Bacheloroppgave

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

Sara Hjelle
Stian Ådnanes
Ådne Eide Stavseng

Utvikling av en webapplikasjon for planlegging og gjennomføring av fagaktiviteter ved bruk av React, TypeScript og ASP.NET Core

Bacheloroppgave i Dataingeniør
Veileder: Ole Christian Eidheim
Mai 2020

NTNU
Kunnskap for en bedre verden

Sara Hjelle
Stian Ådnanes
Ådne Eide Stavseng

Utvikling av en webapplikasjon for planlegging og gjennomføring av fagaktiviteter ved bruk av React, TypeScript og ASP.NET Core

Bacheloroppgave i Dataingeniør
Veileder: Ole Christian Eidheim
Mai 2020

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

NTNU

Kunnskap for en bedre verden

Forord

Denne bacheloroppgaven er gjennomført i samarbeid med Norconsult Informasjonsystemer, som en avsluttende oppgave for dataingeniørstudiet ved Norges teknisk-naturvitenskapelige universitet, våren 2020.

Det å utvikle et system, som skulle tas i bruk internt i en bedrift, har vært en spennende utfordring og en lærerik prosess. Vi fikk bryne oss på nye teknologier og erfare hvordan det er å jobbe på en arbeidsplass, sammen med de ansatte.

Underveis i prosjektperioden bød den pågående COVID-19-pandemien på uforventede problemer, som stilte krav til omorganisering av prosjektet. En brå overgang til hjemmekontor førte til nye måter å kommunisere på, både innad i gruppen, med veiledere og andre kontaktpersoner.

Vi vil takke Andreas Ravnestad, Elias Sundby Aukan og Kristian Winther fra Norconsult Informasjonssystemer for teknisk veiledning, tett oppfølging og en spennende oppgave. Vi ønsker også å takke øvrige ansatte hos Norconsult Informasjonssystemer, for hjelp og tilbakemeldinger på systemet. Til slutt vil vi rette en stor takk til de faglige veilederne Helge Hafting og Ole Christian Eidheim fra Norges teknisk-naturvitenskapelige universitet, for oppfølging og rådgivning gjennom prosjektperioden.

Sara Hjelle | Stian Ådnanes | Ådne Eide Stavseng

Oppgavetekst

Oppgaven går ut på å utvikle en webapplikasjon for å håndtere planlegging og gjennomføring av fagaktiviteter, for Norconsult Informasjonssystemer. Systemet skal bygge på funksjonalitet i deres eksisterende system, men inneholde utvidet funksjonalitet for kostnadskontroll og langtidsplanlegging.

Sammendrag

Norconsult Informasjonssystemer har sterkt fokus på å utvikle de ansattes fagkunnskap, og har i dag en etablert prosess for planlegging og gjennomføring av fagaktiviteter. For å holde oversikt over prosessen, har de tidligere brukt et system med begrenset funksjonalitet, som de ønsket å erstatte. I den forbindelse har vi utviklet et nytt system, med utvidet funksjonalitet for kostnadskontroll og langtidsplanlegging. Systemet vil hjelpe bedriften å holde oversikt over de ansattes fagaktiviteter. Utviklingen av systemet har vært sterkt påvirket av smidig utviklingsmetodikk og kontinuerlig testing på sluttbrukerne.

Gjennom prosjektet har vi sett på utfordringene i et webutviklingsprosjekt med flere utviklere, og fokusert på hvordan statisk typesjekking og testing kan minske utfordringene. For å undersøke fordelene ved statisk typesjekking, ble kodebasen konvertert til TypeScript underveis i prosessen. På denne måten fikk vi sammenlignet utviklingsprosessen med og uten statisk typesjekking.

Resultatene våre viser at innføring av statisk typesjekking kan tvinge frem en ryddigere og mer oversiktlig kode. Ved bruk av testing kan en sikre at koden er feilfri før den blir delt med de andre utviklerne. I tillegg sikrer en at endringer ikke påvirker andre deler av systemet.

Innhold

Figurer

Tabeller

Akronymer og forkortelser

NTNU: Norges teknisk-naturvitenskapelige universitet
NoIS: Norconsult Informasjonssystemer
REST: Representational State Transfer
ORM: Object-Relational Mapping
DTO: Data Transfer Objects
SOC: Separation of concerns **SQL:** Structured Query Language
Azure AD: Azure Active
EFC: Entity Framework Core
LINQ: Language Integrated Query
DOM: Document Object Model
RTL: React Testing Library
IDE: Integrated Development Environment
CSV: Comma-separated values
HTTPS: Hypertext Transfer Protocol Secure

Kapittel 1
Introduksjon og relevans

1.1 Bakgrunn for oppgaven

Norconsult Informasjonssystemer (NoIS) har et sterkt fokus på å utvikle de ansattes fagkunnskaper, og har en etablert prosess for planlegging og gjennomføring av fagaktiviteter. Tidligere benyttet NoIS en selvutviklet applikasjon for å holde oversikt over fagaktivitetene de gjennomfører nå og fremover i tid, samt en historikk over tidligere fagaktiviteter. Alle ansatte kan bruke denne applikasjonen, og alle kan se hverandres fagaktiviteter. På denne måten får de synliggjort arbeidet som foregår, og får en god visualisering av fremdriften i arbeidet.

Det tidligere systemet ble benyttet for å planlegge eventuelle utgifter tilknyttet fagaktiviteter, og var derfor et verktøy for budsjettplanlegging. I tillegg var den med på å gi et innblikk i hvilke sertifiseringer og kurs som var populære. Systemet hadde begrenset funksjonalitet, og behøvde blant annet utvidet funksjonalitet for kostnadskontroll og langtidsplanlegging. I forbindelse med dette valgte de å pensjonere det gamle systemet. De ønsket derfor et helt nytt system for gjennomføring og planlegging av fagaktiviteter.

1.2 Problemstilling

Under utviklingen av en webapplikasjon med flere utviklere, kan det oppstå flere utfordringer. Det kan for eksempel være utfordrende å holde oversikt, og tidkrevende å sette seg inn i andres arbeid. Vi ønsker derfor å se nærmere på følgende problemstilling:

Hvilke støtteverktøy er fordelaktig å anvende i utvikling av en webapplikasjon med flere utviklere?

Problemstillingen er et åpent spørsmål, og er derfor konkretisert i to forskningsspørsmål, som vi ønsker å fokusere på:

1. *Er det fordelaktig å ta i bruk statisk typesjekking i det dynamisk typede programmeringsspråket JavaScript?*
2. *Hvordan kan testing og statisk typesjekking minske utfordringene i et webutviklingsprosjekt med flere utviklere?*

1.3 Struktur

Kapittel 1: Introduksjon og relevans, introduserer bakgrunnen for oppgaven, problemstillingen, forskningsspørsmålene og rapportens struktur.

Kapittel 2: Teori, forklarer relevant teori som er benyttet i prosjektet.

Kapittel 3: Valg av teknologi og metode, beskriver valg av teknologier underveis, og bakgrunnen for valgene.

Kapittel 4: Resultater, presenterer resultatene fra prosjektet

Kapittel 5: Diskusjon, diskuterer resultatene, som ble lagt frem i kapittel 4

Kapittel 6: Konklusjon og videre arbeid, svarer på problemstillingen og forskningsspørsmålene

Kilder, legger fram alle kildene som er brukt

Vedlegg, viser alle vedleggene til sluttrapporten

Kapittel 2

Teori

I dette kapittelet vil vi ta for oss det teoretiske arbeidet som har vært viktige for å løse problemstillingen og å utvikle sluttproduktet. Først vil vi ta for oss teorien knyttet til problemstillingen, etterfulgt av teori som legger grunnlaget for design- og produktvalg. Til slutt tar vi for oss det teoretiske grunnlaget for valg av utviklingsmetodikk.

2.1 Typesjekking

Alle programmeringsspråk har et eget typesystem: Et sett med regler for typene i programmeringsspråket. Typesystem blir brukt for å redusere antall feil ved å verifisere at data er representert riktig i et program. Typer (datatyper) forteller kompilatoren hvordan variabler skal brukes, og hvilke data de kan tilegnes. Typen bestemmer hvilke operasjoner som kan utføres på verdiene, hvor mye av minnet variabelen legger beslag på og hvordan bitmønsteret skal tolkes. [1]

Typesjekking er prosessen som verifiserer at alle variabler tilhører en bestemt type. I tillegg sjekker en at typen gir mening i programmet, slik at programmet kan kjøres typesikkert. Denne prosessen utføres enten statisk eller dynamisk.

2.1.1 Dynamisk typesjekking

Ved dynamisk typesjekking utføres typesjekkingen samtidig som koden kjøres. Eventuelle feil avdekkes først når den problematiske delen av koden eksekveres, og kan derfor føre til at programmet avbrytes. Til tross for dette er dynamiske språk populære fordi de er fleksible, og programmeringsspråket er ofte enklere i bruk. I tillegg slipper en å vente på kompilatoren før en kan prøve ut den nye funksjonaliteten.

2.1.2 Statisk typesjekking

I et statisk typet språk må typen til en variabel deklareres før variabelen kan brukes, og typen vil da være immutabel. Dersom variabelen tildeles en verdi som ikke samsvarer med typen, vil dette føre til feil ved kompilering. Statisk typesjekking utføres ved kompilering, og eventuelle typefeil vil forhindre kjøring av programmet. Dette

fører til en typesikker kodebase, der eventuelle typefeil avdekkes fortløpende under utvikling. Selv ved typesikker kode, kan andre svakheter føre til et annet resultat enn forventet. [2]

I matematikken er et bevissystem *sound* hvis alt som kan bevises er sant. Et bevissystem er *complete* dersom alt som er sant kan bevises.[3] I et typesystem er *soundness* typesjekkerens evne til å avdekke alle feil som kan forekomme under kjøring. Programmet er typesikkert dersom typesjekken stemmer for alle typer. *Completeness* er derimot typesjekkerens evne til å avdekke alle feil som kommer til å oppstå under kjøring, og er typesikkert dersom alle typer kan typesjekkes. [4]

2.1.3 Statisk typesjekking i JavaScript

I et dynamisk programmeringsspråk, som JavaScript, bestemmes typen til en variabel under kjøring. Dette gjør JavaScript fleksibelt, men kan føre til at programmet fungerer annerledes enn utviklerens hensikt. TypeScript og Flow er to hjelpemidler for å gjennomføre statisk typesjekking i JavaScript.

2.1.3.1 TypeScript

Typescript er et typet supersett til JavaScript. Det vil si at TypeScript er et programmeringsspråk som inneholder alle funksjonene til JavaScript, men skiller seg fra JavaScript ved at det bruker et avansert typesystem. TypeScript støtter mange av JavaScripts kodemønstre, og er fleksibel når det gjelder bruk av disse.

Typescript implementerer statisk typesjekking og en kilde-til-kilde kompilator (*transpiler*) som oversetter kildekoden til JavaScript. [5] TypeScript har en modus som heter *strict*, som aktiverer en strengere og mer omfattende typesjekking. Dette gir større garanti for en typesikker kodebase. [6] I de fleste tilfeller vil TypeScript være *sound*, men godkjenner at visse operasjoner er ukjente ved kompilering.

2.1.3.2 Flow

Flow er en statisk typesjekker for JavaScript, og ikke et eget programmeringsspråk. Flow tillater typeannotering, men gjennomfører kun typesjekken, og er avhengig av et annet verktøy for å fjerne annoteringene. Dersom annoteringene legges i kommentarer kan kildekoden kompileres uten andre verktøy, og Flow vil oppdage annoteringene.

Flow bruker dataflytanalyse (*flow analysis*) for å samle informasjon om verdier, som blir kalkulert på forskjellige steder i programmet. Denne informasjonen benyttes av

kompilatoren for å finne feil som kan gjøre at programmet oppfører seg uventet, og gir beskjed om hvordan utvikleren skal håndtere dette. [2]

Flow prøver å være både *sound* og *complete*, men ettersom JavaScript ikke er designet rundt et typesystem, hender det at Flow må ta et valg. I de fleste tilfeller velger Flow *sound* over *complete*, for å forsikre seg om at koden ikke har noen feil. [4]

2.2 Testing

Testing er en viktig del i utviklingen av et programvaresystem. Gjennom testing kan man oppdage og forhindre feil i programmet, sørge for at funksjonaliteten fungerer som den skal, og at man møter oppdragsgivers krav.

