjektet, var alt opprettholdt godt i systemet - men etterhvert som utviklingen foregikk og arbeidsmengden begynte å øke, begynte vi å ha lite tid for opprettholding. Dette forårsakte at enkelte deler av systemet som milepæler ble eventuelt ikke brukt videre i prosjektet; kun scrumboard og dets issues var de eneste aspektene igjen for vedlikehold.

Allikevel, lærte vi fortsatt mye om arbeidsmetodikk og utviklingsprosessen gjennom dette. Vi lærte mye om hvor viktig det var å ha fleksibilitet og god kommunikasjon gjennom utviklingsperioden, sammen med importansen av å ha en solid alternativ plan for 'når ikke går som planlagt'. I korte ord, dersom vi skulle evaluere oss på utviklingsprosessen, ville vi ha sagt at vi er 'god, men trenger justeringer'.

10.4.2 Rapport

Vi mener at rapporten kunne ha vært bedre og mer omfattende dersom den hadde blitt skrevet på mer gradvis over en lengre periode. Istedenfor å dokumentere

arbeidet vi gjorde grundig underveis, hadde gruppa en tendens til å fokusere mest på utviklingen av applikasjonen. Dette førte til at rapporten ble hovedsakelig skrevet mot slutten av prosjektet, som videre betydde et stort tidspress på oss helt opp til leveringen av oppgaven.

Å fokusere mer på utviklingen tidlig i prosjektet var nødvendig for å både finne arkitekturen som oppgaven skulle basere seg på, samtidig som å få en god start på eventuell videreutvikling. Vi må allikevel innrømme at en bedre balanse mellom utvikling og skriving av rapport ville ha styrket dokumentasjonen vår betydeligere, og vi ville ha unngått mye stress senere i prosjektperioden. En bedre tilnærming til rapportskrivingen kunne ha inkludert:

- Oppdateringer om fremdrift etter hver sprint.
- Oppdateringer om utfordringer vi møtte på underveis.
- Løpende dokumentasjon av beslutninger vi gjorde.
- Identifisering kritiske områder for diskusjon og analyse.

10.4.3 Annet arbeid

Noe vi har lagt merke til for seint i prosjektperioden, er at vi ikke har ført timer over arbeidet vi hadde gjort for prosjektet. Selv om det fantes et system for det tidligere i prosjektet, så ble dette også glemt når prosjektet begynte å kreve mer arbeid - noe som kan knyttes til vedlikeholdsproblemet vi hadde beskrevet tidligere. Selv om det var ikke et så stort problem utover prosjektperioden, erkjenner vi at det ville ha vært nyttig både for rapportskriving og dokumentasjon. Dette ville ha gitt våre lesere og andre utviklere som kan jobbe videre med programvaren en kvantitativ mening om hvor mye innsats ble satt inni prosjektet. Selvfølgelig, skal vi ta dette med oss til fremtiden ved profesjonelle sammenheng og prosjekter.

Det samme kan sies om våre møtereferater. De ble laget tidlig i prosjektet, men når utviklingen tok av - så gikk det også bort. Møtereferater ville ha vært svært nyttig både for rapporten, men også for utviklingsprosessen. Med å skrive dem, ville vi ha hatt bedre kontroll over eksakt hvilket arbeid måtte gjøres, sammen med offentlig dokumentasjon på sitater fra møtene til å bruke i rapporten. For oss var det ikke et stort problem under utviklingsprosessen gjennom vår scrumboard og produktkø, men nå som vi skriver dette, anerkjenner vi her også at det ville ha vært fint å inkludere dem.

10.5 Konklusjon

Gjennom denne bacheloroppgaven har vi fått god innsikt og erfaring innenfor webutvikling, som vil være uvurderlig når vi nå går inn i arbeidslivet. Vi har forsøkt å integrere prinsipper for bærekraft gjennom hele utviklingsprosessen, fra valg av teknologier til optimalisering av ressursbruk. Vårt arbeid har bidratt til å stryke vitenskapelig forskning gjennom å forenkle tilgangen til avanserte verktøy for visualisering av fargematchfunksjoner.

Noe av det viktigste vi har lært er gjennom dette prosjektet hvor viktig det er med fleksibilitet og å kunne tilpasse seg nye utfordringer. Selv om vi hadde en solid plan på plass møtte vi stadig på tekniske utfordringer. Dette har lært oss at å kunne juster tidsrammer og arbeidsmetoder ofte er avgjørende for å sikre at prosjektmålene oppnås.

Vi har også fått kjenne på hvor viktig det er med tidsstyring og kontinuerlig dokumentasjon. Mangelen på et system for timeføring gjorde det litt utfordrende å identifisere tidskrevende arbeidsoppgaver.

Vi har samarbeidet godt og det har vært en avgjørende faktor for prosjektet og har vært en viktig faktor for vår suksess. Selv om vi møtte på noen kommunikasjonsutfordringer, karte vi å løse disse gjennom regelmessige møter og klar fordeling av oppgaver. Dette har lært oss hvor viktig det er med effektiv kommunikasjon ar lagarbeid.

Samlet sett har vi prøvd å integrere prinsipper for bærekraft gjennom hele utviklingsprosessen, fra valg av teknologier til optimalisering av ressursbruk. Ved å styrke vitenskapelig forskning og forbedre tjenligheten til enkle verktøy for visualisering av fargematchfunksjoner, bidrar vi til en mer bærekraftig fremtid.

Bibliografi

[1] I. Farup, J. H. Wold, T.-H. Yang, G. D. de La Riva og Bartek, *CIE Functions*, versjon 1.0.2, Basisprogrammet for oppgave. adresse: https://github.com/ifarup/ciefunctions.

[2] K. Hofstad, «måling (naturvitenskap),» *Store norske leksikon*, 2021, Hentet 19. mai 2024.

[3] T. Holtebekk, «kolorimeter - fargebestemmelse,» *Store norske leksikon*, 2023, Hentet 19. mai 2024 fra https://snl.no/kolorimeter_-_fargebestemmelse. adresse: https://snl.no/kolorimeter_-_fargebestemmelse.

[4] J. Setchell, «8 - Colour description and communication,» i *Colour Design*, ser. Woodhead Publishing Series in Textiles, J. Best, red., Woodhead Publishing, 2012, s. 219–253. adresse: https://www.sciencedirect.com/science/article/pii/B9781845699727500084.

[5] M. Polo, *CIE1931 RGBCMF.svg*, Normalized RGB functions for monochromatic beams of light of specific wavelength in en: CIE 1931 color space., nov. 2007. adresse: https://commons.wikimedia.org/wiki/File:CIE1931_RGBCMF.svg.

[6] S. Sulyman, «Client-Server Model,» *IOSR Journal of Computer Engineering*, årg. 16, s. 57–71, jan. 2014. DOI: 10.9790/0661-16195771.

[7] M. Team, *Microsoft® Application Architecture Guide*. Microsoft Press, 2009, ISBN: 9780735642799. adresse: https://books.google.no/books?id=D9on897Ep7AC.

[8] H. Mili, A. Elkharraz og H. Mcheick, «Understanding separation of concerns,» jan. 2004. adresse: https://www.researchgate.net/publication/244446574_Understanding_separation_of_concerns.

[9] IBM, «Architectural characteristics of web-based applications,» jan. 2024, Hentet 18.05.2024. adresse: https://www.ibm.com/docs/en/db2-for-zos/13?topic=environment-architectural-characteristics-web-based-applications.

[10] Bart, *Overview of a three-tier application vectorVersion.svg*. adresse: https://commons.wikimedia.org/wiki/File:Overview_of_a_three-tier_application_vectorVersion.svg.

[11] J. Duckett, *HTML CSS - Design and Build Websites*. Wiley Press, 2011.

[12] J. Duckett, *JavaScript JQuery - Interactive front-end web development*. Wiley Press, 2014.

[13] D. Singh, «Understanding Browser Rendering: The Critical Rendering Path,» adresse: https://www.linkedin.com/pulse/understanding-browser-rendering-critical-path-divyansh-singh.

[14] Z. Gollwitzer. «How your browser loads, parses, and renders a webpage.» (2024), adresse: https://www.fullstackfoundations.com/blog/how-browser-loads-parses-renders-webpage (sjekket 20.05.2024).

[15] «Populating the page: how browsers work.» (), adresse: https://developer.mozilla.org/en-US/docs/Web/Performance/How_browsers_work (sjekket 20.05.2024).

[16] P. Irish og T. Garsiel. «How your browser loads, parses, and renders a webpage.» (2011), adresse: https://web.dev/articles/howbrowserswork (sjekket 20.05.2024).

[17] M. Kosaka. «Inside look at modern web browser (part 3).» (2018), adresse: https://developer.chrome.com/blog/inside-browser-part3/ (sjekket 20.05.2024).

[18] W. Preiser og E. Ostroff, *Universal Design Handbook* (M-H handbooks). McGraw-Hill, 2001, ISBN: 9780071376051. adresse: https://archive.org/details/universaldesignh0000unse/page/n27/mode/2up.

[19] E. Marcotte, «Responsive Web Design,» *A List Apart*, mai 2010. adresse: https://alistapart.com/article/responsive-web-design/.

[20] H. Kang, G. Liu, W. Quan, L. Meng og J. Liu, «Theory and Application of Zero Trust Security: A Brief Survey,» *Entropy*, årg. 25, nov. 2023. DOI: 10.3390/e25121595.

[21] I. C. Instructor, «The Five Languages or Dimensions of Interaction Design,» *Interaction Design*, 2016. adresse: https://www.interaction-design.org/literature/article/the-five-languages-or-dimensions-of-interaction-design.

[22] *Interaction Design: Beyond Human-Computer Interaction*. Wiley, 2002.

[23] IBM og M. Goodwin, «What is an API?» *IBM Explainers*, 2024. adresse: https://www.ibm.com/topics/api.

[24] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, Doctoral dissertation, University of California., 2000. adresse: https://ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf.

[25] F. Belqasmi, R. Glitho og C. Fu, «RESTful Web Services for Service Provisioning in Next-Generation Networks: A Survey,» *IEEE Communications Magazine*, årg. 49, s. 66–73, des. 2011. DOI: 10.1109/MCOM.2011.6094008.

[26] D. K. Wagh og R. Thool, «A Comparative study of SOAP vs REST web services provisioning techniques for mobile host,» *Journal of Information Engineering and Applications*, årg. 2, s. 12–16, jul. 2012.

[27] W3, *WebAssembly Core Specification*, W3C Recommendation, des. 2019. adresse: https://www.w3.org/TR/wasm-core-1/.

[28] D. Pockstaller, S. Huber og L. Demetz, «Comparing the Energy Consumption of WebAssembly and JavaScript in Mobile Browsers,» nov. 2023, s. 121–127. DOI: 10.5220/0012205600003584.

[29] GeeksForGeeks, «Floating point error in Python,» des. 2023, Hentet 19. Mai 2024. adresse: https://www.geeksforgeeks.org/floating-point-error-in-python/.

[30] P. J. Eby, «PEP 3333 - Python Web Server Gateway Interface v.1.0.1,» PEP 3333, 2010. adresse: https://peps.python.org/pep-3333/.

[31] Wikipedia contributors, *Human–computer interaction — Wikipedia, The Free Encyclopedia*, [Online; accessed 19-May-2024], 2024. adresse: https://en.wikipedia.org/w/index.php?title=Human%E2%80%93computer_interaction&oldid=1222972301.

[32] J. Nielsen. «Usability 101: Introduction to Usability.» (2012), adresse: https://www.nngroup.com/articles/usability-101-introduction-to-usability/ (sjekket 10.04.2024).

[33] J. Ighalo. «Using react-responsive to implement responsive design.» (2021), adresse: https://blog.logrocket.com/using-react-responsive-to-implement-responsive-design/ (sjekket 20.05.2024).

[34] GeeksForGeeks. «React Introduction.» (2024), adresse: https://www.geeksforgeeks.org/reactjs-introduction/ (sjekket 17.05.2024).

[35] I. Akinyemi. «Build Responsive Web Pages With React-Responsive And TypeScript.» (2021), adresse: https://blog.openreplay.com/build-responsive-web-pages-with-react-responsive-and-typescript/ (sjekket 20.05.2024).

[36] J. Lanctot. «The key ingredients of RESTful APIs: resources, representations, and statelessness.» (2023), adresse: https://www.torocloud.com/blog/the-key-ingredients-of-restful-apis-resources-representations-and-statelessness (sjekket 20.05.2024).

