Bacheloroppgave

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

David Spataru
Peter Behncke Nes
Gustav Isaksen Wallden
Herman Lindskog

Cyberwindow

Bacheloroppgave i Bachelor i digital infrastruktur og cybersikkerhet
Veileder: Hanno Langweg

Mai 2022

NTNU
Kunnskap for en bedre verden

David Spataru
Peter Behncke Nes
Gustav Isaksen Wallden
Herman Lindskog

Cyberwindow

Bacheloroppgave i Bachelor i digital infrastruktur og cybersikkerhet
Veileder: Hanno Langweg
Mai 2022

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

NTNU

Kunnskap for en bedre verden

SAMMENDRAG

Tittel :	Cyberwindow	Dato : 19-05-22
Deltaker(e) :	Gustav Isaksen Wallden, David Spataru, Peter Behncke Nes, Herman Lindskog	
Veileder(e) :	Hanno Langweg	
Evt. oppdragsgiver :	NTNU-SOC v/ Christoffer Vargtass Hallstensen	
Stikkord/nøkkelord (3-5 stk) :	Programmering, SOC, NTNU, Webapplikasjon, Sikkerhet	

Antall sider/ord : 64	Antall vedlegg : 5	Publiseringsavtale inngått : Åpen

Kort beskrivelse av bacheloroppgaven :

Denne oppgaven omhandler utviklingen av en webapplikasjon med formål om å hente informasjon fra de forskjellige fagapplikasjonene til NTNU SOC sine APIer, for å så fremstille de grafisk på en oversiktlig måte. Oppgaven kom utifra et behov fra NTNU sin SOC om å aggregere og samle informasjon på ett sted. Oppgaven dekker utviklingsløpet fra spesifikasjon av krav til design, implementasjon og kvalitetssikring. Løsningen er utviklet med moderne webteknologi slik som Django og React, med Scrumban som vår utviklingsmodell for å sikre kontinuerlig fremgang underveis. Ettersom applikasjonen skulle brukes av NTNUs SOC, har applikasjonssikkerheten vært et gjennomgående tema under utviklingen. Prosjektet legger til rette for videreutvikling av applikasjonen.

ABSTRACT

Title :	Cyberwindow	Date : 19-05-22
Participants :	Gustav Isaksen Wallden, David Spataru, Peter Behncke Nes, Herman Lindskog	
Supervisor(s) :	Hanno Langweg	
Employer :	NTNU-SOC v/ Christoffer Vargtass Hallstensen	
Keywords : (3-5)	Programming, SOC, NTNU, Webapplication, Security	
Number of pages/words : 64	Number of appendix :5	Availability :Open

Short description of the bachelor thesis :

This report describes the development of a web application that retrieves information from the various APIs through NTNU SOC's applications, and renders the data in a clear manner. The thesis was requested by NTNU SOC as they had a desire to aggregate information from different systems to one place, and to display that information in an easy digestible way. The thesis covers the development from requirement specification to design, implementation and quality assurance. The application is developed using modern webtechnology such as Django and React, with Scrumban as our development model to ensure continuous progress. As the application was intended to be used by NTNU SOC, application security has been an important aspect to our development. This thesis facilitates further development of the application.

Forord

Oppgaven er skrevet av Gustav Isaksen Wallden, Herman Lindskog, Peter Behncke Nes og David Spataru ved Institutt for Institutt for informasjonssikkerhet og kommunikasjonsteknologi ved NTNU i Gjøvik.

Vi ønsker å takke de som har hjulpet oss gjennom bacheloroppgaven. Først og fremst ønsker vi å rette en takk til veilederen vår, Hanno Langweg ved NTNU i Gjøvik. Møtene med Hanno har bidratt til god progresjon og pushet oss til å legge inn ekstra innsats. Vi ønsker også å takke vår arbeidsgiver med Christoffer Vargtass Hallstensen i spissen, og Hans Åge Martinsen for SOC'en. De har gitt oss muligheten for en avansert, men spennende oppgave. De har hjulpet oss på veien, og vist interesse for å sjekke hvordan prosessen har gått. Vi ønsker også å rette en stor takk til medstudenter og venner. Deres hjelp har vært veldig verdifull for oss.

Helt til slutt takker vi oss selv for utfordrende og lærerrike fem måneder, samt et godt samarbeid.

Innhold

Forord . . . . . **iii**
Innhold . . . . . **iv**
Figurer . . . . . **vii**
Tabeller . . . . . **viii**
Kodelister . . . . . **ix**
Akronymer . . . . . **x**
Ordliste . . . . . **xi**
1 Introduksjon **1**
1.1 Bakgrunn 1
1.2 Oppgavebeskrivelse 1
1.3 Mål og rammer 1
1.3.1 Resultatmål 1
1.3.2 Effektmål 2
1.3.3 Rammer 2
1.3.4 Avgrensing 2
1.3.5 Gruppens bakgrunn 2
1.4 Prosjektorganisering 2
1.4.1 Ansvarsforhold 2
1.4.2 Gruppeinndeling 3
1.5 Rapportens oppbygging 4
2 Utviklingsprosess **5**
2.1 Systemutviklingsmodell 5
2.1.1 Valg av modell 5
2.1.2 Kanban-brett 5
2.1.3 Iterasjoner 7
2.1.4 Overblikk av prosjektfasen 7
2.2 Møter 8
2.2.1 Interne møter 8
2.2.2 Veiledningsmøter 8
2.2.3 Iterasjonsmøter 9
2.3 Product backlog 9
3 Kravspesifikasjon **10**
3.1 Funksjonelle krav 10
3.2 Ikke funksjonelle krav 11

3.3 Sikkerhetskrav . . . 11
3.4 Use Cases . . . 13
3.4.1 Beskrivelse av use caser . . . 14
3.5 Domene modell . . . 15
4 Teknologier . . . 16
4.1 React . . . 16
4.2 Django . . . 17
4.3 PostgreSQL . . . 17
4.4 Create React App . . . 18
4.5 Node.js . . . 18
4.6 npm . . . 18
4.7 Andre teknologier . . . 18
4.7.1 Git . . . 18
4.7.2 Teams . . . 18
4.7.3 Figma . . . 19
5 Systemdesign . . . 20
5.1 Systemarkitektur . . . 20
5.2 Frontend rammeverk . . . 21
5.2.1 Frontendens oppbygging . . . 21
5.3 GUI . . . 21
5.3.1 Innloggingsside . . . 22
5.3.2 Administrasjonsgrensesnitt . . . 22
5.3.3 Toolbar . . . 23
5.3.4 Sidebar . . . 23
5.3.5 Vindu fremvisning . . . 23
5.3.6 Legg til ny tile . . . 24
5.3.7 Presentasjonsgrensesnitt . . . 25
5.4 React-komponenter . . . 25
5.4.1 Index . . . 25
5.4.2 App . . . 25
5.4.3 PresentationUI . . . 26
5.4.4 Tile . . . 26
5.4.5 Sidebar . . . 26
5.4.6 Toolbar . . . 27
5.4.7 Login/Tokenhåndtering . . . 27
5.5 Backend design . . . 27
5.5.1 Database . . . 27
5.6 API . . . 28
5.7 API-Mellomledd . . . 28
5.8 Bruker autentisering . . . 28
6 Brukergrensesnitt . . . 30
6.1 Introduksjon . . . 30
6.2 UX-design . . . 30
6.2.1 Skisser . . . 30

6.2.2 Metode . . . 31
6.2.3 Prototype . . . 31
6.2.4 Brukertesting . . . 32
6.3 Brukervennlighet . . . 32
6.3.1 Heuristiske prinsipper . . . 32
7 Implementasjon . . . 37
7.1 Verktøy og kodebase . . . 37
7.2 Node moduler . . . 37
7.3 Initell oppsett av frontend . . . 37
7.4 Frontend . . . 39
7.4.1 React-komponentene . . . 39
7.4.2 Kommunikasjon med database . . . 45
7.4.3 Kommunikasjon med NTNU API . . . 45
7.4.4 Fremstilling av data . . . 46
7.4.5 Innloggingssystem og tokenhåndtering . . . 49
7.4.6 API . . . 51
7.5 Database . . . 51
7.5.1 Design . . . 52
7.5.2 Database implementasjon . . . 53
8 Kvalitetssikring . . . 56
8.1 Kodekvalitet, lesbarhet og dokumentasjon . . . 56
8.2 Sikkerhetsaspekter under utviklingenn . . . 56
8.3 SonarQube . . . 57
8.3.1 Dependabot . . . 59
9 Konklusjon . . . 60
9.1 Måloppnåelse . . . 60
9.1.1 Oppsummering av målene . . . 61
9.1.2 Oppnåelse av resultatmål . . . 61
9.1.3 Oppnåelse av effektmål . . . 61
9.2 Diskusjon . . . 61
9.2.1 Rapport . . . 62
9.2.2 Applikasjon . . . 62
9.2.3 Møter . . . 63
9.2.4 Arbeidsmiljø . . . 63
9.3 Videre arbeid . . . 63
Bibliografi . . . 65
A Konfidensialitet avtale . . . 68
B Prosjektavtale . . . 71
C Møtereferater . . . 78
D Forprosjekt . . . 82
E Timelister . . . 96

Figurer

1.1 Ansvarsforholdet i et diagram 3
1.2 Rollene til gruppemedlemmene 3

2.1 Detaljert beskrivelse av et kort 6
2.2 Oversikt av Kanban-brettet på GitHub 6

3.1 Use case diagram 13
3.2 Domene modell for applikasjonen 15

5.1 Oversikt over komponentene i systemarkitekturen 20
5.2 Innloggingsside 22
5.3 Forsiden til administrasjonsgrensesnittet 22
5.4 Informasjonen til et valgt vindu 23
5.5 Lag en ny tile 24
5.6 Presentasjonsgrensesnittet 25

6.1 Eksempel på skisse vist frem i demo 31
6.2 Eksempel på status symbol design 33

7.1 Komponent struktur 39
7.2 Flytdiagram over tile komponentens funksjoner 41
7.3 En fremstilling av ulike IP'er som har sendt data over nettverket. 47
7.4 Databasedesign 52
7.5 Generert filstruktur etter de to kommandoene 53

8.1 Første resultat ved bruk av SonarQube 57
8.2 Eksempel på sikkerhetssårbarhet i SonarQube 57
8.3 Resultat etter fikset feil og gjennomgang av sikkerhetshull 58
8.4 Oversikt over alle sikkerhetshull som dependabot fant 59

Tabeller

2.1 Product backloggen 9

3.1 Funksjonelle krav 10
3.2 Applikasjonen består av to grensesnitt 11
3.3 Andre krav 11
3.4 Påloggings autentisering 12

7.1 Liste over node moduler 38

9.1 ID-listen med oppnåd resultater 60

Kodelister

7.1 Div i index.html . . . 40
7.2 Legger til appkomponenten i root elementet . . . 40
7.3 AdminUI returverdi . . . 40
7.4 useEffect for AdminUI . . . 40
7.5 renderTiles funksjonen . . . 41
7.6 Tile Komponent: render-funksjon . . . 42
7.7 Tile Komponent: Mount-funksjon . . . 42
7.8 Parsing av mapper og vinduer . . . 43
7.9 Kode som fremstiller en mappe . . . 43
7.10 Legge til ny mappe eller vindu . . . 44
7.11 Slett mappe eller vindu . . . 44
7.12 Toolbar komponenten . . . 45
7.13 API-Mellomledd - Get forespørsel til NTNU API . . . 46
7.14 Tile komponent: displayData funksjon . . . 48
7.15 Tile Komponent: parseData-funksjonen . . . 48
7.16 Innloggingsspørring på API . . . 49
7.17 LoginAPI i APIhåndtereren . . . 49
7.18 Autoriseringsfunksjon . . . 50
7.19 Token lagring i databasen . . . 50
7.20 Sjekker om tokenen er gyldig . . . 51
7.21 Tile modellen . . . 54
7.22 Tile serializer . . . 54
7.23 Tile view . . . 54
7.24 URL definisjoner . . . 55

Akronymer

ACL Access Control List. 56

CORS Cross-Origin Resource Sharing. 51

CSS Cascading Style Sheets. 16

CVE Common Vulnerability Exposures. 59

GDPR General data protection regulation. 11

HTML HyperText Markup Language. 16, 25, 39, 40, 43

HTTP HyperText Transfer Protocol. 51

JS JavaScript. 16, 17, 39

JSON JavaScript Object Notation. 1, 10, 17, 46, 47, 61

JSX JavaScript XML. 43, 45

LDAP Lightweight Directory Access Protocol. 28, 38

MTTA Mean Time to Acknowledge. 2, 61

MTTD Mean Time to Detect. 2, 61

MTTI Mean Time to Investigate. 2, 61

RMM Remote Monitoring and Management. 21

SOC Security Operations Center (NTNU-SOC). 1, 2, 12, 25, 30, 32

UX User Experience. 7

Ordliste

administrasjonsgrensesnitt Grensesnittet som oppdragsgiver skal kunne styre og konfigurere web applikasjonen fra. v, vii, 2, 8, 11, 14, 21–24, 26, 32, 34

API Endepunkt til en applikasjon, blir i denne applikasjonen brukt for hente data og innloggingsautentisering. vi, xii, 1, 7, 10, 14, 21, 24, 42, 45, 47, 49, 51, 52, 56, 57, 60–64

backend Et utrykk innen programvareutvikling som henviser til koden/funksjonaliteten bak et programvare som brukeren ikke har tilgang til. 16, 17, 20, 51

brukergrensesnitt Betegnelse for kontaktflaten mellom brukeren og datamaskinens operativsystemer og programvare[1]. 17, 32

CIA-triaden CIA-Triaden er en sikkerhetsmodell som tar utgangspunkt i informasjonen som skal beskyttes. Konfidensialitet (Confidentiality), integritet (integrity) og tilgjengelighet (Availability) er de aspektene ved informasjonen CIA triaden vurderer.. 59

frontend Motparten til backend som henviser til strukturen, utformingen og funksjonaliteten av et programvare som brukeren kan se og interagere med [2]. 16, 20, 21, 51

open source Åpen kildekode som kan deles og redigeres av andre parter. 1, 16, 17, 35, 36, 61

presentasjonsgrensesnitt Grensesnittet som skal brukes som fremvisning på skjermene til oppdragsgiver. 8, 10, 11, 15, 17, 21, 25, 26, 32–35, 39, 44

product backlog Product backlog er en logg som beskriver kravspesifikasjonen til det ferdige produktet[3]. 9

render Datamaskin grafikk som er gjenskapt gjennom kalkulasjon av program- og maskinvare. 17

RESTful Representational State Tansfer er en arkitektur stil som tilfører standarder mellom datasystemer på internett slik at de skal kunne kommunisere lettere[4]. 20

state React applikasjonen oppdaterer hva som blir vist basert på hvilken tilstand applikasjonen har. 21

tiles/widgets Inneholder data som er hentet fra NTNU APIene og visuelt fremstiller denne dataen. xii, 1, 10, 11, 14, 21, 23, 26, 33, 35, 40, 41, 51–53, 61

token En unik linje med tekst som blir lagret lokalt som gjør at brukeren kan aksessere applikasjonen til den utløper. 12, 27, 39, 44, 45, 49–51, 53, 56, 62

URL (Uniform Resource Locator) Refererer som oftest til en nettadresse[5]. 12, 24, 25, 34, 51

windows Et vindu i applikasjonen som inneholder en liste med tiles/widgets. 1

Kapittel 1

Introduksjon

1.1 Bakgrunn

NTNU SOC er et sikkerhetsoperasjonssenter lokalisert på NTNU i Gjøvik og ligger under seksjon for digital sikkerhet ved IT-avdelingen. Det fungerer som en digital sikkerhet, beredskapsfunksjon og mottakssenter[6]. Senteret har ansvar for monitorering, analysering og håndtering av sikkerhetshendelser. Det er en beredskapsfunksjon som også utfører digital etterforskning og trusseletterretning.

1.2 Oppgavebeskrivelse

NTNU sin SOC har etterspurt en dynamisk konfigurerbar webapplikasjon hvor brukeren enkelt skal kunne definere ulike såkalte windows, disse vinduene skal inneholde ulike tiles/widgets. De skal fremstille ulike data som hentes fra SOC'en sine allerede eksisterende applikasjoner. Applikasjonen skal også ha en demo funksjon som skjuler all sensitiv informasjon fra vinduene som er oppe, i tilfelle eksterne skal besøke operasjonsrommet.

1.3 Mål og rammer

1.3.1 Resultatmål

Tidsbegrensningen til prosjektet er mellom 10. januar og frem til 20. mai. Gruppen har bestemt seg for at all utvikling skal være ferdig innen 1. mai. Arbeidsgiver har etterspurt følgende krav:

- Applikasjonen skal være dynamisk konfigurerbar
- Applikasjonen skal bli publisert som open source
- Applikasjonen skal bruke NTNU sin merker og fargetema
- Applikasjonen skal bruke moderne webteknologi
- Applikasjonen skal kunne kontrolleres gjennom et API
- Applikasjonen skal kunne lese data fra API'er (JSON)
- Applikasjonen skal følge beste praksis sikkerhetsprinsipper

1.3.2 Effektmål

Ønskelig utfall er at SOC'en skal kunne benytte seg av løsningen, og at den er med på å bedre oversikten over trusler/alarmer som rammer NTNU sine systemer. Løsningen skal være av høy kvalitet for å enkelt kunne videreutvikles av andre.

Derfor definerer vi disse effektmålene for prosjektet:

- Gi en oversikt over informasjon, data og alarmer NTNU SOC må agere på til enhver tid.
- Forbedre SOC KPIer: Mean Time to Detect (MTTD), Mean Time to Acknowledge (MTTA) og Mean Time to Investigate (MTTI). [7]
- Frigjøre digital arbeidsflate på SOC'en arbeidsplasser.
- Applikasjonen skal kjøres på en bærekraftig måte.

1.3.3 Rammer

Administrasjonsgrensesnittet skal kun bli administrert fra en datamaskin, og det legges ikke opp til noe grensesnitt for smarttelefoner for eksempel. Touch funksjonalitet ble vurdert og diskutert på et tidspunkt, men ble uaktuelt da gruppen fant ut at skjermene på SOC ikke hadde denne type funksjonaliteten uansett. Moderne webteknologi skal benyttes, da noe av hensikten er at dette skal være en webapplikasjon som skal kunne utvikles videre på i framtiden, samt av sikkerhetsårsaker.

1.3.4 Avgrensing

Analyse av dataen vil ikke være en del av prosjektet. Funksjonalitet som skal fremheve dataen automatisk via analyse er ikke relevant for oppgaven. Oppgaven skal også være avgrenset til serversiden av applikasjonen. Applikasjonen skal være tilgjengelig på et nettverk via nettlesere, men skal ikke ta stilling til oppsett av klienter. Oppgavens fokus vil være dynamisk framstilling av data og innhenting av den. Forskjellige måter å vise dataen på skal være linje-grafer, søylediagrammer, kakediagrammer eller som tabeller.

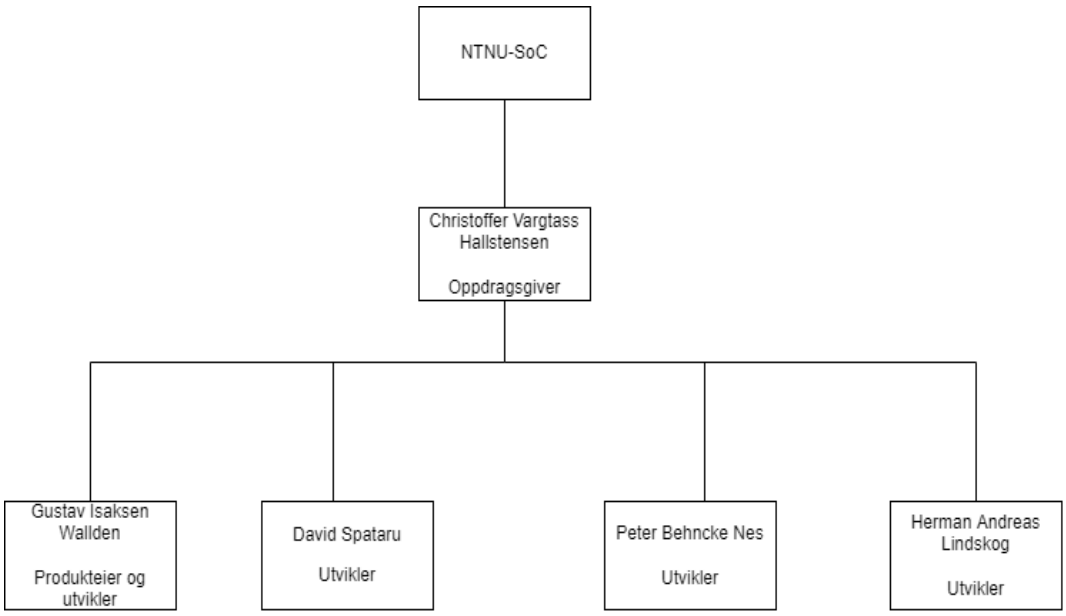
1.3.5 Gruppens bakgrunn

Gruppen består av fire studenter på NTNU Gjøvik på studieprogrammet Digital Infrastruktur og Cybersikkerhet. Vi har fulgt samme studieplan frem til femte semesteret, der vi endte opp med noe variasjon i ulike valgemner som vi valgte. Felles for alle medlemmene er at vi tok valgemner som har en fordypning i matte/statistikk, sikkerhetsemner og algoritmer.

1.4 Prosjektorganisering

1.4.1 Ansvarsforhold

Selv om gruppen består av studenter fra samme studiet så er det fortsatt noen ulikheter innad i gruppa. Disse ulikhetene kommer av forskjellige valgemner, fritidsinteresser og generell kompetanse. Dermed har vi innført som en del av prosjektplanleggingen at gruppemedlemmene har fått tildelt forskjellige roller og forskjellig ansvar under prosjektet.



Figur 1.1: Ansvarsforholdet i et diagram

I ansvarsforhold diagrammet kommer det frem at Christoffer Vargtass Hallstensen er kontaktperson for oppdragsgiveren (NTNU-SOC) som vi har fått til vår bacheloroppgave, og står dermed oppført som oppdragsgiver. Gustav vil fungere som prosjekteier og har ansvaret for prosjektet i sin helhet, samt at han er en del av utviklingsteamet. De øvrige medlemmene er utviklere.

1.4.2 Gruppeinndeling

Figur 1.2: Rollene til gruppemedlemmene

- Prosjektleder
  - Skal holde prosjektet i gang og innenfor de gitte rammene.
  - Organisere møter.
  - Skal ha avgjørende stemme i situasjoner uten flertall.
- Kommunikasjonsansvarlig
  - Skal opprettholde kontakt med arbeidsgiver.

    - Skal opprettholde kontakt med veileder.
    - Skal håndtere all kontakt til og fra gruppa.
- Dokumentansvarlig
    - Ansvar for at dokumenter og koden som er knyttet til prosjektet blir oppbevart på en hensiktsmessig måte.
- Referent
    - Sørge for at det blir skrevet møtereferater under møter.
    - Oppbevare møtereferatene.

1.5 Rapportens oppbygging

Rapporten er satt opp sekvensielt og derfor bør den leses i kronologisk rekkefølge. Under kommer det en rask oversikt over innholdet i hvert enkelt kapittel.

Kapittel 1 - Introduksjon Presentere oppgaven, gruppen og rammer rundt applikasjonen. Gir en innføring i hva bacheloroppgaven handler om og gir en kjapp innføring i hva som kommer videre.

Kapittel 2 - Utviklingsprosess Kapittel to handler om hvordan utviklingsprosessen har vært og om systemutviklingsmodellen. Her gås det også gjennom ulike typer møter som gruppen har hatt.

Kapittel 3 - Kravspesifikasjon Definerer de ulike kravene til applikasjonen, og hvordan applikasjonen er bygd opp i store trekk.

Kapittel 4 - Teknologier Går gjennom teknologiene som har vært sentrale for gruppen gjennom prosjektet.

Kapittel 5 - Systemdesign Systemdesign handler om valgene våre for design av applikasjonen, samt komponenter knyttet til applikasjonen.

Kapittel 6 - Brukergrensesnitt Hvordan vi satte opp de to ulike grensesnittene for den lokale webløsningen vår.

Kapittel 7 - Implementasjon Kapittelet går gjennom hvordan gruppen implementerte de ulike funksjonene for applikasjonen, og teorien bak.

Kapittel 8 - Kvalitetssikring Tiltakene og metodene som ble brukt for å kvalitetssikre løsningen blir beskrevet i dette kapittelet.

Kapittel 9 - Konklusjon Konkluderer og diskuterer rundt resultatet av prosjektet, og drøfter videre arbeid for applikasjonen.

Kapittel 2

Utviklingsprosess

2.1 Systemutviklingsmodell

For å ha en effektiv utvikling på et større prosjekt benytter vi oss av en systemutviklingsmodell. Dette skal avgjøre arbeidsflyten gjennom oppgaveperioden og gi en strukturert arbeidsmetode. Den kommer også til å definere spesifikke arbeidsoppgaver gjennom prosjektet, hvordan de skal fordeles, hvor lenge de skal vare og hvordan samhandling mellom oss i gruppen som utviklere og oppdragsgiver skal foregå.[8]

Det eksisterer flere systemutviklingsmodeller, hvor hver enkelt har fordeler og ulemper ved seg. Av de ulike systemutviklingsmodellene som vi har vurdert kan de kategoriseres som fossefalls- eller smidige-metoder. Vi har valgt å gå med en smidig systemutviklingsmodell, ettersom vår vurdering av en fossefalls metodologi behøver mer erfaring innenfor utvikling; det kreves mer planlegging og at en kan forutse problemer nedover i utviklingsperioden. Av de ulike smidige systemutviklingsmodellene har vi besluttet å benytte oss av Scrumban.

2.1.1 Valg av modell

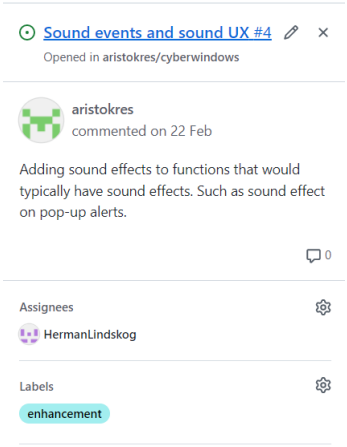
Vi valgte å gå for Scrumban som utviklingsmodell. Scrumban er en kombinasjon av to utviklingsmodeller, scrum og kanban. Ved å anvende den iterasjonsdrevne ideologien bak scrum, og oversiktligheten til kanban metoden får vi en dynamisk utviklingsmodell, som vi kan lett tilpasse til de problemene som oppstår. Vi ønsker ikke å anvende den samme administrative strukturen i Scrum, derav scrum-master og produkteier, men ta i bruk iterasjons-konseptet. Vi ønsker heller ikke å bruke de samme restriksjonene innenfor Kanban hvor man har maksimalt x antall poster under et felt.[9]

En annen fordel med å gå for Scrumban som utviklingsmodell er at vi har brukt den tidligere i andre emner, og er dermed kjent med hvordan den fungerer.