2.2.1 Akseptansetesting

Akseptansetesting utføres typisk av oppdragsgiver for å verifisere at systemet møter kravene som er avtalt i kontrakten. Slike tester kan foregå manuelt gjennom demonstrasjon, forklaring eller ved at oppdragsgiver får teste koden. [7]

2.2.2 Enhetstesting

Enhetstesting går ut på å teste individuelle enheter i koden for å sørge for at de fungerer som tiltenkt. Enhetstesting kan føre til ryddigere kode med høy kohesjon og løse koblinger, ettersom en tvinger frem en kode som muliggjør testing av den enkelte enheten. [8]

2.2.3 Brukertesting

Feilfri kode tilsier ikke at systemet er feilfritt, derfor er det essensielt å teste sluttbrukerne av systemet. Brukertesting benyttes for å teste at brukerne forstår, og er i stand til å bruke systemet. Under brukertesting får reelle brukere definerte oppgaver i kontrollerte omgivelser, for å se hvordan de samhandler med systemet. Brukertesting kan gjennomføres flere ganger i løpet av prosjektperioden for å evaluere systemet opp mot brukerkravene. [9]

2.3 Systemutvikling

2.3.1 Kontinuerlig integrasjon

Kontinuerlig integrasjon er en utviklingspraksis som innebærer at ny kode lastes opp og fusjoneres med eksisterende kode underveis i utviklingen, gjerne opptil flere ganger om dagen. Før automatisk bygging av koden gjennomføres tester for å verifisere koden, og dersom feil oppstår avbrytes prosessen. Dette sikrer en kodebase med færre feil, da disse kan oppdages raskere og rettes opp fortløpende. [10]

2.3.2 Kontinuerlig utrulling

Kontinuerlig utrulling er det påfølgende steget til kontinuerlig integrasjon. Dersom den nye koden passerer automatisk bygging og testing uten feil, vil den automatisk lastes opp og kjøres på en test- eller produksjonsserver. Å bruke dette i et utviklingsprosjekt kan føre til hyppigere leveranser. Dette vil skape flere små endringer for brukerne av systemet, fremfor større oppdateringer med mye nytt å sette seg inn i. Det fører til at ny funksjonalitet kan tas i bruk tidligere, og feil vil lokaliseres fortløpende. I mange tilfeller vil kostnader senkes, og overgangen til et nytt system kan bli enklere for brukerne.

2.3.3 Representational State Transfer

Representational State Transfer(REST) er et arkitekturmønster for distribuerte systemer, som består av ressurser og enkle, definerte operasjoner en kan utføre på disse. Et system kan kalles *RESTful* dersom det følger følgende prinsipper[11]:

- **Klient - server**: Ved å skille mellom klient- og serversiden blir systemet mer fleksibelt, og en kan endre på den ene uten å påvirke den andre.
- **Tilstandsløs**: Serveren lagrer ingen informasjon om tilstanden til klienten. For at kommunikasjonen skal være tilstandsløs må hver forespørsel fra klienten inneholde all nødvendig informasjon.
- **Hurtiglager**: *Cacheable* responser skal lagres i hurtiglageret slik at den kan brukes ved senere anledninger. Dette blir mindre ressurskrevende, ettersom data ikke hentes fra serveren på nytt.
- **Uniformt grensesnitt**: Ved å standardisere måten komponentene kommuniserer, kan man gjøre systemarkitekturen enklere og synligere. Dette kan gjøres ved å identifisere individuelle ressurser i forespørsler. Ressursene som returneres til klient trenger ikke å være representert på samme måte som i databasen.

- **Lagdelt system**: Klienten er uvitende om den kommuniserer direkte med sluttserveren eller om den går via andre tjenester underveis. I en RESTful applikasjon vil ikke servere som er plassert mellom klient og sluttserver påvirke kommunikasjonen.
- **Kode på etterspørsel**(valgfri): Funksjonalitet på klientsiden kan utvides ved å laste ned og kjøre kode fra skript.

2.3.4 Object-Relational Mapping

Object-Relational Mapping (ORM) er en programmeringsteknikk, som konverterer data mellom to inkompatible systemer ved bruk av objektorientert programmering. ORM håndterer data til og fra en relasjonsdatabase, og lager passende objekter som programmereren kan jobbe med. På denne måten kan en utføre operasjoner og spørringer på objektet i stedet for direkte mot databasen. Bruken av ORM sikrer systemet mot angrep med SQL-injeksjon. [12]

2.3.5 Data Transfer Object

Data Transfer Object (DTO) brukes for å innkapsle data som skal sendes mellom to subsystemer, fremfor å sende data hentet direkte fra databasen. Klienten kan dermed ikke legge til flere egenskaper på objekter, og egenskaper klienten ikke skal ha tilgang til fjernes.

2.3.6 Interaksjonsdesign

Interaksjonsdesign handler om å designe samhandling mellom mennesker og digitale systemer, objekter, tjenester og miljøer, for å lage systemer som er enkle, effektive og behagelige å bruke. [13]

Jakob Nielsens 10 prinsipp for interaksjonsdesign
Prinsippene under er hentet fra Nielsen Norman Group [14], og er huskeregler for brukervennlighet, ikke spesifikke retningslinjer.

1. **Synlighet av systemets status**: Brukerne skal få tilbakemelding fra systemet om hva som foregår til enhver tid.
2. **Likhet mellom systemet og den virkelige verden** - Systemet må gi tilbakemelding til brukerne på et språk som brukerne forstår.
3. **Brukerkontroll og frihet**: Dersom brukerne utfører en handling ved et uhell, må de ha en tydelig utvei fra tilstanden som systemet er i.

4. **Konsekvens og standardisering**: Ulike ord og funksjoner skal ikke endre betydning på tvers av kontekst.
5. **Forebygging av feil**: Design som hindrer situasjoner som kan føre til feil.
6. **Gjenkjennelse fremfor hukommelse**: Synliggjøre objekter, handlinger og alternativer slik at det ikke er nødvendig for bruker å memorere informasjon.
7. **Fleksibilitet og effektivitet**: Snarveier for erfarne brukere, som er usynlige for uerfarne brukere.
8. **Minimalistisk design**: Informasjonsbokser skal kun innholde den nødvendige informasjonen som er relevant for brukeren.
9. **Hjelpe brukeren etter feil har oppstått**: Feilmeldinger bør vise nøyaktig hva som er galt og foreslå en løsning.
10. **Hjelp og dokumentasjon**: Et system bør kunne brukes uten dokumentasjon, men i noen tilfeller kan det være nødvendig med hjelp. Dokumentasjonen skal være oversiktlig og tilpasset brukerens oppgaver.

2.3.7 Separation of concerns

Separation of concerns er et designmønster hvor systemet deles opp i moduler med distinkt funksjonalitet. Antall moduler avhenger av systemets størrelse. Dette reduserer komplekse problemer til håndterlige moduler. [15]

2.4 Utviklingsmetodikk

2.4.1 Smidig utvikling

Smidig utvikling er en generell betegnelse for metoder basert på iterativ utvikling. I en iterativ prosess går en syklisk gjennom de ulike fasene. Det er tilbakekoblinger til gamle steg, slik at en enkelt kan gjøre endringer underveis. Dette kan være tidsbesparende ettersom man kan unngå store endringer senere. Gjennom prosessen har utviklerne et tettere samarbeid med kunden og får kontinuerlig tilbakemelding på produktet. Gjennom læring og endring underveis kan smidige metoder øke kvaliteten og redusere risiko. [16]

2.4.2 Scrum

"Scrum er et prosessrammeverk hvor mennesker kan adressere komplekse, adaptive problemer, mens de produktivt og kreativt leverer produkter med høyest mulig verdi" [17]. Scrum bygger på verdiene i manifestet for smidig programvareutvikling, og bruker en heuristisk[1] tilnærming for å løse komplekse problemer og håndtere uforutsigbarheter.

Et Scrum team består av tre roller: Produkteier, Scrum master og utviklerteamet. De er ikke avhengig av parter utenfor Scrum teamet. Produkteier er ansvarlig for å styre prioriteten til oppgavene og representere de ulike interessentene i prosjektet. Scrum master skal veilede utviklerteamet, sørge for at alle prosesser følges, og at de er så produktive som mulig. Utviklerteamet gjør selve arbeidet med å ferdigstille funksjonalitet ut fra kravene i produktkøen (*Product Backlog*). Utviklerteamet skal være selvorganiserte og kryssfunksjonelle. Dette innebærer at de selv velger hvordan arbeidet skal organiseres, og at de innehar den nødvendige kunnskapen for å gjennomføre prosjektet. [17]

Scrum definerer tre artefakter som gir informasjon de involverte må være klar over for å forstå produktet og aktivitetene som inngår. Produktkøen (*Product backlog*) er en liste over alle funksjonaliteter systemet skal ha. Listen er sortert etter prioritering, hvor de viktigste funksjonalitetene står øverst, og skal implementeres først. Produktkøen justeres og endres i takt med produktet, slik at den alltid identifiserer funksjonaliteter som systemet trenger. Sprintkøen (Sprint backlog) inneholder de elementene fra produktkøen som skal implementeres i løpet av en sprint. Et inkrement er summen av alle elementer fra produktkøen som implementeres i løpet av en sprint, og det som har blitt implementert i tidligere sprinter. [17]

Scrum definerer fem seremonier for å skape rutine og minimere behovet for møter utenom de faste seremoniene. En sprint varer i en til fire uker, og kan bli sett på som et lite prosjekt hvor man skal oppnå et sprintmål. I løpet av denne tiden skal et brukbart inkrement produseres. Hver sprint starter med en sprintplanlegging (*Sprint Planning*). Målet med planleggingen er å finne ut hva man klarer å levere i løpet av den kommende sprinten. Arbeidet som skal gjennomføres velges fra produktkøen og legges inn i sprintkøen. Hver arbeidsdag i sprinten gjennomfører utviklerteamet et Scrum-møte (Daily Scrum) for å synkronisere arbeidet sitt og legge en plan for det neste døgnet. Scrum master sørger for at utviklerteamet har møtet, men utviklerteamet er selv ansvarlig for å gjennomføre.

[1] Heuristikk er en enkel fremgangsmåte eller strategi som en problemløser kan ta i bruk for å øke sjansen til å løse en oppgave. [18]

Etter endt sprint gjennomføres det en sprintgjennomgang (Sprint Review) hvor utviklerteamet viser frem den nye funksjonaliteten. Produkteier og eventuelle interessenter får muligheten til å inspisere og gi tilbakemeldinger, som utviklerteamet tar med seg videre til neste sprintplanlegging. Produkteier har nå mulighet til gjøre endringer i produktkøen. Etter gjennomgangen vil utviklerteamet evaluere den gjennomførte sprinten i et retrospekt (Sprint Retrospective). Her diskuterer de hva som har gått bra, hva som kunne gått bedre, og hva de skal forplikte seg til å gjøre i neste sprint. Dette er en mulighet for teamet til å forbedre seg, og det er derfor viktig at alle deltar for å dele sine synspunkter.

I Scrum benyttes korte iterasjoner hvor man kontinuerlig kontrollerer resultatene og tilpasser produktkøen underveis, fremfor langsiktig planlegging. Scrum muliggjør gjennomføring av prosjekter hvor kravene ikke er fullstendig utarbeidet på forhånd, og gjør det enklere å håndtere endringer. [19]

Kapittel 3
Valg av teknologi og metode

I dette kapittelet vil vi begrunne valg av teknologi og metoder, som har vært sentrale i utviklingen av sluttproduktet. Første del vil ta for seg valgene knyttet til serversiden og andre del vil ta for seg valg knyttet til klientsiden. Til slutt vil vi begrunne valg av utviklingsmetodikk og rollefordeling.

3.1 Valg av teknologier til serversiden

3.1.1 Azure

Azure er en gruppe skytjenester skapt av Microsoft for bygging, testing, utrulling og administrering av applikasjoner. Tjenestene kjøpes på et forbruksbetalt grunnlag. Det vil si at en kun betaler for det en bruker. Tilgang til tjenestene gis gjennom en sikker Internett-tilkobling. [20] NoIS benytter Azure for autentisering og *hosting* av eksisterende systemer. Oppdragsgiver ønsket derfor at det nye systemet også skulle benytte tjenester levert av Azure.

3.1.1.1 Azure SQL Database

Azure SQL Database er en intelligent, skalerbar relasjonsdatabase, levert som en tjeneste. Databasen er basert på den siste versjonen av Microsoft SQL Server. Gjennom portalen tilbyr Microsoft en rekke verktøy for å håndtere databasen. Dette inkluderer tilgang til tilkoblingsinformasjon, statistikk, dataadministrasjon og oversikt over hele databasen. For å tilpasse ytelsen bruker databasen maskinlæring. [21]

3.1.1.2 Azure Active Directory

Azure Active Directory (Azure AD) er Microsofts skybaserte tjeneste for identitets- og tilgangsadministrasjon. Bedrifter kan bruke *single sign-on* for tilgang til alle Microsofts tjenester og andre applikasjoner som benytter tjenesten. Azure AD ble brukt i vårt system for å integrere pålogging for bedriftens eksisterende brukere.