[37] Wikipedia contributors, *Flask (web framework) — Wikipedia, The Free Encyclopedia*, [Online; accessed 19-May-2024], 2024. adresse: https://en.wikipedia.org/w/index.php?title=Flask_(web_framework)&oldid=1222790474.

[38] Wikipedia contributors, *Model–view–controller — Wikipedia, The Free Encyclopedia*, [Online; accessed 19-May-2024], 2024. adresse: https://en.wikipedia.org/w/index.php?title=Model%E2%80%93view%E2%80%93controller&oldid=1222042229.

[39] C. Gackenheimer, *Introduction to React*. Springer Science, 2015.

[40] A. Banks og E. Porcello, *Learning React: Functional Web Development with React and Redux*. O'Reilly Media, 2017, ISBN: 9781491954577. adresse: https://books.google.no/books?id=pMTADgAAQBAJ.

[41] D. Banerjee. «Using strongly typed vs. statically typed code.» (2023), adresse: https://blog.logrocket.com/using-strongly-typed-vs-statically-typed-code/ (sjekket 20.05.2024).

[42] N. Raval. «TypeScript vs JavaScript: Know The Difference.» (2023), adresse: https://radixweb.com/blog/typescript-vs-javascript (sjekket 18.05.2024).

[43] Wikipedia contributors, *Visual Studio Code — Wikipedia, The Free Encyclopedia*, [Online; accessed 21-May-2024], 2024. adresse: https://en.wikipedia.org/w/index.php?title=Visual_Studio_Code&oldid=1223317495.

[44] Wikipedia contributors, *JetBrains — Wikipedia, The Free Encyclopedia*, [Online; accessed 19-May-2024], 2024. adresse: https://en.wikipedia.org/w/index.php?title=JetBrains&oldid=1221276587.

[45] R. H. Inc. «What is Docker?» (2018), adresse: https://www.redhat.com/en/topics/containers/what-is-docker (sjekket 17.05.2024).

[46] Docker. «Use containers to Build, Share and Run your applications.» Offisiell dokumentasjon fra Docker. (), adresse: https://www.docker.com/resources/what-container/ (sjekket 17.05.2024).

[47] Wikipedia contributors, *Monad (functional programming) — Wikipedia, The Free Encyclopedia*, [Online; accessed 20-May-2024], 2024. adresse: https://en.wikipedia.org/w/index.php?title=Monad_(functional_programming)&oldid=1216198666.

[48] GeeksForGeeks. «What is StrictMode in React ?» (2023), adresse: https://www.geeksforgeeks.org/what-is-strictmode-in-react/ (sjekket 20.05.2024).

[49] G. van Rossum, B. Warsaw og A. Coghlan, «PEP 8 – Style Guide for Python Code,» PEP 0008, 2001. adresse: https://peps.python.org/pep-0008/.

[50] J. E. Ravndal og J. H. Halleraker. «FNs bærekraftsmål i Store Norske Leksikon på snl.no.» Hentet 20.05.2024. (mai 2023), adresse: https://snl.no/FNs_b%C3%A6rekraftsm%C3%A5l.

Vedlegg A

Definisjoner

Dette vedlegget skal inneholde definisjoner og forklaringer på ord og uttrykk brukt i denne rapporten. Denne lista skal ikke inkludere tekniske begrep og deres definisjoner, men heller uttrykk brukt spesifikt i rapporten.

- **'Alt-i-nettleser':** Et uttrykk som brukes for den klient-baserte arkitekturen, hvor utregninger kjøres hos klientens egen maskin istedenfor en server.
- **Hvitpunktet:** Uttrykk i fargematchfunksjoner. Spesifikk verdi for de tre stimulus i øyner for å oppnå fargen hvit.
- **Purpurlinja:** En purpur linje som oppstår i kromatisitetsdiagram.
- **'Dockerisering'**: Et uttrykk brukt for å omgjøre et program om til en Docker kontainer.
- **'Klient-tjener':** Både et uttrykk som brukes for arkitekturmodellen med samme navn, men også for den tjener-baserte arkitekturen, hvor utregninger kjøres hos server gjennom forespørsler fra klient.
- **Søkestrenger:** Addisjonelle parametere i URL, også kjent som 'query string' på engelsk.

Vedlegg B

Standardavtale

NTNU

Norges teknisk-naturvitenskapelige universitet

Fastsatt av prorektor for utdanning 10.12.2020

STANDARDAVTALE

om utføring av studentoppgave i samarbeid med ekstern virksomhet

Avtalen er ufravikelig for studentoppgaver (heretter oppgave) ved NTNU som utføres i samarbeid med ekstern virksomhet.

Forklaring av begrep

Opphavsrett
Er den rett som den som skaper et åndsverk har til å fremstille eksemplar av åndsverket og gjøre det tilgjengelig for allmennheten. Et åndsverk kan være et litterært, vitenskapelig eller kunstnerisk verk. En studentoppgave vil være et åndsverk.

Eiendomsrett til resultater
Betyr at den som eier resultatene bestemmer over disse. Utgangspunktet er at studenten eier resultatene fra sitt studentarbeid. Studenten kan også overføre eiendomsretten til den eksterne virksomheten.

Bruksrett til resultater
Den som eier resultatene kan gi andre en rett til å bruke resultatene, f.eks. at studenten gir NTNU og den eksterne virksomheten rett til å bruke resultatene fra studentoppgaven i deres virksomhet.

Prosjektbakgrunn
Det partene i avtalen har med seg inn i prosjektet, dvs. som vedkommende eier eller har rettigheter til fra før og som brukes i det videre arbeidet med studentoppgaven. Dette kan også være materiale som tredjepersoner (som ikke er part i avtalen) har rettigheter til.

Utsatt offentliggjøring
Betyr at oppgaven ikke blir tilgjengelig for allmennheten før etter en viss tid, f.eks. før etter tre år. Da vil det kun være veileder ved NTNU, sensorene og den eksterne virksomheten som har tilgang til studentarbeidet de tre første årene etter at studentarbeidet er innlevert.

1. Avtaleparter

Norges teknisk-naturvitenskapelige universitet (NTNU) Institutt: Institutt for datateknologi og informatikk
Veileder ved NTNU: Ivar Farup e-post og tlf. Ivar.farup@ntnu.no, 61 13 52 27
Ekstern virksomhet: CIE TC1-97 Ekstern virksomhet sin kontaktperson og e-post; Jan Henrik Wold Jan.h.wold@ntnu.no
Student: Lars Edvin Jonsson Hoff Fødselsdato: 18.12.1996
Student: Nikolay Savchuk Fødselsdato: 13.11.2002
Student: Anders Mariendal Brunsberg Fødselsdato: 28.09.1996

Partene har ansvar for å klarere eventuelle immaterielle rettigheter som studenten, NTNU, den eksterne eller tredjeperson (som ikke er part i avtalen) har til prosjektbakgrunn før bruk i forbindelse med utførelse av oppgaven. Eierskap til prosjektbakgrunn skal fremgå av eget vedlegg til avtalen der dette kan ha betydning for utførelse av oppgaven.

2. Utførelse av oppgave

Studenten skal utføre: (sett kryss)

Masteroppgave	
Bacheloroppgave	X
Prosjektoppgave	
Annen oppgave	

Startdato: 8.1.2024
Sluttdato: 21.5.2024

Oppgavens arbeidstittel er: Webapp CIE

Ansvarlig veileder ved NTNU har det overordnede faglige ansvaret for utforming og godkjenning av prosjektbeskrivelse og studentens læring.

3. Ekstern virksomhet sine plikter

Ekstern virksomhet skal stille med en kontaktperson som har nødvendig faglig kompetanse til å gi studenten tilstrekkelig veiledning i samarbeid med veileder ved NTNU. Ekstern kontaktperson fremgår i punkt 1.

Formålet med oppgaven er studentarbeid. Oppgaven utføres som ledd i studiet. Studenten skal ikke motta lønn eller lignende godtgjørelse fra den eksterne for studentarbeidet. Utgifter knyttet til gjennomføring av oppgaven skal dekkes av den eksterne. Aktuelle utgifter kan for eksempel være reiser, materialer for bygging av prototyp, innkjøp av prøver, tester på lab, kjemikalier. Studenten skal klarere dekning av utgifter med ekstern virksomhet på forhånd.

Ekstern virksomhet skal dekke følgende utgifter til utførelse av oppgaven:

Dekning av utgifter til annet enn det som er oppført her avgjøres av den eksterne underveis i arbeidet.

4. Studentens rettigheter

Studenten har opphavsrett til oppgaven[1]. Alle resultater av oppgaven, skapt av studenten alene gjennom arbeidet med oppgaven, eies av studenten med de begrensninger som følger av punkt 5, 6 og 7 nedenfor. Eiendomsretten til resultatene overføres til ekstern virksomhet hvis punkt 5 b er avkrysset eller for tilfelle som i punkt 6 (overføring ved patenterbare oppfinnelser).

[1] Jf. Lov om opphavsrett til åndsverk mv. av 15.06.2018 § 1

I henhold til lov om opphavsrett til åndsverk beholder alltid studenten de ideelle rettigheter til eget åndsverk, dvs. retten til navngivelse og vern mot krenkende bruk.

Studenten har rett til å inngå egen avtale med NTNU om publisering av sin oppgave i NTNUs institusjonelle arkiv på Internett (NTNU Open). Studenten har også rett til å publisere oppgaven eller deler av den i andre sammenhenger dersom det ikke i denne avtalen er avtalt begrensninger i adgangen til å publisere, jf. punkt 8.

5. Den eksterne virksomheten sine rettigheter

Der oppgaven bygger på, eller videreutvikler materiale og/eller metoder (prosjektbakgrunn) som eies av den eksterne, eies prosjektbakgrunnen fortsatt av den eksterne. Hvis studenten

---

skal utnytte resultater som inkluderer den eksterne sin prosjektbakgrunn, forutsetter dette at det er inngått egen avtale om dette mellom studenten og den eksterne virksomheten.

Alternativ a) (sett kryss) Hovedregel

X	Ekstern virksomhet skal ha bruksrett til resultatene av oppgaven

Dette innebærer at ekstern virksomhet skal ha rett til å benytte resultatene av oppgaven i egen virksomhet. Retten er ikke-eksklusiv.

Alternativ b) (sett kryss) Unntak

	Ekstern virksomhet skal ha eiendomsretten til resultatene av oppgaven og studentens bidrag i ekstern virksomhet sitt prosjekt

Begrunnelse for at ekstern virksomhet har behov for å få overført eiendomsrett til resultatene:

6. Godtgjøring ved patenterbare oppfinnelser

Dersom studenten i forbindelse med utførelsen av oppgaven har nådd frem til en patenterbar oppfinnelse, enten alene eller sammen med andre, kan den eksterne kreve retten til oppfinnelsen overført til seg. Dette forutsetter at utnyttelsen av oppfinnelsen faller inn under den eksterne sitt virksomhetsområde. I så fall har studenten krav på rimelig

godtgjøring. Godtgjøringen skal fastsettes i samsvar med arbeidstakeroppfinnelsesloven § 7. Fristbestemmelsene i § 7 gis tilsvarende anvendelse.

7. NTNU sine rettigheter

De innleverte filer av oppgaven med vedlegg, som er nødvendig for sensur og arkivering ved NTNU, tilhører NTNU. NTNU får en vederlagsfri bruksrett til resultatene av oppgaven, inkludert vedlegg til denne, og kan benytte dette til undervisnings- og forskningsformål med de eventuelle begrensninger som fremgår i punkt 8.

8. Utsatt offentliggjøring

Hovedregelen er at studentoppgaver skal være offentlige.

Sett kryss

X	Oppgaven skal være offentlig

I særlige tilfeller kan partene bli enige om at hele eller deler av oppgaven skal være undergitt utsatt offentliggjøring i maksimalt tre år. Hvis oppgaven unntas fra offentliggjøring, vil den kun være tilgjengelig for student, ekstern virksomhet og veileder i denne perioden. Sensurkomiteen vil ha tilgang til oppgaven i forbindelse med sensur. Student, veileder og sensorer har taushetsplikt om innhold som er unntatt offentliggjøring.

Oppgaven skal være underlagt utsatt offentliggjøring i (sett kryss hvis dette er aktuelt):

Sett kryss		Sett dato
	ett år	
	to år	
	tre år	

Behovet for utsatt offentliggjøring er begrunnet ut fra følgende:

Dersom partene, etter at oppgaven er ferdig, blir enig om at det ikke er behov for utsatt offentliggjøring, kan dette endres. I så fall skal dette avtales skriftlig.