2.1.2 Kanban-brett

Et Kanban-brett er et verktøy for å holde oversikt på arbeidsoppgaver, og prioritere disse. Brettet inneholder mange ulike "kort" med en kort beskrivelse av arbeidsoppgaven som kan tildeles

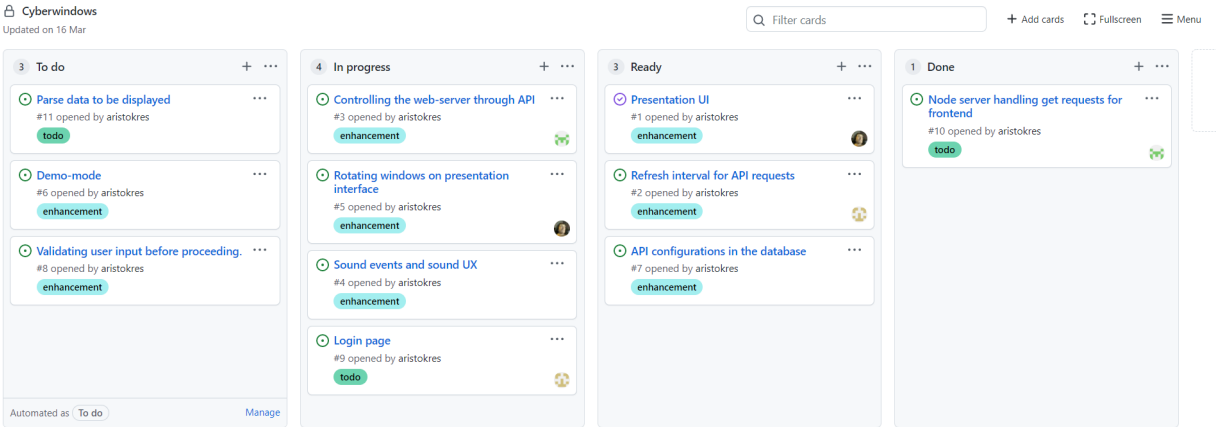
mellom gruppemedlemmene, og dermed gjør det lettvint å holde oversikten på hvem som gjør hva. Gruppen satte opp kanban-brettet via GitHub ved å bruke arbeidsoppgaver, der det legges inn en kort beskrivelse av kortet som overskrift, og det finnes en mer detaljert beskrivelse om en trykker seg inn på kortet. Det er også mulig å legge inn etiketter med relevante ord for å beskrive arbeidsoppgaven.



Figur 2.1: Detaljert beskrivelse av et kort

Kanban-brettet vil bestå av fire felt for arbeidsoppgavene:

- Ikke begynt - To do
- Påbegynt - In progress
- Klar for godkjenning - Ready
- Fullført - Done



Figur 2.2: Oversikt av Kanban-brettet på GitHub

2.1.3 Iterasjoner

En iterasjon og en sprint er to begreper som brukes om hverandre, og de har stort sett samme funksjon. En iterasjon er gjentagelse av oppgaver i en tidsbegrenset periode, i vårt tilfelle hadde de en varighet på to uker. Vi valgte å sette opp åtte iterasjoner, der vi kjørte iterasjonsmøter på starten av iterasjonen. På iterasjonsmøtene flyttet vi kortene til riktig felt etterhvert som vi hadde jobbet med de, definerte ikke-begynt oppgaver, og gruppen fordelte også poster i ikke-begynt feltet på kanban brettet. Målet deretter var at alle postene på ikke-begynt feltet på kanban brettet skal bli fullført før iterasjonen er over. Iterasjonene vil dermed oppdateres hele veien i forhold til de fire feltene som er definert.

Det var vanskelig å definere nye oppgaver hele veien og vurdere lengden på disse, dermed ble det en mer flytende prosess som vi tok tak i etter behov. Vi prøvde uansett å definere oppgaver så godt det lot seg gjøre under oppstarten og iterasjonsmøtene, men det opplevdes som krevende i perioder på grunn av mangel på nok kompetanse innefor området.

2.1.4 Overblikk av prosjektfasen

Gruppen hadde som nevnt åtte ulike iterasjoner i tillegg til prosjektplan-prosessen.

Prosjektplan, 10.jan - 31.jan Det første gruppen gjorde på prosjektet var å sette opp en prosjektplan, og det brukte vi stort hele januar til. Det var viktig for planleggingen og defineringen av oppgaven sin del at vi brukte god tid på denne delen. Den kan leses i vedlegg D.

Iterasjon 1, 1.feb - 7.feb Den første iterasjonen på prosjektet brukte vi på å sette opp UX-design som vi skulle bruke på applikasjonen vår. Gruppen satte også opp utviklingsmiljøet vårt med de teknologiene og løsningene som vi kommer tilbake til i kapittel 4.

Vi brukte også mye tid på å lese oss opp på de ulike teknologiene som vi skulle bruke, da vi hadde lite erfaring med akkurat dette feltet.

Iterasjon 2, 8.feb - 20.feb Under denne iterasjonen begynte vi å sette opp en prototype av brukergrensesnittet og en fungerende webserver med de mest essensielle funksjonene. Denne prototypen ble brukt til å vise frem til oppdragsgiveren vår, slik at vi kunne få tilbakemelding på løsningen vår.

Iterasjon 3, 21.feb - 6.mars Her fortsatte vi kodingen på løsningen vår med de tilbakemeldingene som vi fikk under forrige iterasjon og fortsatte med funksjoner som vi hadde sett for oss på applikasjonen. NTNU APIet var ennå ikke oppe, slik at vi måtte lage noe enkel testdata selv for å teste hvordan håndteringen av API dataen skulle skje.

Iterasjon 4, 7.mars - 20.mars Nå var vi nødt til å se på API-implementeringen for løsningen vår, og hvordan vi kunne gå gjennom dataen gjennom API'et som SOC'en skulle sett opp for oss. Det var under denne iterasjonen det oppstod komplikasjoner, og vi opplevde flere problemer med API'et som vi skulle hente data fra, som vi kommer tilbake til i subseksjon **??**

Iterasjon 5, 21.mars - 3.april Iterasjon fem bestod av å bli ferdig med backend delen av administrasjonsgrensesnittet og pirke litt på frontend delen av dette grensesnittet.

Iterasjon 6, 4.april - 24.april Gruppen sluttstilte presentasjonsgrensesnittet og satte inn støtet med å skrive på rapporten. Under iterasjon fire oppstod det problemer med API-nøkkelen og vi fortsatte å undersøke sammen med arbeidsgiver om når en fungerende løsning ville være tilgjengelig.

Iterasjon 7, 25.april - 8.mai Her hadde vi egentlig planlagt at vi skulle fullføre brukertestingen og sikkerhetsimplementasjonen på applikasjonen vår, men med problemene som oppstod var ikke dette mulig. Dermed begynte alvoret med innspurten med å finpusse koden vi allerede hadde lagd og dytte de siste endringene til live. Vi satte dermed en midlertidlig strek for videre utvikling. Dermed begynte alvoret med å skrive på rapporten.

Iterasjon 8, 9.mai - 20.mai Under den siste iterasjonen så finpusset gruppen rapporten, og fikk andre til å lese over før rapporten ble klargjort for levering.

2.2 Møter

Det ble arrangert ulike typer møter gjennom prosjektet. Vi hadde hovedsaklig møter med gruppen internt, møter med veileder og møter/meldingsutvekslinger med oppdragsgiver. Vi så på det som viktig å sette opp møter for å gi statusoppdateringer på hvordan vi lå an. Møtene våre ble primært gjennomført digitalt over discord og Teams. Se vedlegg C for eksempler på møtereferater.

2.2.1 Interne møter

Interne møter ble gjennomført litt sporadisk etter behov, og hadde en rimelig flytende agenda som gjorde at det ble rom for diskusjon på møtene. På disse møtene hadde vi statusoppdateringer, og videre planlegging av arbeid fremover. Det var gruppeleders ansvar å kalle inn til møtene, og ta opp saker om det var nødvendig. Møtene fungerte bra for å skaffe seg et større overblikk på prosessen.

2.2.2 Veiledningsmøter

Det ble satt opp møte med veilederen vår Hanno Langweg hver fredag fra 10:30 til 11:00, der vi i starten ble enige om at vi skulle møtes etter behov. Her skal arbeid gjort i tidligere uker gjennomgås og ulike spørsmål angående arbeidet kan stilles. Det ble ikke satt noe fast agenda på disse møtene, men fokuset lå ofte på helhetlig progresjon, spesielt det med rapportskriving var et viktig punkt under møtene.

2.2.3 Iterasjonsmøter

Etter at iterasjonene var ferdige så hadde vi interne møter der vi gikk gjennom hva som hadde blitt gjort. Vi brukte stort sett meldingsutveksling på Teams for å gi oppdragsgiver tilbakemelding på hva som hadde blitt gjort under iterasjonene. Vi synes dette fungerte fint, og det at vi brukte Teams som kommunikasjonsplattform gjorde at oppdragsgiver og veileder hadde muligheten til å få med seg hva vi jobbet med til enhver tid.

2.3 Product backlog

Denne product backlogen ble produsert i starten av prosjektet, og inneholder ikke de ekstra beskrivelsene av elementer som gruppen kom på senere tidspunkt.

ID	Description	Estimate (size)	Estimated days	Priority
1	Node server handling get request for frontend	Medium	15	1
2	Toolbar	Small	5	1
3	Sidebar	Small	4	1
4	Tiles	Medium	12	1
5	Presentation UI	Medium	15	2
6	API configurations in the database	Medium	12	2
7	Controlling the web-server through API	Large	25	2
8	Setting up the database	Medium	12	2
9	Refresh interval for API requests	Small	3	3
10	Login page	Medium	10	3
11	Demo-mode	Medium	15	3
12	Rotating windows on presentation interface	Small	2	4
13	Sound events and sound UX	Small	2	4
14	Parse data to be displayed	Medium	16	4
15	Validating user-input before proceeding	Small	5	5

Tabell 2.1: Product backloggen

Kapittel 3

Kravspesifikasjon

Ut fra oppgaveteksten og samtaler vi har hatt med arbeidsgiver så har vi utarbeidet en kravspesifikasjon. Denne inneholder alle funksjonelle og ikke funksjonelle krav, og sikkerhets krav. Den inneholder også use caser, beskrivelse av disse og en use case-modell, samt en domenemodell.

Mer nøyaktig beskrivelser av det som blir definert i kravspesifikasjonen vil være forklart i større detalj utover i rapporten, dette gjelder da alt fra funksjoner til beskrivelser av ulike teknologier vi har valgt å bruke.

3.1 Funksjonelle krav

For å gjøre det lettere for leseren å holde oversikten over kravene, valgte vi å tildele en unik ID for de ulike kravene og dermed komme tilbake til kravene i kapittel 9.

ID	Beskrivelse
F1	Skal kunne lage ulike vinduer, disse skal inneholde tiles/widgets.
F2	Skal kunne lage ulike tiles i vinduene, disse skal fremstille data som er hentet fra APIer.
F3	Skal kunne endre og slette de ulike tilsene, ikke bare hvilke data som fremstilles men hvordan den spesifikke tilen skal se ut f.eks størrelsen.
F4	Skal kunne styres via API slik at man enkelt kan integrere applikasjonen med andre verktøy.
F5	Skal håndtere å hente inn ulik data fra en rekke APIer i JSON-format. Disse skal parses for så å bli fremstilt.
F6	I presentasjonsgrensesnitt så skal det være mulig å gjemme alt sensitivt innhold i tilfelle f.eks eksterne besøkende.

Tabell 3.1: Funksjonelle krav

ID	Beskrivelse
A1	Administrasjonsgrensesnittet skal kun være tilgjengelig for autentiserte brukere. Det er her administratoren kan velge hvilket vinduer og tiles/widgets som skal vises. De kan legge til nye, slette og endre tiles.
A2	Presentasjonsgrensesnitt er det som skal vises på skjermene i SOC-avdelingen. Der skal de ulike tilsene bli fremstilt.

Tabell 3.2: Applikasjonen består av to grensesnitt

ID	Beskrivelse
U1	Applikasjonen må bruke NTNU sine farger og logoer.
U2	Skal publiseres som åpen kildekode slik at det kan arbeides på videre etter at vi er ferdige.
U3	Må ta i bruk moderne webteknologier.

Tabell 3.3: Andre krav

3.2 Ikke funksjonelle krav

3.3 Sikkerhetskrav

- Owasp top ten 2021 er en globalt anerkjent standard for å bevistgjøre de ti mest sentrale risikoene som finnes for nettapplikasjoner.[10] Gruppen har redgjort for alle de ti ulike risikoene, og kommer tilbake til hvilke vi anser som mest sentrale for vår applikasjon senere i 5.7.
    - **1. Brutt aksess kontroll**
        - Handler om at brukere kun kan foreta handlinger innenfor sitt tillatelse området.
    - **2. Kryptografiske feil**
        - Sensitive data er kryptert etter prioritering på viktigheten av dataen, bruk av GDPR og andre lover og regler.
    - **3. Injeksjon**
        - Injeksjoner er at en angriper sender spørringer til en applikasjon med mål om å endre på applikasjonen ved å utføre enkelte kommandoer.
    - **4. Usikkert design**
        - Dårlig utført kontroll av designet til applikasjonen, som kan føre til at det er lettvint for angripere å missbruke.
    - **5. Feilkonfirgurasjon av sikkerheten**
        - Feilkonfirgurasjon handler om for eksempel at unødvendige porter er åpne, og brukere med admin rettigheter eller manglende sikkerhetstiltak på tvers av

applikasjonen.

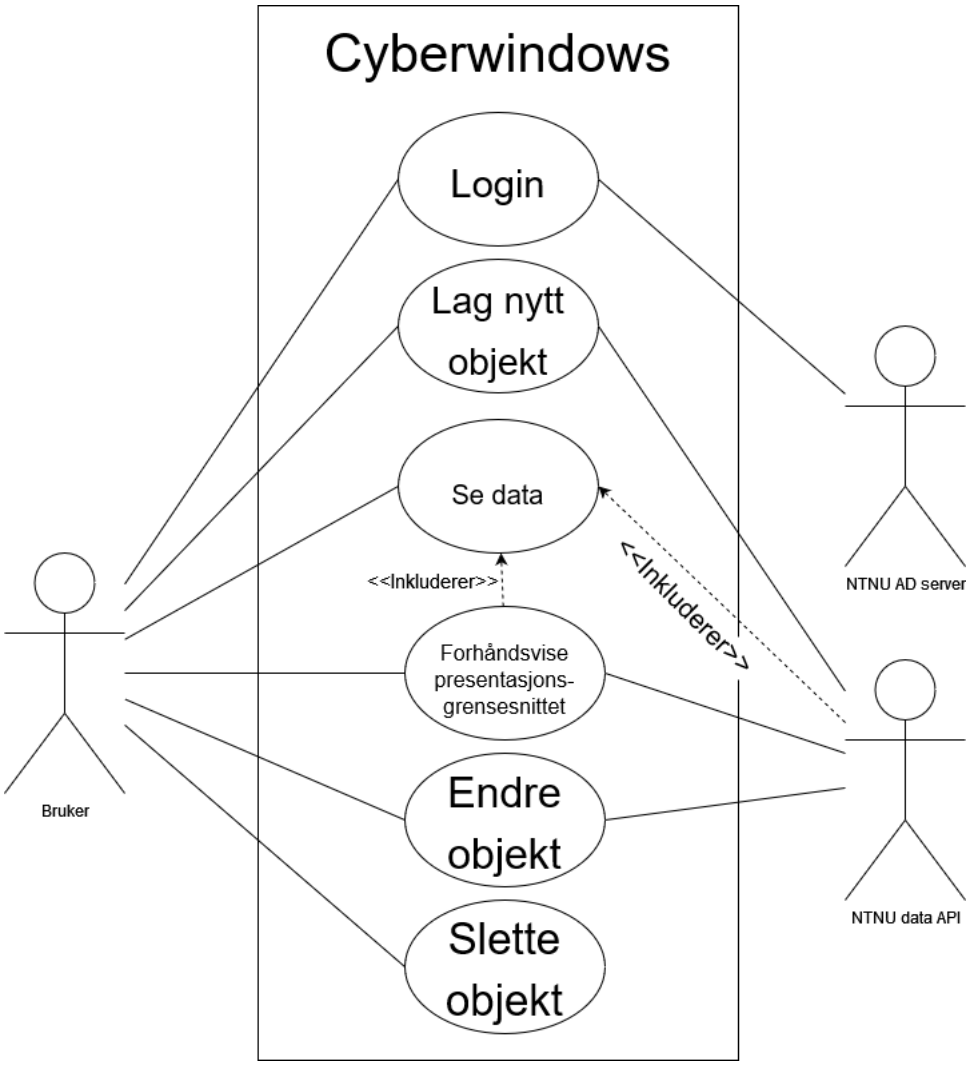
- **6. Sårbare og utdaterte komponenter**
  - Dreier seg om komponenter som ikke er oppdaterte etter bransjestandard, eller komponenter som ikke scannet etter sårbarheter.
- **7. Identifisering og autentiserings feil**
  - Innebærer at det er svakheter som gjør det lettere å missbruke brukere som har for dårlig sikkerhet rundt seg.
- **8. Programvare og dataintegritetsfeil**
  - Det representerer kode som ikke er beskyttet mot integritetsbrudd, som for eksempel applikasjoner som bruker plugins eller moduler som kommer fra uautoriserte kilder.
- **9. Sikkerhetslogging og overvåkningsfeil**
  - Handler om å detektere, eskalere og respondere på sikkerhetsbrudd og mulige trusler som oppstår ved en applikasjon.
- **10. Forfalskning av forespørsler på serversiden**
  - Oppstår ved at applikasjonen ikke sjekker om URL-adressen som kommer inn er validert etter beste praksis, og dermed oppstår muligheten til å sende en forespørsel til en beskyttet destinasjon.

ID	Beskrivelse
S1	Alle som skal ha tilgang til webapplikasjon må logge på med sin NTNU bruker. Alle som vil ha tilgang må også være medlem av SOC på NTNU.
S2	Når bruker blir autentisert så vil de få en token/cookie. Den skal ha en levetid.

Tabell 3.4: Påloggings autentisering

3.4 Use Cases

Vi velger å bruke use caser fordi det gir en god oversikt over hvilke funksjonalitet som applikasjonen skal ha. Vi bruker også et use case-diagram(figur 3.1) for å visuelt fremstille sammenhengen mellom de ulike casene, og gi et overblikk over hvilket caser de ulike brukertypene skal kunne utføre. Brukere i denne sammenhengen kan ses på som ansatte på SOC'en.



Figur 3.1: Use case diagram

3.4.1 Beskrivelse av use caser

Navn: Login
Aktør: Administrator
Mål: Logge på slik at administratoren kan bruke administrasjonsgrensesnittet
Beskrivelse: Administrator skriver inn sitt NTNU brukernavn og passord. Disse blir sendt videre i applikasjonen og blir autentisert via NTNU sitt bruker API. Hvis brukeren eksisterer og er medlem av riktig gruppe så vil innloggingen godkjennes.

Navn: Lage ny vindumappe
Aktør: Administrator
Mål: Lage en ny mappe på sidebaren
Beskrivelse: Administrator lager en ny mappe på sidebaren. I denne kan administratoren legge til flere mapper eller mapper. Det må velges navn.

Navn: Lage et nytt vindu
Aktør: Administrator
Mål: Lage et nytt vindu på sidebaren
Beskrivelse: Lager et nytt vindu med navn. Administratoren velger hvilken mappe denne skal legges til i, potensielt om den ikke skal høre til en mappe og bli lagt i root-mappen.

Navn: Lage en ny tile
Aktør: Administrator
Mål: Lage en ny tile som legges til i et vindu
Beskrivelse: Administratoren kan legge til en ny tile. Når det blir laget nye tiles/widgets så vil administratoren få en del valg over hvordan denne nye tilen skal se ut, blant annet hvilket API som den tilen skal hente data fra. Når administratoren har valgt et API så skal de ha muligheten til å velge hvilket data som skal fremstilles ved å huke av ulike valg.

Navn: Endre på en tile
Aktør: Administrator
Mål: Endre på en tile sine data
Beskrivelse: Velger hvilken til som skal endre data på og kan deretter forandre på alle dataene til den tilen.

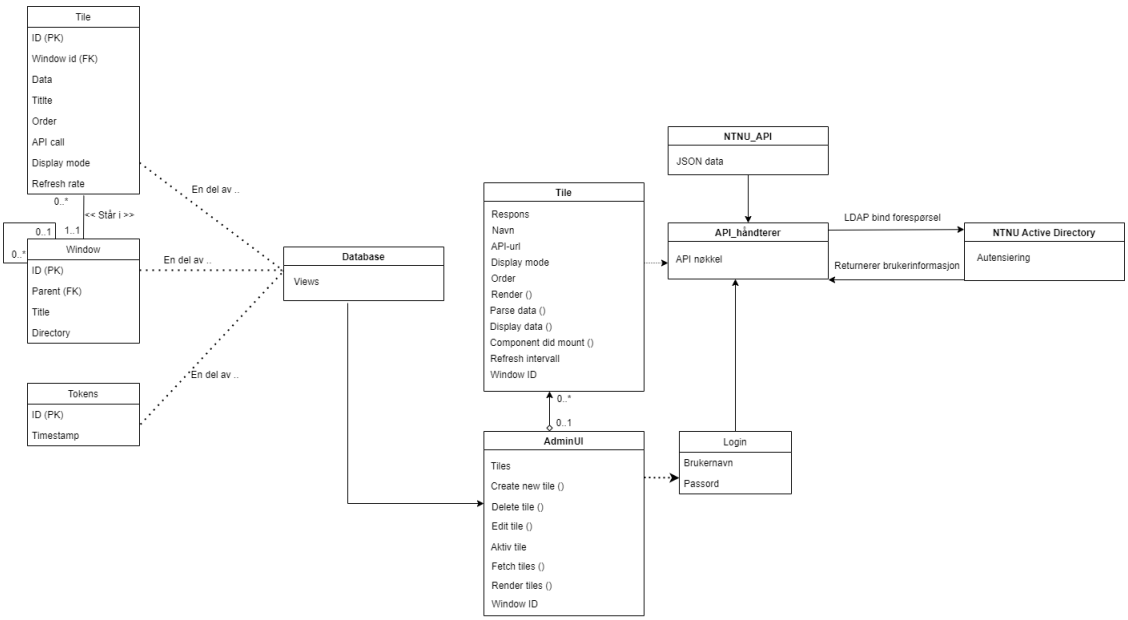
Navn: Slett en tile
Aktør: Administrator
Mål: Slette en tile fra vinduet
Beskrivelse: Velger en tile som skal slettes fra vinduet. Når tilen slettes fra vinduet så vil den slettes fra databasen også.

Navn: Slett vindu/mappe
Aktør: Administrator
Mål: Sletter enten et vindu eller en mappe fra sidebaren
Beskrivelse: Kan slette enten et vindu eller en mappe fra sidebaren. Når et vindu slettes så vil alle tilsene som er knyttet til dette vinduet slettes fra databasen. Når en mappe slettes så vil da alle dens undermapper og vinduer rekursivt slettes fra databasen.

Navn: Forhåndsvisning av presentasjons grensesnitt
Aktør: Administrator
Mål: Se hvordan presentasjonsgrensesnittet vil se ut
Beskrivelse: Administratoren kan velge å få en forhåndsvisning på hvordan presentasjonsgrensesnitt vil se ut når det vises

3.5 Domene modell

En domene modell er en modell som beskriver hvordan de ulike kompotentene henger sammen[11]. Utifra kravspesifikasjonene ble det laget en modell for hvordan alle komponentene i vårt prosjekt er knyttet sammen.



Figur 3.2: Domene modell for applikasjonen

Kapittel 4

Teknologier

Gjennom prosjektet er det brukt en rekke ulike frontend og backend verktøy. Når det kommer til utvikling av web applikasjoner er det ofte en variert kombinasjon av programmeringsspråk og rammeverk som blir brukt. En sammensetning av HTML og CSS brukes for å utvikle selve presentasjonen, strukturen og designet som brukeren faktisk ser og kan samhandle med. Dette er også kjent som frontend og er koden som utformer det brukeren ser visuelt på skjermen sin[2]. JavaScript (JS) på den andre siden blir svært ofte brukt for backend og håndterer da mesteparten av funksjonaliteten som foregår i bakgrunnen av en web applikasjon. Over 97 prosent av nettsider bruker JavaScript for funksjonaliteten på klient siden[12].

Det ble bestemt veldig tidlig i utviklingsprosessen å ta nytte av et JavaScript bibliotek. Et slikt bibliotek har som hensikt å tilby mer brukervennlig funksjonalitet til utvikleren samt å effektivisere utviklingsprosessen, slik at man ikke ender opp med å bruke mye tid på å utvikle funksjonalitet og løsninger som allerede er lett tilgjengelig for gjenbruk. De tilbyr med andre ord ofte gode løsninger for kjente problemstillinger. Majoriteten av nettsider bruker også JavaScript biblioteker. I følge W3Techs[13] bruker over 77 prosent av nettsider et JS bibliotek kalt *jQuery*. Det er desidert ett av de mest populære JS bibliotekene der ute. Til tross for det ble det tatt en avgjørelse om å heller gå med et nokså populært bibliotek kalt *ReactJS* som er noe nyere og mer moderne. Vi begrunner dette videre i delkapittel 4.1 under.

Før vi går videre kan det være hensiktsmessig å differensiere forskjellen mellom et JavaScript bibliotek og et rammeverk. Et bibliotek består av veldefinerte funksjoner som kan kalles på og brukes i egen kode. Et rammeverk på den andre siden tilbyr et skjelett hvor det er meningen at man skal selv fylle ut deler av skjelletet for å koble alt sammen. Mye av hensikten med et rammeverk er å gjøre det raskere å starte med utviklingen av et programvare ved å eliminere mye av oppstartsarbeidet rundt oppkoblingen av ulike elementer.

4.1 React

React (eller *ReactJS*) er et open source og gratis JavaScript bibliotek utviklet og vedlikeholdt av Meta (tidligere kjent som Facebook)[14]. Det ble utgitt for fullt den 29. mai 2013 og har stadig fått mer anerkjennelse og brukere siden[15].

Som en gruppe tok vi en avgjørelse for å implementere dette biblioteket rimelig tidlig i livsløpet

til prosjektet. Ingen på gruppen har hatt tidligere erfaring med det, så det krevde litt tid for å sette seg inn i funksjonaliteten som tilbys. Til tross for det, valgte vi å ta i bruk dette biblioteket med god samvittighet basert på en rekke faktorer. En av hovedfaktorene til avgjørelsen var at React virket veldig moderne og kom med god støtte og oppfølgning av utviklerne. Samtidig er det et nokså populært bibliotek som gjør det enkelt å finne svar på lignende problemstillinger på nettet som andre brukere har støtt på. Industri giganter som f.eks Apple, Netflix og Paypal har også benyttet seg av ReactJS som JavaScript bibliotek[16].

Andre faktorer går på selve funksjonaliteten som React tilbyr. React har banet vei for at man lettvint skal kunne skape interaktive og komplekse brukergrensesnitt. For oppgaven vår var dette av stor betydning da oppdragsbeskrivelsen krever et presentasjonsgrensesnitt som er dynamisk konfigurerbart.

Noen av fordelene ved React:

- Komponent-basert gjør det mer intuitivt å utvikle et helhetlig brukergrensesnitt ved å stykke det opp i flere deler
- Modulær struktur som gjør det mer fleksibelt og tidssparende å jobbe med
- Tilbyr god ytelse for hurtig server-side rendering av ulike komponenter
- Lettvint å bruke og lære dersom man har JavaScript erfaring fra før

4.2 Django

Django er et Python basert nettrammeverk med hensikt om å oppfordre hurtig og fornuftig utvikling av web applikasjoner.[17] Det ble utviklet i perioden 2003 til 2005 og publisert sommeren 2005 som open source.