3.1.2 GitHub

GitHub er en utviklingsplattform hvor en kan *hoste*, administrere og bygge programvare. [22] Vi benyttet oss av GitHub for å lagre koden eksternt, og versjonskontrollsystemet Git for å holde oversikt over versjonene. GitHub Actions[1] brukes for å integrere automatisk testing, bygging og utrulling til Azure.

3.1.3 ASP.NET Core

ASP.NET Core er et rammeverk med åpen kildekode, bygget på Microsofts .NET Core. Rammeverket kan brukes på tvers av de største operativsystemene Windows, MacOS og Linux. NuGet er en pakkebehandler for ASP.NET Core, som gjør det mulig å produsere utvidelser eller benytte utvidelser andre har laget. [23] Oppdragsgiver anbefalte å benytte ASP.NET Core, fordi den støtter flere plattformer og har god integrasjon mot Azure. Ettersom rammeverket er utbredt hos NoIS, blir det enklere for de å videreutvikle systemet. Vi ønsket å lære mer om denne teknologien, og valgte derfor å bruke ASP.NET Core til serversiden.

3.1.4 Entity Framework Core

Entity Framework Core (EF Core) er et ORM-rammeverk som gjør det mulig å jobbe med en database ved bruk av .NET objekter. EF Core bruker Language Integrated Query (LINQ) til spørringer mot databasen, som muliggjør databasespørringer i C#. Rammeverket aksesserer databasen ved bruk av modeller. En modell består av et kontekstobjekt, og entitetsklasser som representerer tabellene i databasen. Dersom en modell endres, kan man bruke migrering for å endre databasestrukturen. Dette gjør det mulig å endre databasen selv om systemet er i produksjon. Rammeverket kan brukes på flere plattformer, og fungerer med ASP.NET Core-applikasjoner. [24]

3.1.5 xUnit

xUnit er et testerverktøy brukt for å lage enhetstester i .NET Core. Testene kjøres mot en kopi av databasen som opprettes *in-memory*. Databasen nullstilles for hver test, slik at testene ikke påvirker hverandre. Det er ikke nødvendig å sette opp og slette testmiljøet for hver test, som skaper ryddigere kode uten duplikater. [25]

[1] Verktøy for å sette opp automatiske arbeidsflyter knyttet til en kodebase

3.1.6 REST

For å sørge for en oversiktlig oppbygning valgte vi å bruke arkitekturmønsteret REST. Som nevnt i kapittel 2.3.3 skiller en mellom klient og server, som gjør det mulig å endre på den ene uavhengig av den andre. Dette er en fordel når flere skal jobbe med utviklingen parallelt.

3.2 Valg av teknologier til klientsiden

3.2.1 React

React er et JavaScript-bibliotek for å lage dynamiske brukergrensesnitt uten å forholde seg til DOM-en. Brukergrensesnittet bygges opp av mindre, gjenbrukbare komponenter med hver sin tilstand. Hovedformålet til React er å være raskt, skalerbart og enkelt. Det eksisterer flere biblioteker og rammeverk til React, som effektiviserer utformingen av brukergrensesnittet. Gjennom faget TDAT2003 Systemutvikling 2 med webapplikasjoner fikk vi en introduksjon til React, og ønsket derfor å utvikle vår kunnskap innen denne teknologien. [26]

3.2.2 React Testing Library

React Testing Library (RTL) er et lettvektsbibliotek for å utføre enhetstester på React-komponenter. Ved bruk av RTL kan en teste komponentene i DOM-en, på samme måte som brukerne ville testet komponentene. Dette fører til at testene blir likere hvordan en bruker ville testet systemet. [27]

3.2.3 Material UI

Material UI er et React-rammeverk med ferdiglagde komponenter som kan brukes til å utforme brukergrensesnittet. Material UI er godt dokumentert, med eksempler på hvordan tema og komponenter kan tilpasses eget behov. Ved å bruke et komponentrammeverk sparer en tid på utformingen av brukergrensesnittet, samtidig som en kan tilpasse komponentene og skape et responsivt design.

3.2.4 TypeScript

Vi valgte å bruke TypeScript for å gjennomføre statisk typesjekking i React. TypeScript er et utbredt programmeringspråk, som er kompatibelt med flere rammeverk. Dette gjør det enklere å finne svar på spørsmål. TypeScript har bred støtte for bruk i IDE'er, slik at IDE'ene kan gi forslag og informasjon underveis. I tillegg har

Material UI eksempler for hvordan deres komponenter kan integreres ved bruk av TypeScript.

3.2.5 ESLint

ESLint er et statisk analyseverktøy for å se etter problematiske mønster og verifisere at koden overholder spesifikke retningslinjer. Siden JavaScript ikke gir feilmeldinger, kan ESLint brukes for å automatisk analysere koden for vanlige feil. Det er ikke nødvendig å kjøre koden eller åpne nettleseren for å bruke ESLint. [28]

3.2.6 Babel

Babel er en JavaScript-kompilator som kompilerer nyere JavaScript-kode til Javascript ES5, som er kjørbar i eldre nettlesere. Babel gjør det derfor mulig å utvikle med den nyeste versjonen av JavaScript, uten å ta hensyn til eldre nettlesere. [29]

3.3 Digitale samhandlingsverktøy

3.3.1 Slack

Slack er kommunikasjonsplattformen oppdragsgiveren bruker for å kommunisere innad i bedriften. Vi fikk tilgang til deres kanal, og kommunikasjonen med oppdragsgiver skulle foregå over denne plattformen. Vi valgte derfor å bruke Slack for kommunikasjon innad i gruppen.

3.3.2 Microsoft Teams

For å kunne gjennomføre veiledningsmøter uten å møtes fysisk, brukte vi Microsoft Teams. Her kan en sende invitasjon via e-post, som legges til i kalenderen. I tillegg kan Microsoft Teams brukes uten å opprette en bruker. Plattformen støtter skjermdeling, slik at arbeid kan presenteres over internett.

3.4 Valg av utviklingsmetodikk

3.4.1 Scrum

Oppdragsgiver ønsket at det nye systemet skulle tilpasses den prosessen de har i dag, og inkludere den viktigste funksjonaliteten fra det eksisterende systemet. Utover dette var det ingen spesifikke krav å forholde seg til, annet enn ønsker fra

produkteier og øvrige ansatte. Vi ønsket derfor å velge en metode som ville gjøre det enklere å håndtere endringer. Som nevnt i kapittel 2.4.2 egner Scrum seg for prosjekter hvor kravene utarbeides og endres underveis. I tillegg hadde vi erfaring med rammeverket, og Scrum var derfor et naturlig valg.

På grunn av størrelsen på teamet og usikkerhet rundt hvilke dager vi ville ha mulighet til å møtes og jobbe sammen, ble vi enige om å ha Scrum-møte to ganger i uken, fremfor hver dag. En ansatt fra NoIS fylte rollen som Scrum master. For å holde oversikt over produktkøen og sprintkøene brukte vi Trello, et online tavleverktøy for å administrere prosjekter digitalt.

3.4.2 Rollefordeling

Ettersom gruppen kun består av tre personer, ønsket vi ikke å definere spesifikke roller eller ansvarsområder. Arbeidet skulle fordeles rettferdig mellom medlemmene i gruppen, og oppgaver skulle påbegynnes fortløpende. På denne måten fikk alle kunnskap om de ulike arbeidsområdene, som muliggjør samarbeid om oppgaver og problemer.

Kapittel 4

Resultater

4.1 Vitenskapelige resultater

I dette delkapittelet vil vi ta for oss de vitenskapelig resultatene i prosjektet. Resultatene er basert på observasjoner og erfaringer gjennom prosjektet, i lys av problemstillingen.

4.1.1 Innføring og bruk av statisk typesjekking

I utviklingens startfase var ikke problemstillingen definert, og programmeringsspråket JavaScript ble derfor brukt på klientsiden. Ved ferdigstilt problemstilling konverterte vi kildekoden til TypeScript for å undersøke effekten av statisk typesjekking.

Alle filene ble konvertert manuelt ved å legge til typer på alle variabler og funksjoner. Filene var ikke kjørbare før kodebasen var typesikker, som førte til at vi ikke kunne teste systemet i nettleseren underveis. Under overgangen oppdaget vi at flere av rammeverkets metoder ble brukt uten nødvendige avhengigheter. Feilene ble ikke oppdaget tidligere fordi metodene fungerte til vårt bruk, men i spesielle situasjoner kunne de gitt et annet resultat enn forventet. I tillegg ble det definert objekter med faste attributter, som var tidkrevende fordi vi måtte inspisere objektene som ble sendt fra serveren.

Etter innføringen av TypeScript, opplevde vi at koden var mer oversiktlig og det var enklere å sette seg inn i andres arbeid. De definerte objektene kunne brukes i flere sammenhenger og effektiviserte arbeidet. Ved å deklarere typer til hver variabel oppdaget vi feil i avhengige komponenter dersom typene ikke var kompatible.

4.1.2 Utfordringer i et prosjekt med flere utviklere

Gruppemedlemmene hadde noen ganger ulik oppfatning av hvordan en utfordring skulle løses. Dette førte til at oppbygningen av koden varierte ut fra hvem som skrev den. Det ble dermed vanskelig å sette seg inn i andres kode, og behovet for å spørre om hjelp fra de andre gruppemedlemmene økte. Dette kunne gå på bekostning av tid og arbeidsflyt.

Etter noen endringer opplevde vi at brukergrensesnittet så ut til å fungere slik det skulle, men returnerte feil svar. For å ta hånd om dette innførte vi klientsidetester for de gjenbrukbare komponentene og kritiske inntastingsfelt. Slik fikk vi sikret at funksjonalitet ble bevart ved endringer, og at feil datatype ikke kunne registreres i inntastingsfeltene.

Brukertester ble benyttet for å teste funksjonalitet utover det som ble dekket av enhetester. Vi fikk da testet både at systemet oppfylte brukernes behov, og at brukerne forstod systemet. I tillegg fikk vi testet at komponentene fungerte som tiltenkt, og at systemet i sin helhet fungerte i sluttmiljøet.

REST-endepunktene ble testet i en tidlig fase og oppdatert kontinuerlig. Testene ble integrert i den automatiske utrullingen for å sikre fungerende endepunkter i produksjonserveren. Videre sikret testene at dataen som ble sendt til klientsiden var riktig, og minimerte området vi måtte feilsøke ved eventuelle feil.

4.2 Ingeniørfaglige resultater

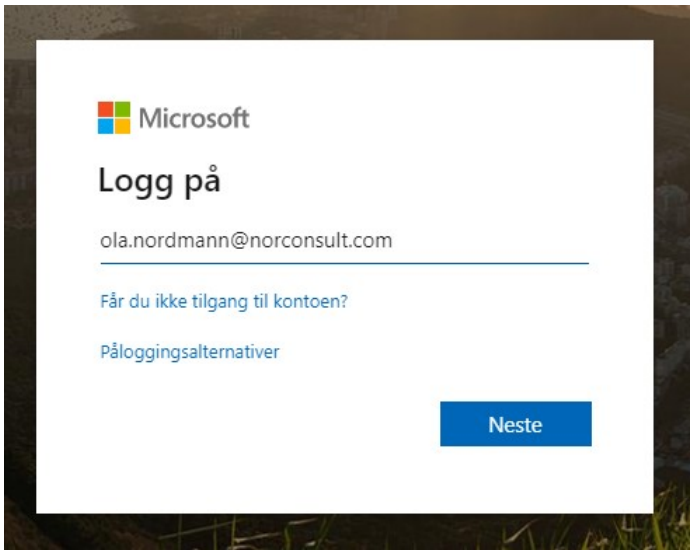
4.2.1 Funksjonelle krav

Denne delen beskriver måloppnåelsen på de funksjonelle kravene beskrevet i Vedlegg A - Visjonsdokument, kapittel 5. De funksjonelle kravene er gruppert etter sidene i systemet. Måloppnåelsen rangeres fra 1 til 5, hvor 1 er lav og 5 er høy.

4.2.1.1 Tilgang

Tilgangsadministrasjonen blir gjort ved bruk av Azure AD, hvor NoIS har et eget domene. De ansatte kan derfor logge inn med eksisterende brukere. Dersom brukeren er innlogget i et annet system gjennom Azure AD, vil brukeren automatisk få tilgang til systemet.