Vedlegg til oppgaven kan unntas ut over tre år etter forespørsel fra ekstern virksomhet. NTNU (ved instituttet) og student skal godta dette hvis den eksterne har saklig grunn for å

be om at et eller flere vedlegg unntas. Ekstern virksomhet må sende forespørsel før oppgaven leveres.

De delene av oppgaven som ikke er undergitt utsatt offentliggjøring, kan publiseres i NTNUs institusjonelle arkiv, jf. punkt 4, siste avsnitt. Selv om oppgaven er undergitt utsatt offentliggjøring, skal ekstern virksomhet legge til rette for at studenten kan benytte hele eller deler av oppgaven i forbindelse med jobbsøknader samt videreføring i et master- eller doktorgradsarbeid.

9. Generelt

Denne avtalen skal ha gyldighet foran andre avtaler som er eller blir opprettet mellom to av partene som er nevnt ovenfor. Dersom student og ekstern virksomhet skal inngå avtale om konfidensialitet om det som studenten får kjennskap til i eller gjennom den eksterne virksomheten, kan NTNUs standardmal for konfidensialitetsavtale benyttes.

Den eksterne sin egen konfidensialitetsavtale, eventuell konfidensialitetsavtale den eksterne har inngått i samarbeidprosjekter, kan også brukes forutsatt at den ikke inneholder punkter i motstrid med denne avtalen (om rettigheter, offentliggjøring mm). Dersom det likevel viser seg at det er motstrid, skal NTNUs standardavtale om utføring av studentoppgave gå foran. Eventuell avtale om konfidensialitet skal vedlegges denne avtalen.

Eventuell uenighet som følge av denne avtalen skal søkes løst ved forhandlinger. Hvis dette ikke fører frem, er partene enige om at tvisten avgjøres ved voldgift i henhold til norsk lov. Tvisten avgjøres av sorenskriveren ved Sør-Trøndelag tingrett eller den han/hun oppnevner.

Denne avtale er signert i fire eksemplarer hvor partene skal ha hvert sitt eksemplar. Avtalen er gyldig når den er underskrevet av NTNU v/instituttleder.

Signaturer:

Instituttleder: v Emneansvarlig Dato: 5/2-24
Veileder ved NTNU: Ivar Farup

Dato: 31.01-2024

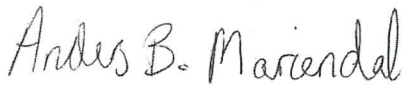
Ekstern virksomhet:
Jan H. Wold

Dato: 01.02-2024

Student: Lars Edvin Jonsson Hoff

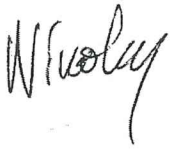
Dato: 31.01.24

Student: Anders Brunsberg Mariendal



Dato: 31.01.24

Student: Nikolay Savchuk
Dato: 31.01.2024



Vedlegg C

Prosjektplan

NTNU
Kunnskap for en bedre verden

Department of Computer Science

IDATG2900 - Bachelor thesis

Web app for visualization of CIE-functions - Project plan

Authors:
Lars Edvin Jonsson Hoff (470525)
Anders Brunsberg Mariendal (472499)
Nikolay Savchuk (562942)

01.02.2024

Table of Contents

1 Goals and restrictions

1.1 Background

Each and every being on Earth perceives the world in a different way. Some insects are able to see UV light usually invisible to us humans, while others are completely blind. However, most of us (especially humans) perceive the world through the visible light spectrum - though not all of us see it in the same way.
How someone sees the world depends on a great amount of factors; so, in an attempt to create representations on how one sees colours, the International Commission on Illumination (CIE) created standards and mathematical functions that do exactly this.

With this, a referential application was made in collaboration within the CIE TC1-97 Farup et al. n.d., in which one can use these functions. Our task is to use this application, and design a web application which houses all of the same functionalities as it - so that scientists who use it can easily adapt and adopt it as an alternative choice of colour functions.

1.2 Project Goals

1.2.1 The Effect

We wish our application has the effect of giving more comfort and usability to the scientists within CIE all around the world. While it is possible for most scientists to use the desktop application version, for some it may seem more attractive to rather use a web application version of it - less to download, available on most (if not all) devices that support the web, and so on.
We wish to make this with them in mind, so that we know for sure that the effect is oriented towards them.

1.2.2 The End-Result

The ultimate end result of our project is to create an web application that has all of the same functionalities as the desktop application this thesis is based on. It should be using an user interface that should be no stranger to users of the original application, while still being adapted for the web browser through responsive design.

1.2.3 The Learning Outcome

We can learn a great deal of matters through this project, but some that we wish to put emphasis on would be:

- Testinging, experimenting and discussing different approaches on how to tackle a project:
  - The tried and true backend approach through subjects such as websockets or APIs.
  - Newer and experimental approaches within popular open source alternatives.
- Communication and teamwork in an actual project, both between team members as well as the team and product owner.
- The usage of scrum as a development model in a more professional environment than before.
- How to combine our strengths in different fields to create a product our product owner would be satisfied with.

1.3 Framework

- The web application should support HTML5, and be accessible through most mainstream web browsers.
- The final version of the application - this implies a chosen prototype that we deemed fit for the project, and thus developed completely - should be completely developed by 16th of April, one month before the thesis's deadline.

2 Scope

2.1 Problem

How a person perceives colour depends on a great set of parameters, and so to make it easier to represent this, the CIE declared a set of standards and mathematical functions regarding colour vision. Most scientists however, do not wish to do the math by hand on these highly sophisticated functions (as well as represent them through plots and tables by extension), so they naturally resort to computers, using applications such as the referential application this project is going to be based on.

While the desktop application is sufficient and gets the job done for most, desktop applications has some inherent flaws such as accessibility and convenience. Through this project we wish to expand on the accessibility and usability of the core application. This will be done by developing a web-based version. The final project will be accessible on most devices. We will achieve this by developing a few different prototypes to fine the optimal framework for our project.

2.2 Task Description

The objective is to create a web application which houses the very same functionalities as the desktop application. This includes and implies:

- The application has to be able to compute various CIE functions, with their own unique and modifiable inputs.
- For each function, the application has to be able to display them as either plots and/or tables.
  - Adapt the user interface from the original application, into a faithful recreation within the web browser.
  - Plots are interactive; they can be moved around, zoomed in/out on, labeled, and gridded, among other features. Additionally, the plot can be saved as an image and further adjusted.
  - Tables have to support a given precision of values, unique to a function - sometimes ranging between 5-7 decimal figures.
  - For each function, the program has to be able to display an informational screen that includes basic information regarding the function. This includes parameters, symbols in the function, wavelengths, normalizations, etc.

In addition to this we have to make some underlying decisions with regards to the framework of the web application. This is primarily:

- Use various frameworks to create working prototypes with the given functionalities above.
- Test the prototypes, and discuss positives/negatives with each one to then choose one that fits ideally for the project.

2.3 Limitations

- While our application has to have the same functionalities as the one it is based on, it shouldn't have any extra large visible functionalities that does not exist in the original application. If we find some aspects that can clearly be improved upon without compromising the integrity the the program potential changes can be discussed with the product owner.
- Featuees and functionalities that are "behind the scenes" and invisible to the user, such as caching, can be developed freely.

3 Project organising

3.1 Routines and group rules

Work Methodology
For our development process we chose to utilise Scrum. This is because of its flexibility and efficiency in managing complex projects. Scrum's iterative approach allows for regular reassessment and adaptation, which is crucial in a dynamic project environment. It facilitates team collaboration ensuring that everyone is aligned and contributing effectively.

Attendance
All team meetings are mandatory, unless otherwise agreed. Project supervisor and product owner may request to move time and place of the meeting. All group members may call for extraordinary meetings if needed.

Absence
If one is prevented from attending at the agreed times and places, a message should be sent to the group members in advance. Planned absences should be discussed with the team members collectively. UIO n.d.

Consequences
In case of breach of the group's rules, a verbal warning will be given first. In case of repeated breaches, the offender will receive a written warning. If a group member receives two written warnings, a meeting with the group advisor will be arranged. Read conflict management.

Meetings
Meeting invitations should be sent out at least 2 days before the meeting. The minutes of the meeting should be ready no later than the next day after a meeting has been held. At meetings, we all agree that time should be used efficiently. We want everyone to come prepared to each meeting.

Decisions
All decisions regarding structure and technology used should be made collectively where all group members must be present. All group members have one vote, and the majority decides.

Conflict Management
All irregularities should initially be discussed at group meetings. All group members can bring up things they believe deviate from the agreements that have been made. If agreement or a solution is not reached, it should be brought to the group supervisor who is responsible for whether the concerned problem child becomes a resource for the team again or the group is split in two. If the group is split, both parties will gain access to all the work previously done.

3.2 Responsibilities and roles

- Product owner
  - This person represents the wants and needs of the CIE organisation.
  - He is also responsible for communicating the requirements of the web application.
- Supervisor
  - The supervisor is responsible for providing academic guidance and helping the students understand and apply different development concepts and theories.
  - He may also assist with administrative aspects. This includes procedures for submission and assessment criteria.
  - The most crucial responsibility of the supervisor is critical assessment and providing constructive feedback to the students.
- Group leader
  - Coordinates the overall project, ensuring that tasks are distributed, deadlines are met, and everyone is on track.
  - Facilitate effective communication among group members, ensuring that everyone is informed about project updates, meetings, and tasks.
  - Maintain open communication with the thesis advisor, seeking guidance when needed and sharing progress updates.
  - Keep track of the project's progress, monitor individual and group achievements, and provide updates to the group and the advisor.
  - Mediate any conflicts or issues that may arise among group members, promoting a harmonious working environment.
  -
- Scrum master
  - Make sure that all Scrum events occur while adhering to the designated time frames.
  - Exert their influence to eliminate any obstacles hindering the Scrum Team's progress.
  - Prioritise the creation of valuable increments that align with the Definition of Done.
  - Ensure that the work produced by group members meets the required standards and aligns with the project's objectives. scrum.org 2024
- Developers
  - Responsible for creating clean efficient code and design.
  - Recognise, rank, and execute tasks throughout the software development life cycle.
  - Conduct validation and verification tests.
  - Document different phases of development.
  - Execute their tasks in accordance with fixed routines and code practices.
- Editor
  - Oversee the documentation of the project, including meeting minutes, progress reports, and any necessary documentation for the thesis itself.
  - Responsible for planing, coordination, and review of material before delivery.
  - Has the final say on wording and phrasing.
  - Collaborate with group members in preparing for the presentation of the bachelor thesis, ensuring that everyone is well-prepared and confident.

4 Planning, follow-up and reporting

The project description defines the wanted functionality of the end product very well, as the end product essentially is a translation from desktop software to web application. A key aspect to the project however is the research and deciding on different technologies that will yield the best implementation of the web application. As such there is quite a bit of uncertainty surrounding the specific details of the application at the start of the project. Therefore flexibility and adaptability are key aspects.

4.1 Main division of the project

After carefully reviewing different frameworks for project management in correspondence with the project description, it was decided that Scrum would be a beneficial approach for the project.

The reasoning for this decision is that scrum is an agile method with key principles such as flexibility, teamwork, iterative progress and continuous improvement. Atlassian n.d. This ensures that the group is able to adapt based on feedback from product owner, supervisor or internal discussion. Even if the project is well defined, it is beneficial to be able to adapt and make changes to the software if more suitable solutions or technology is discovered. The frequency of meetings also ensures that everyone involved stays updated on the progress of the project, and facilitates better collaboration. Working by the scrum principles also gives a good overview of the project's overall progression which can help with identifying concerns and issues early and thereby make the necessary adjustments.

Scrum will be implemented in the following way:

- **Roles**
  - **Scrum master**: Facilitates the process and ensures that Scrum practices are followed.
  - **Product owner**: Represents the stakeholder, defines requirements.
  - **Development team**: Cross-functional team members who develops the product.
- **Artifacts**
  - **Product backlog**: A prioritized list of features describing functionality of end product.
  - **Sprint backlog**: A subset of the product backlog that is to be completed during the sprint.
  - **Increment**: The sum of the product backlog items completed during previous sprints.
- **Events**
  - **Sprint**: The time period allocated for a given sprint backlog to be worked on.
  - **Sprint planning**: Meeting between team and product owner at the start of each sprint where the sprint backlog is decided.
  - **Daily Stand-Up**: Short meeting between team members for status updates.
  - **Sprint Review**: Meeting between team and product owner at the end of the sprint to inspect the outcome of the sprint. Feedback from product owner, updating the product backlog.
  - **Sprint retrospective**: Meeting between team members. Takes place between the end of one sprint and the start of the next sprint. Evaluate the previous sprint and make plans for improvements for the coming sprint.