Det som var attraktivt umiddelbart ved dette rammeverket er at det skulle tilby en enkel og effektiv løsning for å kunne sette opp backend delen av applikasjonen.

Felles for Django og React er at de begge vektlegger gjenbruk av såkalte komponenter for å minimere koderepetisjon og sørge for mer effektiv utvikling. På den måten samspiller begge teknologiene svært bra.

4.3 PostgreSQL

PostgreSQL, ofte forkortet til kun Postgres, er en åpen kildekode objekt-relasjons database som ble laget ut fra POSTGRES prosjektet i 1986. Som navnet tilsier samsvarer teknologien med SQL-standarden og oppfyller 170 av de 179 kravene som er spesifisert i kjerne kravspesifikasjonene for SQL. Valget av postgres ble gjort etter samtaler med oppdragsgiver, men også grunnet postgres sin støtte for bruker-definerte objekter. Dette gjør databasen mer fleksibel etter brukerens behov, men også for dens moderne implementasjon. Dette er spesielt hensiktsmessig for vår del når det kommer til støtte for JSON-data i databasen. [18]

4.4 Create React App

Create React App (CRA) er et verktøy for å sette opp og bygge en webapplikasjon som bruker React, og håndterer en stor del av konfigurasjonen til React og ulike pakker som React er avhengig av. Hensikten med dette verktøyet er å effektivisere kodeprosessen slik at endringer i koden automatisk forårsaker en rekompilasjon av koden. Det er også for at bygging av applikasjonen er avhengig av at den blir strømlinjet gjennom et punkt.[19]

4.5 Node.js

Node er et åpen-kildekode system for å kjøre servere og nettverksapplikasjoner. Systemet er hendelsedrevet og vil behandle spørringer ettersom de kommer og ellers sove i mellomtiden. Løsningen bruker JavaScript og gjør det enkelt ved å inkludere ulike moduler for å behandle kjernefunksjonalitet i koden. I tillegg til Node sine egne grunn moduler, er det lett for utviklere å utvide applikasjonen sin ved å installere pakker og bibliotek fra "npm registry" via "npm" pakkesystemet.[20]

4.6 npm

"npm", originalt forkortet fra Node Package Manager, er en pakkebehandler som automatiserer installering, konfigurering og oppgradering av kode lastet ned fra et kodearkiv kjent som "NPM registry". Det inneholder gratis offentlige pakker, samt private pakker som krever betaling for å få tilgang til.[21]

4.7 Andre teknologier

I tillegg til teknologier som er direkte involvert i funksjonaliteten og utviklingen av systemet, ble det brukt applikasjoner for å bidra til utviklingen av systemt indirekte. Dette innebærer applikasjoner for kommunikasjon, versjonskontroll og design-verktøy.

4.7.1 Git

Git er en åpen kildekode versjonskontrollsystem som ofte brukes av programmerere for å gjøre det lettere å jobbe sammen på et prosjekt.[22] Gruppen har tidligere brukt Git i flere emner, og valget var da lett da vi skulle velge teknologi/plattform for prosjektet.

4.7.2 Teams

Microsoft Teams ble brukt som kommunikasjonskanalen mellom oppdragsgiver, veileder og gruppen. I tillegg ble det også brukt som et samlingssted for ulike dokumenter brukt til organisasjon av utviklingen og for å dele dokumenter mellom oppdragsgiver, veileder og gruppen.

4.7.3 Figma

Figma er et designverktøy som gjør det enkelt å samarbeide i kreasjonen av ulike diagrammer og figurer for et grensesnitt. Verktøyet har en rekke innebygde funksjoner for å sette sammen tekst, ruter, bilder o.l samt endringer skjer i samtid med alle som arbeider på det samme grensesnittet.

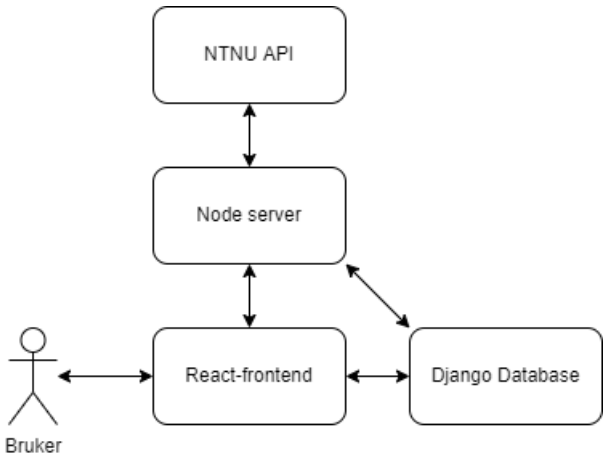
Kapittel 5

Systemdesign

I dette kapittelet vil valg angående design av applikasjonen gjort rede for. Det som dekkes er avgjørelsene for systemarkitekturen og komponenter i systemarkitekturen, vurderingene som ble utført før valg ble tatt, og en evaluering av valgene. Sluttproduktet består av flere komponenter og sub-komponenter, men kan kategoriseres i to deler; “Frontend” og “Backend”.

5.1 Systemarkitektur

Løsningen vi kom fram til innebar å separere frontend og backend helt. Ettersom brukere skal kunne konfigurere de ulike datapunktene som skal vises i hvert vindu, og så ha tilgang til det vinduet fra en annen maskin, valgte vi en sentralisert framgangsmåte. Backenden skal inneholde konfigurasjonen for disse vinduene som frontenden skal hente ut og så gjengi, der all kommunikasjon mellom backend og frontend kun gjøres via RESTful API. Under vises en modell på systemarkitekturen for løsningen vår.



Figur 5.1: Oversikt over komponentene i systemarkitekturen

Av sikkerhetsmessige grunner er det en node.js server som kjører som et mellomledd mellom nettleseren og NTNU's APIer. Det er viktig at API nøkler holdes skjult for brukeren, og skal kun nås ved en konfigurasjonsfil som lagres lokalt på den enkelte sin maskin.

5.2 Frontend rammeverk

Et av de første valgene som ble foretatt var valg av rammeverk for frontenden i applikasjonen. De rammeverkene som ble vurdert var React, Angular, Vue og Ember. Det ble valgt å implementere frontenden i React av følgende grunner; React har et av de største utviklersamfunnet innenfor webutvikling som betyr at det er lettere å finne støtte på diverse forum, samt at gjenbruk av publisert åpen kildekode gjør utviklingen hurtigere. I tillegg gir React en mulighet for Single-Page webapplikasjoner som reduserer trafikken og belastningen på serveren som hoster nettsiden.

5.2.1 Frontendens oppbygging

Brukegrensesnittet til applikasjonen består hovedsakelig av to separate grensesnitt, et administrasjonsgrensesnitt og et presentasjonsgrensesnitt. Administrasjonsgrensesnittet skal kun være tilgjengelig for administratorer. I dette grensesnittet så skal administratoren kunne velge hvilke vinduer som eksisterer og i hvilke mapper de skal ligge. De bestemmer også hvilke tiles/widgets som skal bli fremstilt i hvert enkelt vindu og hvilken informasjon de skal inneholde. Presentasjonsgrensesnittet skal man kun ha mulighet til å velge hvilket vindu som blir vist, informasjonen som blir vist i dette vinduet er allerede bestemt av administrator. Begge grensesnittene vil inneholde hemmelig informasjon, som vil si at man må være en autorisert bruker for å få tilgang til begge.

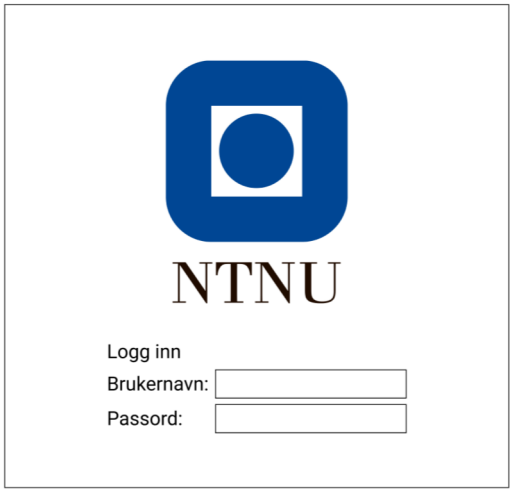
Vi har valgt å bruke React som rammeverk for frontenden vår. React er bygd opp av mange ulike komponenter med statiske og dynamiske deler. Disse komponentene blir oppdatert når komponentens state endrer seg. State-variabler er variabler der en endring medfører en gjenframstilling av komponenten.

Da vi designet brukergrensesnittet hentet vi inn inspirasjon fra flere applikasjoner som gjør noe lignende. Vi fant en type programvare som gjør veldig likt som det vi skal gjøre, denne typen programvare kalles for RMM. Det omhandler om å fremstille informasjon som man kan monitorere fra en avsideliggende lokasjon. Det finnes flere typer RMM-programvare, men den vi tok størst inspirasjon fra var N-Able sin RMM programvare [23].

5.3 GUI

Her beskrives de forskjellige GUI-elementene og kort sammenhengen mellom disse. Nøyaktige beskrivelser på funksjonalitet og React komponentene er beskrevet i implementasjons kapittelet 7.

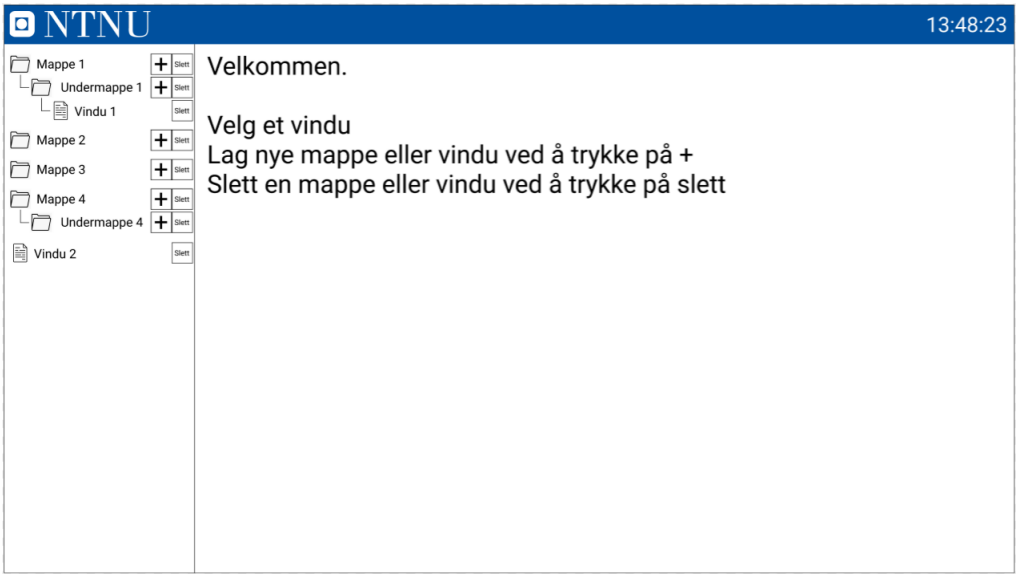
5.3.1 Innloggingsside



Figur 5.2: Innloggingsside

Det første man vil møte på når man starter applikasjonen er en innloggingsside. Her blir man bedt om å logge på med sitt NTNU brukernavn og passord. Nøyaktiv hvordan autentiseringen fungerer og hvem som skal få logge på, blir beskrevet senere i rapporten.

5.3.2 Administrasjonsgrensesnitt



Figur 5.3: Forsiden til administrasjonsgrensesnittet

Når en bruker er blitt autorisert vil de bli videresendt til administrasjonsgrensesnittet. Det er her all informasjon om de ulike mappene og vinduene ligger. Forsiden er bygd opp av ulike react-komponenter.

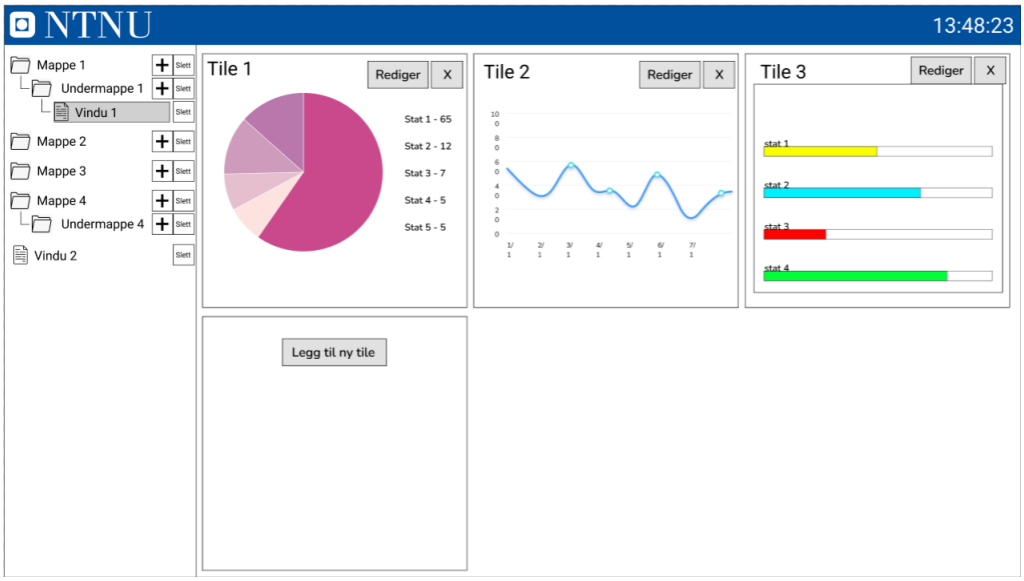
5.3.3 Toolbar

Denne komponenten inneholder en klikkbar NTNU-logo som sender deg tilbake til forsiden og en klokke som er stilt etter system tiden på datamaskinen applikasjonen kjører på.

5.3.4 Sidebar

Her ligger alle mappene og vinduene. Hver mappe vil ha en legg til og slett knapp, hvis man trykker på slett knappen på en overmappe så vil alle dens underliggende mapper og vinduer bli slettet, og brukeren vil bli spurt om de er sikre før de sletter den. Når brukeren skal lage en ny mappe eller vindu så gjør de det på samme sted. For å skille om brukeren skal lage en mappe eller et vindu så må de enkelt huke av om det er en mappe.

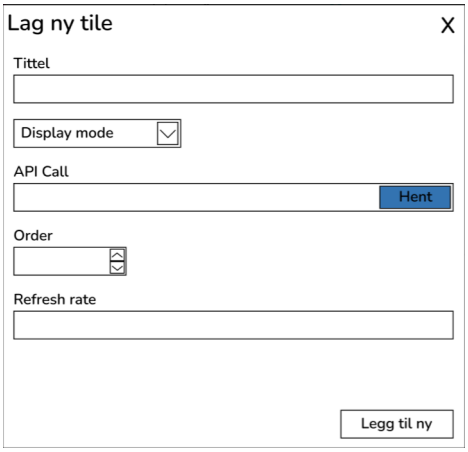
5.3.5 Vindu fremvisning



Figur 5.4: Informasjonen til et valgt vindu

Dette er skjermen som brukeren vil se hvis de velger et vindu. Når de klikker på et vindu så vil alle tilsene til det vinduet komme opp der velkomstteksten var på forsiden. Her kan brukeren legge til en ny tile, og slette eller redigere eksisterende tiles/widgets. Hver tile fremstiller ulik data på forskjellige måter som er definert av brukeren selv. Hvordan denne dataen blir hentet og parset vil bli beskrevet senere i rapporten.

5.3.6 Legg til ny tile



Figur 5.5: Lag en ny tile

Dette er et vindu som lar brukeren definere informasjonen til en tile. Hvis brukeren velger å redigere en tile så er det et identisk vindu de vil komme til.

- **Tittel** - Navnet til tilen
- **Display mode** - Dropdown meny hvor brukeren kan velge hvilken form dataen skal bli fremstilt på, f.eks linjediagram eller søylediagram.
- **API Call** - URL til hvilket API som skal hentes informasjon fra. Etter brukeren har skrevet inn en URL og klikket på "hentså vil det bli fremvist en sjekkliste under. Der brukeren kan huke av og velge spesifikt hvilke data fra det APIet som skal fremstilles i tilen.
- **Order** - I presentasjons og administrasjonsgrensesnittet så vil de forskjellige tilsene alltid bli vist i en spesifikk rekkefølge. Dette alternativet definerer hvor denne tilen befinner seg i rekkefølgen.
- **Refresh rate** - Hvor ofte tilen skal oppdatere seg ved å gjøre en ny spørring på APIet. Dette alternativet finnes fordi situasjonsbildet alltid må være oppdatert. I tillegg så vil det være en minsteverdi på refresh raten som blir satt hvis brukeren ikke skriver inn en refresh rate eller hvis innskrevet verdi er mindre enn minste raten. Dette er fordi at APIene ikke skal bli overbelastet av for mange spørringer på en gang.

5.3.7 Presentasjonsgrensesnitt

Figur 5.6: Presentasjonsgrensesnittet

Dette er grensesnittet som skal vises på de ulike skjermene i SOCen. Presentasjonsgrensesnittet er ikke interaktivt og man bruker det kun for å vise frem de ulike vinduene.

5.4 React-komponenter

React er som nevnt over satt sammen av mange ulike komponenter. For gruppens del er det viktig å definere hva disse skulle være tidlig i prosessen, slik at vi får et overblikk over hvilken funksjonalitet som trengs.

5.4.1 Index

Det aller første som skjer når applikasjonen lastes inn er at index filen blir lastet inn. Det er en HTML-fil med en div som kalles for root. Denne div'en må få en react komponent lastet inn i seg for å definere hva som skal være hovedkomponenten. I vårt tilfelle vil det være react-komponenten "Appsom blir lastet inn der.

5.4.2 App

Dette er hovedkomponenten til applikasjonen. Denne vil konstant være lastet inn og det er den som bestemmer via routing hvilke komponenter som skal være lastet inn til hvilken tid. Routingen i denne komponenten, som bestemmer hvilke komponenter som skal lastes inn, vil være satt opp basert på URLer. Den vil være satt opp slik at sidebar og toolbar komponentene

alltid være lastet inn. Den vil også ha routes til administrasjonsgrensesnitt og presentasjonsgrensesnitt komponentene som vil bli lastet inn basert på brukerens valg. I figur 5.3 kan man se forsiden. Home vil være en egen komponent som inneholder teksten der det står velkommen og gir litt informasjon om oppgaven. Det er denne delen av applikasjon som vil bli oppdatert basert på routing, toolbaren og sidebaren vil være der konstant, med mindre brukeren velger å se presentasjonsgrensesnittet.

AdminUI

Dette er komponenten for administrasjonsgrensesnittet. For å laste inn denne komponenten så må brukeren velge et vindu. Når brukeren har valgt et vindu så vil den se noe slik som i figur 5.4. Denne komponenten er den viktigste i applikasjonen. Det er her administratoren kommer til å lage nye tiles og bestemme innholdet til disse. I denne komponenten så vil det blir gjort mange state forandringer basert på hva brukeren gjør og hvor langt applikasjon har kommet i ulike prosesser. Den vil også inneholde flere API kall, det inneholder databasekall for å lagre og hente inn tiles og den informasjonen til disse tilsene. De tilsene som blir lastet inn er bygd på Tile komponenten som blir nevnt under.

5.4.3 PresentationUI

Denne komponenten tilhører presentasjonsgrensesnittet, som er det grensesnittet som skal bli vist på de forskjellige skjermene i SOC avdelingen. Denne komponenten skal være en enklere versjon av AdminUI-komponenten. Det betyr at brukeren ikke skal få lov til å forandre noe på de ulike tiles/widgetsene, men kun se informasjonen til det vinduet som administratoren har valgt å vise på den spesifikke skjermen. Samme som i AdminUI-komponenten så vil tilesene i denne komponenten være bygd opp på Tile komponenten.

5.4.4 Tile

Denne komponenten definerer hvordan tilesene i applikasjonen fungerer. Den er ikke ansvarlig for å lage tiles/widgets eller lagre disse bort i databasen, det er ansvaret til administrasjonsgrensesnitt. I motsetning til de andre komponentene som ble laget, så er tile komponenten en klassekomponent, kontra en funksjonell komponent. Når man åpner et vindu i administrasjons- eller presentasjonsgrensesnittet så vil applikasjonen gjøre et databasekall som går igjennom alle tilesene som tilhører det spesifikke vinduet og fremstiller disse i den rekkefølgen som er definert. I figur 5.4 så kan man se hvordan en tile vil se ut i administrasjonsgrensesnittet. Der vil man ha muligheten til å redigere, slette og legge til en ny tile. I figur 5.6 så kan man endre hvordan en tile vil se ut i presentasjonsgrensesnittet, i motsetning til i administrasjonsgrensesnittet så vil man ikke kunne slette eller redigere en tile.

5.4.5 Sidebar

Denne komponenten vil være lastet inne på siden hele tiden, utenom da presentasjonsgrensesnittet vises. I denne komponenten vil alle mapper og dens undermapper og vinduer bli fremstilt rekursivt nedover. I figur 5.4 så kan man se hvordan sidebaren vil se ut til en hver tid.

Der vil administratoren ha mulighet til å slette og legge til nye mapper eller vinduer. Hver gang en mappe eller vindu blir laget eller slettet så vil den oppdatere seg ved hjelp av en state forandring.

5.4.6 Toolbar

Toolbar komponenten ligger lastet inn hele tiden, både på administrasjons- og presentasjonsgrensesnittet. Den har en NTNU-logo på seg som vil være klikkbar, den leder tilbake til hjemskjermen som man kan se på figur 5.3. Den skal også ha en klokke på seg som også er react komponent. Den stiller seg i utgangspunktet til servertiden og vil ved intervaller sjekke om den går riktig.

5.4.7 Login/Tokenhåndtering

Hele innloggingsystemet skal være satt sammen av to komponenter, en som håndterer innloggingsinformasjonen som brukeren skriver inn og en som håndterer tokens. I utgangspunktet så skal brukeren skrive inn sin innloggingsinformasjon for sin NTNU bruker, den blir så sendt videre i applikasjonen og det vil bli gjort en spørring mot NTNU APIet. Hvis den godkjennes så skal tokenhåndteringskomponenten gi brukeren en token med en gitt levetid, den tokenen vil så bli sendt videre til apihåndtererer og lagre den bort i databasen.

5.5 Backend design

5.5.1 Database

Det første vi måtte gjøre da vi begynte å designe og utvikle databasen var å bestemme hvilket rammeverk vi skulle bruke. Det var veldig mange rammeverk å velge fra, men vi bestemte oss for å enten gå for Django, Flask, eller web2py, alle disse er Python basert. Django var rammeverket som ble bestemt at vi skulle bruke til slutt. Dette er fordi Django har det største utviklersamfunnet av de tre og vil derfor være best dokumentert, og det vil være flere eksempler for oss å ta av. Flask egner seg egentlig best for Single-Page webapplikasjoner som det vi skal lage, men Flask støtter verken MySQL eller PostgreSQL rett ut av esken.[24] Hovedsakelig på grunn av det så bestemte vi oss for å bruke Django.

Etter valg av rammeverk for backenden så gjenstod det å velge databaseløsning, det stod i mellom MySQL og PostgreSQL. Etter en samtale med arbeidsgiveren vår så spesifiserte han at det var PostgreSQL som skulle brukes. Dette er fordi det er de internt på SOCen som hovedsakelig skal videreutvikle applikasjonen. Der foretrekker de å bruke PostgreSQL på grunn av måten den sin måte å gjøre brukerkontroll og datakontroll gjennom rollestyring. I følge PostgreSQL dokumentasjonen brukes ikke "users"og groups", men roller som inkorporerer de tidligerenvnte til en samlet enhet. Det betyr at man enkelt kan velge hvilke roller som har tilgang til de forskjellige databaseobjektene og enklere avskjære tilgangen til disse.[25]

For kommunikasjon mellom databasen og webapplikasjon så tar vi i bruk RESTapi. For å implementere dette med Django database så ble Django REST frameworkbrukt, dette er en python pakke som man installerer via pip. Den gjør det mye enklere å sette opp en fungerende database som kommuniserer med resten av applikasjonen.

5.6 API

Applikasjonen skal kunne kommunisere med flere eksterne APIer for å få full funksjonalitet. Dette gjelder kommunikasjon for å få ut data som skal bli fremstilt i vinduene, samt å kommunisere med NTNU sitt innloggings API og å hente tiden til Nodeserveren(?). En React-applikasjon kommer ikke med funksjonalitet for å kommunisere med APIer rett ut av esken, man må derfor bruke en node modul som heter Axios. Axios er en løfte-basert HTTP client for node og nettleseren.[26] At den er løfte-basert betyr kort fortalt at den venter på et svar, et løfte har tre states som enten er pending", fulfilledeller rejected".[27] Når den har fått et svar så vil den gå videre til neste steg i spørringen.

5.7 API-Mellomledd

For å forhindre og minske sannsynligheten for at API-nøkler kommer på avveie, så bør det utstedes så få nøkler som mulig. Applikasjonen bør derfor ikke kreve en unik API nøkkel fra hver bruker, men hvordan skal hver bruker få tilgang til dataen med en enkel API-nøkkel? En måte ville vært at hver bruker får tilsendt API-nøkkelen på et vis, men da gjenstår fortsatt problemet med at nøklene kan komme på avveie. En limitasjon av React, og alle applikasjoner, er at alle variabler som lagres i applikasjonen blir eksponert til brukeren med litt leting. Istedenfor å bruke en form for kryptografi for å skjule variabler i applikasjon, var løsningen vi kom fram til etter samtale med oppdragsgiver, at API-nøkler ikke skulle være tilstede direkte i applikasjonen. Eksponering av sensitiv data var en av topp ti største sikkerhetstrusler i 2017 ansett av OWASP[28], og går under kryptografiske feil i 2021 versjonen 3.3. For å forhindre at API-nøkler ble eksponert i nettleseren var det nødvendig å ha et mellomledd mellom webapplikasjonen og APIer den er i kontakt med. På denne måten kan brukere anvende en API-nøkkel uten at de vet verdien av nøkkelen.

En node.js server ble brukt der API-nøkkelen ble lagret som en miljø-variabel og tok for seg alle forespørsler til NTNU's APIer. Applikasjonen sendte en get-forespørsel til node serveren, som videreførte parametrene og gjorde en get forespørsel videre, som så returnerte hva enn den fikk som svar. Sikkerheten av API-nøklene blir da sentrert rundt sikringen av node serveren og maskinen den kjører på. All kommunikasjon som går mellom APIet og node serveren skjer via https.

5.8 Bruker autentisering

Ettersom dataene som blir behandlet av applikasjonen er av sensitiv natur for NTNUs virke, er bruker autentisering en nødvendig del for å oppnå akseptabel sikkerhetsfunksjonalitet. Bruker autentisering gjøres gjennom LDAP autentisering og en spørring og sjekk for brukerens detaljer mot NTNUs ActiveDirectory server. Pakken som er blitt brukt for å oppnå denne funksjonaliteten er 'ldap-authentication' som ble funnet i npm-registry'et[29]. Brukerens innloggingsdetaljer blir tatt inn og en LDAP bind-forespørsel blir sendt, deretter må funksjonen ta hensyn til 3 scenario som kan oppstå.

Scenario 1. Brukeren har tastet feil brukernavn eller passord. Funksjonen returner false og brukeren får ikke tilgang videre i applikasjonen.