Grad av måloppnåelse: 5



Figur 4.1: Innlogging til systemet

4.2.1.2 Registrere fagaktivitet

For å registrere en fagaktivitet må brukeren gjennom seks steg:

1. **Kategori**: Kategori velges fra alternativene: Sertifisering, kurs, konferanse og internt arrangement.
2. **Detaljer**: Aktivitetsnavn kan velges fra en liste over tidligere registrerte fagaktiviteter eller skrives inn selv. Listen med forslag avhenger av hvilken kategori som er valgt. De resterende feltene (arrangør, lenke og sted) er valgfrie.
3. **Dato**: Velg start- og sluttdato i kalenderen.
4. **Kostnader**: Kostnadene knyttet til gjennomføringen av en fagaktivitet fylles inn her. Det er mulig å legge ved en kommentar til kostnadene. Alle inntastingsfeltene har hjelpetekst for å sørge for at riktig informasjon registreres.
5. **Utbytte**: Den ansatte må krysse av for ett eller flere alternativer, som beskriver utbytte av fagaktiviteten.
6. **Fullfør**: Kan velge å fullføre registreringen eller avbryte. Dersom en velger å fullføre registreringen vil statusen på fagaktiviteten settes til "ikke godkjent" frem til en administrator godkjenner den.

Grad av måloppnåelse: 4
Kun utvalgte inntastingsfelter har en tilhørende hjelpetekst, da de resterende feltene var selvforklarende i følge oppdragsgiver.

4.2.1.3 Profil

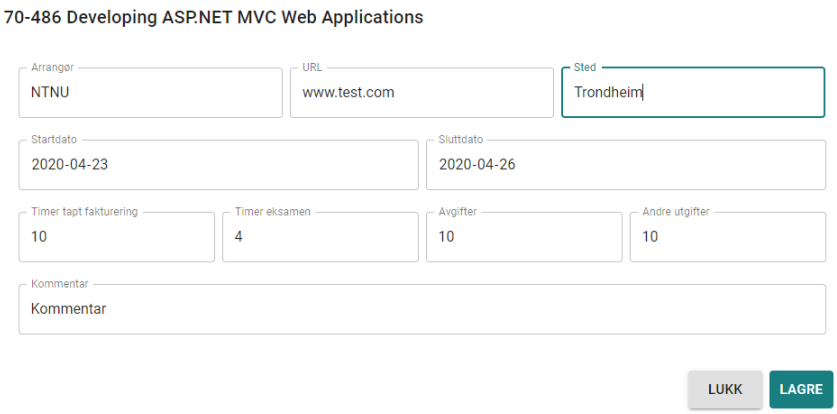
Utformingen av profilsiden er lik, uavhengig om man er på sin egen eller andres profil, med mer funksjonalitet på egen profil. På profil-siden kan man se en oversikt over registrerte fagaktiviteter. Hver fagaktivitet har en status og en tilhørende prosentverdi som viser hvor langt personen har kommet i prosessen. En kan søke på fagaktivitetsnavn, og oversikten kan filtreres etter kategori og status.

Sidepanelet viser avdelingen den ansatte tilhører, antall gjennomførte fagaktiviteter totalt og antall planlagte fagaktiviteter. Bildet i sidepanelet hentes fra CV Partner om den ansatte har et bilde i CV-en sin, hvis ikke brukes initialene til den ansatte.

Figur 4.2: Profilen til en ansatt

Ved klikk på en fagaktivitet i oversikten blir man sendt til en ny side med all informasjonen den ansatte har registrert om denne fagaktiviteten. På egen profil kan en fullføre, avlyse og endre en fagaktivitet. Ved endring får man opp en dialogboks hvor den tidligere registrerte informasjonen er ferdig utfylt i inntastingsfeltene. Databasen vil oppdateres med den nye informasjonen dersom den ansatte bekrefter endringene.

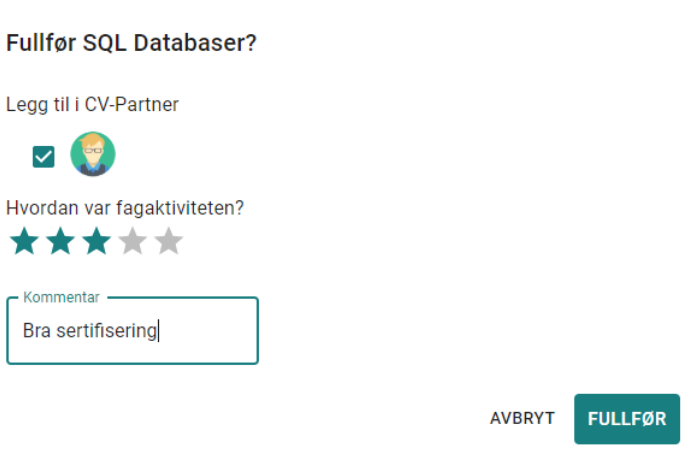
Grad av måloppnåelse: 5



Figur 4.3: Endre en fagaktivitet

4.2.1.4 Synkronisere fagaktivitet med CV Partner

Ved fullføring av en sertifisering eller et kurs, kan den ansatte velge å registrere fagaktiviteten i CV Partner. Fagaktiviteten vil automatisk legges inn i CV-en til den ansatte. I tillegg må den ansatte gi fagaktiviteten en rangering fra en til fem, ut fra hvor nyttig og lærerikt den opplevdes. En kommentar kan registreres dersom den ansatte ønsker å skrive en vurdering eller gi tips til andre.



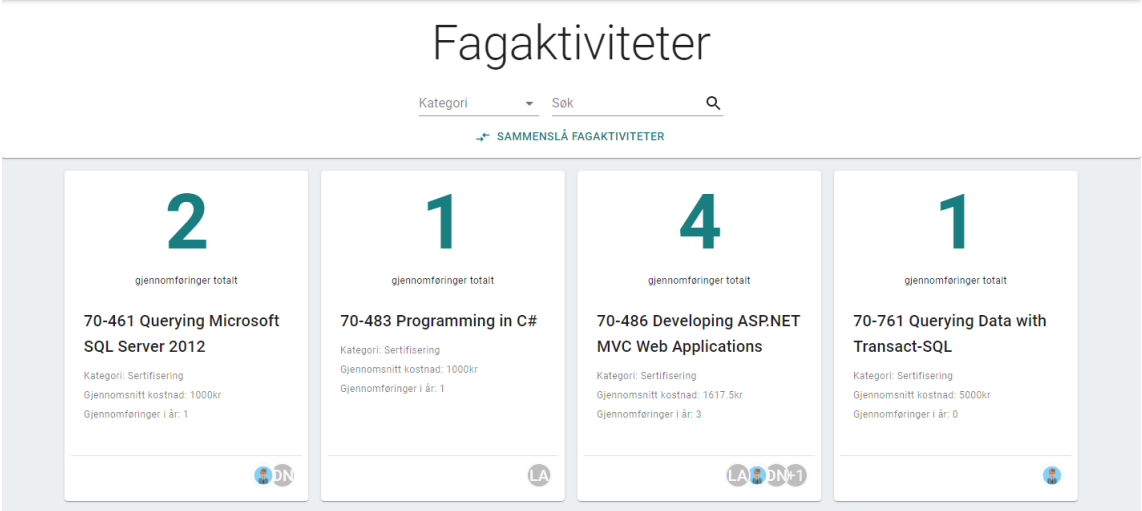
Figur 4.4: Fullfør aktivitet, og legg til i CV Partner

Hvis en ansatt har fagaktiviteter i CV-en sin som ikke er registrert i databasen, har den ansatte mulighet til å synkronisere fagaktivitetene. Kun kategori, navn på fagaktiviteten og sluttdato kan hentes fra CV Partner. Den ansatte kan derfor velge å fylle inn resterende informasjon. Fagaktiviteten vil lagres i databasen om brukeren klikker på "Legg til aktivitet".

Grad av måloppnåelse: 5

4.2.1.5 Fagaktivitet

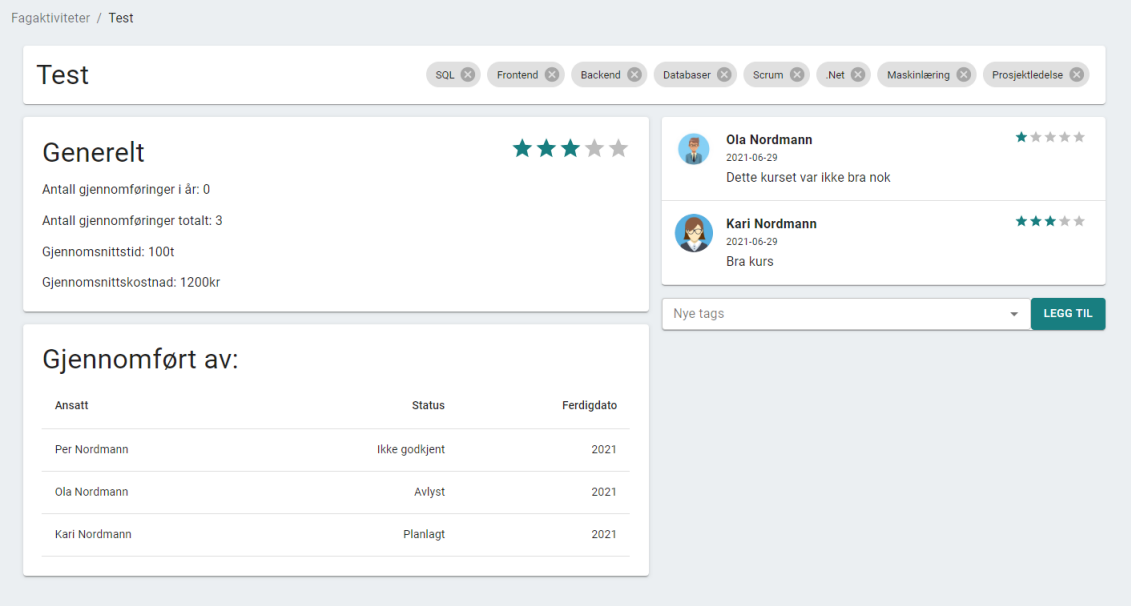
Fagaktivitet-siden gir en oversikt over alle fagaktivitetene som er registrert i systemet. For hver fagaktivitet vises kategori, gjennomsnittskostnad og antall ansatte som har gjennomført fagaktiviteten totalt, og dette året. Det er mulig å søke på fagaktivitetsnavn og filtrere oversikten på kategori. For å unngå duplikater, er det mulig å slå sammen fagaktiviteter som er like, men lagret med ulike navn.



Figur 4.5: Oversikt over alle fagaktiviteter

Ved klikk på en fagaktivitet i oversikten, sendes en til siden med generell informasjon om fagaktiviteten. Her er det en tabell som viser alle ansatte knyttet til fagaktiviteten, statusen på gjennomføringen og sluttdato. I tillegg kan en se kommentarer registrert av ansatte som har fullført fagaktiviteten.

Grad av måloppnåelse: 4
Det er ikke mulig å se mest populære fagaktiviteter i en gitt periode, kun totalt.

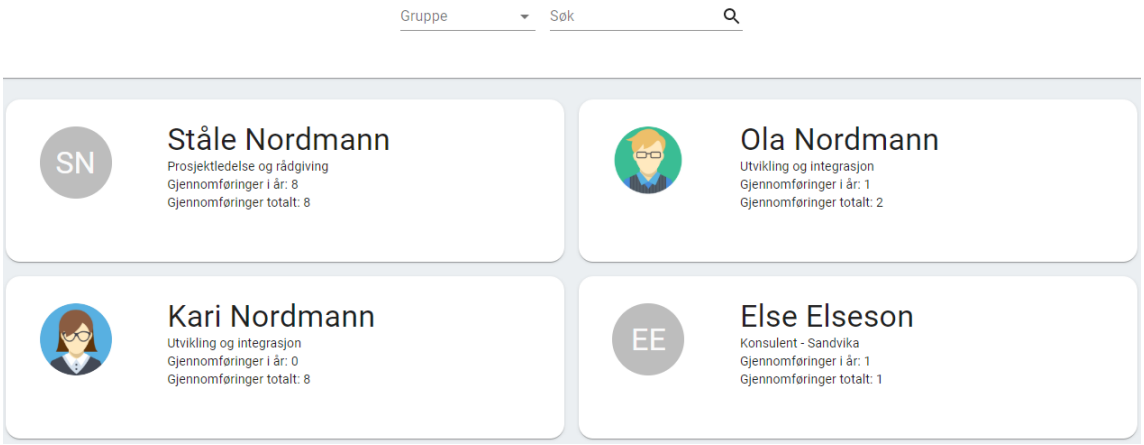


Figur 4.6: Siden til en fagaktivitet

4.2.1.6 Personer

Person-siden gir en oversikt over alle ansatte i systemet, hvilken avdeling de tilhører, antall fagaktiviteter de har gjennomført totalt og antall gjennomføringer i år. Oversikten kan filtreres på avdeling, og det er mulig å søke på navnene til de ansatte. En kan besøke profilen til en ansatt ved å klikke på personen i oversikten.