The group decided on a sprint duration of 2 weeks, as this seems like a good balance between frequency for feedback and discussion with the product owner and allocated time to implement features from the sprint backlog.

Sprint planning will take place at the start of each sprint. In this meeting the team and product owner will discuss and decide upon the sprint backlog that is to be worked on for the coming sprint.

Sprint review will take place at the end of each sprint. In this meeting the team and product owner will discuss the outcome of the sprint. The increment is inspected and the product backlog is updated. This gives a clear overview of the progress made up to this point and the remaining work for future sprints.

Sprint Retrospective will take place after the sprint review. The team members will discuss the sprint itself and evaluate what went well, and what changes to implement to improve future sprints. This ensures continuous improvement throughout the course of the project.

Following the scrum framework there will be held short stand-up meetings between the team members where each member briefly informs what they have been working on since the last stand up. Traditionally these stand-up meetings are held daily when following the scrum framework. Due to the scope of the project/thesis we decided that having stand-up meetings every other day will be more expedient.

In addition to following the Scrum framework as described, the team plans to utilize a Kanban board or Scrum board to keep track of the sprint backlog. This will be created digitally so that every team member can update the board and get a clear overview of how the sprint is going at all times. In this board every item from the sprint backlog will be added and given status labels such as "to-do", "in progress", "code review" and "done", and the visualisation will make it clear what has been done and is left to do.

4.2 Plan for status meetings and decision points during the period

As the sprint duration is two weeks, the sprint planning and sprint review will be held every other week. These two events will be done over one meeting between the team, supervisor and product owner where the meeting starts with a sprint review before moving over to sprint planning. After the Sprint planning and sprint review meeting, the team will do the sprint retrospective.

Every other Wednesday, where there are no sprint events planned, there will be a status meeting between the team and the supervisor.

Every Monday, Wednesday and Friday the team will have stand-up meetings.

Schedule for meetings:

- Stand-up meetings three times every week: Monday, Wednesday and Friday from 11:00 – 11:15
- Sprint planning and sprint review every two weeks on a Wednesday 11:15 - 12:30
- Sprint retrospective every two weeks on a Wednesday 12:30 - 13:00
- Status meeting with supervisor on Wednesdays with no Sprint events from 11:15-12:00

5 Organisation of quality assurance

5.1 Tools

Here is a list of the tools (libraries, software, services) we plan to use within the project. Keep in mind that this is the project planning phase - and thus, the future project may not be limited to these as we start work on the project.

NAME	USAGE
Docker	Docker gets used in the backend to containerize our software to the server.
PyScript	PyScript gets used as one of the possible approaches to the task, by introducing Python to the front-end. Anaconda n.d.
Django	Django gets used as one of the possible approaches to the project, by being a popular backend framework within Python.
Flask	Flask is also one of the possible approaches within backend, also being a popular Python framework.
PyCharm	PyCharm gets used as an IDE for Python development.
Visual Studio Code	Visual Studio Code gets used as a code editor for front-end development, as well as an alternative for Python development.
GitLab	GitLab gets used as the main repository for our project, as well as potentially the main area for us to plan sprints and sprint boards.
Diagrams.net	Diagrams.net gets used to create UML diagrams (as well as general diagrams) in case those are needed.
Figma	Figma gets used to design the application with wireframes and prototypes in case it is needed.
Overleaf	Overleaf gets used to create documentation for reports through LaTeX.

5.2 Quality Assurance of Documentation

To ensure proper quality of the source code, we wish to stick with practices we've learnt over the cource of our degrees. Naturally, we will comment our code with explanations for bits and pieces that is harder to read, while still keeping the shorter comments for bits that are easier to read, but still useful. All comments will be in English due to the project's background of being potentially used for CIE, an international organistation, later. This will make fruther potential development easier.

To ensure correct usage of the application for users, we also wish to include a README.md within the project repository, containing proper documentation on how to run, update and use the project.

5.3 Quality Assurance of Source Code

Here, we also wish to follow standard practices we learnt over the course of our degrees. Each developer gets their own branch in the repository that they're solely responsible for, alongside one main branch that hosts the newest changes. Any developer may work on an issue within the

current sprint backlog, in which they themselves are naturally responsible for updating that issue in addition to developing, testing, and merging their work to main (thus, they themselves are responsible for any merge issues that may arise).
Due to the nature of our project - in which we have to create prototypes of several frameworks - each prototype of different framework may get their own "main" branch that primarily contains its development. For each sprint, current prototypes are assigned a version to manage version controls easier.

We also wish to perform code review of each other's work before we classify something as "completely done". We wish to do this for most changes to the source code, and that any changes to the repository should be generally accepted by the entire group.

In addition, it is important to note that testing of each approach would be different, due to their differences. For example, testing of the PyScript approach requires insight into the performance of the application on a given web browser - while testing of a backend approach requires network testing to see if the client got the right calculations from the server. We intend to carefully document these as well.

5.4 Fixed development routines

We wish to follow the following routines in regards to the development of the project:

- Every developer is to meet up on a given meeting (or, if they cannot, document this to the rest of the group in advance).
- Every developer is responsible for logging their own hours used for the project through the usage of a shared Excel document.
- Every developer is responsible for updating issues they have taken upon themselves to work with in regards to a current sprint, as well as discussing their progress in the scrum meetings.

5.5 Risk analysis

Our risk analysis will consist of a series of potential risks linked to the project. We've decided that every risk will have an ultimate risk level, decided by the risk's potential total danger if realised (in "loss"), as well as the general chance of the risk happening (in "probability").
For the sake of simplicity, all of these have been standardized to "high", "some", or "minimal"; with the final risk level being a culmination of the loss and probability.

NAME	DESCRIPTION	LOSS	PROBABILITY	RISK LEVEL
Asset Loss	There may be some documentation and/or source code that gets lost over the course of development.	Some	Minimum	Some
Unfinished Aspect	There may be a part of the project that is left unfinished until the deadline of our project.	High	Minimum	Some
Obsolete Technology	There may arise a situation where our project uses technology that is either obsolete, or not recieving updates any longer.	Some	Minimum	Minimum
Unexpected Development Problem	There may be an abstract problem within development that causes serious delays and/or unexpected changes in the project over the course of development.	Some	Some	Some

5.6 Risk management plan

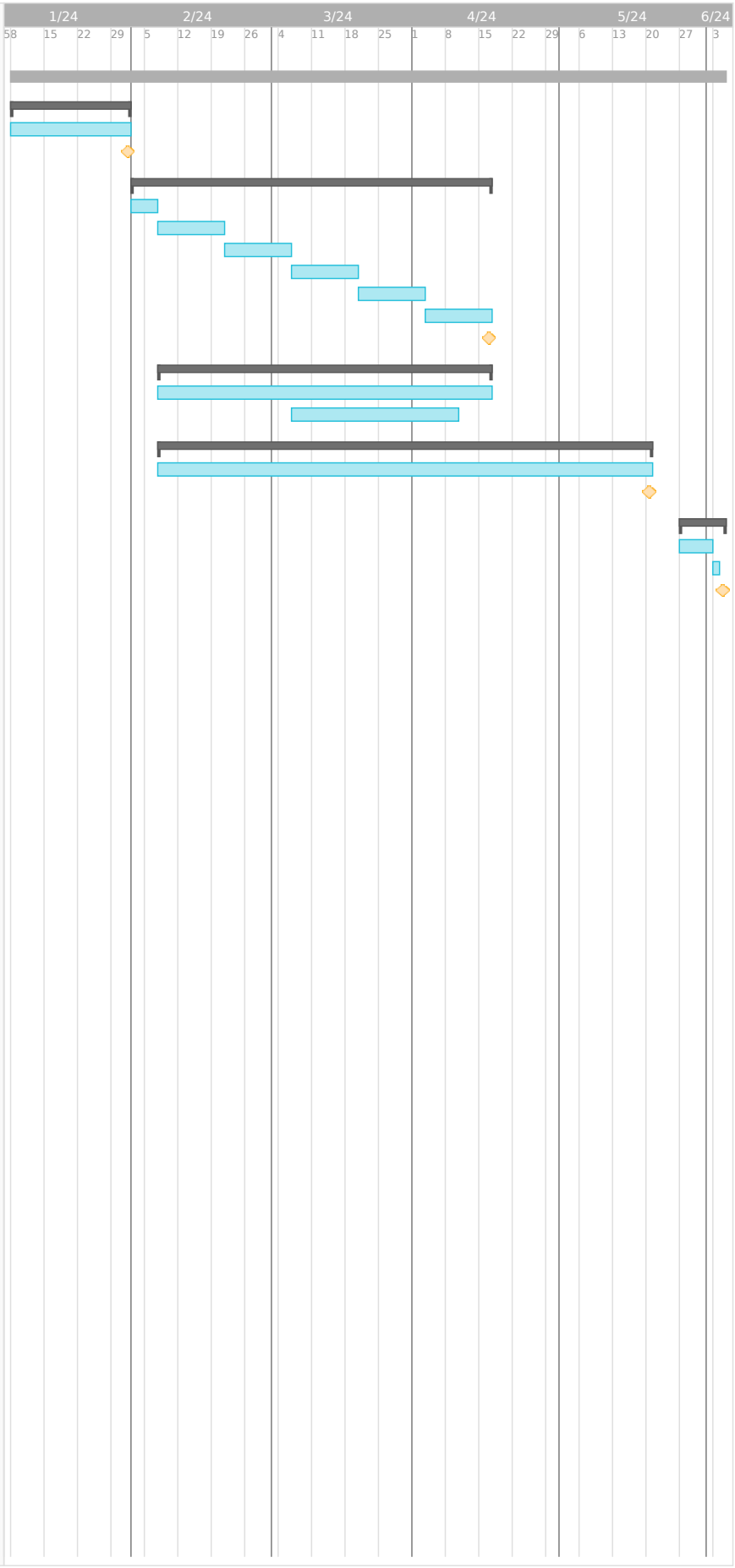
In the risk management above, we discussed risks in general - and while some are in need of critical management plans in case they are realised, some are not so deserving. Thus, we have the table below which references to risks above, and then details a management plan on how to proceed.

NAME	MEASURE?	DESCRIPTION
Asset Loss	Yes	Considering the fact that our project should be fairly easy to recover with the usage of GitLab and commits, there is little need for a measure for it - but to be on the safe side, we wish to store backups based on their version.
Unfiunished Aspect	None	There's no measure you can take to an unfinished aspect except for finish it, or make it at least somewhat viable for common usage.
Obsolete Technology	None	We'll accept this risk, given the nature of the project. In the case where the approach regarding Python in frontend is eventually made obsolete, we at least have other prototypes of other approaches that we can continue development from.
Unexpected Development Problem	None	Problems arise out of the blue, and there is no real measure one can take.

6 Plan for execution

6.1 Gantt chart

teamgantt
Created with Free Edition



Web App	**0h**	**0%**
Project plan	**0h**	**0%**
Project plan	0	0%
Project plan delivery	0	0%
Project	**0h**	**0%**
Research	0	0%
Sprint 1	0	0%
Sprint 2	0	0%
Sprint 3	0	0%
Sprint 4	0	0%
Sprint 5	0	0%
Deadline coding	0	0%
Testing	**0h**	**0%**
Code testing	0	0%
User testing	0	0%
Dissertation writing	**0h**	**0%**
Thesis writing	0	0%
Bachelor thesis delivery	0	0%
Presentation	**0h**	**0%**
Planing	0	0%
Rehearsal	0	0%
Presentation	0	0%

6.2 Milestones, activities and decisions

- **Research Phase:** This initial stage involves gathering necessary information and insights that will inform the rest of the project. It's crucial for understanding the problem space and identifying viable solutions.
- **Sprint Development:** The project is divided into several sprints, each focusing on different aspects of the web app. These sprints involve iterative development, with each sprint building upon the learnings from the previous one.
- **Deadline for Coding:** This milestone marks the completion of the main coding phase. It's a crucial point where the core functionalities of the web app should be implemented and operational.
- **Testing Phase:** This stage is divided into code testing and user testing. Code testing ensures the technical robustness of the app, while user testing focuses on the app's usability and user experience.
- **Dissertation Writing:** Alongside the development, writing the thesis is a continuous process. This phase involves documenting the project's progress, methodologies, findings, and conclusions.
- **Thesis Submission:** This is the final milestone where the completed thesis is submitted. It signifies the culmination of research, development, and writing efforts.
- **Presentation Preparation:** Planning and rehearsing for the final presentation of the project. This is where we showcase your work, highlighting the key aspects and learnings from the project.