Scenario 2. Brukeren har tastet riktig kombinasjon av brukernavn og passord, men er ikke autorisert av NTNU til å ha tilgang til dataen. Dette skjer når brukeren har en NTNU konto, men arbeider ikke for SOC'en i NTNU eller skal av andre årsaker ikke ha tilgang. Brukerens detaljer blir analysert og sjekket for om de er medlem i en gruppe som skal være autorisert til å ha tilgang. Funksjonen returnerer false, og brukeren får ikke tilgang videre i applikasjonen.

Scenario 3. Brukeren har tastet inn riktig kombinasjon av brukernavn og passord og de er autorisert til å se innholdet i webapplikasjonen. Funksjonen returnerer true og brukeren får tilgang til resten av applikasjonen.

Kapittel 6

Brukergrensesnitt

6.1 Introduksjon

Oppgaven spesifiserte et brukervennlig og visuelt grensesnitt, der vi ble fortalt at det skulle være en lokal webløsning. En lokal webløsning vil si at vi lager applikasjonen som kjører lokalt i nettleseren på SOC'en, så lenge maskinen(e) er inne på samme nettverk. Vi brukte anerkjente og etablerte design-prinsipper som basis for hvordan vi bygget applikasjonens utseende.

6.2 UX-design

Oppgaven går ut på at vi skal utvikle en dynamisk web basert løsning som skal fremstille data på ulike skjermer inne i SOC'en. Ved å sette opp en web basert løsning innebærer det at vi må lage /UX design for å ha et utgangspunkt. Dette gjøres både for vår egen del, men også fordi vi ønsket å få tilbakemeldinger fra oppdragsgiver med hvordan de ønsker at sluttproduktet skulle se ut. I delkapitlene under redegjøres det for hvordan vi valgte å sette opp UI/UX designet vårt med ulike teknologier/løsninger.

6.2.1 Skisser

I emnet Programvareutvikling (PROG1004) brukte vi primært programmet BalsamiQ til å sette opp wireframes, noe som også fungerte helt greit. Vi fikk en anbefaling fra en industriell design student om at Figma er et program som lar deg redigere på wireframes i sanntid, og er et anerkjent program som brukes mye innen UI/UX design. Dermed endte vi opp med å lage våre skisser i Figma.

Skissense representer ikke nødvendigvis hvordan sluttproduktet vårt endte opp, men fungerte mer som en pekepinn på hvordan vi ønsket at det skulle bli seende ut (figur 6.1).

Figur 6.1: Eksempel på skisse vist frem i demo

6.2.2 Metode

Ved å gjennomføre en kortere og enklere versjon av en design sprint, kunne vi lande på hvilke skisser vi ønsket å videreutvikle til prototyper i etterkant. En design sprint er normalt satt til fem dager, og fokuset ligger i å øke effektivitet og gjennomførbarheten. Sprinten er ment for å gå fra ide til læring, uten at det skal utvikles noe[30].
Vi valgte å gjennomføre designsprinten vår på tre dager. Første dagen jobbet vi med å forstå hvilke krav som måtte være med i applikasjonen vår. Vi satte opp noen øvelser som vi ønsket å gjennomføre i veldig korte trekk, for å pushe oss litt til å komme i gang med design prosessen.

Andre dagen gikk vi videre på hvordan vi selv ønsket oppsettet av applikasjonen med de kravene som ble satt. Da begynte vi å skissere de ulike ideene som vi hadde i hodet, og kjørte øvelsen som kalles "lightning demos". Denne går ut på at vi samler ideer fra både bransjer som er nærliggende og andre forskjellige bransjer. Deretter kan vi trekke frem det vi liker med de ulike applikasjonene, for å begynne å skissere løsninger.

Siste dagen var ment for oppsummering og fastslå hvilken løsning vi ønsket å gå for, før vi skulle vise skissen til arbeidsgiveren som skulle gi oss en tilbakemelding på hva de tenkte om løsningen vår.

6.2.3 Prototype

Prototyping handler om å sette opp en tidlig utgave av et fremtidig produkt[31]. Ved å gjennomføre sprinten vår, kunne vi sette opp en prototype på selve applikasjonen vår i en slags samling med de viktigste funksjonene som vi skulle ha med i løsningen vår. Her valgte vi å sette opp trådskisser (wireframes) som kommuniserte ut de elementene vi satte opp og bruken av disse, samt plassering av innhold. Poenget med denne lavnivå-prototypen er å tegne et bilde for arbeidsgiver av hvordan sluttproduktet kan bli seende ut, uten å legge for mye arbeid i prosessen. [32]

6.2.4 Brukertesting

Siste fasen av programvareutvikling var brukertestingen av applikasjonen. En workshop ble arrangert slik at de ansatte i SOC'en kunne få et overblikk over applikasjonen og komme med tilbakemeldinger på design og brukervennlighet. Dette gikk under akseptansetestingen, men det var også planlagt å utføre videre brukertesting av en prototype. Grunnet manglende tilgang til NTNU APIet så ble ikke denne planlagte brukertestingen utført.

6.3 Brukervennlighet

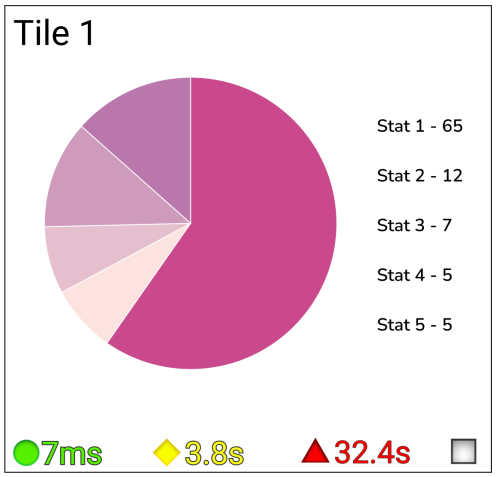
For å kunne skape et brukervennlig brukergrensesnitt ble det fulgt et sett med heuristiske prinsipper utformet og publisert av Jakob Nielsen og Rolf Molich i 1990[33]. Her er det viktig å legge merke til at de blir bemerket som "heuristiske prinsipper" og ikke retningslinjer. Det er fordi de er ment å fungere som veiledning etter beste praksis, ikke håndfaste regler man nødvendigvis må følge. Til tross for alderen er de heuristiske prinsippene fortsatt relevante i dag. Det er allikevel viktig å vurdere om noe kan være utdatert, eller om det foreligger mangler for utviklingen som har skjedd siden. Alt er ikke like relevant for vår oppgave da det er viktig å erkjenne at brukerne av webapplikasjonen (ansatte ved NTNU SOC) er godt erfarne med IT-faget. Dette medfører en unødvendighet for å "idiotsikre" systemet fullstendig. Under går vi gjennom listen[34] med de ti heuristiske metodene og drøfte hvilke prinsipper som har blitt tatt til betraktning og hvorfor. Merk at det blir her drøftet funksjonalitet og implementasjon som er fullført, men også ideer og ønsker som aldri ble iverksatt grunnet tidsbegrensninger og andre prioriteringer.

6.3.1 Heuristiske prinsipper

1. **Synlighet av systemstatus**
   Her skal det foreligge en tydelig indikasjon om systemstatus av webapplikasjonen for brukerne både i administrasjonsgrensesnittet og presentasjonsgrensesnittet. Det kan da være snakk om et grafisk ikon for hver tile som viser status for fremlagt data. Denne statusen skal kunne fortelle brukeren på visuelt vis om dataen blir hentet og oppdatert innenfor et brukerspesifisert oppdateringsintervall, eller om den er utgått og trenger manuell kontroll. Dette intervallet bestemmes forhåndsvis av "refresh rate" egenskapen når man lager en tile.

   Dette er et veldig spesifikt eksempel på implementasjon av synlig systemstatus. Mye av poenget med webapplikasjonen som utvikles, er for at oppdragsgiveren skal kunne konfigurere og fremstille ulike data for monitorering av systemstatus, basert på innhold fra de forskjellige fagapplikasjonene til NTNU SOC. Derfor er det usedvanlig viktig at all informasjon som blir framstilt er oppdatert og nøyaktig. I figur 6.2 under er det gitt et design eksempel på hvordan status indikasjonen kunne ha sett ut for en tile. Grønt ikon vil indikere en god og akseptabel brukerbestemt oppdateringsfrekvens. Gult ikon kan fungere som en varsellampe for at dataen blir hentet ut, og oppdatert tregere enn normalt. Rødt ikon vil falle under et uakseptabelt intervall og betyr at det trengs en kon-

troll. Det grå ikonet til høyre er lagt til som en mulighet for å vise at systemet er nede og tilen henter ikke ut data lenger. Merk at de fargede ikonene er skalert opp i figuren for å gjøre det mer lesbart.



Figur 6.2: Eksempel på status symbol design

2. **Overensstemmelse mellom system og verden**
   Dette går ut på å bruke begreper og et språk i designet av applikasjonen som er enkelt å forstå for de aller fleste. Et eksempel på dette i web applikasjonen er bruken av ordet "vinduer" når det refereres til ulike oppsett av presentasjonsgrensesnittet. Her kunne man fort ha kalt det for noe mer teknisk, men vindu er godt beskrivende og samtidig et begrep som mange er kjent med i data sammenheng. Dette punktet har ikke blitt tatt særlig hensyn til da brukerne av vår webapplikasjon vil være godt kjent med sjargong relatert til faget. Bruken av tekniske utrykk og da spesielt engelske gjenkjennbare utrykk som "API Call" og "Refresh Rate" er heller viktigere å ivareta i denne sammenhengen, istedenfor å skjule de bak oversatte og forvirrende norske begrep.

3. **Bruker kontroll og frihet**
   Dette punktet er av særlig stor betydning for sluttproduktet vårt, da det omfavner hvor konfigurerbar og fleksibel webapplikasjonen er. Jo mer frihet brukeren har til å tilpasse programvaren via forhåndsprogrammerte funksjoner, jo bedre. Det har kommet frem mange idéer både gjennom interne møter med gruppen og via innspill fra oppdragsgiver.

   En idé som oppdragsgiver ønsket var et "drag-and-drop" system utviklet for plassering av tiles/widgets. Tanken er at man intuitivt skal kunne omplassere tiles slik man selv ønsker ved å enkeltvis klikke på de med musa og dra dem til ønsket plass. Videre er det da meningen at plasseringen av de eventuelt andre tilesene skal kunne være såpass fleksible at de tilpasser, og omplasserer seg etter det nye oppsettet. Dette er en løsning som ble nedprioritert og dermed ikke implementert, men det ble bare utforsket ulike løsninger for hvordan det ville fungert.

4. **Samsvar og standarder**
   Her skal man forsøke å holde designet og utformingen av en applikasjon forutsigbar, og gjenkjennelig. Knapper og ikoner som tilbyr lignende funksjonalitet eller symboliserer noe tilsvarende, skal være homogent i utseende. Dette er for å skape forutsigbarhet hos brukeren så det blir lettvint å navigere seg fram, og lokalisere det man leter etter. Det er typisk at knapper for å avbryte en handling, eller logge seg ut av et system skiller seg ut. Måten de ofte skiller seg ut på er i form av plassering, utforming og størrelse. Kanskje den viktigste måten er dog fargebruken. Slike knapper er ofte røde for å tiltrekke oppmerksomhet, og samtidig indikere en advarsel slik at man unngår å trykke på den ved et uhell. Dette kalles ofte for et intuitivt design, der brukeren kan bruke intuisjon og tidligere erfaring til å raskt komme i gang med programvare uten nødvendigheten for ekstern hjelp.

   Utover knapper og ikoner er det flere elementer som inngår i et design som forsøker å være konsekvent. Ord/utrykk, fargebruken og funksjonalitet må også kunne være gjenkjennbar. Eksempelvis har det blitt sørget for i vår webapplikasjon at det skal foreligge minimale ulikheter mellom administrasjonsgrensesnittet og presentasjonsgrensesnittet. De ligner veldig på hverandre og det var tiltenkt slik at når man konfigurerer og tilpasser vinduene i administrasjonsgrensesnittet, skal det være enkelt å se for seg hvordan det endelig ser ut når det videre skal presenteres på andre skjermer. Det har også blitt sørget for at knapper og ikoner holdes mest mulig likt der det gir mest mening. I tillegg til dette krevde oppgaven bruk av NTNU sine logoer og branding. Dette skal også kunne være konfigurerbart slik at man enkelt kan tilpasse det for ulike omgivelser. Her kom oppdragsgiver med innspill om muligheten til en såkalt “dark-mode” da særlig for bruk i administrasjonsgrensesnittet. Dette var også noe som ble nedprioritert og ikke endelig implementert.

5. **Forebygging av feil**
   Dette handler ikke så mye om å forhindre programvarefeil som om å designe programvaren på en slik måte som sørger for at brukerne selv unngår å gjøre feil. Eksempel på dette kan være et tilfelle der brukeren har spesifisert et URL for “API Call” attributten når det skal lages en ny tile. Dersom denne URL’en er feil og peker til noe som de ikke kan hente ut, så er det bedre at det er utviklet en kontrollmekanisme i koden som sørger for at den oppdager dette og samtidig informerer brukeren. På denne måten slipper man å bruke mye tid og ressurser på feilsøking av enkle brukerfeil ved at programvaren kommuniserer dette effektivt tilbake til brukeren der det skulle oppstå.

   Nettopp dette ble nevnt som en løsning i underseksjon 5.3.6 der det er ment at programvaren skal gjøre en sjekk mot den oppgitte URLen, for og så presentere brukeren med en sjekkliste for videre fremvisningsvalg av data. Dersom denne sjekken mislykkes skal det komme opp en feilmelding til brukeren på skjermen som informerer om hva som gikk galt.

6. **Anerkjennelse i stedet for tilbakekall**
   Brukeren av et programvare burde ikke tvinges til å måtte huske overveldende mye informasjon, og detaljer for hver gang man kommer tilbake. Det skal heller sørges for at

tidligere informasjon som er enten påkrevd eller av overordnet betydning for å komme seg videre, er lett tilgjengelig for brukeren hvis nødvendig. Dette kan være brukernavn og passord som blir lagret og forhåndsfylt ved at brukeren huker av en "husk meg" funksjon ved innlogging. En slik funksjon er ikke strengt tatt nødvendig å implementere og heller ikke noe vi fokuserte på, men det er et eksempel på en "quality of life" funksjon som hadde forbedret brukeropplevelsen.

7. **Fleksibilitet og effektivitet i bruken**
   Programvaren skal kunne tilby muligheten for å lagre visse innstillinger, brukerpreferanser og skjermoppsett. Dette er enda et viktig prinsipp for oppgaven vår, da oppdragsgiver spesifiserte en dynamisk konfigurerbar webapplikasjon. Fra møter med oppdragsgiver ble det klart at det var ønskelig med fleksibilitet i oppsettet av ulike tiles/widgets for å kunne lett omplassere og justere dem etter behov. Her er det også viktig med tanke på effektivitet, at en har muligheten til å lagre ulike tilstander av konfigurasjonen. Dette er for å raskt kunne gjenskape vinduer og tiles skulle man være uheldig å miste det.

8. **Estetikk og minimalistisk design**
   For å forhindre et rotete og uoversiktlig design er det viktig å unngå at unødvendig informasjon blir presentert på skjermen. Kun den viktigste informasjonen som brukeren trenger til enhver tid bør vises. Dette er særlig viktig for presentasjonsgrensesnittet der hensikten er at man raskt skal kunne hente seg et oversiktsbilde av det som vises på skjermen(e). Brukeren skal ha friheten til å tilpasse innholdet som vises og på hvilken måte. Tanken er dermed ikke å innføre restriksjoner på antallet eller størrelsen på tiles for å forhindre et rotete grensesnitt, der vil brukeren selv stå ansvarlig for design av oppsettet. Det er klart at det er en underliggende begrensning i designet av tilesene i seg selv, med tanke på at de er utformet som firkanter, som oppfordrer til et estetisk og minimalistisk design. Det vi har prøvd å gjøre gjennom designet på applikasjonen derimot er å tilby et grensesnitt som er enkelt og effektivt i bruken. Informasjonen som vises på skjermen vil til enhver tid konkurrere om plass og oppmerksomhet for annen relevant informasjon. Derfor har det vært forsøkt å forhindre at for mye informasjon vises på skjermen samtidig. Et eksempel på dette er at det dukker opp en dialog boks hver gang man ønsker å legge til en ny tile.

9. **Hjelpe brukere å gjenkjenne, diagnostisere og gjenopprette fra systemfeil**
   I bakgrunn av målgruppen for denne webapplikasjonen og siden den blir publisert som open source, har ikke dette prinsippet blitt vektlagt. Her er dessuten noe av tanken bak prinsippet at man heller skal forklare brukeren hva slags feil som har oppstått, istedenfor å bruke feilkoder. Dette kan virke mot sin hensikt da brukerne i dette tilfellet vil muligens kunne løse programvare- og systemfeil mer smidig dersom de har feilkoder som referansepunkt.

10. **Hjelp og dokumentasjon**
    Det skal foreligge omfattende hjelp og dokumentasjon som er lett tilgjengelig, både i og utenom webapplikasjonen, som brukeren skal kunne oppsøke dersom nødvendig. Innad

i programvaren kan dette være i form av informasjonsbobler som dukker opp når man holder musepekeren over navnet på en funksjon, eller en tekstboks som må fylles inn. Det har ikke blitt særlig vektlagt med dokumentasjon inne i webapplikasjonen, da kildekoden blir publisert som open source. I kildekoden vil det være grundig kommentering for funksjonaliteten som vil fungere som dokumentasjon for eventuelle videreutviklere av programvaren, samt brukerne.

Kapittel 7

Implementasjon

7.1 Verktøy og kodebase

Under utviklingen ble Visual Studio Code brukt som teksteditor ettersom den har innebygd integrering for git, som var versjonskontrollsystemet som ble valgt. For håndtering av pakker, derav nedlasting og installasjon, ble npm brukt for de javascript-pakkene som trengtes i frontenden og pip ble brukt for python-moduler som trengtes i backenden.

7.2 Node moduler

I følgende kapittel så blir de viktigste delene av de node modulene som er brukt for å lage applikasjonen beskrevet. I senere kapiteler så vil disse kun bli nevnt med navn. Alle node moduler blir installert ved å bruke kommandoen "npm install PAKKENAVN", etter dette er gjort så vil de bli lagt til i en modulliste, da trenger man kun å skrive "npm i" for å installere alle modulene i listen. De fleste av modulene har andre moduler som de er avhengige av, disse kalles for "dependencies". Noen av modulene har få, mens noen andre har mange, derfor står det ikke beskrevet hvilke dependencies de ulike har, de står beskrevet under dokumentasjonen til modulene.

7.3 Initell oppsett av frontend

For å begynne kodingen så fort som mulig og på grunn av manglende kunnskap om konfigurering av de ulike teknologiene, så ble Create-React-App brukt som et verktøy for initielt oppsett av webapplikasjonen. To mapper ble laget i github-repo'en, en for frontend og en for backend, og kommandoen "npx create-react-app cyberwindow" ble kjørt i frontend-mappen. Create-React-App lager et start-up script som kan kjøres via "npm run start". Scriptet som blir kjørt er en rekke scripts fra "React-scripts" som setter opp utviklingsmiljøet, testingmiljøet og en server som kjører webapplikasjonen.

Navn og URL	Versjon	Beskrivelse
create-react-app	5.0.0	Blir brukt for initielt oppsett av en react applikasjon. Denne lager engrunnleggende filstruktur og laster ned de viktigste node modulene.
react-pro-sidebar	0.7.1	Gjør det enkelt å lage en dynamisk sidebar. Gir mulighet for å lage submenyer og legge inn menyvalg som er enkelt å lage egen funksjonalitet for.
react-router-dom	6.2.1	Gjør det mulig å sette opp ruter i applikasjonen. I denne applikasjonen blir den brukt til å rute ulike URLer og definere hvilke komponenter som blir lastet inn.
react-bootstrap	2.1.2	Kommer med ferdig definerte react komponenter. Disse har egen CSS og funksjonalitet, slik at man slipper å programmere hver enkelt komponent i applikasjonen fra bunnen av. I denne applikasjonen så brukes modal, forms og button.
axios	0.25.0	Løfte-basert HTTP client. Kan kjøre i nettleseren og node clienter. Blir brukt til å f.eks gjøre API eller databasespørringer hvor det skal bli returnert et svar.
body-parser	1.19.2	Blir brukt for å parsere innkommende forespørsler før applikasjonen og apihåndterer får tilgang til de.
express	4.17.3	Gjør at man kan definere ulike HTTP forespørsler til ulike URLer eller ruter. Blir brukt i apihåndteren for å definere hva de ulike API spørringene til applikasjonen skal håndtere.
cors	2.8.5	Definerer hvilke steder applikasjonen skal kunne hente ressurser fra. I denne applikasjonen så kan man kun hente informasjon fra NTNU APIene.
ldap-authentication	2.2.9	Gjør det enkelt å gjøre LDAP spørringer. Man trenger kun å definere hvilket URL og de ulike dataene som skal bli sendt med i spørringen.

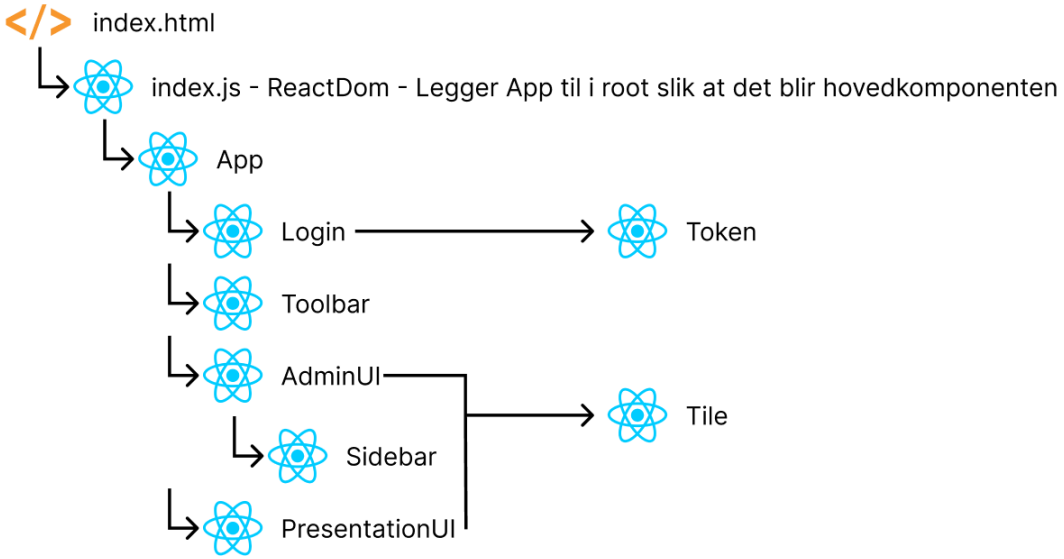
Tabell 7.1: Liste over node moduler

7.4 Frontend

Frontenden er som tidligere nevnt laget med React. og er basert i de Heuristiske prinsippene for å gi brukeren en gjenkjennelig erfaring ved bruk av applikasjonen som beskrevet i 6.3.1.

Brukeren har tilgang til to sider i frontenden, administrasjon- og presentasjonsgrensesnittet.

7.4.1 React-komponentene



Figur 7.1: Komponent struktur

I figur 7.1 kan man se et overblikk over hvordan de ulike komponentene vil snakke med hverandre og i hvilken rekkefølge de lastes inn. Alt begynner i index.html som har en div som via index.js får en hovedkomponent lastet inn. I denne applikasjonen så er det app-komponenten. Etter den blir lagt til i root i index.html så vil den laste inn loginkomponenten. Loginkomponenten vil deretter kommunisere med tokenkomponenten som håndterer tokens. Etter brukeren har fått en token og den er godkjent så vil den laste inn de andre komponentene i App, toolbar og enten AdminUI eller PresentationUI. AdminUI laster inn sidebarkomponenten, og både den og PresentationUI tar i bruk tilekomponenten. Disse komponentene blir beskrevet i mer detalj under.

index.html/index.js

Index.html er første fil som blir lastet inn i applikasjonen når den starter opp. Dette er den eneste HTML filen i hele applikasjon. Dette er fordi resten er bygd opp av React kode og kjører da i JS filer. Det viktigste med denne filen for vår del er div'en med id root. All react kode og komponenter blir lastet i denne div'en.

```
1     <div id="root"></div>
```

Kodeliste 7.1: Div i index.html

Index.js er en sammenhengende javascript fil med index.html. Den spesifiserer at appkomponenten skal bli lastet inn i et HTML element med id root, altså den div'en som bli spesifisert i index.html.

```
1     ReactDOM.render(
2       <React.StrictMode>
3         <App />
4       </React.StrictMode>,
5       document.getElementById('root')
6     );
```

Kodeliste 7.2: Legger til appkomponenten i root elementet

AdminUI

AdminUI komponenten inneholder mesteparten av koden som brukeren er i kontakt med. Den presenterer tiles/widgets for et vindu og har funksjonalitet for å legge til, endre og fjerne tiles. Komponenten returnerer en sidebar komponent, tilesene som skal ligge i det vinduet og en modal komponent som brukes for å legge til og endre tilesene.

```
1  return (
2      <div className="AdminUI">
3          <div className="Sidebar"><Sidebar/></div>
4          <div className="Tiles">{ renderTiles() }
5          <Tile createNew={true} handleCreateNewCB={handleCreateNew} />
6          {customModal(modalState)}
7          </div>
8      </div>
9  );
10 }
```

Kodeliste 7.3: AdminUI returverdi

Når komponenten lastes inn så kjøres det en innhenting av de ulike tilesene som skal vises på det spesifikke vinduet.

```
1  // runs on load + update
2  useEffect( () => {
3      axios
4      .get('http://localhost:8000/api/tiles/')
5      .then((res) => { setTiles(res.data) })
6      .catch((err) => console.log(err))
7  }, []);
```

Kodeliste 7.4: useEffect for AdminUI

Hvert vindu blir lagret i databasen med en unik ID og hver tile har en variabel med hvilket vindu den tilhører. Dette blir brukt av AdminUI'en for å gjengi de tiles/widgetsene som tilhører et spesifikt vindu. All konfigurasjon og data om tilesene blir lastet inn og filtrert på vindu ID'en. Deretter blir det laget en array av tile komponenter i JSX format som blir returverdien av funksjonen.

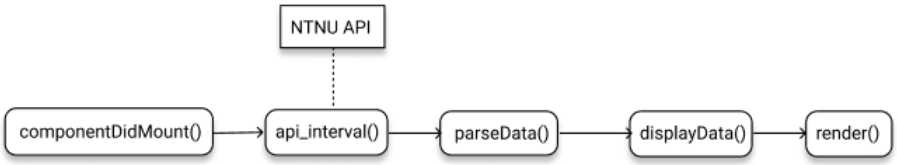
```
1  const renderTiles = () => {
2      const data = tiles.filter(e => e.window_id == params.id);
3
4      const ret = (data.sort((a, b) => a.order-b.order)).map( (tile) => (
5          <Tile
6              data={tile}
7              key={tile.id}
8              deleteTileCB={deleteTile.bind()}
9              updateTileCB={updateTile.bind()}
10         />
11     ))
12     return ret;
13 }
```

Kodeliste 7.5: renderTiles funksjonen

Tile

Tile-komponenten har som jobb å fremstille data fra NTNU APIet. Verdier for hvilken data som skal vises, hvordan dataen skal vises fram og hvor ofte dataen blir oppdatert, blir lagt inn i komponentens egenskaper. Render funksjonen til tilen er den funksjonen som blir kalt når React skal gjengi en komponent.