Grad av måloppnåelse: 5

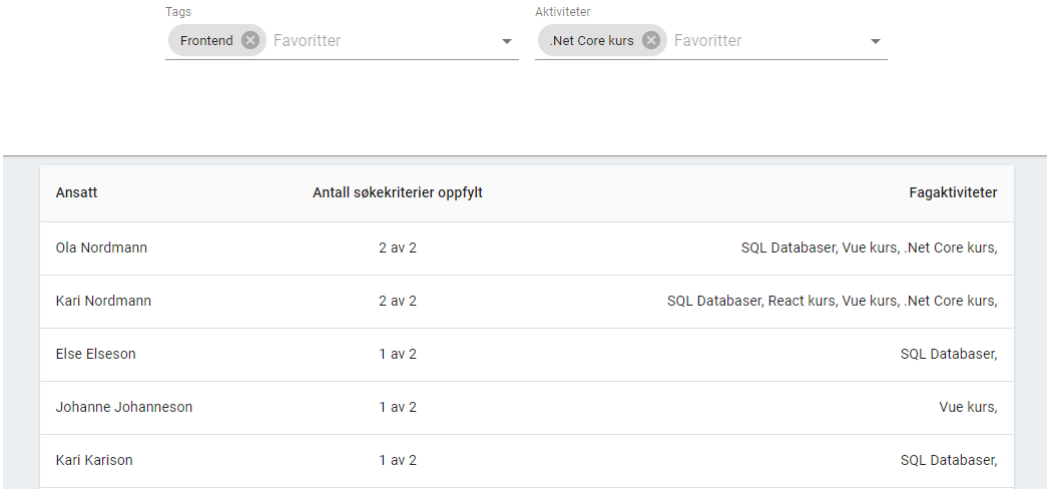


Figur 4.7: Oversikt over alle ansatte i bedriften

4.2.1.7 Kompetanseoversikt

Kompetanseoversikten viser en liste med alle ansatte i systemet, og antall fagaktiviteter de har gjennomført. Når en legger til søkekriterier vil listen sortere de ansatte etter antall kriterier de oppfyller. Listen vil i tillegg vise hvilke fagaktiviteter som gjør at den ansatte oppfyller kriteriene.

Grad av måloppnåelse: 5



Figur 4.8: Kompetanseoversikt: finne personer som oppfyller søkekriteriet

4.2.1.8 Arrangement

Arrangementsiden består av en kalender med kommende arrangementer. For mer informasjon om ett arrangement kan en trykke på arrangementet i kalender.

Grad av måloppnåelse: 5

4.2.1.9 Publisering til Slack

Når statusen på en fagaktivitet oppdateres, vil RSS-feeden oppdateres med den nye informasjonen. Dette publiserer en melding i bedriftens Slack-kanal, med informasjon om oppdateringen.

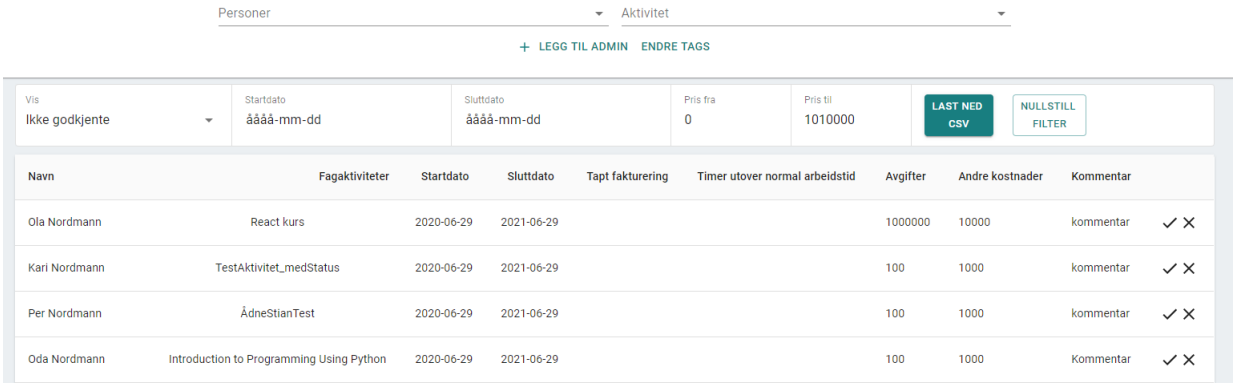
Grad av måloppnåelse: 4
Det er ikke mulig å begrense hvilke statusoppdateringer som skal publiseres.

4.2.1.10 Administrator

Gruppe- og teamledere har fått en egen side for å dekke kravene knyttet til godkjenning og økonomi. Administrator-siden inneholder en tabell, hvor hver rad inneholder

all informasjon en ansatt har registrert om én fagaktivitet. Administratoren kan velge om listen skal vise fagaktiviteter som er godkjente, venter på godkjenning eller alle. Videre kan listen tilpasses etter behov ved hjelp av en rekke filtrerings- og søkemuligheter, som kan brukes om hverandre. Listen kan lastes ned som en CSV-fil slik at informasjonen kan brukes til budsjettplanlegging. Fagaktiviteter godkjennes ved å klikke på hake-symbolet og avslås ved å klikke på krysset. Administratorer kan legge til nye administratorer og *tags*.

Grad av måloppnåelse: 5



Figur 4.9: Oversikt over fagaktiviteter som venter på godkjenning

4.2.2 Ikke-funksjonelle krav

Denne delen vil ta for seg måloppnåelsen til de ikke-funksjonelle kravene beskrevet i vedlegg A - Visjonsdokument, kapittel 6.

4.2.2.1 Brukbarhet

Systemet har blitt utviklet med mål om å være intuitivt og behagelig i bruk. Gjennom to runder med brukertesting har brukergrensesnittet blitt tilpasset de ansattes tilbakemeldinger og ønsker. Elementer som skapte usikkerhet blant noen av testbrukerne ble endret eller forklart ved hjelp av verktøytips og informasjonsbokser. Brukergrensesnittet er utformet slik at det er mulig å bruke med alle skjermstørrelser, men best tilpasset større skjermer. Arrangementsiden er ikke tilgjengelig på mobiltelefon og tilsvarende skjermstørrelser.

Det er lagt inn feltkontroll på alle inntastingsfeltene for å sikre at brukeren fyller inn riktig data. Dette hindrer at feil datatype lagres i databasen.

4.2.2.2 Funksjonell egnethet

Alle brukerbehovene er implementert og fungerer som avtalt med oppdragsgiver, med unntak av muligheten for mørk modus. Dette hadde lav prioritet og kan løses ved bruk av en tilleggsfunksjon i nettleser.

Kravet om at systemet skal hente korrekt data blir sikret gjennom enhetstesting på serversiden. Eventuelle feil blir fanget opp slik at systemet ikke går ned.

4.2.2.3 Ytelse

Systemet krever en stabil internettilgang for at nettsiden skal fungere optimalt. En svak internettilgang kan medføre lengre lastetid og mindre feil på siden. Med en stabil internettilkobling har siden en svært lav responstid. Nettsiden lastes raskt inn og er rask under bruk. Dette ble kommentert av flere testsubjekter under brukertestingen.

Kravet om at systemet skal håndtere flere personer og ikke knele under stor pågang, er sikret ved å benytte en Azure-server med gode spesifikasjoner. I tillegg har serverløsningen lav responstid.

4.2.2.4 Støtte

Kravet om testdekning er oppfylt ved å benytte xUnit for enhetstesting på serversiden og RTL på de gjenbrukbare komponentene på klientsiden. Brukergrensesnittet er testet gjennom brukertesting, for å sikre at det er brukervennlig og møter brukernes behov. Oppdragsgiver har validert systemet ved sprintgjennomganger.

Kravet om dokumentasjon er oppfylt. All funksjonalitet på klientsiden er dokumentert med TypeDoc som genererer dokumentasjon i et klikkbart HTML-format, som er oversiktlig og ryddig. Serversiden er dokumentert ved bruk av kommentarer i kildekoden. I tillegg beskriver Vedlegg E - Systemdokumentasjon systemets oppbygning og hvordan det fungerer.

4.2.2.5 Sikkerhet

Kravet om autentisering er oppfylt ved bruk av Azure AD. Systemet bruker HTTPS for å oppfylle kravet om kommunikasjon over en kryptert forbindelse. Azure er benyttet som skyløsning fordi Microsoft bruker store ressurser på å holde seg oppdatert på sikkerhetsfronten.

4.2.2.6 Vedlikehold

Oppdragsgiver ønsket at systemet skulle være modulbasert, slik at det var mulig å videreutvikle. For å sikre dette, og samtidig oppfylle SOC, er systemet oppbygd av moduler. Modulene er uavhengige, og oppfyller kravet om vedlikehold.

4.3 Administrative resultater

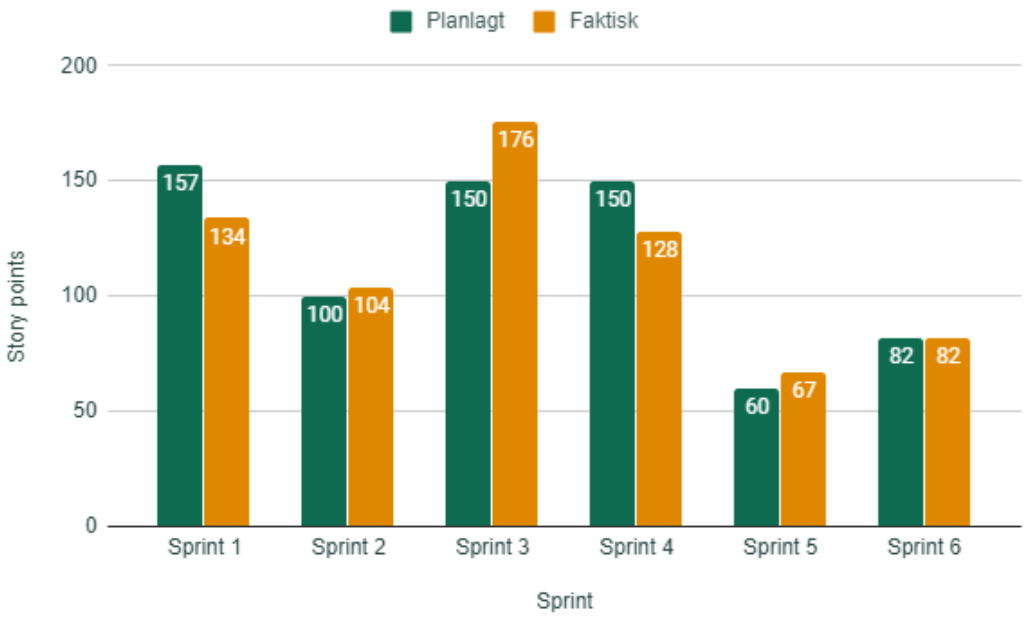
Denne delen tar for seg de administrative resultatene i prosjektet og refererer til Vedlegg D - Prosjekthåndbok. Vedlegget inneholder fremdriftsplan, dokumentasjon av utviklingsprosess, timelister med statusrapport, møteinnkallinger og møtereferat.

4.3.1 Scrum

I begynnelsen av prosjektet fungerte Kristian Winther (veileder fra NoIS) som produkteier. Denne oppgaven ble etterhvert videreført til Elias Sundby Aukan, som også fungerte som Scrum master gjennom hele prosjektet. Alle seremonier, med unntak av Scrum-møter, er dokumentert i Vedlegg D - Prosjekthåndbok, kapittel 2.

Hver sprint varte i omtrent to uker, med noe variasjon i antall arbeidsdager. Hver sprint ble planlagt med hensyn på at antall arbeidsdager varierte. Dette gjorde vi ved å skrive ned summen av antall timer hver enkelt hadde tilgjengelig i neste sprint. Summen av timer gav oss en indikasjon på hvor mange arbeidsoppgaver vi kunne velge ut. Hver oppgave i sprintkøen fikk en estimert poengsum i form av story points, basert på hvor stor arbeidsinnsats hver oppgave krevde. Den estimerte poengsummen ble satt opp i et Burndown Chart for å visualisere fremgangen gjennom sprinten. Hver sprint ble avsluttet med en sprintgjennomgang for produkteier før vi gjennomførte retrospektiv og begynte planleggingen av en ny sprint.

Under de første sprintene avholdt vi Scrum-møte kun to ganger i uken. Dette fungerte fint ettersom vi ikke hadde muligheten til å jobbe med prosjektet hver dag, og satt sammen de dagene vi jobbet. Etterhvert hadde vi kapasitet til å arbeide med prosjektet hver dag. Da så vi større nytte av å avholde Scrum-møte daglig for å synkronisere arbeidet og planlegge det neste døgnet. Dette ble spesielt nyttig da vi ikke lenger hadde mulighet til å møtes fysisk. Scrum master deltok på Scrum-møtene to ganger i uken gjennom hele prosjektperioden.