Early on in the development process we will decide on which framework we will utilise for version control. We believe we have two realistic options. GitHub or GitLab. This is because of their robust version control capabilities and their wide adaptation in the developer scene. GitHub and GitLab facilitates collaborative coding, allowing multiple contributors to work simultaneously without conflict. It also provides a system for tracking changes, reviewing code, and managing updates efficiently. The choice we make will come down to which of the two frameworks is more compatible with Scrum, our work methodology.

By the conclusion of the initial Scrum sprint, we expect to have developed basic yet operational prototypes that demonstrates appropriate framework's capabilities. These prototypes serve as a proof of concept, illustrating how the framework can be utilised in our specific project context. This approach fosters a sense of achievement and momentum early in the project, setting a positive tone for subsequent development phases.

Bibliografi

Anaconda (n.d.). *PyScript*. URL: https://github.com/pyscript/pyscript.

Atlassian (n.d.). *What is scrum?* URL: https://www.atlassian.com/agile/scrum.

Farup, Ivar et al. (n.d.). *CIE Functions*. URL: https://github.com/ifarup/ciefunctions.

scrum.org (2024). *What is a Scrum Master?* URL: https://www.scrum.org/resources/what-is-a-scrum-master (visited on 16th Jan. 2024).

UIO (n.d.). *Eksempel på gruppeavtale*. URL: https://www.uio.no/studier/emner/matnat/ifi/IN1060/v18/timeplan/gruppe-3/eksempel-pa-arbeidskontrakt.pdf.

Vedlegg D

Oppgavebeskrivelse

Oppgave 8 BPROG, BIDATA 3 stk

Oppgavetittel: Webapp for visualisering av CIE-funksjoner

Bedrift: Fargelabben, IDI, NTNU

Kontaktperson: Ivar Farup og Jan H. Wold

E-post: Ivar.farup@ntnu.no
Telefon: 61135227
Lokasjon: Gjøvik

Oppgaven passer til 3 studenter fra BIDATA og/eller BPROG

Beskrivelse av oppgaven:

Bakgrunn

En persons fargesyn varierer med flere parametre, f.eks. alder og feltstørrelse. Standardiseringsorganet CIE har laget standarder og anbefalinger for hvordan funksjoner som representerer fargesynet, kalt fargematchfunksjoner, skal beregnes og representeres. Gjennom bl.a. prosjektet «Individual Colour Vision-based Image Optimisation» støttet av Norges forskningsråd, er det ved NTNU og USN utviklet en referanseimplementasjon programvare for for å kunne beregne disse funksjonene for en såkalt standardobservatør. Programvaren er utviklet i Python med en backend og et enkelt brukergrensesnitt i PyQt.

Oppgave

Vi ønsker gjennom dette bachelorprosjektet å få utviklet en Webapp med i hovedsak samme funksjonalitet som desktopapplikasjonen. Siden den eksisterende beregningskoden er implementert i Python, vil det være hensiktsmessig at backend-delen av løsningen forblir i Python. En viktig del av oppgaven blir å finne ut hva som er et egnet rammeverk for resten av applikasjonen gitt denne føringen. Løsninger som Dash, Flask, og Django kan synes naturlige å sjekke ut, men det er også åpent for andre løsninger.

Vedlegg E

Produktkø

Frontend - functionality for field size for CIE XYZ standard colour-matching functions and CIE xy standard chromaticity diagram
#32 · created 1 week ago by Anders Brunsberg Mariendal
Backend Approach
0 updated 1 week ago

Backend - Deployment
#31 · created 1 month ago by Nikolay Savchuk
Backend Approach In-Progress Produktkø
0 updated 1 month ago

Backend - Error Handling
#30 · created 1 month ago by Nikolay Savchuk
Backend Approach In-Progress Produktkø
0 updated 1 week ago

Backend - Asynkronisitet
#29 · created 1 month ago by Nikolay Savchuk
Backend Approach Godkjenning Sprintkø
Closed 0 closed 1 month ago

Backend - Revamp av API
#28 · created 1 month ago by Nikolay Savchuk
Backend Approach In-Progress
0 updated 1 week ago

Backend - Base precision
#27 · created 1 month ago by Nikolay Savchuk
Backend Approach Godkjenning Produktkø
Closed 0 closed 1 month ago

Backend - Wavelength floating point
#26 · created 1 month ago by Nikolay Savchuk Sprint #5
Backend Approach Godkjenning Produktkø
Closed 0 closed 1 month ago

Backend - Søke om server
#25 · created 2 months ago by Lars Edvin Jonsson Hoff
Backend Approach
0

Backend - Html sidemeny endpoint
#24 · created 2 months ago by Lars Edvin Jonsson Hoff
Backend Approach
Closed 0 closed 1 week ago

Frontend - Lage flere diagrammer
#23 · created 2 months ago by Lars Edvin Jonsson Hoff
Backend Approach Godkjenning
0 updated 1 week ago

Backend - Compute.py modularisering
#22 · created 2 months ago by Nikolay Savchuk Sprint #4
Backend Approach Godkjenning
0 updated 1 week ago

Backend - Endpoints
#21 · created 2 months ago by Nikolay Savchuk Sprint #4
Backend Approach Godkjenning
Closed 0 closed 1 month ago

Backend - Testing
#20 · created 2 months ago by Nikolay Savchuk Sprint #4
Backend Approach Godkjenning Produktkø Sprintkø
Closed 0 closed 1 month ago

Frontend - Redux
#19 · created 2 months ago by Nikolay Savchuk Sprint #3
Backend Approach In-Progress Produktkø
0 updated 2 months ago

Frontend - Checkbox (labels, log10, renormaliserte verdier)
#18 · created 2 months ago by Nikolay Savchuk Sprint #4
Backend Approach Godkjenning
0 updated 1 week ago

Frontend - Modulisering
#17 · created 2 months ago by Nikolay Savchuk Sprint #3
Backend Approach In-Progress Produktkø
0 updated 2 months ago

Frontend - Nedtrekksmeny
#16 · created 2 months ago by Nikolay Savchuk Sprint #3
Backend Approach Godkjenning Produktkø
0 updated 2 months ago

Frontend - Tabell
#15 · created 2 months ago by Nikolay Savchuk Sprint #3
Backend Approach Godkjenning Produktkø
0 updated 1 month ago

Frontend - Sidemeny
#14 · created 2 months ago by Nikolay Savchuk Sprint #3
Backend Approach
Closed 0 closed 1 week ago

Testing av prototyper
#13 · created 3 months ago by Nikolay Savchuk Prototypeavgjørelse
Dokumentasjon Produktkø
Closed 0 closed 2 months ago

Rapportskriving
#12 · created 3 months ago by Nikolay Savchuk Rapportskriving
Dokumentasjon In-Progress
0 updated 1 week ago

Backend Prototype, v3
#11 · created 3 months ago by Nikolay Savchuk Sprint #3
Backend Approach Produktkø
0 updated 2 months ago

Frontend Prototype, v3
#10 · created 3 months ago by Nikolay Savchuk Sprint #3
Frontend Approach Produktkø
Closed 0 closed 2 months ago

Backend Prototype, v2
#9 · created 3 months ago by Nikolay Savchuk Prototypeavgjørelse
Backend Approach Produktkø
Closed 0 closed 2 months ago

Frontend Prototype, v2
#8 · created 3 months ago by Nikolay Savchuk Prototypeavgjørelse
Frontend Approach Produktkø
Closed 0 closed 2 months ago

Backend Prototype, v1
#7 · created 3 months ago by Nikolay Savchuk Sprint #1
Backend Approach In-Progress Produktkø
Closed 0 closed 2 months ago

Frontend Prototype, v1
#6 · created 3 months ago by Nikolay Savchuk Sprint #1
Frontend Approach Produktkø Sprintkø
Closed 0 closed 2 months ago

Arkitektur - Komponentdiagrammer
#5 · created 3 months ago by Nikolay Savchuk Sprint #1
Dokumentasjon Produktkø
Closed 0 closed 2 months ago

Kravspesifikasjoner - Sekvensdiagrammer
#4 · created 3 months ago by Nikolay Savchuk Sprint #1
Dokumentasjon Produktkø
Closed 0 closed 2 months ago

Kravspesifikasjoner - Use-cases og use-case diagram
#3 · created 3 months ago by Nikolay Savchuk Sprint #1
Dokumentasjon Produktkø
Closed 0 closed 2 months ago

Arkitektur - Wireframe
#1 · created 3 months ago by Nikolay Savchuk Sprint #1
Dokumentasjon Produktkø Sprintkø
Closed 0 closed 3 months ago

Vedlegg F

Klient-tjener prototype stresstest

Performance test report - May 19, 2024 (#1)

Open in Postman

Postman collection: performance tst

Report exported on: May 19, 2024, 20:56:40 (GMT+2)

Test setup

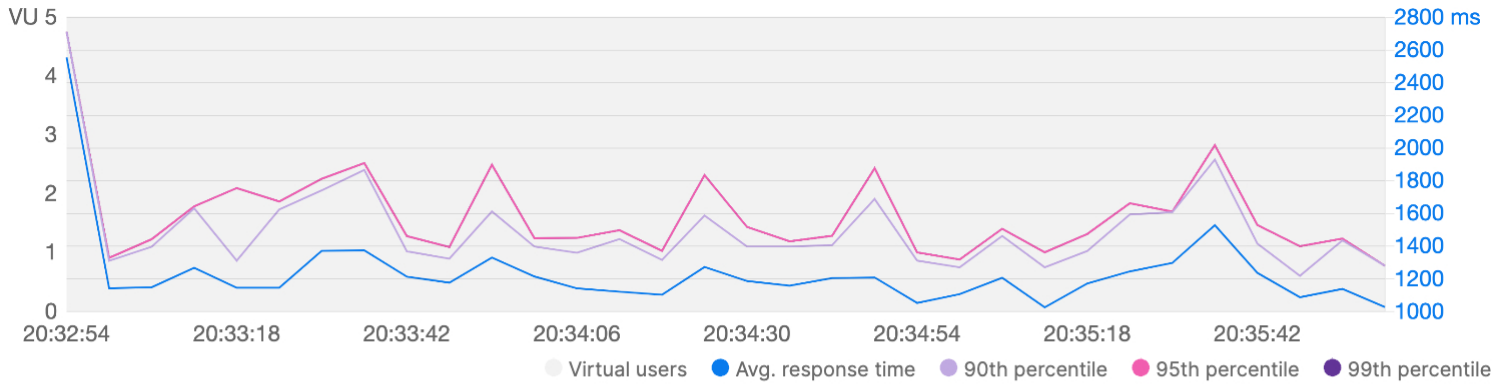
Virtual users	Start time	Load profile
5 VU	May 19, 20:32:56 (GMT+2)	Fixed
Duration	**End time**	**Environment**
3 minutes	May 19, 20:36:02 (GMT+2)	New Environment

1. Summary

Total requests sent	Throughput	Average response time	Error rate
421	2.26 requests/second	1,211 ms	0.00 %

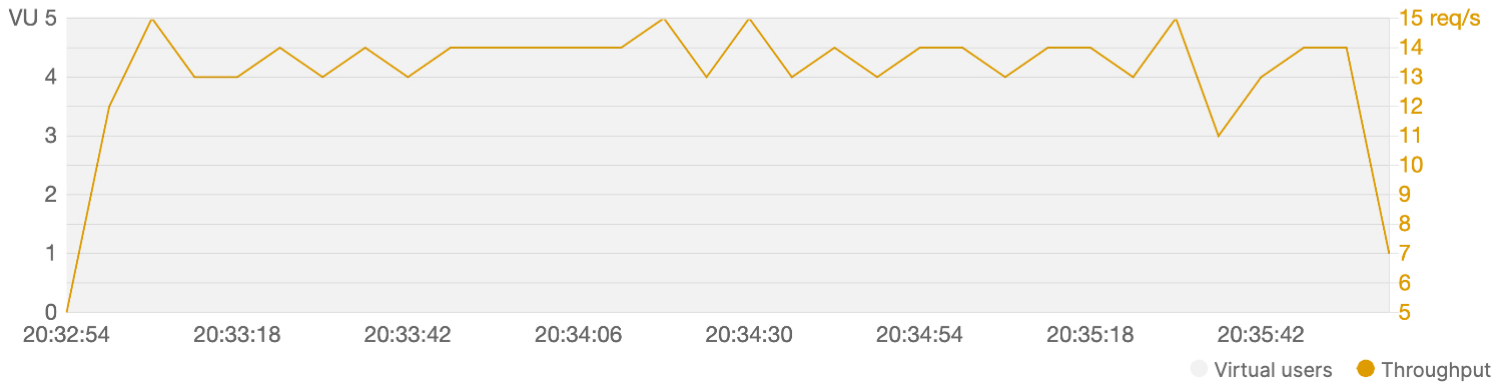
1.1 Response time

Response time trends during the test duration.