Et helhetlig bilde av dataflyten og hvilke funksjoner innebæres i framstillingen av dataen kan ses i figur 7.2.



Figur 7.2: Flytdiagram over tile komponentens funksjoner

```
render() {
    return (
        <div className="tile">
            <div className="tile-header">
                <h3>{this.props.data.title}</h3>
                <button className="delete" id={this.props.data.id} onClick={this.props.
    deleteTileCB}>
                    X
                </button>
                <button className="Edit" id={this.props.data.id} onClick={this.props.
    updateTileCB}>
                    Edit
                </button>
            </div>
            {this.displayData()}
        </div>
    );
}
```

Kodeliste 7.6: Tile Komponent: render-funksjon

Når komponenten blir lastet inn lages det en funksjon som blir kalt i satte intervaller. Denne funksjonen har som oppgave å oppdatere dataen som skal vises fra NTNU APIet. Responsen fra APIet blir lagret i en av komponentens state-variabler som tvinger en ny fremstilling av komponenten slik at den nye dataen blir vist.

```
componentDidMount() {
    let refreshRate = this.props.data.refresh_rate;
    if(refreshRate < 20000) refreshRate = 20000;
    const api_interval = setInterval(() => {
        axios
        .get(this.props.data.api_call)
        .then((res) => {
            this.state.response = JSON.parse(res.data);
        })
        .catch((err) => console.log(err))

        this.parseResponse();
      }, refreshRate);

      return () => clearInterval(api_interval)
}
```

Kodeliste 7.7: Tile Komponent: Mount-funksjon

Sidebar

Sidebar er en react-komponent som inneholder oversikten til alle de ulike mappene og vinduene i databasen. I denne komponenten skal administratoren ha mulighet til å lage nye vinduer eller mapper, slette disse, eller redigere vinduene. Denne komponenten inneholder en rekke funksjoner for å håndtere funksjonaliteten som administratoren skal kunne gjøre.

For å få ut alle mappene og vinduene korrekt så gjøres det først et databasekall på window-tabellen. Der henter den alle mappene og vinduene som ligger i tabellen.

```
const parseDirectorylist = (data) =>
{
    res.map( e => e.children = []);
    for(let i = 0; i < res.length; i++)
    {
        if(res[i].parent !== null)
        {
            let pid = res[i].parent;
            let p = res.findIndex(e => pid == e.id)
            res[p].children.push(res[i]);
        }
    }
    return res;
}
```

Kodeliste 7.8: Parsing av mapper og vinduer

Etter alle mappene og vinduene er hentet fra databasen så må de parseres slik at applikasjonen kan fremstille de riktig. I 7.8 så skjer dette. Hvis man tenker på alle vinduene og mappene som noder i en graf, så er hensikten med denne koden å lage en liste av barn til hver node. Listen av barn for hver node lages og så går funksjonen gjennom alle nodene og legger inn noden i forelderen's liste over barn, dersom noden har en forelder. Etter den har sortert alle mappene og vinduene, så vil den legge til listen i en globalvariabel. For å sette opp sidebaren så ble react-pro-sidebar pakken brukt. Denne pakken gjør det veldig enkelt å sette opp en sidebar, det eneste som trengs å definere er hva som er "subitems" og "menuitems". I dette tilfellet så vil alle mapper være subitems og alle vinduer vil være menuitems. Når et "item" blir laget så vil man enkelt kunne definere HTMLkode på de objektene, f.eks en knapp. Alle mappene blir lagt til med mulighet for å slette og legge til en ny mappe eller vindu, hvert vindu blir lagt til med muligheten for å slette det vinduet.

```
    <SubMenu
        title={e.title}
        key={e.id}
        icon={<FaFolder />}
    >
        <div className="menubuttons">
            <button className="create-window" id={e.id} onClick={handleClickNewUnderFolder}>
    Create new window</button>
            <button className="delete-window" id={e.id} onClick={handleClickDeleteUnderItem}>
    Delete</button>
        </div>
        {renderChildren(e)}
    </SubMenu>
```

Kodeliste 7.9: Kode som fremstiller en mappe

Koden i 7.9 er en del av en funksjon som fremstiller alle vinduene og mappene, hvor denne funksjonen tar den globale listen med alle mappene og vinduene. Denne koden er en del av et return statement, som returnerer koden i JSX format. Funksjonen bruker den globale listen og

går rekursivt igjennom det. Etter at alle vinduene og mappene er lagt til i sidebar komponenten så lastes react-komponenten inn igjen på nytt for å få fremstilt alle mappene og vinduene.

```
1     <FormGroup>
2         <label htmlFor="title">Title</label>
3         <input type="text" className="form-control" id='title' onChange={handleInputChange}/>
4     </FormGroup>
5     <FormGroup>
6         <FormCheck type="checkbox" label='Directory' id='directory' onChange={handleInputChange
      }/>
7     </FormGroup>
```

Kodeliste 7.10: Legge til ny mappe eller vindu

Når brukeren skal lage en ny mappe eller vindu så må de trykke på legge til ny mappe knappen. Etter de har gjort det så vil en modal komponent komme opp på skjermen. Denne komponenten vil være lik for vinduer og mapper, men for å spesifikt lage en mappe så må brukeren huke av en avmerkningsboks for om det skal være en mappe eller ikke (6. linje i 7.10). De må også legge til et navn for mappen eller vinduet. Etter brukeren har gjort dette så vil det bli gjort et databasekall hvor den nye mappen/vinduet blir lagt til. Foreldre IDen til objektet som blir lagt til i databsen vil være IDen til den aktive mappen, mens den aktive mappen blir bestemt av react-states. Når brukeren trykker på lag ny mappe/vindu knappen så vil den aktive react-staten bli satt til den mappen som det skal bli laget ny mappe/vindu i. Hvis brukeren velger å lage en ny mappe/vindu som ikke allerede tilhører noe mappe så vil IDen være NULL, dette er fordi default react staten er NULL. Etter at et nytt objekt er lagt til i databasen så vil sidebar-komponenten oppdateres, og det som er nevnt tidligere angående å fremstille sidebaren vil skje igjen.

```
1     <button className="delete-window" id={e.id} onClick={handleClickDeleteUnderItem}>Delete</
      button>
```

Kodeliste 7.11: Slett mappe eller vindu

For å slette en mappe eller vindu så er det lagt inn en slett-knapp (7.11). Det vil bli sendt en delete request til databasen med den nodens ID som parameter. Databasen er satt opp slik at elementene har en foreign-key, som har "on delete cascade"innstillingen, slik at databasen vil slette alle mapper og vinduer som har det elementet som parentID som kan ses i koden 7.21. Etter objektet er slettet så vil sidebar-komponenten bli lastet inn på nytt igjen.

Toolbar

Toolbar komponenten vil alltid være lastet inn, både i administrasjons og presentasjonsgrense-snittet. Hovedgrunnen til at den alltid vil være lastet inn er fordi det er denne komponenten som sjekker om tokenen til brukeren er gyldig, hver gang den blir lastet inn på nytt igjen så vil den sjekke tokenen.

```
1    <nav className="toolbar">
2        <div className="navleft">
3            <Link to="/">
4                <img src={logo}></img>
5            </Link>
6        </div>
7        <div className="navright">
8            <Clock />
9            <button onClick={() => deleteToken()}>Log out</button>
10       </div>
11   </nav>
```

Kodeliste 7.12: Toolbar komponenten

Koden i 7.12 er det komponenten returner, det er kode i JSX format og den bestemmer hvordan toolbaren skal fungere. Den har et bilde av NTNU logoen, en klokke og en utloggingsknapp. Hvis man klikker på NTNU logoen så vil man bli tatt til hjemskjermen, det er også en klokke som skal stilles etter nodeserveren som apihåndteren kjører på. Hvis man klikker på utloggingsknappen så vil tokenen til brukeren slettet, beskrevet i mer detalj i delen som handler om innloggingssystemet i 7.4.5. Etter brukeren har klikket på utloggingsknappen så vil komponenten bli lastet inn på nytt igjen, siden sjekker for tokenen blir gjort i denne komponenten så vil ikke brukeren lenger ha en token og vil dermed bli ledet til innloggingsskjermen.

7.4.2 Kommunikasjon med database

Hver gang brukeren har laget et vindu eller tile så skal informasjon om vinduet eller tilen lagres i databasen. For å forhindre at det finnes inkongruens mellom frontenden i bruk, og databasen, blir dataen fra databasen hentet ut hver gang brukeren beveger seg på siden eller legger til en ny tile/vindu.

7.4.3 Kommunikasjon med NTNU API

Kommunikasjonen med NTNUs APIer skjer gjennom en API portal laget gjennom Gravitee. For å få tilgang til APIet må man få en nøkkel av de som overser API portalen. Et problem som oppstod under utviklingen var at tilgangen til NTNU APIet var kun tilgjengelig i to uker.

En node server ble satt opp som et mellomledd mellom applikasjonen og APIet, med formål å skjerme API nøkkelen til brukeren av applikasjonen. Node serveren har en rekke endpoints som samsvarer med de endpoints'ene som eksisterer i NTNUs API. Serveren kjører på port 8080 og laster inn verdier fra filen ved navn '.env' som ligger i samme lokasjon der koden for serveren ligger. For å forenkle tilgangen til APIet fra applikasjonen gjøres det en enkel get-forespørsel mot node serveren, som videresendes til NTNUs API med korrekte header verdier.

```
app.get('/ssh', (req, res) => {
    const options = {
        method: 'GET',
        url: 'https://apitest01.soc.ntnu.no:8082/surtest/ssh',
        headers: {
            'X-Gravitee-Api-Key': process.env.API_KEY,
        },
    }
    axios.request(options).then((response) => {
        let data = response.data;
            data = data.replaceAll('\n', ',');
            data = '[' + data;
            data = data.slice(0, data.length - 1);
            data = data + ']';
        res.json(data);
    }).catch((err) => console.log(err))
});
}
```

Kodeliste 7.13: API-Mellomledd - Get forespørsel til NTNU API

Før dataen blir sendt til frontenden så må den omgjøres. Dataen som blir mottatt fra NTNUs APIer er ikke gyldig JSON kode tatt i helhet. NTNUs API er satt opp slik at det blir sendt en linje med alle objekter som skilles ved newline karakterer, men for å jobbe med dataen videre må objektene ligge sammen i en klammeparentes, og skilles ved komma-tegn. Vi omgjør dataen slik at den samsvarer til JSON standarden RFC8259.[35]

Strengen som mottas er på formatet:

```
{ \"element1\":\"verdi1\" ... }\n{ ... }\n ...
```

Strengen omgjøres for videre parsing ved å fjerne omvendt-skråstreker '\' foran apostrofer '"', bytte alle newline karakterer med komma ',' og så sette inn en klammeparentes på starten og slutten av strengen.

Formatet på strengen etter omgjøring ser slik ut:

```
[{ "element1": "verdi1" ... },{ ... }, ...]
```

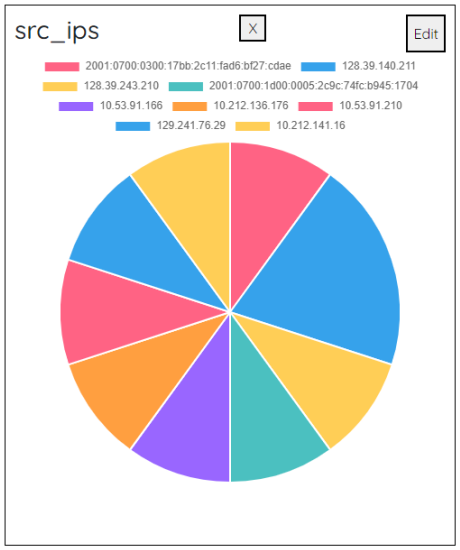
APIet har en maksimum grense for antall forespørsler som kan gjøres per minutt. Hver tile i applikasjonen har en konfigurerbar variabel, refresh rate, som tilsier hvor ofte dataen som skal fremstilles i den tilen blir hentet. Minimumsverdien for denne variabelen er 20 sekunder.

7.4.4 Fremstilling av data

Det er flere behov for hvordan dataen skulle fremstilles, og ulike design ble sendt til og vurdert av oppdragsgiver. Applikasjonen støtter kakediagrammer, søylediagrammer og tabelloversikter. Hva som vises i de ulike diagrammene er basert på brukervalg. Etter brukeren velger hvilket

API som data skal hentes fra, gjøres det en get-forespørsel for å se dataen som sendes fra det APIet. Brukeren velger type diagram og velger deretter hvilke datapunkter som skal vises i henhold til fremstillingsmetoden.

Fremstillingsmetodene har et nomenklatursystem hvor det første ordet beskriver typen diagram, og deretter følger en beskrivelse av hvordan dataen skal selekteres, slik at det står på formen <Diagramtype-Seleksjonsmetode>. Som et eksempel, et type kakediagram skal vise antall forekomster av en verdi x av et datapunkt y. Et API som viser server informasjon kan gi informasjon om serveren er oppe, om den er i en gjenoppstartsprosess eller er nede. Da vil serverstatus være verdien som summeres opp og serverstatusen mulige verdier er de navnet på de ulike datapunktene som skal vises.



Figur 7.3: En fremstilling av ulike IP'er som har sendt data over nettverket.

For fremstilling av dataen brukes Chartjs. En pakke ved navn react-chartjs-2 ble brukt som en metode å bruke chartJS som react komponenter. Chartjs krever at dataen ligger i et JSON objekt. I JSON objektet så trengs det noen nøkkel-verdi par: label og data, og framstillingen kan utbredes med flere nøkkel-verdi par som bestemmer mer om utseende til dataen som fremstilles.

Før dataen vises må dataen parses og legges inn i et format for ChartJS. Basert på hvilke valg brukeren gjør for når de lager tilen, så gjøres dataen klar for fremstilling forskjellig. Under ser man et eksempel på hvordan pie-x-by-yfremstillingen skjer, der denne typen diagram er et kakediagram hvor antall instanser av et datapunkt blir summert.

```
displayData()
{
    this.parseData();
    switch(this.props.data.display_mode)
    {
        case 'pie-x-by-y':
            return(<Pie data={this.state.displaydata}/>);
            break;
        ...
    }
}
```

Kodeliste 7.14: Tile komponent: displayData funksjon

Under er et utklipp av parseData-funksjonen for hvordan dataen som skal vises i et pie-x-by-y diagram blir preparert. Hvert unikt datapunkt i dataen blir lagt til i en array og så summert opp.

```
parseData()
{
    if(Object.keys(this.state.response).length === 0)
        return;

    switch(this.props.data.display_mode)
    {
        case 'pie-x-by-y':
            let sums = [];
            let labels = [];
            // Add all unique x points to labels
            this.state.response.map((e) => {
                if(labels.find(element => element === e[this.state.datapointx]) === undefined)
                    labels.push(e[this.state.datapointx])
            });
            // Sum up all occurences of datapoint x
            for(let i = 0; i < labels.length; i++)
            {
                sums.push(0);
                this.state.response.map((e) => {
                    if(e[this.state.datapointx] === labels[i])
                        sums[i] += 1;
                });
            }
            let data = {
                data: sums,
                backgroundColor: this.state.chartColors,
            }
            this.setState({ displaydata: { "labels": labels, "datasets": [data] } })
            break;
        ...
    }
}
```

Kodeliste 7.15: Tile Komponent: parseData-funksjonen

7.4.5 Innloggingssystem og tokenhåndtering

Innloggingssytemet er bygd på å gjøre spørringer på NTNU sin LDAP-tjener. Det innebærer at brukeren må skrive inn sitt NTNU brukernavn og passord, denne informasjonen blir sjekket opp mot NTNU sitt API. Denne spørringen vil returnere informasjonen til brukeren, og brukeren vil bli autorisert hvis de er medlem av NTNU sin -avdeling. Hvis brukeren blir autorisert så vil de få returnert en token som blir lagret lokalt, denne tokenen blir så lagret i databasen. Tokenen blir lagret som en string sammen med UNIX-tiden den ble generert på. Det er tokenen i databasen som brukeren sin token, blir sjekket opp mot. Den blir sjekket opp ved gitte intervaller innad i applikasjonen, og tokenen blir godkjent hvis den er lik og det er innen en time siden den ble laget. Hvis tokenen ikke blir godkjent så vil den bli slettet lokalt hos brukeren, og de vil deretter bli kastet ut og bedt om å logge inn på nytt igjen.
Hele innloggingsprosessen begynner med at brukeren blir presentert med en innloggingsside, der de blir bedt om å skrive inn sitt NTNU brukernavn og passord. Brukernavnet og passordet blir sendt med som parameter i funksjonen 7.16

```
async function loginUser(creds) {
    const token = await axios.post('http://localhost:8080/login', creds)
    .then((response) => {
        return(response.data.token)
    })
    return(token);
}
```

Kodeliste 7.16: Innloggingsspørring på API

Denne funksjonen tar brukernavnet og passordet, og gjør en spørring på innloggings APIet. Den får returnert en token hvis den blir godkjent, den returnerer så tokenen igjen slik at den kan bli lagret lokalt hos brukeren.
Spørringen i funksjonen 7.16 tar den informasjonen som er blitt sendt med og gjør en spørring på NTNU sitt innloggings API.

```
app.post('/login', (req, res) => {
    (async () => {
        if (await auth(req.body)) {
            res.send({
                token: crypto.randomBytes(64).toString('hex')
              });
        };
    })()
})
```

Kodeliste 7.17: LoginAPI i APIhåndtereren

I 7.17 er den første delen av spørringen på NTNU APIet. Der tar den inn informasjonen som den har fått sendt med og kaller på en asynkron funksjon med den medsendte informasjonen som parameter. Asynkrone funksjoner blir brukt når man skal gjøre løftebaserte funksjoner, i dette tilfellet "await". Det er for å gi beskjed om at programmet ikke skal gå videre før den har fått et svar [36]. Hvis den funksjonen får returnert at brukeren finnes så generer den en token som den sender videre til brukeren.

```
1    let user = await authenticate(options);
2
3    if (user.ntnuMemberOf.find(element => element === "od-it-sikkerhet_soc")) {
4        console.log(user);
5        return true; }
6    else {
7        console.log("Ingen bruker funnet");
8        return false }
```

Kodeliste 7.18: Autoriseringsfunksjon

I funksjonen auth(7.18) så blir det deklarert en variabel som kalles for "options". Denne variabelen inneholder alt som "authenticate" trenger for å gjøre en spørring, den inneholder adressen til NTNU APIet, brukernavnet og passordet til brukeren, og hvor i LDAP hierarkiet brukeren skal bli sjekket opp. "Authenticate" funksjonen som ligger på første linjen i 7.18 er en del av en node modul som heter "ldap-authentication". Denne pakken håndterer enkelt LDAP spørringer slik at vi kun trenger å konfigurere hvor spørringen skal bli gjort og med hvilken informasjon. Den returnerer brukerens informasjon, siden den får sendt med passord så vil den få tilgang til mere informasjon, som f.eks hvilke grupper brukeren er med i. Etter brukeren sin informasjon er returnert så vil programmet gå igjennom gruppene de er med i og sjekke om de er med i gruppen "od-it-sikkerhet_soc". Hvis brukeren er det så vil denne funksjonen returnere true, som er det funksjonen i 7.17 venter på.

Tokenen som blir generert i 7.17 er tokenen som funksjonen i 7.16 returnerer. Denne funksjonen sender tokenen videre til en annen funksjon som lagrer tokenen i "localstorage" hos brukeren. Grunnen til at den blir lagret i localstorage, og ikke sessionstorage, er at den blir lagret helt til applikasjonen sletter den. I motsetning til i sessionstorage der den blir slettet hvis brukeren lukker igjen fanen hvor applikasjonen kjører. Da vil brukeren kunne lukke igjen fanen og gå inn igjen på siden uten å måtte logge inn så lenge tokenen er gyldig. Den funksjonen lagrer også tokenen i databasen. Den gjør dette ved å gjøre et APIkall som er ganske likt som 7.16. I stedenfor å sende med brukerdetaljene så vil den sende med tokenen som parameter på adressen "http://localhost:8080/saveToken/".

```
1    axios.request(options)
2    .then((res) => {
3        console.log(res.status)
4        status = res.status;
5    })
6    .catch((err) => console.log(err))
```

Kodeliste 7.19: Token lagring i databasen

I 7.19 så gjøres en axios forespørsel med "options" som parameter. Denne variabelen definerer adressen som databasespørringen skal bli gjort på, og hvilke data som skal bli lagret. Den dataen som skal bli lagret er tokenen som er sendt med, og den nåværende UNIX-tiden. Den lagrer dette i token tabellen i databasen.

Hver gang grensesnittet oppdateres, eller react-staten blir forandret så vil det bli gjort en sjekk på om tokenen fortsatt er gyldig. Den sjekken fungerer slik at den sender med tokenen som ligger i brukeren sin localstorage vil bli sendte med i en axios spørring ganske likt som i 7.16. Den vil gjøre en spørring på adressen "http://localhost:8080/getToken".

```
1    axios.request(options).then((response) => {
2        if (response.data.find(e => { return ( e.token === parsedToken && time <= e.timestamp +
     parseInt(process.env.TOKEN_DURATION))})) {
3            console.log("VALID TOKEN");
4            res.send(true);
5        }
6        else {
7            console.log("INVALID TOKEN")
8            res.send(false);
9        }
10   }).catch((err) => console.log(err))
```

Kodeliste 7.20: Sjekker om tokenen er gyldig

I 7.20 så gjøres en axios forespørsel med "options" som parameter. Denne variabelen definerer adressen som databasespørringen skal bli gjort på. Den henter alle tokens som ligger lagret i databasen og sjekker om den medsendte tokenen finnes i databasen. Hvis den gjør det så sjekker den også om den fortsatt skal være gyldig. Det gjør det ved å ta den nåværende tiden og sjekker om den lagrede tiden er innen intervallet til levetiden. Levetiden vil være konfigurerbart i .env filen, men normalt så vil den være på en halvtime. Hvis tokenen er gyldig så vil den returnere true og sende det oppover i applikasjonen. Hvis denne funksjonen ikke returnerer true så skal applikasjonen slette tokenen som ligger lagret hos brukeren. Brukeren vil da bli kastet ut av applikasjonen og det vil kreves at de autentiseres på nytt.

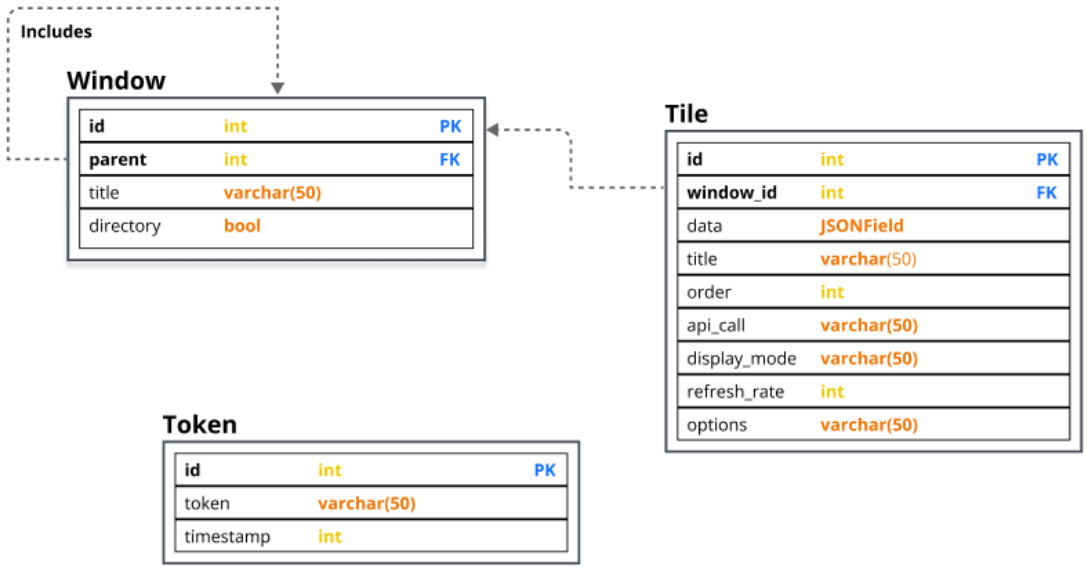
7.4.6 API

API håndtereren

For å håndtere de ulike spørringene mellom frontenden og backenden så lager vi en apihåndteringsfil. Denne apihåndtereren skal kjøre på en egen node server. Apihåndtereren er bygd opp på Express. Express er et node webrammeverk som gjør det mulig å gjøre ulike HTTP spørringer på ulike URLer, man kan også definere hvilken port som frontenden må gjøre spørringer på for å få kontakt med de ulike APIene.[37] For at Express skal kunne hente informasjon fra eksterne APIer så må CORS tas i bruk. CORS er en HTTP-header basert mekanisme som lar en server selv indikere hvilke domener utenom sitt eget, som en nettleser skal få lov til å laste inn informasjon fra.[38] Det er veldig viktig at applikasjonen klarer å håndtere dette, fordi hovedfunksjonaliteten til denne skal være å fremstille informasjon som den henter inn eksternt.
API håndtereren er applikasjonens mellomvare. Den skal håndtere all kommunikasjon mellom frontend og backend, inkludert å hente data fra eksterne APIer.

7.5 Database

Databasen vår lagrer i utgangspunktet kun de ulike vinduene og hvilke tiles/widgets og informasjon de skal inneholde. Det ble også bestemt at tokenhåndtering skal skje i databasen, det vil si at når en token blir generert så blir den lagret i databasen, det er denne tokenen som brukeren sin token blir sjekket opp mot.



Figur 7.4: Databasedesign

7.5.1 Design

I figur 7.4 kan man se sammenhengen mellom de ulike tabellene. Databasen består av de tre tabellene "Window", "Tile" og "Token".

Window

Denne tabellen skal holde oversikt over alle vinduene i applikasjonen. Et vindu i applikasjon kan enten være en mappe eller et vindu. En mappe vil inneholde flere undermapper eller vinduer, og et vindu vil inneholde ulike tiles/widgets. Hvis brukeren vil ha en mappe så må de huke av på directory boolen. Hvis den mappen eller vinduet er barnet til en annen mappe, så vil foreign key parent variabelen bli satt til IDen til den mappen. Hvis man ikke velger forelder så vil den mappen eller vinduet ligge som en root folder. Når en mappe blir slettet så vil alle dens undermapper og vinduer rekursivt bli slettet.

Tile

Denne tabellen holder oversikt over alle tilsene som ligger i applikasjonen. Hver tile kan kun høre til et vindu. Order variabelen setter hvilken rekkefølge den tilen skal bli fremstilt i når man viser frem et vindu. Api_call variabelen bestemmer hvilket API som den tilen skal hente informasjon fra, man kan velge hvilke data som skal bli fremstilt i data variabelen, der vil brukeren få mulighet til å huke av ønsket variabler. Refresh_rate variabelen bestemmer hvor ofte den tilen skal oppdateres. Minsteverdien til denne variabelen vil være ti sekunder, det er fordi applikasjonen ikke skal overbelaste APIet som den henter informasjon fra. Grunnen til at denne variabelen ligger i denne tabellen og ikke i Window-tabellen er fordi at noen

tiles/widgets er mindre kritisk at blir oppdatert en andre, det vil si at det sparer på trafikk på APIet.