Figur 4.10: Planlagt arbeid kontra faktisk arbeid målt i story points for hver sprint.

4.3.2 Fremdriftsplan

Fremdriftsplanen i form av et GANNT-diagram ligger i Vedlegg D - Prosjekthåndbok, kapittel 1. Fremdriftsplanen for prosjektet viser planen slik den var i begynnelsen av prosjektet, med oversikt over hovedaktiviteter og milepæler.

Milepæl	**Planlagt**	**Faktisk**
Visjonsdokument	Uke 5	Uke 4
Forstudierapport	Uke 6	Uke 7
Kravdokumentasjon	Uke 6	Uke 4
Brukertesting på wireframes	Uke 7	Uke 8
Problemstilling formulert	Uke 8	Uke 15
Brukertesting på sluttprodukt	Uke 14	Uke 16
Ferdig produkt	Uke 17	Uke 18

Tabell 4.1: Oversikt over måloppnåelse av milepæler.

Fremdriftsplanen definerer åtte milepæler vi skulle oppnå gjennom prosjektet. Tabell 4.1 gir en oversikt over hvilken uke vi planla å oppnå milepælen, og når milepælen faktisk ble oppnådd.

4.3.3 Timeforbruk

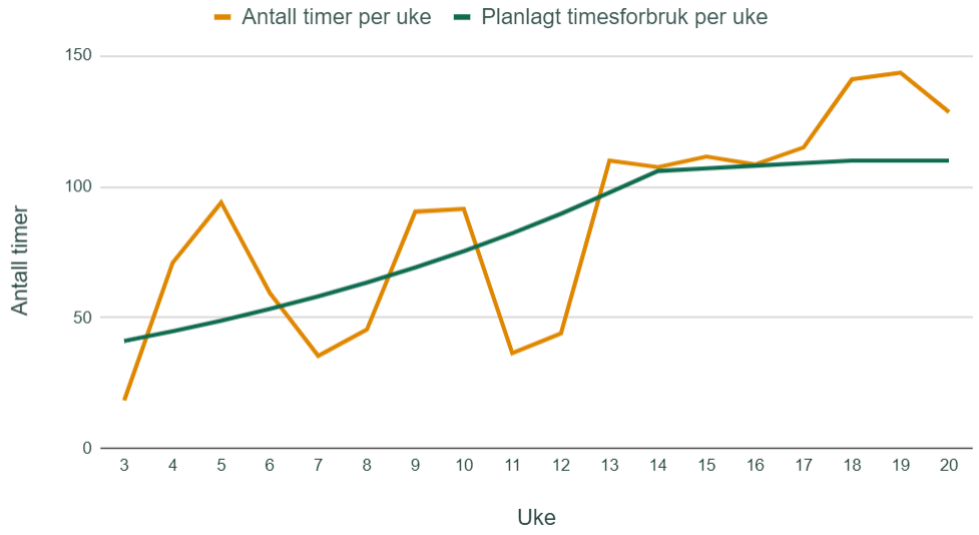
Timeforbruket gjennom prosjektet har blitt dokumentert i form av timelister i Vedlegg D - Prosjekthåndbok, kapittel 4. Timeforbruket er fordelt på aktivitetene i fremdriftsplanen. For hver uke følger en kort statusrapport om hva hver enkelt har jobbet med gjennom uken.

Differansen mellom planlagt timeforbruk per aktivitet og faktisk timeforbruk er gitt av tabell 4.2.

Aktivitet	**Planlagte timer**	**Brukte timer**	**Differanse**
Oppstart/planlegging	60	117	57
Dokumentasjon	70	75	5
Utvikling	600	582,5	-17,5
Brukertesting/undersøkelser	50	57,5	7,5
Administrativt	70	81	11
Forskning	50	41	-9
Hovedrapport	600	580	-20
Totalt	**1500**	**1552**	**52**

Tabell 4.2: Timeforbruk per aktivitet i fremdriftsplanen

Totalt timeforbruk per uke gjennom hele prosjektperioden er vist i figur 4.11.



Figur 4.11: Timeforbruk per uke kontra ideelt timeforbruk

Kapittel 5
Diskusjon

5.1 Vitenskapelige resultater

I dette delkapittelet vil vi diskutere resultatene vi kom frem til i kapittel 4.1 knyttet til problemstillingen, beskrevet i kapittel 1.2. Problemstillingen er konkretisert i to forskningsspørsmål, som vil legge grunnlaget for diskusjonen.

5.1.1 Er det fordelaktig a ta i bruk statisk typesjekking i det dynamisk typet programmeringsspråket JavaScript?

Vi innførte statisk typesjekking underveis i prosjektet, og fikk på denne måten testet programmering med både dynamisk og statisk typesjekking. Ved å manuelt gå gjennom alle filene for å konvertere de til TypeScript og typesette variablene, utviklet vi en bedre forståelse av egen kildekode. Dette gjorde det enklere å rydde opp i kildekoden og kommentere attributter, funksjoner og komponenter.

I dynamiske språk gjennomføres typesjekking under kjøring. Dette kan føre til at typefeil ikke oppdages umiddelbart. Slike feil kan forplante seg videre, og skape en mer kompleks korrigeringsprosess. Ved statisk typesjekking avdekkes typefeil under kompilering. Endringer i kildekoden vil dermed typesjekkes umiddelbart, og kan gjøre det enklere å oppdage andre feil tidlig i prosessen.

Ved å bruke TypeScript oppdaget vi at enkelte av rammeverkets funksjoner var implementert uten nødvendige avhengigheter. Siden vi ikke var klar over avhengighetene da funksjonen ble tatt i bruk, ble koden tilpasset slik at komponenten fungerte slik vi ønsket. Dette gjorde at vi ikke oppdaget problemet tidligere. Slike feil kan medføre problemer ved gjenbruk av komponenten, fordi koden kun er tilpasset én situasjon vil den ikke nødvendigvis fungere som tiltenkt i andre situasjoner.

Gjennom bruk av TypeScript får en mer funksjonalitet i IDE'en, for eksempel å gi en oversikt over hvilke argumenter en funksjon tar inn og gi beskjed dersom noe mangler. Hvis en komponent endres vil IDE'en markere det som eventuelt må endres i en annen komponent. På denne måten er det mulig å endre eller legge til funksjonalitet i komponenter uten å vite alle sammenhenger. Dette er nyttig i et

system med flere gjenbrukbare komponenter, for å sikre at alle deler av systemet fungerer etter en endring.

Ved å gå over til TypeScript underveis i prosjektet måtte vi gjøre en rekke endringer for å konvertere koden. Dette var utfordrende ettersom ingen hadde erfaring med TypeScript fra tidligere, men etterhvert som en kom mer inn i det ble prosessen mer effektiv. Det mest tidkrevende var å definere objekter med spesifikke attributter, som kunne brukes i flere komponenter. Dersom vi hadde brukt TypeScript fra begynnelsen av kunne denne tiden blitt brukt på andre oppgaver.

Når TypeScript skal sjekke om to objekter, x og y, er kompatible, ser kompilatoren på hver attributt i objekt x for å finne en tilsvarende kompatibel attributt i objekt y. Dersom alle attributter i objekt x har en tilsvarende kompatibel attributt i objekt y, vil TypeScript definere objektene som kompatible. Objekt y kan dermed inneholde flere attributter enn x, og likevel brukes i funksjoner som tar inn objekt x. Dette er *unsound* fordi kompilatoren kun tar hensyn til eksisterende attributter i målobjektet. *Unsound* tillates i denne sammenhengen for å beholde litt av fleksibiliteten til JavaScript, som skaper større frihet i datautvekslingen mellom komponenter.

5.1.2 Hvordan kan testing og statisk typesjekking minske utfordringene i et webutviklingsprosjekt med flere utviklere?

Ved bruk av statisk typesjekking ble kildekoden mer oversiktlig, som sammen med bruk av typer gjorde det enklere å sette seg inn i andres arbeid. Dette minket behovet for hjelp, og frigjorde tid slik at andre oppgaver kunne prioriteres. Utover dette sørget typesjekkingen for at eventuelle typefeil ble oppdaget før koden ble delt med de andre.

Enkelte tester kan være tidkrevende å implementere, men vil bidra til å oppdage feil tidligere. Dette fører til at feil kan rettes opp med en gang, slik at de ikke forplanter seg videre og skaper mer arbeid senere. En kan dermed spare tid i lengden ved å implementere omfattende tester.

Gjennom testing kan en sikre kvalitet på sluttproduktet, og at systemet oppfyller oppdragsgivers krav. I et prosjekt med flere utviklere er det viktig å sørge for at de ulike delene fungerer sammen, og at endringer ett sted ikke påvirker andre deler av systemet. Det er derfor viktig med omfattende tester, og prøve å få så høy testdekning som mulig. En ulempe kan være at det er vanskelig å vite hvor mye en skal

teste systemet og definisjonen av "nok tester" vil variere fra prosjekt til prosjekt.

Ulike tester dekker forskjellige aspekter av et system. Enhetstesting tester kun enhetene enkeltvis, men tester ikke samspillet seg imellom og systemet som en helhet. Ved å teste på sluttbrukere får man testet at systemet fungerer som tiltenkt i miljøet det skal brukes og at brukergrensesnittet er intuitivt. I tillegg bør en inkludere tester som sikret at kravene til oppdragsgiver dekkes. Hvilke tester en bør inkludere vil derfor variere, og må tilpasses med tanke på disponibel tid og prosjektets omfang.

5.2 Ingeniørfaglige resultater

I dette delkapittelet vil vi diskutere de ingeniørfaglige resultatene, som vi kom frem til i kapittel 4.2, og valg som førte til det ferdige produktet.

5.2.1 Funksjonelle krav

Systemet samsvarer godt med de funksjonelle kravene, beskrevet i Vedlegg A - Visjonsdokument, kapittel 5. I denne delen vil vi diskutere måloppnåelsen beskrevet i kapittel 4.2.1.

5.2.1.1 Tilgang

Det funksjonelle kravet om tilgang gjennom Azure AD er oppfylt. Oppdragsgiver kan dermed administrere tilgangen til systemet gjennom Azure-portalen, og de ansatte kan logge inn med eksisterende brukere.

5.2.1.2 Registrere fagaktivitet

De funksjonelle kravene for "Registrering" er delvis oppfylt. Det er mulig å registrere en fagaktivitet med tilhørende informasjon gjennom et skjema med flere steg. Skjemaet er delt opp etter ønske fra oppdragsgiver, for å sikre at all nødvendig informasjon blir fylt inn. Kravet om at hvert felt i registreringen skulle ha et informasjonsfelt, som forklarer hva feltene skulle inneholde, er ikke oppfylt. Etter avtale med oppdragsgiver ble vi enige om at informasjonsfelt kun var nødvendig på feltene knyttet til kostnader.

5.2.1.3 Profil

De funksjonelle kravene for "Profil" er oppfylt. Profiler var ikke en del av det tidligere systemet og ble inkludert etter ønske fra de ansatte. Ved bruk av profiler får de

oversikt over egne fagaktiviteter, og muligheten til å se hvilke fagaktiviteter andre ansatte har registrert.

5.2.1.4 Synkronisere fagaktiviteter med CV Partner

De funksjonelle kravene knyttet til synkronisering med CV Partner er oppfylt. NoIS bruker CV Partner for å samle CV-ene til de ansatte. De ansatte er selv ansvarlige for å holde CV-en oppdatert, med alle fagaktiviteter og prosjekter. Ved å integrere CV partner i systemet, reduserer vi arbeidet de ansatte må gjøre. Samtidig sikrer dette at systemet og CV-ene alltid er synkroniserte.

5.2.1.5 Fagaktiviteter

De funksjonelle kravene for "Fagaktiviteter" er delvis oppfylt. Det er ikke mulig å se en oversikt over de mest populære fagaktivitetene i en periode. Topplisten er kun mulig å sortere etter totalt antall gjennomføringer eller rangering. Dette ble nedprioritert, for å inkludere funksjonalitet de ansatte savnet under brukertesting. Resterende krav er oppfylt.

5.2.1.6 Personer

De funksjonelle kravene for "Personer" er oppfylt. Person-siden gjør det mulig å finne andre ansatte, slik at de ansatte kan besøke hverandres profil. For å finne en bestemt person, kan en søke eller filtrere på avdeling.

5.2.1.7 Kompetanseoversikt

De funksjonelle kravene knyttet til kompetanseoversikten er oppfylt. Kompetanseoversikt var i utgangspunktet kun tiltenkt administratorbrukere, men etterhvert viste flere av de ansatte interesse for en slik oversikt. Ettersom all informasjonen i oversikten var tilgjengelig i andre deler av systemet, ble det bestemt at kompetanseoversikten skulle være åpen for alle ansatte.