1.2 Throughput

Rate of requests sent per second during the test duration.



1.3 Requests with slowest response times

Top 5 slowest requests based on their average response times.

Request	Resp. time (Avg ms)	90th (ms)	95th (ms)	99th (ms)	Min (ms)	Max (ms)
POST New Request http://127.0.0.1:5000/compute_all_specific_data	1,211	1,518	1,663	2,360	841	2,713

2. Metrics for each request

The requests are shown in the order they were sent by virtual users.

Request	Total requests	Requests/s	Min (ms)	Avg (ms)	90th (ms)	Max (ms)	Error %
POST New Request http://127.0.0.1:5000/compute_all_specific_data	421	2.26	841	1,211	1,518	2,713	0

3. Errors

This run has no errors

All requests were sent successfully and returned a 2xx response code.

Testing API performance on Postman

Postman enables you to simulate user traffic and observe how your API behaves under load. It also helps you identify any issues or bottlenecks that affect performance.

Learn more about testing API performance.

Vedlegg G

Kodestykker av parametersystem

Kodeliste G.1: Backend - Utsnitt av parameterfunksjon, del 1.

```
# parameterfunksjon tatt fra prosjektet,
        # kommentarer i dette vedlegget er annerledes enn de i programmet
        # for hensikten av enklere formidling til leser
def createAndCheckParameters(disabled, calculation, request):
    # funksjon som forsøker å lese til gitt type,
        # lik til Rust sin <Option> type (enten Some(...) or None)
    def string_to_type_else(string, type, other):
        try:
            return type(string)
        except ValueError:
            return other
        except TypeError:
            return None
    # forskjellig behandling av parametere for standardfunksjoner
    if disabled:
        # henter inn *obligatoriske* parametere enten som None eller som float med
                # bruk av det over
        parameters = {
            "field_size": string_to_type_else(request.args.get('field_size'),
                float, None),
            "age": string_to_type_else(request.args.get('age'), float, None)
        }
        # sjekker om de finnes
        for (name, value) in parameters.items():
            if value is None:
                raise SanicException(
                    ("Value error.",
                    "Invalid input for '{}' either due to absence or invalid
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣type".format(name),
                     "Control that '{}' is present, and is of the 'float'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣type.".format(name)), status_code=422)
                # runder av i tilfellet det ble gitt et desimaltall
        parameters['age'] = round(parameters['age'])
```

Kodeliste G.2: Backend - Utsnitt av parameterfunksjon, del 2.

```
                # deretter, forsøker å finne de valgfrie parametere som gjelder
                # blir enten til:
                        # float dersom oppgitt verdi *og* det er et tall
```

```
                        # -1 dersom verdi er oppgitt, men *ikke* et tall
                        # None dersom ingenting er oppgitt
        optionals = {
            "min": string_to_type_else(request.args.get('min'), float, -1),
            "max": string_to_type_else(request.args.get('max'), float, -1),
            "step_size": string_to_type_else(request.args.get('step_size'), float, -1),
        }
                # utfører takling av valgfri parameter over;
        for (name, value) in optionals.items():
            # om feil type, send feilsvar tilbake til brukeren
            if value is -1:
                raise SanicException((
                "Type␣error.",
                "Invalid␣input␣for␣'{}'␣due␣to␣invalid␣type".format(name),
                "Control␣that␣the␣value␣is␣of␣a␣'float'␣type,␣or␣remove␣it␣to
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣use␣default␣settings."), status_code=422)
            # om ingenting er oppgitt, gi det en standardverdi og oppdater dict
            if value is None:
                given = 0
                if name == "min": given = 390.0
                if name == "max": given = 830.0
                if name == "step_size": given = 1.0
                parameters.update({name: given})
            # ellers, oppdater parameter dict til å inneholde verdien som den er
            else:
                parameters.update({name: float(value)})

        # starten av feilhåndtering
        if parameters['field_size'] > 10 or parameters['field_size'] < 1:
            raise SanicException(("Value␣error.", "Invalid␣value␣for␣'field_size'.",
                                  "Control␣that␣the␣value␣is␣between␣1.0␣and␣10.0."),
                                  status_code=422)
                # evt. flere andre feilhåndteringer ...
```

Kodeliste G.3: Backend - Utsnitt av LMS utregningsfunksjon

```
def compute_LMS_Modular(parameters):
    def inner_LMS(variation):
        # compute.py line 1565
         _all  = my_round(np.arange(390., 830. + .01, .1), 1)
        # compute.py line 1575
        LMS_base_all = LMS_energy(parameters['field_size'], parameters['age'])[0]
        if parameters['base']:
            # compute.py line 1572
            LMS_base_all = LMS_energy(parameters['field_size'],
            parameters['age'], base=True)[0]
        # compute.py line 1585
        ( _all , L, M, S) = LMS_base_all.T
        # compute.py lines 1586-1588
        L_spline = scipy.interpolate.InterpolatedUnivariateSpline( _all , L)
        M_spline = scipy.interpolate.InterpolatedUnivariateSpline( _all , M)
        S_spline = scipy.interpolate.InterpolatedUnivariateSpline( _all , S)
        # compute.py lines 763-768
        LMS_sf = 6
        logLMS_dp = 5
        if parameters['base']:
            LMS_sf = 9
            logLMS_dp = 8
        # compute.py lines 770-772
        (La, Ma, Sa) = np.array([sign_figs(L_spline(variation), LMS_sf),
                                 sign_figs(M_spline(variation), LMS_sf),
                                 sign_figs(S_spline(variation), LMS_sf)])
        # compute.py line 773
        LMS = chop(np.array([variation, La, Ma, Sa]).T)
        if parameters['log']:
            LMS[:, 1:][LMS[:, 1:] == 0] = -np.inf
            LMS[:, 1:][LMS[:, 1:] > 0] = my_round(
                np.log10(LMS[:, 1:][LMS[:, 1:] > 0]), logLMS_dp)
            LMS[:, 0] = my_round(LMS[:, 0], 1)
            return chop(LMS)
        else:
            LMS[:, 0] = my_round(LMS[:, 0], 1)
            # compute.py line 779
            return chop(LMS)
        return LMS
    dict = {
        "result": inner_LMS(np.arange(parameters['min'], parameters['max'] + .01,
        parameters['step_size'])),
        "plot": inner_LMS(my_round(np.arange(parameters['min'], parameters['max'] + .01,
        .1), 1))
    }
    return dict
```

Vedlegg H

Kodestykker for diagramressurs

Følgende kode dekker `macleod_graph()` fra `graph.py` modulen i prosjektet.

H.1 Python

Kodeliste H.1: Backend - Utsnitt av Pythonkode for MacLeod-Boynton diagrammet.

```
def macleod_graph(parameters):
    temp = parameters.copy()
    plots_json = cieapi.new_calculation_JSON(compute_MacLeod_modular, temp)
    temp['info'] = True
    info_json = cieapi.new_calculation_JSON(compute_MacLeod_modular, temp)
    # retrieves relevant datapoints from a range
    points = retrievePoints(macleod_graph, parameters)
    title = (
        "MacLeod-Boynton␣ls␣chromaticity␣diagram<br>␣Field␣size:
␣␣␣␣␣␣␣␣␣␣␣␣{}  ,␣Age:␣{}␣yr,␣Domain:␣{}␣nm␣-␣{}␣nm,␣Step:␣{}␣nm".
        format(parameters['field_size'], parameters['age'],
                parameters['min'], parameters['max'],
                parameters['step_size']))
    xaxis = "l<sub>MB</sub>.␣{},␣{}".format(
        parameters['field_size'], parameters['age'])
    yaxis = "S<sub>MB</sub>.␣{},␣{}".format(
        parameters['field_size'], parameters['age'])
    # creates the raw html
    raw = head(macleod_graph) + """ ...
```

Kodeliste H.2: Backend - Utsnitt av Pythonkode for head() fra graph.py

```
def head(function):
    return """
    <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src='https://cdn.plot.ly/plotly-2.32.0.min.js'></script>
    <style>
        #plot {
            border: 1px solid black;
        }
```

```
        #plot * {
            margin-left: auto;
            margin-right: auto;
        }
        #inputrow {
            display: flex;
            flex-direction: row;
            justify-content: space-around;
        }
    </style>
</head>
<body>
    <div id='plot'></div>
    """ + checkboxes(function) + """
    <script>
    """
```

H.2 JavaScript

N.B: Bruken av `plots_json`, `info_json`, `title`, `xaxis`, `yaxis` og `points` er fra Pythonkoden, og er elementer som settes inn for å dynamisk generere diagrammet basert på parametere.

Kodeliste H.3: Backend - Utsnitt av JavaScriptkode for MacLeod-Boynton diagrammet.

```
    const results = '""" + plots_json + """';
    const info = '""" + info_json + """';
    const plot = JSON.parse(results)['plot'];
    const information = JSON.parse(info);

    // from StackOverflow, see module description.
    output = plot[0].map((_, colIndex) => plot.map(row => row[colIndex]));

    const config = {responsive: true}
    var layout = {
        showlegend: false,
        autosize: true,
        title: '""" + title + """',
        margin: { l: 50, r: 10, b: 50, t: 50, pad: 4 },
        height: 800,
        width: 800,
            yaxis: {
                scaleanchor: "x",
                zeroline: false,
                title: '""" + yaxis + """'
            },
            xaxis: {
                zeroline: false,
                nticks: 10,
                title: '""" + xaxis + """',
            }
        }

    // macleod curve
    var curve = {
        x: output[1],
```

```
        y: output[3],
        xaxis: "x-aspect",
        yaxis: "y-aspect",
        mode: 'lines',
        line: {
                color: 'rgb(0, 0, 0)'
        }
    }

    const y_points = []
    const x_points = []
    const pointes = """ + str(points) + """;
    for (let i = 0; i < pointes.length; i++) {
        let index = output[0].indexOf(pointes[i]);
        x_points.push(output[1][index]);
        y_points.push(output[3][index]);
    }

    var points = {
        x: x_points,
        y: y_points,
        mode: 'markers',
        type: 'scatter',
        textposition: 'top right',
        marker: {
            color: 'rgb(255, 255, 255)',
            size: 10,
            line: {
                color: 'rgb(0, 0, 0)',
                width: 2
            }
        }
    }
    var illuminantE = {
        x: [information['white'][0]],
        y: [information['white'][2]],
        mode: 'markers',
        type: 'scatter',
        textposition: 'top right',
        marker: {
            color: 'rgb(255, 255, 255)',
            size: 10,
            line: {
                color: 'rgb(0, 0, 0)',
                width: 2
            }
        }
    }
    var purpleline = {
        x: [information['tg_purple'][0][1], information['tg_purple'][1][1]],
        y: [information['tg_purple'][0][2], information['tg_purple'][1][2]],
        mode: 'lines',
        line: {
            color: 'rgb(150, 32, 240)',
            width: 3
        }
    }

    // adds event listeners for grid and label button as done earlier
```

```
    grid.addEventListener('change', (event) => {
        var grid = document.getElementById('grid').checked;
        layout['yaxis']['showgrid'] = grid;
        layout['xaxis']['showgrid'] = grid;

        Plotly.react('plot', [curve, points, illuminantE, purpleline], layout, config);
    })

    label.addEventListener('change', (event) => {
    var label = document.getElementById('label').checked;
        if (label) {
            points['mode'] = 'markers+text';
            points['text'] = """ + str(list(map(str, points))) + """;
            illuminantE['mode'] = 'markers+text';
            illuminantE['text'] = ['E'];
        } else {
            points['mode'] = 'markers';
            points['text'] = [];
            illuminantE['mode'] = 'markers';
            illuminantE['text'] = [];
        }
        Plotly.react('plot', [curve, points, illuminantE, purpleline], layout,
            config);
    })

    Plotly.react('plot', [curve, points, illuminantE, purpleline], layout, config);
```

Vedlegg I

Stresstest

Performance test report - May 17, 2024 (#5)

Open in Postman

Postman collection: deployed-tests

Report exported on: May 17, 2024, 21:42:50 (GMT+2)