Token

Denne tabellen skal lagre tokens som bli generert. Token variabelen er tilfeldig generert string som blir laget i APIhåndtereren. Timestamp variabelen er tiden tokenen ble laget i unixtid. Applikasjonen vil ved intervaller sjekke hvor lenge siden det var tokenen ble laget, hvis den tiden overskrider grensen så vil tokenen bli slettet og brukeren kastet ut av applikasjonen.

7.5.2 Database implementasjon

Som nevnt i systemdesign kapitellet så er databasen vår bygd opp i Django og kjører i PostgreSQL. Django installeres via python og PostgreSQL laster man ned manuelt, når PostgreSQL er lastet ned så blir pgAdmin installert, det er en applikasjon som gjør det enkelt å administrere databasen. Vi brukte aldri pgAdmin til noe annet å tømme tabeller etter testing, all funksjonalitet som vi trenger skal fungere i direkte i python.

Initiell oppsett

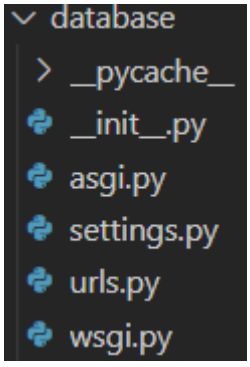
Det initielle oppsettet til databasen gjorde vi i python og Django, etter vi hadde installert det som trengtes av programvare og pythonpakker.
For å lage Django applikasjon må man først kjøre kommandoen:

```
...\> django-admin startproject database
```

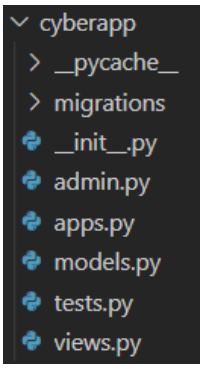
Den genererer automatisk noen filer og instillinger for Django som trengs. Vi valgte å kalle prosjektet for database
Etter at Django er satt opp i applikasjonen så må man kjøre kommandoen:

```
...\> py manage.py startapp cyberapp
```

Den generer automatisk filer som bestemmer hvordan de ulike tabellene skal settes opp og hvordan de fungerer.



(a) Generert av første kommando



(b) Generert av andre kommando

Figur 7.5: Generert filstruktur etter de to kommandoene

Hovedsakelig så er det models.py og views.py fra figur 7.5b som jobbes med, men også urls.py fra figur 7.5a. For å gjøre det initielle oppsettet av Django så fulgte vi det som stod i Django dokumentasjonen[39].

models.py

Denne filen inneholder informasjonen til tabellene i databasen. Det blir definert hvilket tabeller som finnes og de ulike variablene og hvilken datatype de er. Hver tabell i databasen har sin egen model i models.py.

```
1  class Tile(models.Model):
2    window_id     = models.ForeignKey('Window', on_delete=models.CASCADE)
3    data          = models.JSONField(null=True)
4    title         = models.TextField()
5    order         = models.IntegerField()
6    api_call      = models.TextField()
7    display_mode  = models.TextField()
8    refresh_rate  = models.IntegerField(default=5000)
9    options       = models.TextField(blank=True)
10 }
```

Kodeliste 7.21: Tile modellen

I koden 7.21 blir tile modellen definert. De ulike variablene får også ulike instillinger gitt som f.eks delete_models.CASCADE, det vil si at når det vinduet som den har en foreign key til slettes så vil det vinduet også slettes. Alle modeller definert på denne måten vil også ha et standard integer ID-felt, med mindre dette blir definert selv.

views.py

I denne filen så blir det definert "serializers" og "views" til tabellene. Serializers oversetter data som f.eks modeller til Python datatyper som enkelt kan bli lagt inn i f.eks JSON[40].

```
1  class TileSerializer(serializers.ModelSerializer):
2    class Meta:
3      model = Tile
4      fields = ('id', 'window_id', 'data', 'title', 'order', 'api_call', 'display_mode', '
       refresh_rate', 'options')
```

Kodeliste 7.22: Tile serializer

Views tar en forespørsel og returnerer en response. I applikasjonen så er klasse baserte views brukt. Klasse basserte views blir brukt til å strukturere og gjenbruke klassen som man selv har lyst til. [41]

```
1  class TileView(viewsets.ModelViewSet):
2    serializer_class = TileSerializer
3    queryset         = Tile.objects.all()
```

Kodeliste 7.23: Tile view

I vårt tilfelle så blir hver view definert til å ta inn tilsvarende serializer også gjøre om de objektene til et queryset.

urls.py

På figur 7.5a så ligger det en fil som heter urls.py. I denne filen deklareres URLer og definerer hvordan de URLene fungerer[42].

```
1 router = routers.DefaultRouter()
2 router.register(r'windows', views.WindowView)
3 router.register(r'tiles', views.TileView)
4 router.register(r'events', views.EventView)
5 router.register(r'tokens', views.TokenView)
```

Kodeliste 7.24: URL definisjoner

I applikasjonen så brukes Django REST framework pakken sin Router funksjon. Den blir brukt for å enklere definere hvilke URLer som skal lede hvor.

Grunnen til at man må gjøre alt det over er at man skal kunne gjøre spørringer direkte på de URLene som er blitt satt opp i urls.py filen. For eksempel så vil man kunne gjøre spørringer på "http://localhost:8000/api/tiles/" for å hente informasjon fra Tiles tabellen i databasen.

Kapittel 8

Kvalitetssikring

8.1 Kodekvalitet, lesbarhet og dokumentasjon

For stil ble instillingen i VSCode for “format"operasjonen endret til å bruke Google JavaScript Style Guide. Etter å ha skrevet kode ble format operasjonen kjørt slik at koden automatisk fulgte google’s javascript stil. Koden ble også skrevet i henhold til Google JavaScript Style Guide, som for eksempel navnekonvensjoner og andre formatteringskonvensjoner.

For å forhindre sikkerhetsfeil, bugs og for å ha en generell høyere kodekvalitet ble en extension for VSCode brukt kalt “SonarLint". SonarLint gir beskjeder til utvikleren i “real-time” om problemer, ubrukte variabler, ubrukte import’er og forenklinger som kan gjøres i koden. Dersom SonarLint utlyste et problem ble koden endret slik at problemet ble løst. Dokumentasjonen ble gjort i henhold til Doxygen-standarden.

8.2 Sikkerhetsaspekter under utviklingenn

Applikasjonen og koden ble sikkerhetsvurdert underveis. Et eksempel på dette vil være det å håndtere brukerens brukernavn og passord og tokenhåndteringen. Vanligvis så blir variabler laget i React-kode lagt inn i React-states. Det vil si at variablene vil være mulig å finne hvis uautoriserte aktører vet hvor de skal lete. F.eks i en nettleser så kan man laste ned en utvidelse som fremstiller de ulike react komponentene, via den kan man enkelt se alle de ulike variablene som er lagret. På grunn av dette så blir aldri brukernavnet og passordet lagret i en state, det vil bli sendt direkte via en axios-spørringen til APIet. Hvordan denne spørringen fungerer står beskrevet i 7.4.5.
Applikasjonen vil kun være tilgjengelig hvis brukeren har en gyldig token lagret, disse blir automatisk generert og det er ingen måte for en aktør å gjette hva den er. Disse vil bli lagret i databasen også bli lagret lokalt hos brukeren. Siden de bli lagret i databasen og så sendt til brukeren så vil det ikke være noen måte for brukeren å forandre på hvordan tokenen som blir lagret er.

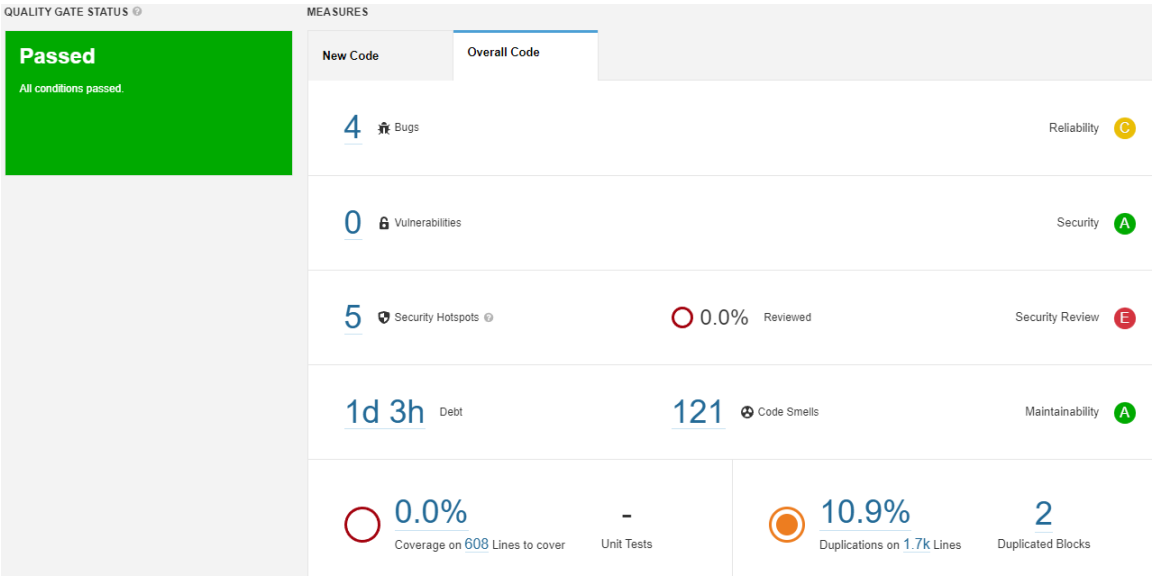
Applikasjonen kjører lokalt så for å sikre applikasjonen mot autorisert tilgang så er det laget en ACL, slik at det kun er mulig for den maskinen som applikasjonen kjører på å få tilgang til APIet.

Applikasjonen er også blitt laget med tanke på OWASP top 10, hva dette er står beskrevet mer i 3.3. Hoveddelene som vi har tatt hensyn til er punkt 1, som omhandler det at brukerene kun kan aksessere områder de er autorisert til. Det er også tatt hensyn til punkt 2 som omhandler eksponering av sensitiv data. Som for eksempler API-nøkler skal være gjemt slik at det ikke er mulig å hente de ut.

For kodekvalitet i tillegg til sikkerhetsfeil, så brukes en programvare som kalles for SonarLint. Denne er en utvidelse som blir lagt til i VSCode, og har i oppgave å detektere sikkerhetsfeil i koden som er skrevet. I motsetning til SonarQube så fungerer SonarLint i sanntid mens koden blir skrevet, SonarQube vil gi en mer helhetlig rapport av alt som er skrevet.

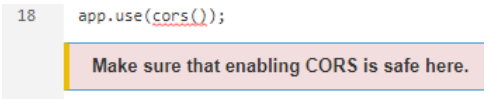
8.3 SonarQube

SonarQube blir brukt for å finne kodekvalitet og sikkerhetsfeil i koden. Programvaren er blitt bruk mot slutten av utviklingsfasen for å se om det er noen forbedringen.



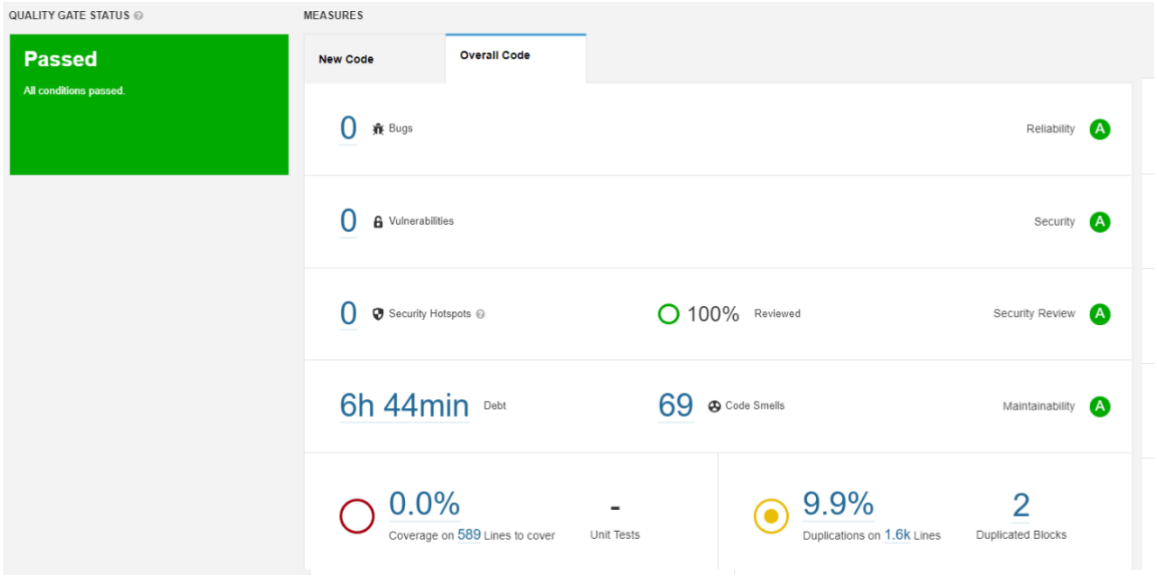
Figur 8.1: Første resultat ved bruk av SonarQube

I 8.1 så ligger resultatene til første bruk av SonarQube. Den viste at generelt sett så var koden bra, men det var noen sikkerhetsfeil og kodefeil, den viste også at det var en del kodeduplisering.



Figur 8.2: Eksempel på sikkerhetssårbarhet i SonarQube

SonarQube viser intuitivt hva som er galt. I figur 8.2 kan man se et eksempel på en sikkerhetssårbarhet i koden. I dette tilfellet så er det bruken av CORS som blir forklart i 7.4.6. Alle "security hotspots" som blir vist i 8.1 er lignende dette og er ikke direkte sårbarheter i applikasjonen, men er mer at den gir beskjed om ting man må passe på.

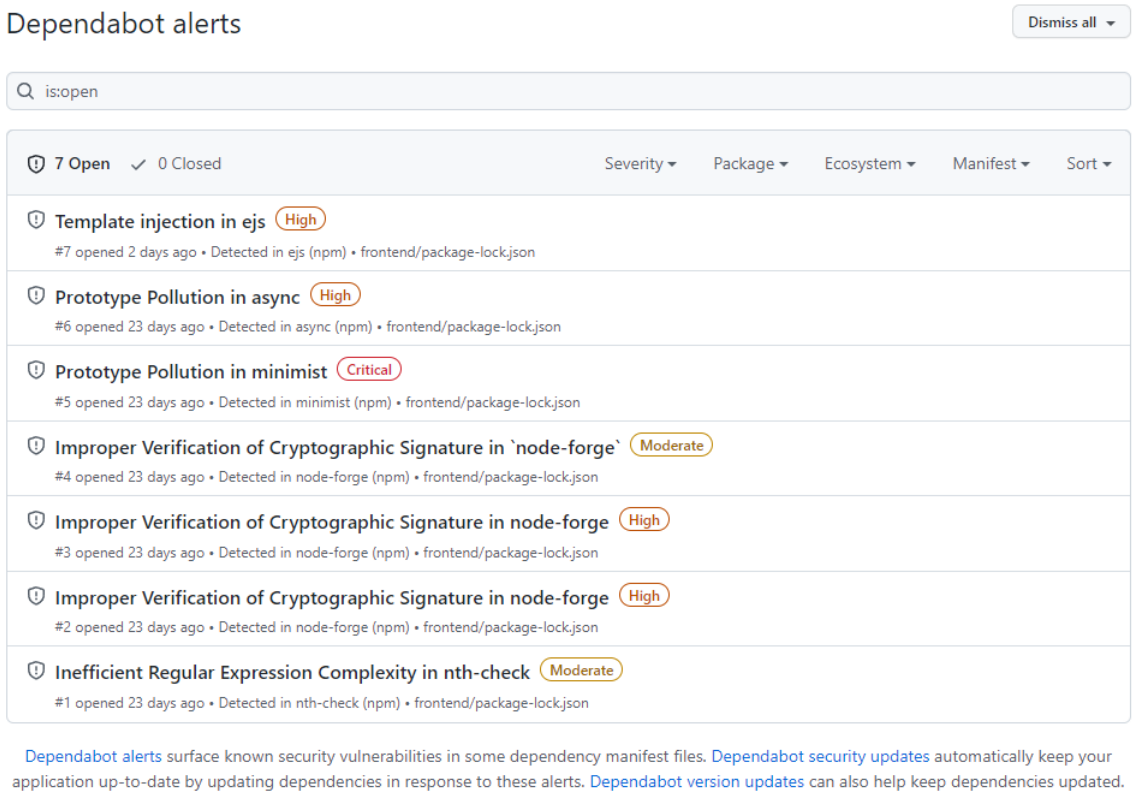


Figur 8.3: Resultat etter fikset feil og gjennomgang av sikkerhetshull

I 8.3 så kan man se resultatet av en gjennomgang med SonarQube etter at alt er blitt gjennomgått. Der er alle bugs rettet opp, og sikkerhetssårbarheter sjekket om de utgjør noe trussel for applikasjonen.

8.3.1 Dependabot

Et av de større punktene innenfor sikkerheten av applikasjonen er alle pakkene som brukes direkte av applikasjonen, men de pakkene har også pakker som de er avhengige av. For å oppdage sikkerhetsmangler og feil i alle pakkene som blir direkte og indirekte brukt av applikasjonen, anvendte vi Github's Dependabot.



Figur 8.4: Oversikt over alle sikkerhetshull som dependabot fant

Dependabot går gjennom alle pakkene og rapporterer alle pakker som har en oppdaget sårbarhet i CVE databasen. Rapporten fra Dependabot gir informasjon om hvilken angrepsvektor sårbarheten blir utnyttet gjennom, kompleksiteten til angrepet, samt hvilke informasjonskomponenter i CIA-triaden som er utsatt. Hver sårbarhet får også en sårbarhetsscore for å si hvor alvorlig sårbarheten er.

Kapittel 9

Konklusjon

9.1 Måloppnåelse

Applikasjonen er ikke et ferdig produkt. Det ble vrient å ferdigstille applikasjonen når vi kun hadde API-tilgangen i to uker. Derfor ble vi bedt om å flytte fokuset fra applikasjonen til rapporten. Gruppen ønsker likevel å måle oppnåelsen av både resultat og effektmål for prosjektet.

Utifra kravspesifikasjonene så er disse kravene oppnådd:

ID	BESKRIVELSE	OPPNÅELSE
F1	Lage vinduer	
F2	Lage tiles	
F3	Endre instillinger i en tile	
F4	Styre applikasjonen gjennom et API.	
F5	Hente data fra APIer i JSON format	
F6	Gjemme sensitiv data	
A1	Administrasjonsgrensesnitt	
A2	Presentasjonsgrensesnitt	
U1	Applikasjonen skal bruke NTNU sine farger og logoer.	
U2	Skal publiseres som åpen kildekode	
U3	Må ta i bruk moderne webteknologier.	
S1	Kun SOC-NTNU brukere skal kunne logge inn i applikasjonen	
S2	Token/Cookie basert system for automatisk utlogging	

Tabell 9.1: ID-listen med oppnåd resultater

9.1.1 Oppsummering av målene

Resultatmålene:

- Applikasjonen skal være dynamisk konfigurerbar
- Applikasjonen skal bli publisert som open source
- Applikasjonen skal bruke NTNU sin merker og fargetema
- Applikasjonen skal bruke moderne webteknologi
- Applikasjonen skal kunne kontrolleres gjennom et API
- Applikasjonen skal kunne lese data fra APIer (JSON)
- Applikasjonen skal følge beste praksis sikkerhetsprinsipper

Effektmålene:

- Gi en oversikt over informasjon, data og alarmer NTNU SOC må agere på til enhver tid
- Forbedre SOC KPIer: MTTD, MTTA og MTTI
- Frigjøre digital arbeidsflate på SOC arbeidsplasser
- Applikasjonen skal kjøres på en bærekraftig måte

9.1.2 Oppnåelse av resultatmål

Resultatmålene ble definert tidlig i prosessen, og de fleste kravene er oppfylt. Gruppen har konsekvent brukt GitHub for åpen kildekode, og brukt NTNU sine retningslinjer for videografisk profil [43]. Applikasjonen er dynamisk konfigurerbar, da brukeren kan sette opp konfigurerbare tiles/widgets og vinduer etter eget ønske.

Django og React er begge to mye brukte teknologier som er anerkjent, som vi brukte i applikasjonen vår. Disse er tidligere beskrevet i kapittel 4.1.

Det har blitt redegjort for Owasp sin topp ti liste med sikkerhetsprinsipper og drøftet om hva gruppen så som mest sentrale sikkerhetprinsipper for denne applikasjonen. Gruppen har vist at det har vært mulig å lese data fra NTNU sitt API via en API håndterer, men som tidligere nevnt så funker ikke dette nå lenger på grunn av problemer med leveransen av API'et.

9.1.3 Oppnåelse av effektmål

Effektmålene som er definert, måler effekten av applikasjonen etter at den er ferdigstilt. Siden applikasjonen ikke er komplett og ikke er klar til å tas i bruk er det ikke mulig å måle effekten av den.

9.2 Diskusjon

Det vil alltid være rom for forbedring under store prosjekter, spesielt i et såpass omfattende prosjekt som ingen av gruppemedlemmene har erfaring med. Gruppen ønsker å redegjøre for og diskutere aspekter ved prosjektet som har forbedringspotensiale og prosessen rundt dette.

9.2.1 Rapport

Både oppdragsgiver og veileder gjorde det veldig tydelig og klart fra tidlig av at det burde prioriteres å skrive rapport under hele prosjektet, og ikke bare rett før fristen. Dermed ble det skrevet et par sider hver uke, og det ble ikke like mye arbeid på slutten med rapporten som fryktet.

9.2.2 Applikasjon

I forhold til kravspesifikasjonen(3) så oppnår applikasjonen de fleste av kravene. Vi har en applikasjon som gir brukeren mulighet til å lage vinduer, lage tiles som fremstiller dataen i vinduene, redigere og forandre disse. Applikasjonen har også et presentasjonsgrensesnitt som fremstiller de ulike tilsene til et vindu. I tilegg så er det laget et login og tokenhåndteringssystem som bruker NTNU sitt innloggingsAPI. Dette fikk vi beskjed om ganske sent i utviklingsprosessen, i utgangspunktet så skulle applikasjonen kun være tilgjengelig fra spesifiserte IP-adresser, men når vi snakket med arbeidsgiver i mars så ville de heller ha et innloggingssytem. På grunn av dette så måtte vi forandre prioriteringen våre da vi synes at det å ha en sikker måte å aksessere applikasjonen på var viktigere. Ved ettertanke så burde vi gått hardere ned på at vi skulle holde oss til de kravene som allerede var spesifisert slik at vi ikke måtte forandre på planen.

Selv om vi har implementert det meste som står beskrevet i kravspesifikasjonen så er det fortsatt noe som ikke er implementert eller ikke helt ferdig. Hovedsakelig så gjelder dette det å kunne styre applikasjonen gjennom API. Dette ble nedprioritert fordi det var viktigere å få til hovedfunksjonaliteten, men også fordi vi skulle få retningslinjer av arbeidsgiver på nøyaktig hva som skulle gjøres med APIet. Da vi var på et statusmøte med arbeidsgiver i midten av februar for å vise frem første utkast av applikasjonen, så diskuterte vi litt hva som forventes angående det. Der nevnte de at noe av APIet kunne bli brukt for var å sende “Events” til applikasjonen. Disse eventene skulle fungere som advarsler som kommer opp på skjermen og gir beskjed om ulike ting som er definert av brukeren. Vi valgte å fokusere på å få hovedfunksjonaliteten først, og da vi kom tilbake til arbeidsgiveren under iterasjon seks om akkurat denne problemstillingen så fikk vi beskjed om at vi skulle gå videre med annet arbeid, inkludert rapportskriving.

Den største mangelen vi har av hovedfunksjonalitet kommer til det å fremstille data i de ulike tilesene. Vi har kode for det og har testet det flere ganger med testdata, men vi har slitt med det på grunn av tilgang til APIene som det skal hentes data fra. Tilgangen til dette APIet kom veldig sent for det var noe arbeidsgiveren kontinuerlig jobbet på. Problemet var at APIet var oppe i veldig kort tidsrom før det gikk ned igjen, og det er per dags dato fortsatt ikke fikset. Denne feilen kommer av at en portal som de bruker for å lage det APIet ble oppdatert og etter oppdateringen ble ikke API-nøkler lenger godtatt. Vi klarer dermed ikke å generere en API-nøkkel som gjør at vi kan aksessere APIet. Vi har heldigvis testdata som gjør at vi kan få til noe funksjonalitet, men den er såpass mangelfull at det kun er formatet som egentlig er brukbart.

Det er flere grunner til at applikasjonen ikke er helt ferdig. Som nevnt er noe av dette API tilgang, men etterhvert så måtte vi prioritere ferdigstillingen av rapporten.

9.2.3 Møter

Møtene med veileder fungerte bra, selv om vi aldri hadde muligheten til å møtes fysisk. Hanno hadde vært veileder for grupper tidligere, og det kom tydelig frem ved at han pushet oss hele veien på å stille krav til oss selv. Spesielt det med rapportskriving var en høy prioritet for veilederen vår. Hanno har mye kunnskap om informasjonssikkerhet, og på oppsett av ulike applikasjoner. Han hadde dermed mulighet til å veilede oss på de fleste områder på prosjektet. Møtene med veileder fungerte som en plass å dele progresjon på rapport og sluttprodukt, og diskutere ulike løsninger på problemstillingene våre.

Møtene med oppdragsgiver ble avholdt etter behov, og vi hadde dermed ingen faste tidspunkter for disse møtene. Gruppen holdt en god dialog med oppdragsgiver utover semesteret

9.2.4 Arbeidsmiljø

Gruppen har aldri vært i nærheten av å jobbe med såpass store og komplekse prosjekter over en lang periode, og det gjorde at vi visste om problemstillingen på forhånd. Likevel var det flere ting som opplevdes som kompliserte under prosjektet, spesielt det med å beregne hvor lang tid gruppen skulle bruke på de ulike iterasjonene var vrient.

Gruppen hadde avtalt arbeidstider på forhånd, og fulgte disse mer eller mindre utover semesteret. Hvis gruppemedlemmene ble forhindret i å jobbe med prosjektet så var personen selv ansvarlig for å jobbe inn timeantallet ifh til avtalt reglement i prosjektavtalen. I enkelte tilfeller har det blitt glemt å loggføre timene sine i systemet vårt, og det kan dermed være noen avvik på timeregistreringen.

En funksjon som gruppen burde implementert er å knytte arbeidstimene til spesifikke deler av prosjektet, som feks koding, skriving av rapport etc.. Da hadde prosjektleder oppnådd større kontroll over hva medlemmene jobbet med i perioder, og gruppen kunne planlagt bedre fordi vi hadde bedre overblikk over hvor lang tid enkelte problemer tok.

I starten av prosjektet så var det fortsatt restriksjoner rundt covid-pandemien og dermed ble det også naturlig at vi jobbet digitalt da det ikke skjedde noe på campus uansett. Etterhvert som samfunnet åpnet opp mer fortsatte gruppen å jobbe hjemmefra, fordi det var bedre tilrettelagt arbeidsmiljø hjemmefra. Tilgang på stasjonsær datamaskin med flere skjermer, bedre internett tilgang og justerbare pulter. Enkelte gruppemedlemmer har uttrykt at gruppen burde hatt en bedre balanse med bruk av fysisk oppmøte på campus for å skape variasjon i arbeidsmiljøet. (I retrospekt så burde nok gruppen benyttet seg mer av fysisk oppmøte, da det ville skapt et helt annet miljø enn å jobbe hjemmefra).