Denne oversikten kan brukes for å finne ansatte med en spesfikk kompetanse til eksterne oppdrag. For å utvide funksjonaliteten til oversikten implementerte vi *tags*, for å beskrive hver fagaktivitet. Disse kan brukes som søkeord, i tillegg til fagaktivitetsnavn.

5.2.1.8 Arrangement

Kravet om oversikt over kommende arrangementer er oppfylt, men oversikten er ikke tilgjengelig på mindre skjermer. Under brukertesting viste det seg at kalenderen var

en overflødig funksjonalitet, og de fleste foretrakk å bruke andre kalendere. Sammen med oppdragsgiver ble vi enige om å beholde siden, men ikke utvide funksjonaliteten.

5.2.1.9 Publisering til Slack

Det funksjonelle kravet om varslinger er oppfylt. For hver oppdatering i systemet vil det sendes en varsling i Slack, og det er ingen innstillinger for å endre hva som skal sendes. Kravet spesifiserte ikke hvilke statusoppdateringer det skulle varsles om, men med mer tid ville vi inkludert en mulighet for å tilpasse varslinger.

5.2.1.10 Administrator

De funksjonelle kravene for administratorer er oppfylt. Kostnadskontroll var en av de største manglene ved det tidligere systemet, og kravet om nedlasting av CSV-fil var derfor viktig for oppdragsgiveren. Administratorsiden har fått en rekke filtreringsmuligheter, slik at han kan laste ned en CSV-fil med innholdet han ønsker.

En administratorbruker må godkjenne alle fagaktiviteter som registreres. På denne måten beholdes oversikten over hva de ansatte planlegger, og sørge for at de ansatte gjennomfører nyttige fagaktiviteter. Dersom en fagaktivitet ikke kan godkjennes, blir den avslått.

5.2.2 Ikke-funksjonelle krav

I denne delen vil vi omtale hvorfor systemet tilfredsstiller de ikke-funksjonelle kravene, beskrevet i Vedlegg A - Visjonsdokument, kapittel 6.

5.2.2.1 Brukbarhet

I visjonsdokumentet er det stilt krav til produktets brukbarhet: Systemet skal være intuitivt og enkelt å navigere i, brukes uten opplæring og sikre at brukerne ikke kan fylle inn feil datatype i inntastingsfelter. I tillegg skulle brukergrensesnittet i hovedsak tilpasses PC-skjermer, men mulig å bruke på andre enheter.

Som nevnt i kapittel 4.2.2.1 er det gjennomført brukertester for å sikre at systemet er intuitivt og at det er enkelt å navigere mellom de ulike sidene. Systemet skal kunne brukes uten opplæring, men i noen tilfeller kan det være nødvendig med hjelp. Det er derfor opprettet en brukerhåndbok (Vedlegg F - Brukerhåndbok). For å forebygge brukerfeil, blir det utført feltkontroll på inntastingsfeltene.

Som nevnt i kapittel 5.2.1.8 var interessen rundt en arrangement-side lav. Ettersom arrangement-siden ikke var brukervennlig på mobil, foreslo produkteier at siden

kunne deaktiveres på mobil.

5.2.2.2 Funksjonell egnethet

I visjonsdokumentet er det stilt krav til produktets egnethet: Alle brukerbehovene med høy prioritet skal være implementert, systemet skal hente korrekt data på alle spørringer og håndtere feilmeldinger. Som nevnt i kapittel 4.2.2.2 er alle brukerbehovene implementert, med unntak av mørk modus. For å kontrollere at systemet tilfredsstilte de ansattes behov, ble det gjennomført brukertester i flere omganger.

For å sikre kravet om at systemet skal hente korrekt data, og håndtere feil, er det innført enhetstester på alle endepunktene på serversiden. Testene blir kjørt før ny versjon utrulles, for å hindre publisering av eventuelle feil. Det er lagt inn sjekker for eventuelle feilmeldinger slik at disse blir håndtert riktig.

5.2.2.3 Ytelse

I visjonsdokumentet er det stilt krav til produktets ytelse: Data skal hentes umiddelbart, og systemet skal håndtere stor pågang. Som nevnt i kapittel 4.2.2.3 kommenterte flere av de ansatte at systemet responderte raskt under bruk. Dette har vi oppnådd ved å kun laste inn nødvendig data og kun oppdatere den aktuelle komponenten. I tillegg brukte vi en serverløsning fra Azure med lav responstid. Serverløsningen kan skaleres i senere tid, for å takle stor pågang uten å knele.

5.2.2.4 Støtte

I visjonsdokumentet er det stilt krav til produktets støtte: Det skal være god testdekning, og systemet skal være godt dokumentert. For å teste systemet har vi implementert enhetstesting på serversiden, for å sikre at riktig informasjon hentes fra databasen. De gjenbruke komponentene på klientsiden er testet for å sikre at de fungerer som tiltenkt. I tillegg lot vi sluttbrukerne klikke seg fritt rundt i systemet, for å avdekke eventuelle svakheter i brukergrensesnittet.

Hele systemet er godt dokumentert med kommentarer, som forklarer hver funksjon og klasse. På større, avanserte funksjoner er det lagt inn ekstra kommentarer.

5.2.2.5 Sikkerhet

I visjonsdokumentet er det stilt krav til sikkerhet: Kun ansatte hos NoIS skal ha tilgang til systemet, data skal sendes over en kryptert forbindelse, og det skal benyttes en skyløsning.

Kapittel 4.2.2.5 viser at kravene er oppfylt. Ved bruk av skytjenestene Azure AD og Azure Portal styres tilgangen til systemet. Applikasjonen kjører på en HTTPS-forbindelse, som krypterer dataene som blir sendt mellom server og klient.

5.2.2.6 Vedlikehold

I visjonsdokumentet er det stilt krav om et system som kan videreutvikles og er enkelt å vedlikeholde. Systemet er bygget opp av komponenter og moduler, slik at delene kan oppdateres og endres uavhengig av hverandre. I tillegg er systemet godt dokumentert, som kan gjøre det enklere å sette seg inn i, se kapittel 4.2.2.4.

5.2.2.7 Styrker med produktet

Systemet bruker skytjenester fra Microsoft Azure, slik at minnekapasiteten og prosessorkraften kan skaleres i senere tid. I tillegg benytter Azure maskinlæring til å foreslå hvordan systemet burde skaleres for optimal ytelse. Gjennom Azure-portalen kan en administrere tilgangen til systemet, slik at en kan legge til og fjerne brukere.

Produktet har tilleggsfunksjonalitet utover de funksjonelle kravene beskrevet Vedlegg A - Visjonsdokument. Vi innførte blant annet *tags*, se kapittel 5.2.1.7, og kommentarfelt på hver fagaktivitet etter ønsker fra flere ansatte. Ansatte som har gjennomført en fagaktivitet får mulighet til å legge ved en kort kommentar om hvordan fagaktiviteten var, eller nyttige tips til andre. Denne kommentaren blir da tilgjengelig for de andre ansatte, sammen med rangeringen som ble gitt.

Gjennom hele prosjektperioden har vi hatt et tett samarbeid med sluttbrukerne (ansatte hos NoIS), for å sikre at systemet møter deres behov. For å sikre at kravene fra oppdragsgiver og produkteier møtes har vi hatt regelmessige møter med statusoppdatering og demonstrasjon av foreløpig resultat.

5.2.2.8 Svakheter med produktet

For å hente og legge til informasjon i CV Partner, benyttet vi oss av deres API. Dette gjør at funksjonaliteter i systemet er avhengig av noe vi ikke har kontroll over. Dersom noen av endepunktene i API-et endres, vil ikke lenger synkroniseringen med CV Partner fungere, og systemet må oppdateres.

Tidlig i prosjektperioden ble det fastslått at alle ansatte skulle se informasjon til de andre ansatte. Det er derfor ikke investert like mye tid på sikkerhet innad i systemet, og fokuset er heller rettet mot at ingen uvedkommende skal få tilgang til systemet.

En funksjonalitet de ansatte savnet ved det tidligere systemet var at informasjonen

om en fagaktivitet automatisk ble utfylt under registrering. Dette er heller ikke implementert i vårt system, ettersom informasjonen som skal legges inn er knyttet til hver bruker og ville i flere tilfeller blitt feil for andre.

5.2.2.9 Valg av teknologi

Valg av teknologi har vært avgjørende for systemet som ble laget, og det var viktig at vi valgte teknologi som gjorde systemet rustet for videre utvikling. For klientsiden brukte vi først React med JavaScript, men gikk over til TypeScript for å innføre statisk typesjekking. Dette var en fordel både for oss, se 5.2, og for de som skal videreutvikle systemet. Vi brukte komponentrammeverket Material UI for å få et ryddig og oversiktlig design. Ved å bruke ferdige komponenter, med egne tilpasninger, kunne vi ha større fokus på funksjonaliteten til systemet.

For serversiden brukte vi ASP.NET Core, hvor programmeringsspråket C# benyttes. Til tross for at ingen i gruppen hadde tidligere kjennskap til ASP.NET Core eller C#, fikk vi satt opp prosjektstrukturen raskt og kom fort i gang med utviklingen. For å kommunisere med databasen brukte vi EF Core, som benytter LINQ i stedet for SQL, som vi har brukt tidligere. Vi brukte en del tid på å finne ut hvordan dette fungerte, men når vi først lærte det gikk det raskt å lage nye spørringer.

Skytjenester fra Microsoft Azure er byttet for å *hoste* databasen, klientsiden og serversiden, og autentisering. Alle tjenestene er godt dokumentert, og dermed greie å sette seg inn i.

Vi er fornøyde med våre teknologivalg. Til tross for at mye tid gikk med på å sette seg inn i ny teknologi, har vi lært mye nytt og føler at det lønnet seg totalt sett. Alle teknologivalg har hjulpet oss å fokusere på å utvikle produktet oppdragsgiver ønsket.

5.2.2.10 Profesjonsetiske problemstillinger

Systemet vil inneholde sensitiv informasjon, kun ment for de ansatte i bedriften. Som dataingeniører må vi sørge for at tilgangen til systemet er begrenset, slik at informasjonen ikke havner i feil hender.

For å lage det ønskede systemet, var vi avhengige av tilganger til sensitive ressurser i bedriften. Dette inkluderer tilgang til både sensitiv informasjon og økonomiske privilegier. Som dataingeniører har vi et ansvar ovenfor oppdragsgiver å sikre at informasjonen holdes privat og at privilegiene ikke blir misbrukt.

5.3 Administrative resultater

I dette kapittelet vil vi diskutere de administrative resultatene vi kom frem til i kapittel 4.3.

5.3.1 Scrum

Vi valgte å gjennomføre prosjektet ved bruk av den iterative utviklingsmetodikken Scrum. Dette fungerte bra ettersom produkteier ikke hadde noen spesifikke krav ved prosjektets start. Etter samtale med oppdragsgiver og inspeksjon av det eksisterende systemet fikk vi formulert de funksjonelle kravene beskrevet i Vedlegg A - Visjonsdokument, kapittel 5. I Scrum er det vanligvis produkteier som styrer produktkøen, men i dette prosjektet hadde vi kontroll og kunne endre produktkøen ved behov. Alle elementene i produktkøen var basert på de funksjonelle kravene som produkteier hadde godkjent, og vi hadde regelmessige møter for å sikre at vi var på riktig vei.

I kapittel 4.3.1 er det en oversikt over hvor vellykket de ulike sprintene var. Ettersom vi tok i bruk ny teknologi og flere av oppgavene var nye for alle i gruppen, ble det vanskelig å gi nøyaktige estimater. Noen oppgaver var mer komplisert, og tok lengre tid enn først antatt, mens andre tok mindre tid enn forventet. Til tross for en del feilestimering, ble flertallet av sprintene suksessfulle, og ekstra funksjonalitet ble implementert.

Det var en fordel å jobbe iterativt, fordi det ga frihet til å tilrettelegge arbeidsoppgaver ved behov og planlegge underveis, fremfor å følge en fast plan. I tillegg hjalp scrummetodikken oss å jobbe strukturert, planlegge og holde oversikt over arbeidet. Gjennom bruk av Trello, kunne vi til enhver tid se hva de andre arbeidet med og gjenværende arbeidsoppgaver.

Gjennom hele prosjektperioden hadde vi god kommunikasjon med representantene fra NoIS. Rollene som produkteier og Scrum master ble etterhvert fylt av samme person. Til tross for at dette ikke er vanlig praksis, ble det en fordel for oss siden produkteier/Scrum master alltid var oppdatert. Vi fikk dermed hyppigere og mer presise tilbakemeldinger på både prosess og produkt.