Test setup

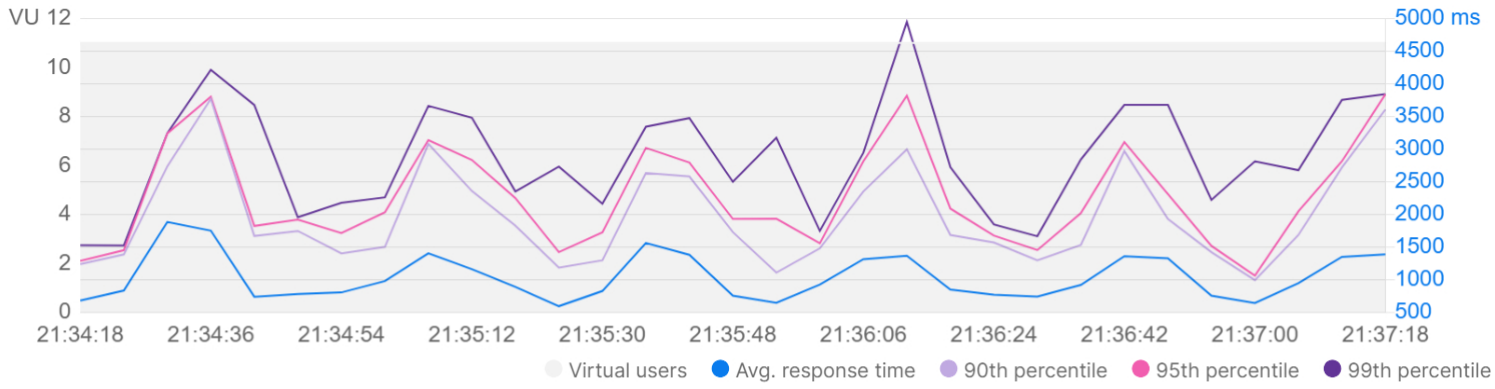
Virtual users	Start time	Load profile
11 VU	May 17, 21:34:11 (GMT+2)	Fixed
Duration	**End time**	**Environment**
3 minutes	May 17, 21:37:19 (GMT+2)	-

1. Summary

Total requests sent	Throughput	Average response time	Error rate
1,836	9.80 requests/second	958 ms	0.00 %

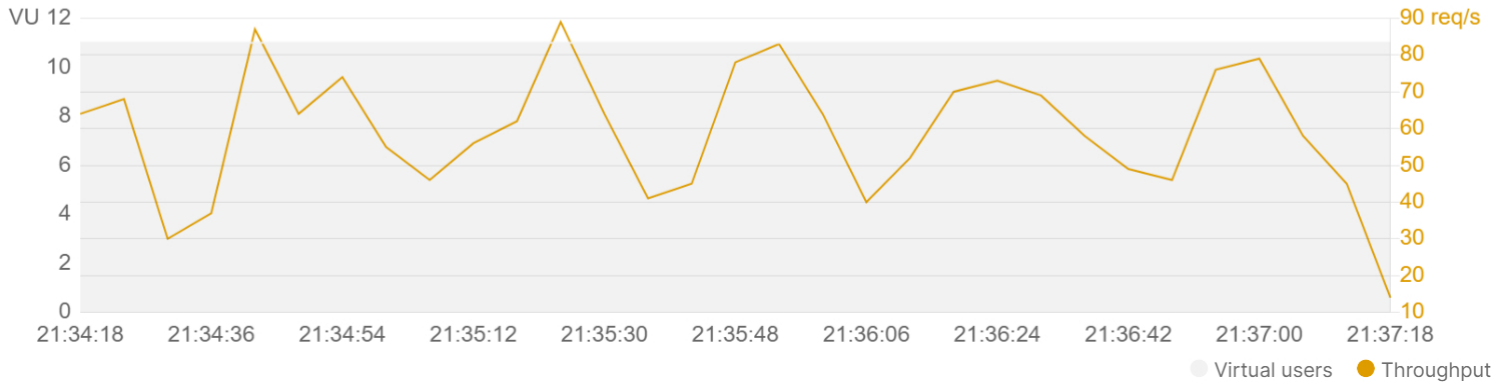
1.1 Response time

Response time trends during the test duration.



1.2 Throughput

Rate of requests sent per second during the test duration.



1.3 Requests with slowest response times

Top 5 slowest requests based on their average response times.

Request	Resp. time (Avg ms)	90th (ms)	95th (ms)	99th (ms)	Min (ms)	Max (ms)
GET xy-plot http://10.212.136.66:8000/api/v2/xy/plot?field_size=2.0&age=23&max=700&min=400	2,370	3,663	3,804	4,946	1,029	4,946
GET xyz-plot http://10.212.136.66:8000/api/v2/xyz/plot?field_size=2.0&age=23&log10&max=700&min=400	1,969	2,998	3,113	3,607	979	3,607
GET xyp-plot http://10.212.136.66:8000/api/v2/xy-p/plot?field_size=2.0&age=23&max=700&min=400	1,882	2,503	2,836	3,771	1,118	3,771
GET xyz-std-plot http://10.212.136.66:8000/api/v2/xyz-std/plot?field_size=2.0	1,538	2,833	3,552	3,841	454	3,841
GET xy-std-plot http://10.212.136.66:8000/api/v2/xy-std/plot?field_size=10	1,439	2,302	2,500	3,343	607	3,343

2. Metrics for each request

The requests are shown in the order they were sent by virtual users.

Request	Total requests	Requests/s	Min (ms)	Avg (ms)	90th (ms)	Max (ms)	Error %
GET lms http://10.212.136.66:8000/api/v2/lms/calculation?field_size=1.0&age=64&min=390.0&max=830.0&step_size=0.5&optional=log,base	70	0.37	242	624	1,179	1,458	0
GET maxwellian http://10.212.136.66:8000/api/v2/lms-mw/calculation?field_size=2.0&age=23&log10&max=700&min=400	70	0.37	112	433	888	1,036	0

GET macleod http://10.212.136.66:8000/api/v2/lms-mb/calculation?field_size=2.0&age=23&log10&max=700&min=400	70	0.37	193	446	760	1,141	0
GET xyz http://10.212.136.66:8000/api/v2/xyz/calculation?field_size=2.0&age=23&log10&max=700&min=400	70	0.37	321	735	1,058	1,530	0
GET xy http://10.212.136.66:8000/api/v2/xy/calculation?field_size=2.0&age=23&max=700&min=400	70	0.37	302	762	1,285	1,633	0
GET xy-std http://10.212.136.66:8000/api/v2/xy-std/calculation?field_size=10	70	0.37	167	590	1,102	1,291	0
GET xyp-calculation http://10.212.136.66:8000/api/v2/xy-p/calculation?field_size=2.0&age=23&max=700&min=400	70	0.37	355	956	1,404	1,877	0
GET xyzp-calculation http://10.212.136.66:8000/api/v2/xyz-p/calculation?field_size=2.0&age=23&max=700&min=400	70	0.37	526	987	1,461	2,050	0
GET xyz-std http://10.212.136.66:8000/api/v2/xyz-std/calculation?field_size=2.0	70	0.37	268	705	1,161	2,319	0
GET lms-plot http://10.212.136.66:8000/api/v2/lms/plot?field_size=2.0&age=23&log10&max=700&min=400	70	0.37	322	857	1,450	2,815	0
GET maxwell-plot http://10.212.136.66:8000/api/v2/lms-mw/plot?field_size=2.0&age=23&log10&max=700&min=400	70	0.37	291	854	1,496	2,720	0
GET macleod-plot http://10.212.136.66:8000/api/v2/lms-mb/plot?field_size=2.0&age=23&log10&max=700&min=400	69	0.37	422	970	1,504	2,970	0
GET xyz-std-plot http://10.212.136.66:8000/api/v2/xyz-std/plot?field_size=2.0	69	0.37	454	1,538	2,833	3,841	0

GET xyz-plot http://10.212.136.66:8000/api/v2/xyz/plot?field_size=2.0&age=23&log10&max=700&min=400	67	0.36	979	1,969	2,998	3,607	0
GET xy-plot http://10.212.136.66:8000/api/v2/xy/plot?field_size=2.0&age=23&max=700&min=400	66	0.35	1,029	2,370	3,663	4,946	0
GET xyp-plot http://10.212.136.66:8000/api/v2/xy-p/plot?field_size=2.0&age=23&max=700&min=400	65	0.35	1,118	1,882	2,503	3,771	0
GET xyzp-plot http://10.212.136.66:8000/api/v2/xyz-p/plot?field_size=2.0&age=23&max=700&min=400	64	0.34	558	1,408	2,795	3,820	0
GET xy-std-plot http://10.212.136.66:8000/api/v2/xy-std/plot?field_size=10	63	0.34	607	1,439	2,302	3,343	0
GET lms-sidemenu http://10.212.136.66:8000/api/v2/lms/sidemenu?field_size=2.0&age=23&log10&max=700&min=400	63	0.34	24	487	1,382	2,313	0
GET xy-std-sidemenu http://10.212.136.66:8000/api/v2/xy-std/sidemenu?field_size=10	63	0.34	73	366	872	1,313	0
GET macleod-sidemenu http://10.212.136.66:8000/api/v2/lms-mb/sidemenu?field_size=2.0&age=23&log10&max=700&min=400	61	0.33	162	680	1,435	2,620	0
GET maxwell-sidemenu http://10.212.136.66:8000/api/v2/lms-mw/sidemenu?field_size=2.0&age=23&log10&max=700&min=400	60	0.32	146	585	1,195	1,841	0
GET xyz-sidemenu http://10.212.136.66:8000/api/v2/xyz/sidemenu?field_size=2.0&age=23&max=700&min=400	60	0.32	307	797	1,523	1,786	0
GET xy-sidemenu http://10.212.136.66:8000/api/v2/xy/sidemenu?field_size=2.0&age=23&max=700&min=400	60	0.32	314	988	1,578	2,448	0

GET xyp-sidemenu http://10.212.136.66:8000/api/v2/xy-p/sidemenu?field_size=2.0&age=23&max=700&min=400	59	0.31	583	1,298	1,933	2,252	0
GET xyzp-sidemenu http://10.212.136.66:8000/api/v2/xyz-p/sidemenu?field_size=2.0&age=23&max=700&min=400	59	0.31	560	1,196	1,952	2,251	0
GET xyz-std-sidemenu http://10.212.136.66:8000/api/v2/xyz-std/sidemenu?field_size=2.0	59	0.31	22	396	977	1,684	0
GET api-main http://10.212.136.66:8000/api/v2	59	0.31	46	511	1,675	3,174	0

3. Errors

This run has no errors

All requests were sent successfully and returned a 2xx response code.

Testing API performance on Postman

Postman enables you to simulate user traffic and observe how your API behaves under load. It also helps you identify any issues or bottlenecks that affect performance.

Learn more about testing API performance.

Vedlegg J

Belastningstest

Performance test report - May 17, 2024 (#7)

Open in Postman

Postman collection: deployed-tests

Report exported on: May 17, 2024, 22:04:26 (GMT+2)

Test setup

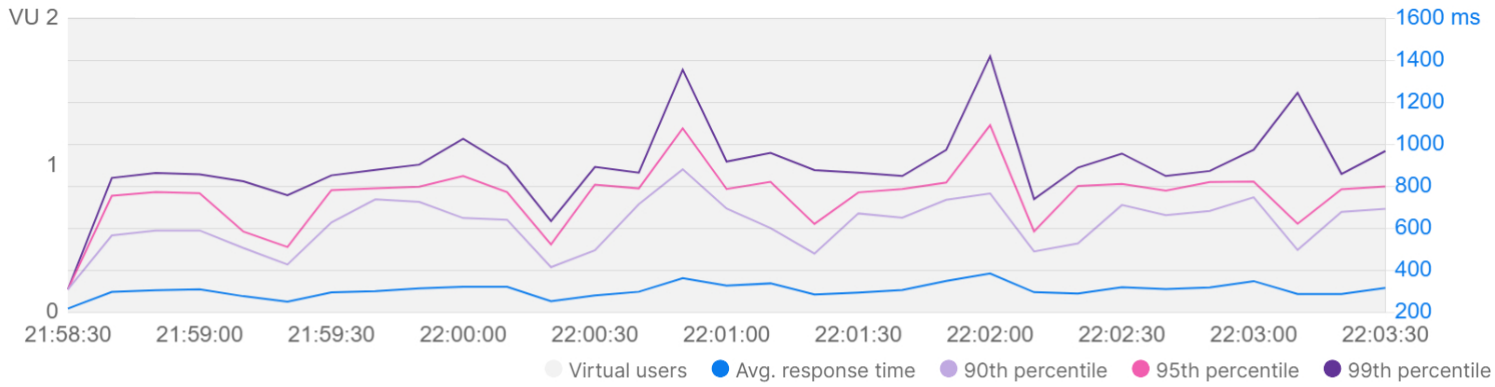
Virtual users	Start time	Load profile
2 VU	May 17, 21:58:31 (GMT+2)	Fixed
Duration	**End time**	**Environment**
5 minutes	May 17, 22:03:39 (GMT+2)	-