9.3 Videre arbeid

Når det kommer til videre arbeid så er det noen deler som ikke ble helt ferdig, noe som kan forbedres også er det noen deler som har vært utenfor gruppen sin kontroll.

Hovedsakelig så er det arbeid på dataen som skal fremstilles. Som nevnt over så har det vært problemer med APIet som applikasjonen skal hente dataen som skal fremstilles fra. I starten av oppgaven så var ikke APIet oppe og når vi fikk tilgang til det så var det bare snakk om uker før det gikk ned igjen, og måneder senere når denne delen av rapporten skrives så

er det enda ikke oppe. Gruppen har hatt testdata som gir en noenlunde ide om hvilke data som APIet skal gi ut, men det er ikke representativt for all dataen som APIet har mulighet til å sende ut. Det er også noe mer funksjonalitet når det kommer til tilesene i applikasjonen som ikke er helt ferdiggjort. Dette gjelder at det er flere metoder for å fremstille dataen på. Det skal også være mulig å ha tilgang til applikasjonen gjennom et API, dette skal bli brukt for å legge inn advarsler hvis det er noe kritisk som må vises i applikasjonen.

Et annet element som gruppen hadde sett på om vi skulle jobbet videre med prosjektet er å få det til å se finere ut rent estetisk. Det vil si å gå nøyere inn på HTML og CSS'en for applikasjonen, og utbedre utseendet. Under dette så er det også noen tilgjengelighets funksjonaliteter som muligheten for mørkmodus, og lyder som f.eks skal gi beskjed om ulike varsler som kommer inn.

Bibliografi

[1] E. Rossen, «brukergrensesnitt,» 8. feb. 2022. adresse: https://snl.no/brukergrensesnitt (sjekket 05.04.2022).

[2] M. Granevang, «frontend,» 31. jul. 2020. adresse: https://snl.no/frontend (sjekket 28.03.2022).

[3] K. Krogh, «Smidig systemutviklingsmetode i praksis,» 1. jun. 2014. adresse: https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/253817/751707_FULLTEXT01.pdf?sequence=2&isAllowed=y (sjekket 10.05.2022).

[4] «What is REST?,» adresse: https://www.codecademy.com/article/what-is-rest (sjekket 19.05.2022).

[5] I. M. Liseter og T. H. Nätt, «URL,» 23. nov. 2021. adresse: https://snl.no/URL (sjekket 28.04.2022).

[6] «Utviklingsmodeller i IP,» adresse: https://www.ntnu.no/adm/it/ntnu-soc (sjekket 18.05.2022).

[7] «MTBF, MTTR, MTTA, and MTTF,» adresse: https://www.atlassian.com/incident-management/kpis/common-metrics (sjekket 16.05.2022).

[8] «Utviklingsmodeller i IP,» adresse: https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/148732/utviklingsmodeller-i-ip.pdf?sequence=1&isAllowed=y (sjekket 28.04.2022).

[9] C. Ladas, «Agile Scrumban,» adresse: https://www.agilealliance.org/scrumban/ (sjekket 16.05.2022).

[10] N. S. Andrew van der Stock Brian Glas og T. Gigler, «Owasp top ten,» adresse: https://owasp.org/Top10/ (sjekket 02.05.2022).

[11] H. O. Aarstein, «Hva er en domenemodell - egentlig?,» 12. feb. 2013. adresse: https://www.kantega.no/blogg/hva-er-en-domenemodell-egentlig (sjekket 18.05.2022).

[12] W3Techs, «Usage statistics of JavaScript as client-side programming language on websites,» 2022. adresse: https://w3techs.com/technologies/details/cp-javascript (sjekket 09.03.2022).

[13] W3Techs, «Usage statistics of JavaScript libraries for websites,» 2022. adresse: https://w3techs.com/technologies/overview/javascript_library (sjekket 09.03.2022).

[14] C. Dawson, «JavaScript's History and How it Led To ReactJS,» 25. jul. 2014. adresse: https://thenewstack.io/javascripts-history-and-how-it-led-to-reactjs/ (sjekket 28.02.2022).

[15] RisingStack, «The History of React.js on a Timeline,» 11. okt. 2021. adresse: https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/ (sjekket 04.04.2022).

[16] U. Pisuwala, «The benefits of ReactJS and reasons to choose it for your project,» 30. mar. 2022. adresse: https://www.peerbits.com/blog/reasons-to-choose-reactjs-for-your-web-development-project.html (sjekket 23.04.2022).

[17] M. contributors, «Django introduction,» 25. mar. 2022. adresse: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction (sjekket 04.04.2022).

[18] «PostgreSQL About,» 12. mai 2022. adresse: https://www.postgresql.org/about/ (sjekket 18.05.2022).

[19] «facebook/create-react-app,» adresse: https://github.com/facebook/create-react-app (sjekket 11.05.2022).

[20] «About Node.js,» adresse: https://nodejs.org/en/about/ (sjekket 19.05.2022).

[21] «About npm,» adresse: https://www.npmjs.com/about (sjekket 19.05.2022).

[22] «GitHub,» adresse: https://en.wikipedia.org/wiki/Git (sjekket 16.05.2022).

[23] «N-Able sitt Remote Monitoring and Management software,» adresse: https://www.n-able.com/features/rmm-remote-monitoring (sjekket 04.05.2022).

[24] D. Ghimire, «Comparative study on Python web frameworks: Flask and Django,» adresse: https://www.theseus.fi/handle/10024/339796 (sjekket 05.05.2022).

[25] «Chapter 18. Database Roles and Privileges,» adresse: https://www.postgresql.org/docs/8.1/user-manag.html (sjekket 10.02.2022).

[26] «Axios API introduction,» adresse: https://axios-http.com/docs/intro (sjekket 05.04.2022).

[27] «Promis introduction,» adresse: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise (sjekket 05.04.2022).

[28] «Owasp Top Ten.» (), adresse: https://owasp.org/www-project-top-ten/ (sjekket 06.05.2022).

[29] «Ldap-authentication.» (), adresse: https://www.npmjs.com/package/ldap-authentication (sjekket 06.05.2022).

[30] «Hva er en design sprint?» *Annalen der Physik*, adresse: https://www.lope.design/design-sprint-no (sjekket 27.02.2022).

[31] K. Hofstad, «Prototyp,» 2019. adresse: https://snl.no/prototyp (sjekket 21.03.2022).

[32] V. Simensen, «Hva er wireframes,» 12. feb. 2017. adresse: https://thepitch.no/wireframes/ (sjekket 18.05.2022).

[33] J. Nielsen og R. Molich, «Heuristic evaluation of user interfaces,» apr. 1990. adresse: http://concreta.com.uy/wp-content/uploads/nielsenheuristicsCHI.pdf (sjekket 25.04.2022).

[34] R. Walsh, «Nielsen's 10 Heuristics for UI/UX Design,» 16. nov. 2020. adresse: https://bonsaimediagroup.com/blog/nielsens-10-heuristics-for-uiux-design/ (sjekket 27.04.2022).

[35] T. Bray, *The JavaScript Object Notation (JSON) Data Interchange Format*, RFC 8259, des. 2017. DOI: 10.17487/RFC8259. adresse: https://www.rfc-editor.org/info/rfc8259 (sjekket 05.05.2022).

[36] «Async function, JavaScript documentation,» adresse: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function (sjekket 19.04.2022).

[37] «Express introduction,» adresse: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction (sjekket 05.04.2022).

[38] «Cross-Origin Resource Sharing,» adresse: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS (sjekket 05.04.2022).

[39] «Writing your first Django app,» adresse: https://docs.djangoproject.com/en/4.0/intro/tutorial01/ (sjekket 06.04.2022).

[40] «Serializers, Django REST Framework documentation,» adresse: https://www.django-rest-framework.org/api-guide/serializers/ (sjekket 06.04.2022).

[41] «Class-based views, Django documentation,» adresse: https://docs.djangoproject.com/en/4.0/topics/class-based-views/ (sjekket 06.04.2022).

[42] «URL dispatcher, Django documentation,» adresse: https://docs.djangoproject.com/en/4.0/topics/http/urls/ (sjekket 06.04.2022).

[43] «Logo - maler - grafisk profil,» adresse: https://i.ntnu.no/logo-og-maler (sjekket 05.05.2022).

Vedlegg A

Konfidensialitet avtale

NTNU

Norges teknisk-naturvitenskapelige universitet

Fastsatt av prorektor for utdanning 10.12.2020

STANDARDMAL ved avtale om konfidensialitet mellom student og ekstern virksomhet i forbindelse med studentens utførelse av oppgave (master-, bachelor- eller annen oppgave) i samarbeid med ekstern virksomhet, jf. punkt 9 i standardavtale om utføring av oppgave i samarbeid med ekstern virksomhet.

Student ved NTNU: Gustav Isaksen Wallden Fødselsdato: 120796
Student ved NTNU: Herman Andreas Lindskog Fødselsdato: 180598
Student ved NTNU: Peter Behncke Nes Fødselsdato: 250598
Student ved NTNU: David Spataru Fødselsdato: 090796
Ekstern virksomhet: Seksjon for Digital sikkerhet, NTNU IT

1. Studenten skal utføre oppgave i samarbeid med ekstern virksomhet som ledd i sitt studium ved NTNU.

2. Studenten forplikter seg til å bevare taushet om det han/hun får vite om tekniske innretninger og fremgangsmåter samt drifts- og forretningsforhold som det vil være av konkurransemessig betydning å hemmeligholde for den eksterne virksomheten. Det er den eksterne sitt ansvar å sørge for å synliggjøre og tydeliggjøre hvilken informasjon dette omfatter.

3. Studenten er forpliktet til å bevare taushet om dette i 5 år regnet fra sluttdato.

4. Kravet om konfidensialitet gjelder ikke informasjon som:

a) var allment tilgjengelig da den ble mottatt
b) ble mottatt lovlig fra tredjeperson uten avtale om taushetsplikt
c) ble utviklet av studenten uavhengig av mottatt informasjon
d) partene er forpliktet til å gi opplysninger om i samsvar med lov eller forskrift eller etter pålegg fra offentlig myndighet.

Signaturer

Student: Gustav Isaksen Wallden
Dato: 31.01.22

X Gustav I Wallden

Student: Herman Andreas Lindskog
Dato: 31.01.22

X Herman Lindskog

Student: Peter Behncke Nes
Dato: 31.01.22

X Peter B. Nes

Student: David Spataru
Dato: 31.01.22

X

Seksjon for Digital sikkerhet:
Dato: 31.01.2022

NTNU

NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERSITET

Vedlegg B

Prosjektavtale

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 Informasjonssikkerhet og Kommunikasjonsteknologi
Veileder ved NTNU: e-post og tlf.
Seksjon for Digital sikkerhet, NTNU IT Christoffer Vargrass Hallstensen, Faggruppeleder SOC Epost: Christoffer.hallstensen@ntnu.no Tel: 61135145
Student: Gustav Isaksen Wallden Fødselsdato: 120796
Student: Herman Andreas Lindskog Fødselsdato: 180598
Student: Peter Behncke Nes Fødselsdato: 250598
Student: David Spataru Fødselsdato: 090796

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:
Sluttdato:

Oppgavens arbeidstittel er: **Cyberwindows**

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).

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

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

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

	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

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

Begrunnelse: Seksjon for Digital sikkerhet beholder eiendomsretten til resultatet for å sikre at ressurser betalt av offentlige midler skal gå til fellesskapets beste etter NTNUs strategi om «Kunnskap for en bedre verden». Seksjon for Digital sikkerhet forplikter seg til å lisensiere kode og rapport som åpen kildekode/creative commons slik at studentene kan ta med seg arbeidet nedlagt i prosjektet videre etter studier, men samtidig ivaretar at fremtidige studenter og andre kan bygge videre på arbeidet.

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: Dato:	
Veileder ved NTNU: Dato:	NTNU NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERSITET
Seksjon for Digital sikkerhet, NTNU IT: Dato: 31.01.2022	
Student: Gustav Isaksen Wallden Dato: 31.01.22	Gustav I Wallden
Student: Herman Andreas Lindskog Dato: 31.01.22	Herman Lindskog
Student: Peter Behncke Nes Dato: 31.01.22	Peter B. Nes
Student: David Spataru Dato: 31.01.22	

Vedlegg C

Møtereferater

Gruppen hadde flere møter med veileder og oppdragsgiver, vedlagt ligger eksempler på referater fra disse møtene.

Møtereferat

Dato: 18.02.22

Tid: 14:00

Tema: Tilbakemelding med oppdragsgiver

Til stede: Peter, Gustav, David, Herman, Christoffer, Hans Åge, Daniel + noen andre fra SOC'en

Referent: David

Agenda:

1. Vise frem applikasjonen ved en demo
2. Få tilbakemeldinger på applikasjonen
3. Åpen diskusjon om videre prosess

Oppsummering:

Sak 1: Demo

Vi viste frem applikasjonen uten å ha planlagt noe særlig, men gikk grundig gjennom hver enkelt funksjon og besvarte spørsmål som dukket opp underveis.

Spørsmål om hva som er fastsatt av funksjoner som det ikke er meningen å endre på, tydelige på at oppdragsgiver kun gir noen faste retningslinjer og at vi står fritt til å sette opp resten som vi selv ønsker. Vi presiserte at alt av frontend ikke var fastsatt, da vi fokuserte på å sette opp hovedfunksjonaliteten til applikasjonen.

Det meste av utseende er satt opp slik vi ser for oss at vi ønsker at sluttproduktet skal bli seende ut, vi viste de også UX-designet som vi hadde lagd i Figma for å gi de et større innblikk i hvordan vi hadde tenkt.

De virker fornøyde med demoen som vi viste frem, og de liker «mockup'en» med funksjoner som vi hadde satt opp hittil.

Sak 2: Tilbakemeldinger

De ønsket blant annet at en klokke øverst på presentasjonsgrensesnittet, også litt småpirking om at vi kan se på white/black mode om vi fikk tid. De ønsket seg også en globus som de hadde sett på Git for å vise frem lokasjonen til enhetene deres, men at dette muligens var out of scope for vår del.

Videre kom de med tilbakemeldinger på hvordan administrasjonsgrensesnittet kan ha noe interaksjon, mens presentasjonsgrensesnittet bør ha minst mulig interaksjon.

Sak 3: Åpen diskusjon

Vi tok opp når de klarte å levere et API eller sende oss testdata som vi kunne bruke til å teste de ulike funksjonene på applikasjonen. De skulle se på saken og gi oss tilbakemelding på når det begynte å bli klart, men de hadde mye annet på agendaen fremover.

Videre så nevnte de at vi burde sette opp en database med blant annet konfigurasjonsfilen for applikasjonen, og at vi burde bruke PostGressSQL til å sette det opp. De nevnte også det med å sette

opp et intervall for hvor ofte den oppdaterer dataen som kommer inn, et naturlig intervall for dette vil være typ 10 sekunder.

Christoffer fra SOC'en foreslo også et slags pop-up vindu eller notification når det dukket opp en alarm eller lignende, slik at de ble observant på at det var uregelmessig aktivitet på enhetene. Det hadde også vært en fordel med en slags lyd som fattet oppmerksomheten til de som satt ved skjermen, men viktig at den gikk bort etter hvert siden det skal være minst mulig interaksjon.

Vi avsluttet møtet med at vi valgte å holde en god dialog over Teams og bruke det som kommunikasjonskanal slik at de kinne følge med på progresjonen vår.

Møtereferat

Dato: 08.02.22

Tid: 10:00

Tema: Iterasjonsmøte

Til stede: Peter, Gustav, David og Herman

Referent: David

Agenda:

1. Gå gjennom kanban-brettet
2. Diskutere nye funksjoner

Sak 1: Kanban-brett

Går gjennom «issues/kort» fra forrige periode og kjører en kjapp statusoppdatering på hvordan det har gått. Oppdaterer plassering på kortene ut ifra statusen, og det gis mulighet til å vise frem hvordan det ble og koden som er brukt.

Sak 2: Nye funksjoner

Diskuterer hva som bør gjøres og om det er noe som bør prioriteres til neste iterasjon.

Kode:

- Kommentere koden
- Presentasjonsgrensesnitt
- Node server får inn get forespørsler fra frontenden
- Småjusteringer av toolbar
- Kreditere koden som vi har brukt fra andre

Rapport:

- Skrive minst 3 sider til neste iterasjonsmøte
- Sette opp struktur på rapporten
- Lese gjennom andre rapporter for inspirasjon

Annet:

- Sette opp databasen
- Lage relasjons-diagram

Vedlegg D

Forprosjekt

Prosjektplan

Gustav Isaksen Wallden Peter Behncke Nes David Spataru
Herman Andreas Lindskog

January 31, 2022

1 Mål og rammer

1.1 Bakgrunn

For å fullføre vårt bachelor studie skal vi skrive en avsluttende oppgave. Gjennom arbeidet skal vi lære og forbedre måten vi kan jobbe systematisk og målrettet på. Den kommer som en forberedelse på arbeidslivet som venter oss etter fullført skolegang.
NTNU SOC er et operasjonssenter for Seksjonen for Digital Sikkerhet på NTNU som har som hovedoppgave å detektere, analysere og respondere på digitale trusler rettet mot NTNU. Gjennom sitt arbeid har de en rekke applikasjoner som innhenter data eller gir varsler om eller under sikkerhetshendelser.

1.2 Prosjektmål

Ved et fullført prosjekt skal vi ha laget en løsning som aggregerer dataene sendt fra ulike applikasjoner ved å kommunisere med APIer og deretter kunne presentere dataene i en god og oversiktlig måte.

Resultatmål

Resultatet fra et gjennomført prosjekt skal være en løsning som aggregerer data fra en rekke systemer som finnes i NTNUs systemer. Løsningen skal være dynamisk og konfigurerbar slik at dataen skal kunne vises til brukerens behov. Dataen blir hentet inn via APIer. Videre så skal løsningen kunne styres via API, slik at brukerens systemer kan sende sikkerhetseventer og denne dataen blir fremhevet. APIet og applikasjonen skal også ha en sensur funksjonalitet som fjerner sensitiv data fra skjermer.

Effektmål

- Gi en oversikt over informasjon, data og alarmer NTNU SOC må agere på til enhver tid
- Forbedre SOC KPIer: Mean time to Detect (MTTD), Mean Time to Acknowledge (MTTA) og Mean Time to Investigate (MTTI)
- Frigjøre digital arbeidsflate på SOC Arbeidsplasser
- Legge til rette for å etablere et felles digitalt situasjonsbilde

1.3 Rammer

Tidsrammen for oppgaven er fra 11. Januar til 20. Mai.
Teknologiske krav og rammer er som følger:

- Applikasjon kan styres via API
- Applikasjonen kan konsummere data fra APIer i form av JSON
- Applikasjonen må publiseres som åpen kildekode

2 Omfang

2.1 Fagområde

Hovedsaklig så er gruppens arbeidsgiver ansvarlig for å opprettholde digital sikkerhet på NTNU. SOC har noen skjermer som de ønsker å fylle med fagligrelevant informasjon for dem. Målet med oppgaven er å lage en applikasjon som gjør nettopp dette for dem. Hovedfokuset vil for gruppen være å utvikle en dynamisk webapplikasjon som de på SOC enkelt kan konfigurere som de vil. Som utviklere av applikasjonen så vil ikke gruppen ha noe ansvar for å finne/lage informasjonen som skal fremstilles, dette vil hentes gjennom APIer fra data som SOC allerede har. Siden dette er en utviklerfokusert oppgave så vil den dekke ulike programmeringsteknologier:

- Skriptspråk som Python og JavaScript
- Apache webserver
- Django er et web rammeverk som er bygd på Python
- React er et JavaScript bibliotek som blir brukt for å bygge brukergrensenitt

2.2 Avgrensning

Analyse av dataen vil ikke være en del av prosjektet. Funksjonalitet som skal fremheve dataen automatisk via analyse vil ikke være relevant for oppgaven.

Oppgaven skal også være avgrenset til serversiden av applikasjonen. Applikasjonen skal være tilgjengelig på et nettverk via nettlesere, men skal ikke ta stilling til oppsett av klienter. Oppgavens fokus vil være dynamisk framstilling av data og innhenting av den. Forskjellige måter å vise dataen på skal være linje-grafer, søylediagrammer, kakediagrammer eller som tabeller.

2.3 Oppgavebeskrivelse

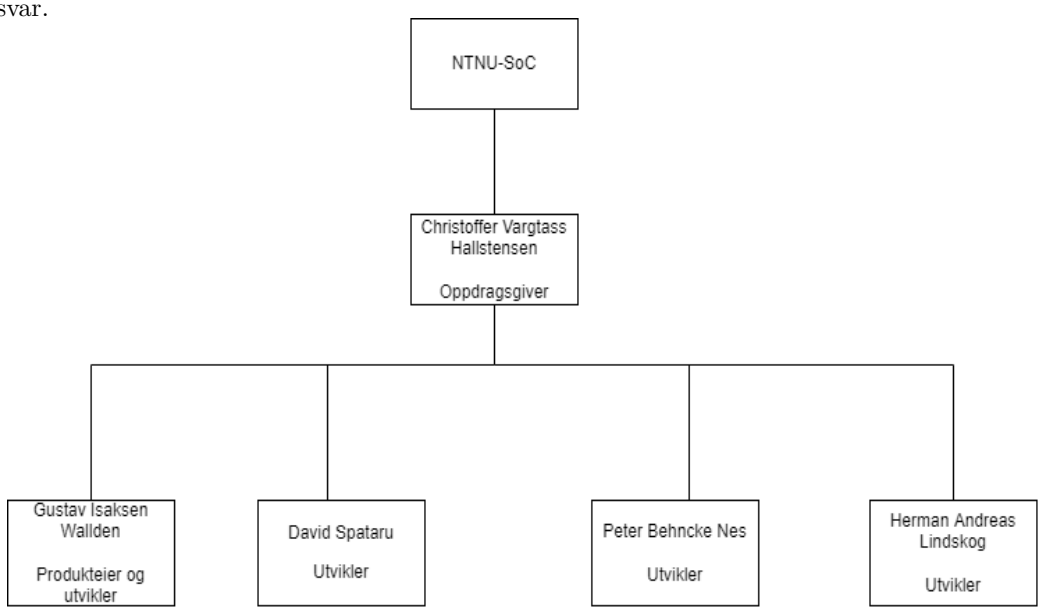
NTNU sin SOC har etterspurt en dynamisk og konfigurerbar webapplikasjon hvor brukeren enkelt skal kunne definere ulike "windows", disse vinduene skal inneholde ulike "tiles/widgets". De skal fremstille ulike data som den henter fra SOC sine allerede eksisterende applikasjoner. Applikasjonen skal også ha en funksjon som gjemmer all sensitiv informasjon fra vinduene som er oppe i tilfelle eksterne skal besøke operasjonsrommet.

3 Prosjektorganisering

Ansvarsforhold

Gruppen består av fire studenter som går på studieprogrammet Digital infrastruktur og cybersikkerhet, på NTNU Gjøvik. Selv om gruppen består av studenter fra samme studiet så er det fortsatt noen ulikheter innad i gruppa. Disse ulikhetene kommer av forskjellige valgemner, fritidsinteresser og generell kompetanse.

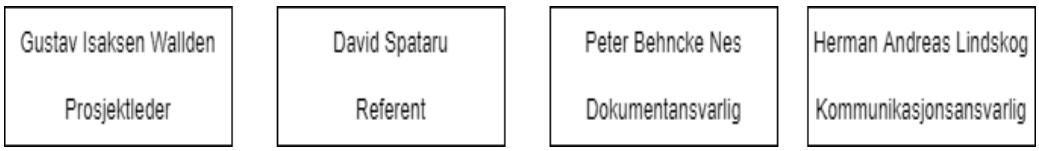
Som del av prosjektplanleggingen så har gruppen tildelt forskjellige roller og forskjellig ansvar.



Ansvarsforhold diagram:

Christoffer Vargtass Hallstensen er kontaktperson for oppdragsgiveren (NTNU-SoC) som vi har fått til vår bacheloroppgave, og står dermed oppført som oppdragsgiver.

Gustav vil fungere som prosjekteier og har ansvaret for prosjektet i sin helhet, samt at han er en del av utviklingsteamet. De øvrige medlemmene er utviklere.



- Prosjektleder
  - Skal holde prosjektet i gang og innenfor de gitte rammene
  - Organisere møter
  - Skal ha avgjørende stemme i situasjoner hvor det trengs
- Kommunikasjonsansvarlig
  - Skal opprettholde kontakt med arbeidsgiver
  - Skal opprettholde kontakt med veileder
  - Skal håndtere all ekstern kontakt til og fra gruppa
- Dokumentansvarlig
  - Skal sørge for organiseringen av ulike dokumenter underveis

  - Skal opprettholde backup av hovedrapport og andre viktige dokumenter

- Referent
  - Notere ned under møter og lage møtereferater
  - Sende ut møteinnkallinger

3.1 Rutiner og grupperegler

Se vedlegg med grupperegler

4 Planlegging, oppfølging og rapportering

4.1 Systemutviklingsmodell

For et større prosjekt er det hensiktsmessig å ha en konkret systemutviklingsmodell. Dette skal avgjøre arbeidsflyten gjennom oppgaveperioden og gi en strukturert arbeidsmetode. Den kommer også til å definere spesifikke arbeidsoppgaver gjennom prosjektet, hvordan de skal fordeles, hvor lenge de skal vare og hvordan samhandling mellom oss i gruppen som utviklere og oppdragsgiver skal foregå.

Det er flere mulige systemutviklingsmodeller som fins, hvor hver enkelt har fordeler og ulemper. Av de ulike systemutviklingsmodellene som vi har vurdert kan de kategoriseres som fossefalls- eller smidige-metoder. Vi har valgt å gå med en smidig utviklingsmodell, ettersom vår vurdering av en fossefalls metodologi behøver mere erfaring innenfor utvikling; det kreves mere planlegging og at man kan forutse problemer nedover i utviklingsperioden. Av de ulike smidige utviklingsmodellene har vi besluttet å benytte oss av Scrumban.

4.2 Scrumban

Scrumban er en kombinasjon av to utviklingsmodeller, scrum og kanban. Ved å anvende den iterasjonsdrevne ideologien bak scrum og oversiktligheten til kanban metoden får vi en dynamisk utviklingsmodell som vi kan lett tilpasse til de problemene som kan oppstå. Vi ønsker ikke å anvende den samme administrative strukturen i Scrum, derav scrum-master og produkteier, men ta i bruk sprint konseptet. Vi ønsker heller ikke å bruke de samme restriksjonene innenfor Kanban hvor man har maksimalt x antall poster under et felt.

Kanban-brettet vil bestå av 4 felt:

- ikke begynt
- påbegynt
- klar for godkjenning
- fullført

Iterasjoner
Vi har valgt å sette iterasjoner på 5? dager. På starten av en iterasjon så vil det bli hatt et møte hvor man legger til poster i ikke-begynt feltet på kanban brettet. Målet deretter er

at alle postene på ikke-begynt feltet på kanban brettet skal bli fullført før iterasjonen er over.

Møte med veileder
Annenhver fredag har er det planlagt møte med veileder fra 10:30 til 11:00. Her skal arbeid gjort i tidligere uker gjennomgås og ulike spørsmål omgående arbeidet kan stilles.

5 Organisering av kvalitetssikring

5.1 Dokumentering

Det er viktig for oss og arbeidsgiver at vi holder god fokus på dokumentskriving og rutiner igjennom hele prosjektet. Dette er hovedsaklig fordi prosjektet skal publiseres som åpen kilde og bli utviklet videre. Derfor er viktig å følge riktige standarder for utviklingsverktøyene og de ulike programmeringsspråkene som vi skal ta i bruk igjennom prosjektet. Dokumentasjonen skal også følge utviklingen tett. Alt dette er for å gjøre det enklere å holde de i gruppen oppdatert, samtidig som det vil gjøre det enklere for mennesker som ikke har den samme tekniske kunnskapen som de i gruppa.

5.2 Lagring av dokumenter

- Rapporten skriven i Latex med hjelp av overleaf.
- Bruker github og vil legge all prosjektkode i repositories.
- For timelister så brukes clockify.com, en nettside hvor man enkelt kan legge til hvor mange timer man har jobbet og hva man har jobbet med.
- Diagrammer, møtereferater og kontrakter lagres i en felles teams-kanal hvor arbeidsgiver og veileder enkelt har tilgang

5.3 Testing

For å forsikre at utviklingen går i riktig retning og at sluttproduktet er av høy kvalitet vil testing være en integral del gjennom hele prosessen. Når nye moduler eller deler blir lagt til i applikasjonen er det viktig at den kan kommunisere med de andre delene, og at kvaliteten er god. Det vil bli brukt 4 kjente metoder for testing av applikasjonen:

- Akseptanse - Forsikrer at løsningen oppfyller oppdragsgivers krav basert på oppdragsgivers tilbakemeldinger
- Bruker - Brukere benytter seg av løsningen og gir tilbakemeldinger basert på sin bruk
- Integrasjon - Tester at de ulike delene av løsningen fungerer sammen.
- Enhet - Tester en spesifikk del av løsningen for feil/mangler og at den fungerer som den skal

For å spare tid gjennom prosjektet så er det også hensiktsmessig å lage automatiske tester der det er mulig, spesielt da på integrasjon- og enhetstestingen.

5.4 Kodekvalitet

I denne oppgaven vil vi benytte oss av en variert kombinasjon av rammeverk og programmeringsspråk. Dette er spesifisert tidligere i prosjektplanen, men i all hovedsak så vil programmeringsspråkene dreie seg om Python og JavaScript. Siden vi som gruppe har ulik grad av erfaring og kompetanse med koding, har vi bestemt oss for å benytte oss av programvare som automatisk og kontinuerlig vil analysere kodekvaliteten vår.

Programvaren heter SonarQube og det er et verktøy for å blant annet oppdage feil og sårbarheter i koden, men også et hjelpemiddel for å kunne skrive effektiv og ryddig kode. Det har støtte for 29 programmeringsspråk da inkludert Python, JavaScript, HTML og CSS som blir mest relevant for vår del. For sikkerhetsaspektet så bruker SonarQube blant annet OWASP sin topp 10 som en sjekkliste som den går gjennom ved analyse av kode.

Vi har ingen erfaring med programvaren fra før, men vi er sikre på at det vil være et nyttig verktøy for oss å implementere når utviklingen er godt i gang.

5.5 Evaluering og håndtering av sikkerhetssårbarheter

For evaluering og håndtering av sikkerhetssårbarheter så har vi valgt å bruke Common Criteria sitt ALC_FLR rammeverk. Det er et rammeverk som bygger på "Common Methodology for Information Technology Security Evaluation" (CEM) og forklarer metoder for hvordan man skal implementere de ulike kvalitetssikringene som er beskrevet der, og hvordan man skal håndterene de ulike sikkerhetssårbarhetene. Den beskriver hele prosessen fra en feil blir funnet til den blir rettet opp.

5.6 Risikoanalyse på prosjektnivå

Risikomatrisen (tabell 1) brukes til å definere de ulike gradene av risiko som en kombinasjon av indeksene til sannsynlighet og konsekvens. Denne tilnærmingen av risikograd brukes videre til å kategorisere (med fargekode) de ulike risikoene som har blitt identifisert i tabell 2. I tabell 3 blir risikoene sortert etter risikograd og gitt beskrivende tiltak.

		Konsekvens				
		Minimal (1)	Liten (2)	Moderat (3)	Betydelig (4)	Alvorlig (5)
Sannsynlighet	Svært usannsynlig (1)	1	2	3	4	5
	Usannsynlig (2)	2	4	6	8	10
	Mulig (3)	3	6	9	12	15
	Sannsynlig (4)	4	8	12	16	20
	Svært sannsynlig (5)	5	10	15	20	25

Tabell 1: Risikomatrise

Nr.	Risiko	Konsekvenser	S	K	Nivå
1	Manglende kunnskap/overvurderer egen kompetanse	Tidsrammer og oppgavens krav blir ikke opprettholdt	3	4	12
2	Tap av arbeid/data	Tidligere arbeid må reproduseres	1	5	5
3	Sikkerhetssårbarheter i kode	Oppfyller ikke oppgavens krav om sikkerhetsfunksjonalitet	3	4	12
4	Feiltolking av oppgavens innhold og krav	Sluttprodukt møter ikke spesifiserte krav	2	5	10
5	Uenigheter innad i gruppen	Reduserer effektivitet og øker tidsforbruk	2	2	4
6	Alvorlig sykdom/skade	Mister betraktelig arbeidskraft	2	5	10
7	Mild sykdom	Arbeidsmengden til andre blir overbelastet	4	2	8
8	Dårlig modul design/implementasjon	Vanskeligere å oppfylle produktkrav	2	4	8

Tabell 2: Risiko

Nr.	Tiltak	Nivå før	Nivå etter
1	Sørge for god og systematisk arbeidsmetode, hyppige statusmøter og tidlig utforskning av relevant litteratur	12	6
3	Anvende sikkerhetsprinsipper og ha aktiv kjennskap til "OWASP Top 10", samt ha en sikkerhetsgjennomgang med testing av sikkerhetsfunksjonalitet	12	6
4	Holde aktiv dialog med oppdragsgiver og få regelmessige godkjenninger (akseptansetesting)	10	6
6	Sørge for at alle gruppemedlemmer er godt kjent med arbeidet til hverandre	10	8
8	Systematisk integrasjonstesting samt bruke god tid på design av et robust domene-modell	8	6
7	Gruppemedlemmer jobber inn tapt arbeidstid per gruppeavtale	8	4
2	Systematisk lokal lagring av GitHub, Overleaf og separate dokumenter	5	3
5	Demokratisk avstemming blant gruppen per gruppeavtale	4	2

Tabell 3: Tiltak

- **1. Manglende kunnskap/overvurderer egen kompetanse**
  Denne risikoen har blitt vurdert til å være av svært betydelig grad. Det er ganske sannsynlig at vi som gruppe vil komme til et punkt der manglende kunnskap vil koste oss mye tid. Beste tiltaket for å kunne forhindre dette i så stor grad som mulig er at alle er nøye med å innhente og lese relevant litteratur samt innføring av en jevn og systematisk arbeidsmetodikk gjennom hele oppgaven. Gruppeavtalen mellom gruppemedlemmene kommer inn i spill her ettersom den er rimelig beskrivende med hvor mye arbeid som er forventet fra hvert enkelt medlem i løpet av en uke.

- **2. Tap av arbeid/data**
  Det er alltid en viss sannsynlighet for å miste arbeid som man har gjort underveis. Hvor stor den sannsynligheten er, er veldig avhengig av hvor oppmerksom og metodisk man er ved behandling og lagring av ens arbeid og data. I tilfeller der man jobber i grupper, burde en være enda mer påpasselig da man kan risikere å miste ikke bare eget arbeid, men også andre sitt.
  Her har vi implementert tiltak som sørger for at hvis vi skulle vært uheldige og mistet noe arbeid, så ville det vært begrenset til maksimalt et par arbeidsdager verdt av arbeid/data. Dette har vi gjort ved å inkludere og tildele rollen dokumentansvarlig, som vil være ansvarlig for å ta lokal backup av GitHub og andre dokumenter annenhver dag. Resterende medlemmer må også ta jevnlige lokale backup.

- **3. Sikkerhetssårbarheter i kode**
  Dette er en risiko som berører et svært viktig krav i oppgavebeskrivelsen. Kravet det henvises til lyder som følger "Applikasjonen har innebygget sikkerhetsfunksjonalitet etter beste praksis". Særlig med vekt på "etter beste praksis" vil et tiltak være å anvende OWASP sin topp 10 sikkerhetsrisikoer for webapplikasjoner.
  Det bør gi oss en god pekepinn på hva vi skal være obs på og samtidig fungere som en sjekkliste som man systematisk kan gå gjennom. Utenom det vil god dialog og jevne statusoppdateringer med både veileder og oppdragsgiver sørge for at sikkerhetshull ikke blir oversett.

- **4. Feiltolkning av oppgavens innhold og krav**
  Oppdragsgiver har inkludert en rekke krav til oppgaven som vi vil forsøke å oppfylle etter beste evne. I tillegg til de som allerede er spesifisert i oppgavebeskrivelsen, regner vi med at det vil komme flere krav og innsnevringer lengre ut i oppgavens livsløp. Her kan det komme missforståelser og feiltolkninger underveis som gjør at oppdragsgiver hadde andre/flere forutsetninger til oppgaven enn det vi som gruppe fortolket. For å unngå en slik situasjon(er) i størst mulig grad vil det være viktig å holde en tett og aktiv dialog med oppdragsgiver underveis. Vi har en felles Microsoft Teams kanal med både oppdragsgiver og veileder for å kunne opprettholde kontinuerlig dialog om vår progresjon og samtidig motta tilbakemeldinger og godkjenninger på utviklingen.

- **5. Uenigheter innad i gruppen**
  I løpet av prosjektet så er det middels sannsynlig at gruppemedlemene blir uenige om en avgjørelse. Dette har derimot lave konsekvenser, fordi alle gruppemedlemene er inneforstått med at de ikke alltid kommer til å være enig. Hvis det blir såpass store uenigheter så er gruppemedlemene blitt enige om at prosjektleder skal ha en avgjørende stemme i slike situasjoner.

- **6. Alvorlig sykdom/skade**
  Dette vil være tilfellet der et gruppemedlem skulle bli såpass alvorlig syk eller skadet at vedkommende ikke vil kunne bidra med arbeid til prosjektet over en betraktelig tidsperiode. Et viktig tiltak vi innfører her er å sørge for at alle medlemmene er godt kjent med arbeidet til hverandre og til enhver tid er forberedt på å måtte ta over andres arbeidsoppgaver dersom det skulle vært nødvendig. Vi vil dermed også sørge for at ingen enkeltindivid får for stort ansvar for en del av oppgaven.

- **7. Mild sykdom**

At noen av gruppemedlemene får mild sykdom anser vi å være sannsynlig, men med liten betydning. I de fleste tilfeller så kan man fortsatt jobbe og være på møter hvis man er syk. Hvis en blir så syk at de ikke klarer å jobbe så skal andre i gruppen være foreberedt på å ta over det arbeidet som det gruppemedlemmet skulle ha gjort. Her har vi også et punkt i gruppeavtalen som sier at tapt arbeidstid må jobbes inn innen én uke har passert fra tapt arbeidstid.

- **8. Dårlig modul design/implementasjon**
  Dårlig design av moduler og videre implementasjon kan oppstå dersom vi forhaster oss i starten ved å forsøke å være i god tid med oppgavens målsetninger og krav. Dårlig design i begynnelsen vil gjøre det vanskeligere med arbeid senere i oppgaven. Et godt tiltak her vil være å gjennomføre systematisk integrasjonstesting for å teste at implementasjon av moduler fungerer som forventet. Vi vil også ta oss god tid med design av et robust domene-modell og sørge for at det er nøye gjennomtenkt.

6 Plan for gjennomføring

6.1 Tidsplan og milepæler

Gruppen har laget et estimat for hvordan tidsplanen bør se ut. Dette er fremstilt i et Gantt-diagram med følgende milepæler:

19.05.2022 **Rapportskriving** - Dette vil være et underliggende punkt i alle fasene av prosjektet. Rapporten er det viktigste med prosjektet og vi vil dermed jobbe med denne tett opp til alle punktene som er nevnt under.

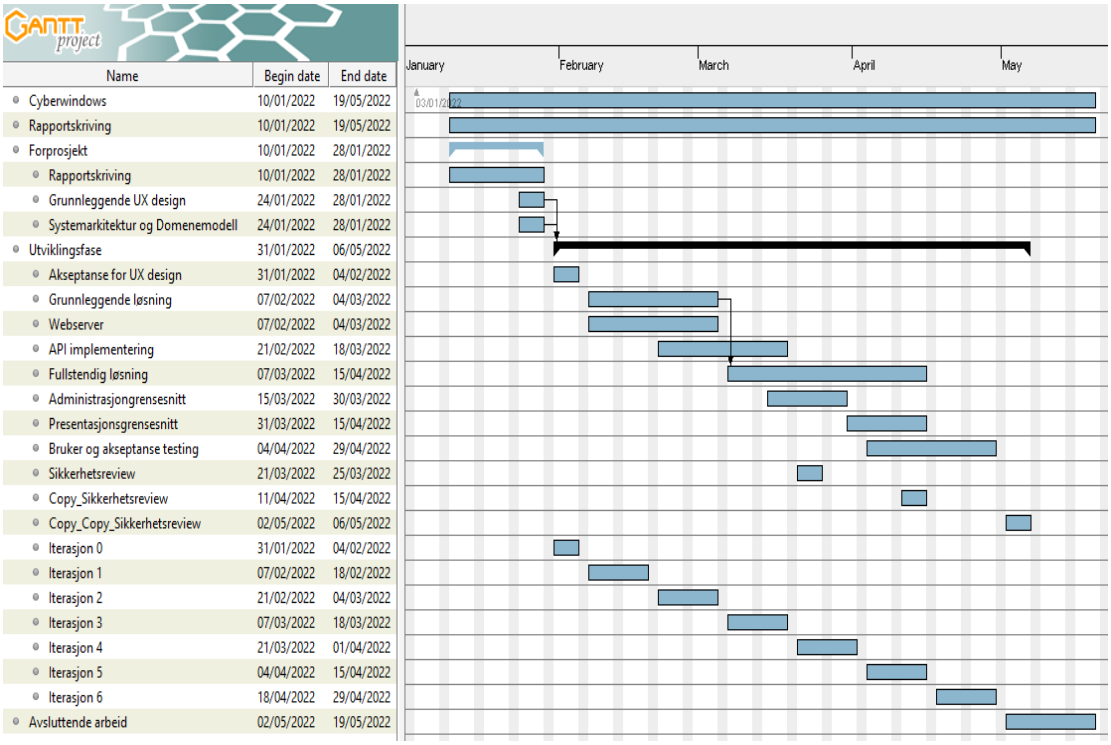
31.01.2022 **Forprosjekt** - Planlegge de ulike delene av prosjektet. Lage planer og bestemme oss for modeller osv. som skal bli tatt i bruk.

- 31.01.2022 Domene modell og valg av system arkitektur
- 04.02.2022 Grunnleggende UX design - Få laget noen versjoner av ulike UX designs som kan vises til arbeidsgiver.

30.04.2022 **Utviklingsfase** - Utvikle applikasjonen, de ulike iterasjonen representerer steg i scrumban modellen.

- 04.02.2022 Akseptanseteste UX - Teste UX, se om den oppfyller de ulike kravene
- 04.03.2022 Prototype - Løsning med noe fungerende funksjonalitet innenfor webserver, API og brukergrensesnitt
- 18.03.2022 API implementering - Implementere løsninger for både interne og eksterne APIer
- 15.04.2022 Fullstendig løsning - En løsning som skal inneholde all funksjonalitet
- 30.03.2022 Administrasjonsgrensesnitt -
- 15.04.2022 Presentasjonsgrensenitt
- 29.04.2022 Bruker og akseptanse testing - Få testet alle de ulike delene av både UX og systemarkitekturen

30.05.2022 **Avsluttende arbeid** - På dette punktet skal all utvikling være over og det skal kun utføres arbeid på rapporten. I denne perioden vil det også være flere gjennomganger av både rapport og produkt.



Name	Begin date	End date
Cyberwindows	10/01/2022	19/05/2022
Rapportskriving	10/01/2022	19/05/2022
Forprosjekt	10/01/2022	28/01/2022
Rapportskriving	10/01/2022	28/01/2022
Grunnleggende UX design	24/01/2022	28/01/2022
Systemarkitektur og Domenemodell	24/01/2022	28/01/2022
Utviklingsfase	31/01/2022	06/05/2022
Akseptanse for UX design	31/01/2022	04/02/2022
Grunnleggende løsning	07/02/2022	04/03/2022
Webserver	07/02/2022	04/03/2022
API implementering	21/02/2022	18/03/2022
Fullstendig løsning	07/03/2022	15/04/2022
Administrasjongrensesnitt	15/03/2022	30/03/2022
Presentasjonsgrensesnitt	31/03/2022	15/04/2022
Bruker og akseptanse testing	04/04/2022	29/04/2022
Sikkerhetsreview	21/03/2022	25/03/2022
Copy_Sikkerhetsreview	11/04/2022	15/04/2022
Copy_Copy_Sikkerhetsreview	02/05/2022	06/05/2022
Iterasjon 0	31/01/2022	04/02/2022
Iterasjon 1	07/02/2022	18/02/2022
Iterasjon 2	21/02/2022	04/03/2022
Iterasjon 3	07/03/2022	18/03/2022
Iterasjon 4	21/03/2022	01/04/2022
Iterasjon 5	04/04/2022	15/04/2022
Iterasjon 6	18/04/2022	29/04/2022
Avsluttende arbeid	02/05/2022	19/05/2022

7 Vedlegg

7.1 Grupperegler

Grupperegler

Medlemmer: Gustav, Peter, David, Herman

Mål: Jobbe effektivt, systematisk og oppnå en god karakter.

Revisjon: 1.2

Ang. Arbeid:

- Mål og planer må bli avklart med de andre i gruppen
- Vi jobber utifra avtalte planer
- Vi fordeler oppgaver og arbeidsmengde likt så langt det lar seg gjøre
- Alle skal ha mulighet til å bidra
- Alle skal jobbe og være tilgjengelig fra 9:00 til 15:00 (mandag-fredag) med mindre det er spesielle omstendigheter.
    - Hvis den enkelte ikke har mulighet til å jobbe under den oppsatte arbeidstiden eller være tilgjengelig så skal de andre i gruppen varsles og hvorfor hvis det ikke er av sensitiv natur
    - Tapt arbeidstid må jobbes inn av personen som har fravær innen en uke i etterkant
- Alle skal føre timelister i clockify gjennom hele prosjektet
- Større valg gjøres i plenum, hvis det ikke er flertall så har prosjektleder en avgjørende stemme

Individuell etos:

- Jeg er ærlig med de andre i gruppa, men spesielt med meg selv
- Jeg skal være punktlig og forberedt til oppmøte ved avtalte tidspunkter
- Jeg tar inititativ til arbeid som må gjøres
- Jeg fullfører alle mine arbeidsoppgaver innen tidsfrister
- Jeg setter bachelor oppgaven som en av de viktigste tingene i mitt liv
- Jeg lar andre hjelpe og bidra
- Jeg viser respekt og ønsker å positivt bidra til å hjelpe de andre i gruppa
- Jeg respekterer avgjørelser
- Hvis jeg er demotivert eller blir frustrert skal jeg ikke la dette gå utover de andre i gruppa
- Hvis jeg merker noen er demotivert eller frustrert skal jeg prøve å oppmuntre de

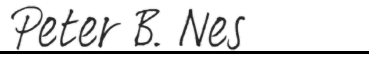
Konsekvenser ved brudd av gruppereglene

1. Samtale innad i gruppen
2. Ved gjentatte brudd holdes en avstemning og det gis skriftlig advarsel med kopi til veileder ved flertall.
3. Hvis brudd fortsetter etter skriftlig advarsel så holdes avstemning om personen må bli ekskludert fra gruppa.

Signaturer

X 

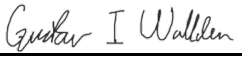
Herman Andreas Lindskog

X 

Peter Behncke Nes

X 

David Spataru

X 

Gustav Isaksen Wallden

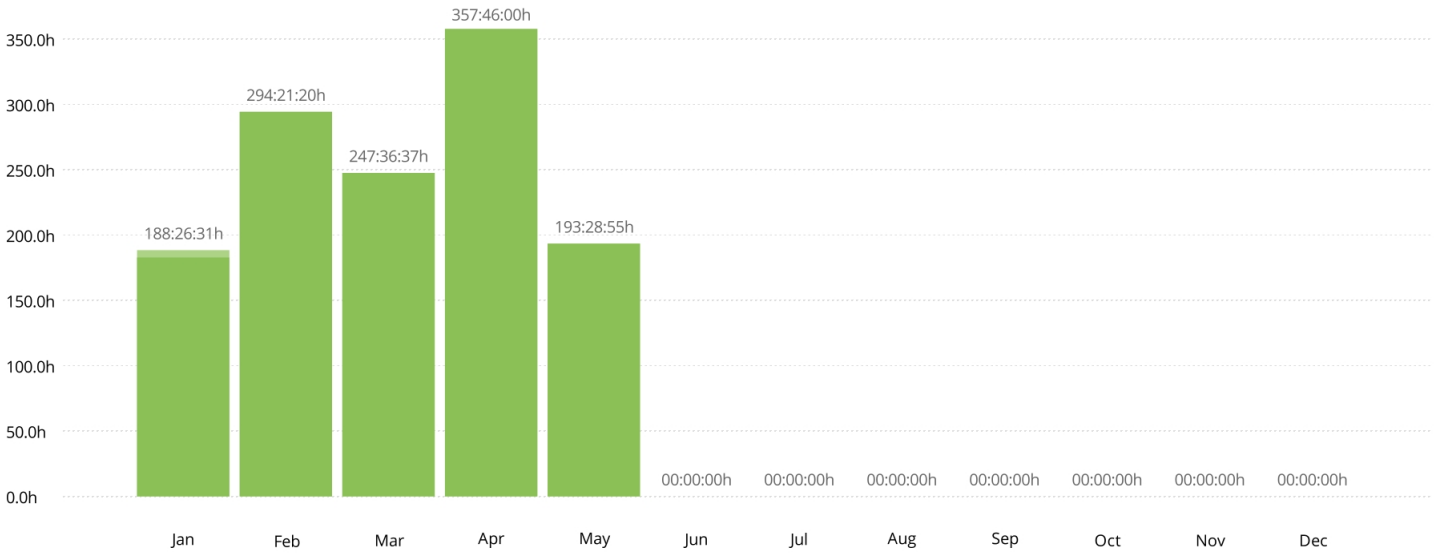
Vedlegg E

Timelister

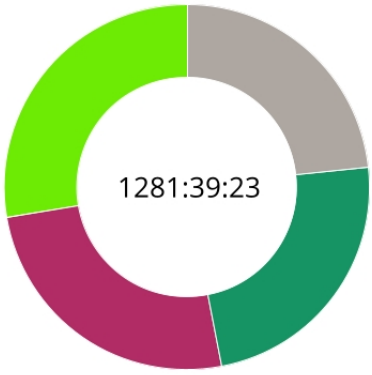
Summary report

01/01/2022 - 31/12/2022

Total: 1281:39:23 Billable: 1276:04:23 Amount: 0.00 USD

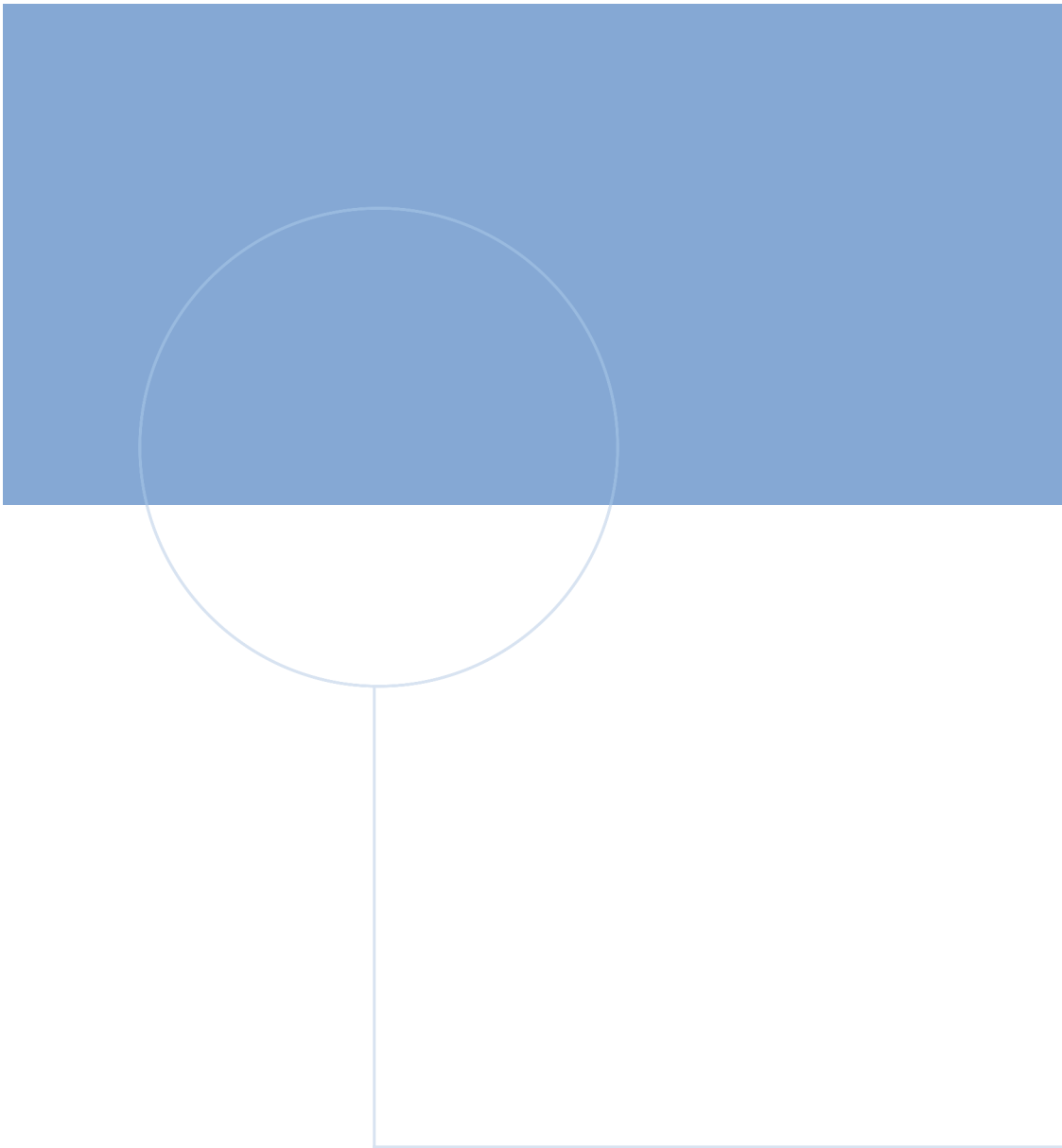


User



Gustaviw	355:10:03	27.71%
Peter Nes	324:17:00	25.30%
Herman Lindskog	304:27:43	23.76%
David Spataru	297:44:37	23.23%

User	Duration	Amount
Gustaviw	355:10:03	0.00 USD
Peter Nes	324:17:00	0.00 USD
Herman Lindskog	304:27:43	0.00 USD
David Spataru	297:44:37	0.00 USD



NTNU

Kunnskap for en bedre verden