1. Summary

Total requests sent	Throughput	Average response time	Error rate
1,403	4.56 requests/second	307 ms	0.00 %

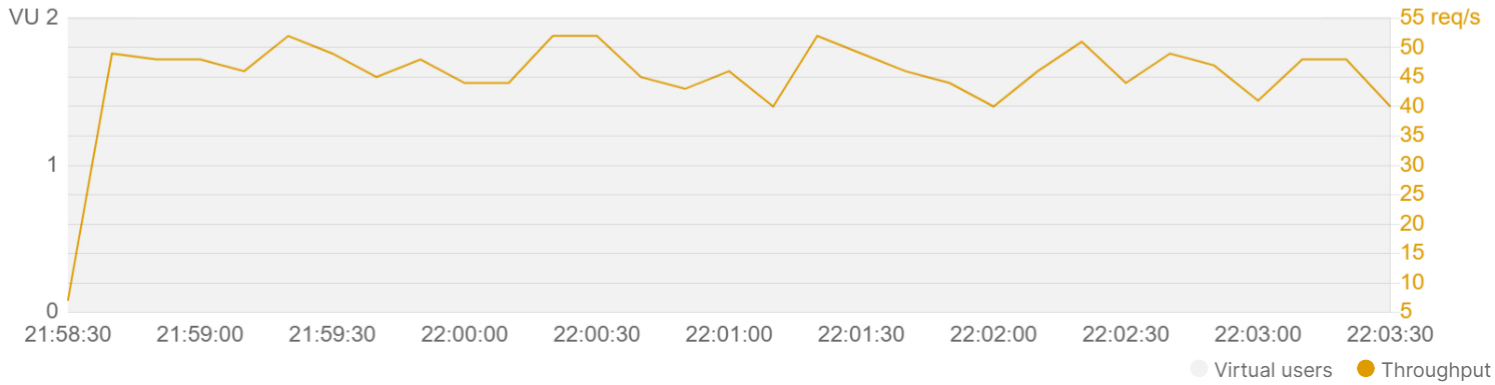
1.1 Response time

Response time trends during the test duration.



1.2 Throughput

Rate of requests sent per second during the test duration.



1.3 Requests with slowest response times

Top 5 slowest requests based on their average response times.

Request	Resp. time (Avg ms)	90th (ms)	95th (ms)	99th (ms)	Min (ms)	Max (ms)
GET xy-plot http://10.212.136.66:8000/api/v2/xy/plot?field_size=2.0&age=23&max=700&min=400	884	974	1,092	1,356	767	1,356
GET xyp-plot http://10.212.136.66:8000/api/v2/xy-p/plot?field_size=2.0&age=23&max=700&min=400	808	878	983	1,340	711	1,340
GET xyz-plot http://10.212.136.66:8000/api/v2/xyz/plot?field_size=2.0&age=23&log10&max=700&min=400	657	737	749	946	539	946
GET xyp-sidemenu http://10.212.136.66:8000/api/v2/xy-p/sidemenu?field_size=2.0&age=23&max=700&min=400	509	533	561	629	475	629
GET xy-std-plot http://10.212.136.66:8000/api/v2/xy-std/plot?field_size=10	429	463	661	1,420	336	1,420

2. Metrics for each request

The requests are shown in the order they were sent by virtual users.

Request	Total requests	Requests/s	Min (ms)	Avg (ms)	90th (ms)	Max (ms)	Error %
GET lms http://10.212.136.66:8000/api/v2/lms/calculation?field_size=1.0&age=64&min=390.0&max=830.0&step_size=0.5&optional=log,base	51	0.17	239	278	313	377	0
GET maxwellian http://10.212.136.66:8000/api/v2/lms-mw/calculation?field_size=2.0&age=23&log10&max=700&min=400	51	0.17	140	167	198	268	0

GET macleod http://10.212.136.66:8000/api/v2/lms-mb/calculation?field_size=2.0&age=23&log10&max=700&min=400	51	0.17	117	173	198	228	0
GET xyz http://10.212.136.66:8000/api/v2/xyz/calculation?field_size=2.0&age=23&log10&max=700&min=400	50	0.16	272	314	360	373	0
GET xy http://10.212.136.66:8000/api/v2/xy/calculation?field_size=2.0&age=23&max=700&min=400	50	0.16	249	287	311	393	0
GET xy-std http://10.212.136.66:8000/api/v2/xy-std/calculation?field_size=10	50	0.16	123	187	221	312	0
GET xyp-calculation http://10.212.136.66:8000/api/v2/xy-p/calculation?field_size=2.0&age=23&max=700&min=400	50	0.16	287	316	336	360	0
GET xyzp-calculation http://10.212.136.66:8000/api/v2/xyz-p/calculation?field_size=2.0&age=23&max=700&min=400	50	0.16	300	352	373	1,008	0
GET xyz-std http://10.212.136.66:8000/api/v2/xyz-std/calculation?field_size=2.0	50	0.16	186	239	284	500	0
GET lms-plot http://10.212.136.66:8000/api/v2/lms/plot?field_size=2.0&age=23&log10&max=700&min=400	50	0.16	224	275	303	435	0
GET maxwell-plot http://10.212.136.66:8000/api/v2/lms-mw/plot?field_size=2.0&age=23&log10&max=700&min=400	50	0.16	223	277	292	1,246	0
GET macleod-plot http://10.212.136.66:8000/api/v2/lms-mb/plot?field_size=2.0&age=23&log10&max=700&min=400	50	0.16	236	278	304	527	0
GET xyz-std-plot http://10.212.136.66:8000/api/v2/xyz-std/plot?field_size=2.0	50	0.16	318	406	491	622	0

GET xyz-plot http://10.212.136.66:8000/api/v2/xyz/plot?field_size=2.0&age=23&log10&max=700&min=400	50	0.16	539	657	737	946	0
GET xy-plot http://10.212.136.66:8000/api/v2/xy/plot?field_size=2.0&age=23&max=700&min=400	50	0.16	767	884	974	1,356	0
GET xyp-plot http://10.212.136.66:8000/api/v2/xy-p/plot?field_size=2.0&age=23&max=700&min=400	50	0.16	711	808	878	1,340	0
GET xyzp-plot http://10.212.136.66:8000/api/v2/xyz-p/plot?field_size=2.0&age=23&max=700&min=400	50	0.16	303	341	355	644	0
GET xy-std-plot http://10.212.136.66:8000/api/v2/xy-std/plot?field_size=10	50	0.16	336	429	463	1,420	0
GET lms-sidemenu http://10.212.136.66:8000/api/v2/lms/sidemenu?field_size=2.0&age=23&log10&max=700&min=400	50	0.16	22	32	42	156	0
GET xy-std-sidemenu http://10.212.136.66:8000/api/v2/xy-std/sidemenu?field_size=10	50	0.16	100	113	129	237	0
GET macleod-sidemenu http://10.212.136.66:8000/api/v2/lms-mb/sidemenu?field_size=2.0&age=23&log10&max=700&min=400	50	0.16	122	156	174	199	0
GET maxwell-sidemenu http://10.212.136.66:8000/api/v2/lms-mw/sidemenu?field_size=2.0&age=23&log10&max=700&min=400	50	0.16	139	153	177	220	0
GET xyz-sidemenu http://10.212.136.66:8000/api/v2/xyz/sidemenu?field_size=2.0&age=23&max=700&min=400	50	0.16	189	202	211	252	0
GET xy-sidemenu http://10.212.136.66:8000/api/v2/xy/sidemenu?field_size=2.0&age=23&max=700&min=400	50	0.16	226	264	278	294	0

GET xyp-sidemenu http://10.212.136.66:8000/api/v2/xy-p/sidemenu?field_size=2.0&age=23&max=700&min=400	50	0.16	475	509	533	629	0
GET xyzp-sidemenu http://10.212.136.66:8000/api/v2/xyz-p/sidemenu?field_size=2.0&age=23&max=700&min=400	50	0.16	381	422	438	612	0
GET xyz-std-sidemenu http://10.212.136.66:8000/api/v2/xyz-std/sidemenu?field_size=2.0	50	0.16	22	29	33	97	0
GET api-main http://10.212.136.66:8000/api/v2	50	0.16	32	59	71	166	0

3. Errors

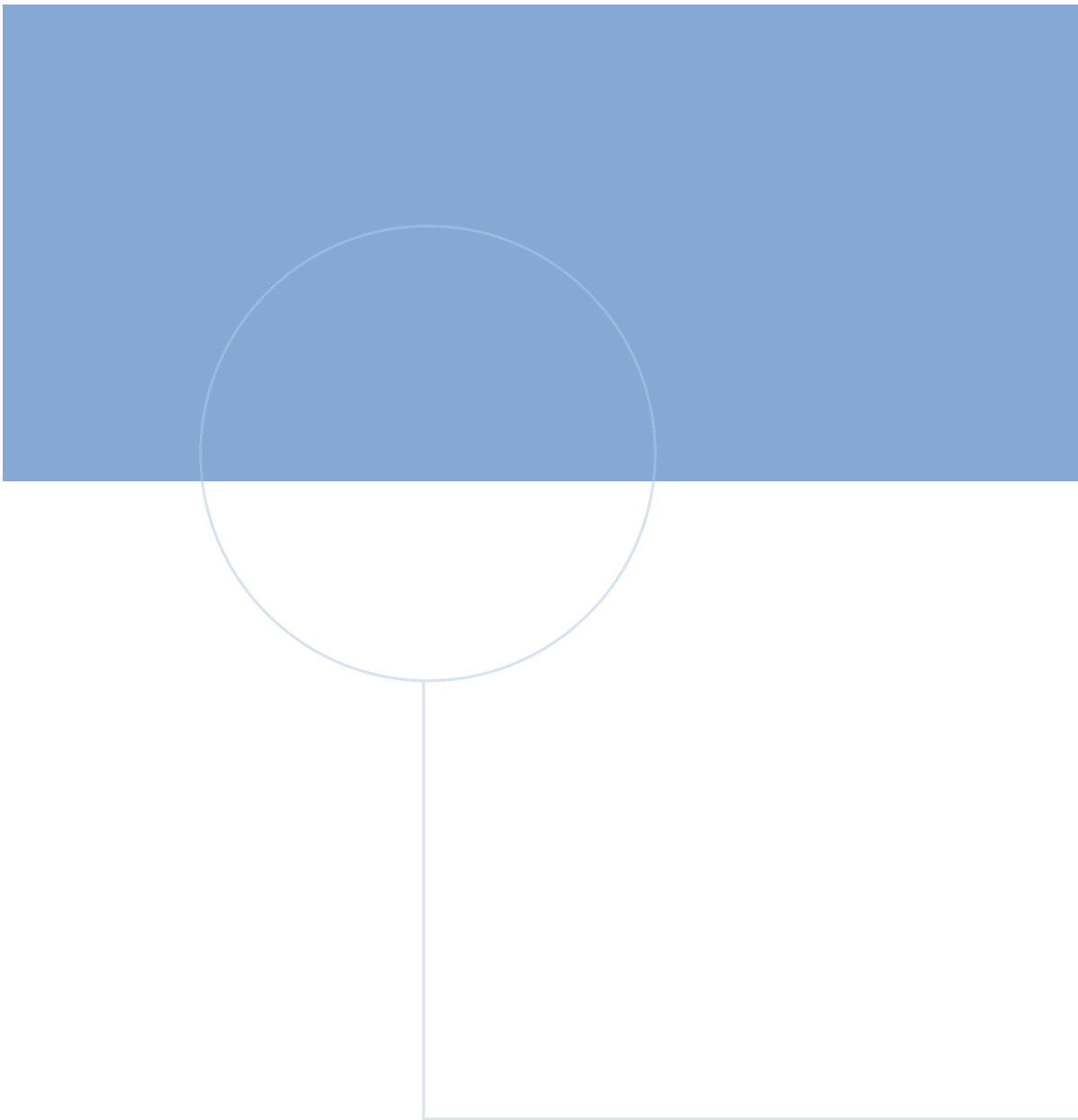
This run has no errors

All requests were sent successfully and returned a 2xx response code.

Testing API performance on Postman

Postman enables you to simulate user traffic and observe how your API behaves under load. It also helps you identify any issues or bottlenecks that affect performance.

Learn more about testing API performance.



NTNU

Kunnskap for en bedre verden