5.3.2 Fremdriftsplan

Fremdriftsplanen var et nyttig verktøy for å måle fremgangen og holde oversikt. På grunn av manglende oversikt i starten av prosjektet, var det vanskelig å sette opp et detaljert GANNT-diagram. Vi valgte derfor å lage en mer overordnet fremdriftsplan, med de viktigste milepælene og fasene i prosjektet.

Som vist i kapittel 4.3.2 var det en del avvik fra planen. Avvikene kan skyldes at flere av oppgavene var mer tidkrevende enn først antatt, og kan være en indikasjon på at målene var litt for optimistiske. Ettersom de ulike aktivitetene i fremdriftsplanen var uavhengige av hverandre, kunne vi arbeide med aktivitetene parallelt. Forsinkelser i én aktivitet førte dermed ikke til forsinkelser i andre aktiviteter.

5.3.3 Timeforbruk

Gjennom hele prosjektperioden har vi prøvd å ha et jevnt timeforbruk og overholde den planlagte timebruken. Oppstartsperioden var mer tidkrevende enn forventet, og vi brukte nesten dobbelt så mye tid som planlagt på aktiviteten "planlegging". Oppstartsaktivitetene gikk ut på å planlegge hvordan videre arbeid skulle gjennomføres og sette seg inn i ny teknologi. Til tross for dette, økte ikke det totale timeforbruket nevneverdig. Tidsbruken på de resterende aktivitetene var omtrent som forventet, og det totale tidsforbruket ble kun 57 timer mer enn planlagt.

I begynnelsen av prosjektperioden ble det lagt ned færre arbeidstimer i uken, grunnet parallell undervisning. Dette tilsvarte omtrent tre fulle arbeidsdager per uke. Etter fullføring av undervisning fokuserte vi kun på prosjektet. Antall arbeidsdager økte dermed til fem dager per uke, med noen avvik grunnet reiser og fridager. Mot slutten av prosjektet arbeidet vi også noen helger, som resulterte i økt timeforbruk per uke. Med et lavere timeforbruk i starten og høyere mot slutten endte det totale timetallet omtrent som planlagt.

5.3.4 Gruppearbeid

Gruppearbeidet har vært effektivt og fungert godt, gjennom hele prosjektperioden. Kommunikasjonen har vært god, alle hjalp hverandre der det var nødvendig og det var lav terskel for å spørre om hjelp. Det var minimalt med uenigheter, og de som oppstod ble løst raskt gjennom diskusjon.

Ved prosjektstart ble vi enige om en flat struktur, uten en bestemt arbeidsfordeling. Selv om alle i gruppen bidro på alle arbeidsområder, oppstod det naturlige ansvarsområder ettersom kompetansen spisset seg. Sara Hjelle fikk hovedansvaret for det

administrative og utformingen av klientsiden, Ådne Eide Stavseng for integrasjon av kontinuerlig utrulling, og Stian Ådnanes tok for seg utformingen av serversiden. Alle har prøvd ny teknologi, og utviklet sin kompetanse gjennom oppgaver de syntes var spennende.

Kapittel 6
Konklusjon og videre arbeid

6.1 Konklusjon

Hovedproblemstillingen for rapporten er et åpent spørsmål som det ikke er mulig å dekke fullstendig gjennom en bacheloroppgave. Basert på dette har vi valgt å snevre inn oppgaven ved å definere to forskningsspørsmål, som fokuserer på effekten av statisk typesjekking og testing i et webutviklingsprosjekt med flere utviklere.

1. *Er det fordelaktig å ta i bruk statisk typesjekking i det dynamisk typede programmeringsspråket JavaScript?*

   Innføringen av statisk typesjekking førte til en mer oversiktlig kode, en tryggere overordnet prosess og enklere individuelt arbeid. Ved å bruke statisk typesjekking fra prosjektets begynnelse, kan en få bedre utnyttelse av fordelene det medfører. I tillegg slipper en å konvertere arbeidet, slik en må dersom statisk typesjekking innføres underveis. Fordelene innebærer at feil oppdages tidligere i prosessen, en får bedre oversikt og typesikker kode, men statisk typesjekking kan fjerne noe av fleksibiliteten til JavaScript. Til tross for tiden brukt på å typesette kildekode i etterkant, førte innføring av statisk typesjekk totalt sett til en mer effektiv utviklingsprosess.

2. *Hvordan kan testing og statisk typesjekking minske utfordringene i et webutviklingsprosjekt med flere utviklere?*

   Vi erfarte at statisk typesjekking kan forenkle det individuelle arbeidet, ved å gjøre det enklere å sette seg inn i andres kildekode. I tillegg sikret det en typesikker kode før den ble tilgjengeliggjort for andre. Testing av kildekoden sikret at funksjonaliteten ble ivaretatt ved endringer, og feil ble oppdaget i en tidlig fase.

Utbyttet av både statisk typesjekking og testing vil øke i takt med grad av utnyttelse. Det vil si at ved å innføre tiltakene tidlig i prosessen, vil utbyttet øke fordi en får utnyttet fordelene over en lengre periode. Ut fra våre resultater kan vi fastslå at statisk typesjekking og testing vil minske utfordringene i webutviklingsprosjekt med flere utviklere. Det er også grunnlag for å anta at fordelene vil øke i takt med størrelsen på prosjektet og antall utviklere.

Det ferdigstilte produktet oppfyller alle kravene fra oppdragsgiver, med noen få unntak som ble avklart underveis. Vi kan dermed konkludere med et vellykket prosjekt, sett fra et ingeniørfaglig perspektiv.

6.2 Videre arbeid

Selv om det ferdigstilte systemet oppfyller alle kravene, er det mer funksjonalitet vi ville implementert dersom systemet skulle videreutvikles.

I systemet får de ansatte ingen form for belønning for fagaktivitetene de fullfører, uavhengig av vanskelighetsgrad. Ved å gi de ansatte en poengsum eller annen belønning ut fra vanskelighetsgraden, kan dette engasjere og motivere de ansatte til å gjennomføre flere fagaktiviteter.

Muligheten for mørk modus i systemet ble nevnt av et par ansatte under prosjektperioden. Dette ble nedprioritert fordi det finnes tilleggsfunksjoner i noen nettlesere som gjør det mulig å aktivere mørk modus. Denne løsningen var ikke optimal, men fungerte for de som ønsket mørkt modus. Ved videre utvikling kan mørk modus implementeres i systemet, slik at det er optimalisert for designet.

I kapittel 5.2.2.8 er det nevnt at automatisk utfylling av informasjon om en fagaktivitet ikke er implementert. Ved å inkludere dette på en god måte, kan en spare de ansatte for tid og sørge for at administrator fortsatt får all nødvendig informasjon.

Det er laget en kalender med oversikt over kommende arrangementer, men utover dette har den begrenset funksjonalitet. For at kalenderen skulle vært nyttigere for bedriften kunne en utvidet funksjonaliteten. Noe av det viktigste ville vært å tilpasse den til mindre skjermer, og gjøre det mulig å melde seg på arrangementer via kalenderen.

Kilder

[1] Else Leirvik og Vegard B. Havdal. «Programmering i Java. 4. utgave.» I: Trondheim: Gyldendal Akademisk og Stiftelsen TISIP, 2009, s. 59.

[2] Panagiotis Vekris. «Precise Type Checking for JavaScript». I: *Precise Type Checking for JavaScript*. UC San Diego Electronic Theses and Dissertations. University of California, San Diego: Association for Computing Machinery, 2017, s. 2–14. URL: https://escholarship.org/uc/item/49c5363t.

[3] *Lecture 39: soundness and completeness*. [Internett] http://www.cs.cornell.edu/ [hentet 13. mai 2020]. Tilgjengelig på: http://www.cs.cornell.edu/courses/cs2800/2016sp/lectures/lec39-sound-complete.html.

[4] *Types Expressions*. [Internett] https://www.typescriptlang.org/ [hentet 13. mai 2020]. Tilgjengelig på: https://flow.org/en/docs/lang/types-and-expressions/.

[5] *Type Compatibility*. [Internett] https://www.typescriptlang.org/ [hentet 13. mai 2020]. Tilgjengelig på: https://www.typescriptlang.org/docs/handbook/type-compatibility.html.

[6] *Strict checks*. [Internett] https://www.typescriptlang.org/ [hentet 18. mai 2020]. Tilgjengelig på: https://www.typescriptlang.org/tsconfig#Strict_Type_Checking_Options_6173.

[7] *Acceptance testing*. [Internett] http://softwaretestingfundamentals.com/ [hentet 06. mai 2020]. Tilgjengelig på: http://softwaretestingfundamentals.com/acceptance-testing/.

[8] *Unit Testing*. [Internett] http://softwaretestingfundamentals.com/ [hentet 16. april 2020]. Tilgjengelig på: http://softwaretestingfundamentals.com/unit-testing/.

[9] *Usability Testing*. [Internett] https://www.interaction-design.org/ [hentet 17. april 2020]. Tilgjengelig på: https://www.interaction-design.org/literature/topics/usability-testing.

[10] Sen-Tarng Lai. Applying Continuous Integration for Increasing the Maintenance Quality and Efficiency of Web App. *International Journal of Software Engineering & Applications*. 2019; 10: s. 37–50.

[11] *Guiding Principles of REST*. [Internett] https://restfulapi.net/ [hentet 8. april 2020]. Tilgjengelig på: https://restfulapi.net/.

[12] Elizabeth J. O'Neil. «Object/Relational Mapping 2008: Hibernate and the Entity Data Model (Edm)». I: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. SIGMOD '08. Vancouver, Canada: Association for Computing Machinery, 2008, s. 1351–1352. URL: https://doi.org/10.1145/1376616.1376773.

[13] *Interaction Design*. [Internett] interaction-design.org [hentet 5. mai 2020]. Tilgjengelig på: https://www.interaction-design.org/literature/topics/interaction-design.

[14] *10 Usability Heuristics for User Interface Design*. [Internett] https://www.nngroup.com/ [hentet 05. mai 2020]. Tilgjengelig på: https://www.nngroup.com/articles/ten-usability-heuristics/.

[15] Jacob Krüger. «Separation of Concerns: Experiences of the Crowd». I: *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. SAC '18. Pau, France: Association for Computing Machinery, 2018, s. 2076–2077. ISBN: 9781450351911. DOI: 10.1145/3167132.3167458. URL: https://doi.org/10.1145/3167132.3167458.

[16] *Hva styrer valg av metodikk for design og utvikling av digitale systemer?* [Internett] https://www.knowit.no/ [hentet 13. april 2020]. Tilgjengelig på: https://www.knowit.no/tjenester/experience/strategi-og-digitalisering/knowitsquartely-take-on-tomorrow/hva-styrer-valg-av-metodikk-for-design-og-utvikling-av-digitale-systemer/.

[17] *What is scrum?* [Internett] Scrum.org [hentet 6. april 2020]. Tilgjengelig på: https://www.scrum.org/index.php/resources/what-is-scrum.

[18] *Heuristikk, Store norske leksikon*. [Internett] snl.no [hentet 6. april 2020]. Tilgjengelig på: https://snl.no/heuristikk.

[19] Leo Adell. *Benefits and Disadvantages of Scrum Methodology in Software Development*. [Internett] Belatrix Software [hentet 13. april 2020]. Tilgjengelig på: https://www.belatrixsf.com/blog/benefits-scrum-software-development.

[20] *Hva er Azure*. [Internett] https://azure.microsoft.com/ [hentet 7. april 2020]. Tilgjengelig på: https://azure.microsoft.com/nb-no/overview/what-is-azure/.

[21] *What is the Azure SQL Database service?* [Internett]https://docs.microsoft.com [hentet 28. april 2020]. Tilgjengelig på: https://docs.microsoft.com/en-us/azure/sql-database/sql-database-technical-overview.

[22] *What is GitHub?* [Internett]https://docs.microsoft.com [hentet 2. mai 2020]. Tilgjengelig på: https://www.w3schools.com/whatis/whatis_github.asp.

[23] *ASP.NET Documentation*. [Internett] docs.microsoft.com [hentet 7. april 2020]. Tilgjengelig på: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-3.1.

[24] *Entity Framework Core*. [Internett] docs.microsoft.com [hentet 7. april 2020]. Tilgjengelig på: https://docs.microsoft.com/en-us/ef/core/.

[25] *Unit testing C in .NET Core using dotnet test and xUnit*. [Internett] https://docs.microsoft.com/ [hentet 8. april 2020]. Tilgjengelig på: https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-dotnet-test.

[26] *React*. [Internett] https://reactjs.org/ [hentet 7. april 2020]. Tilgjengelig på: https://reactjs.org/.

[27] *Rect Testing Library*. [Internett] https://testing-library.com/ [hentet 18. mai 2020]. Tilgjengelig på: https://testing-library.com/docs/react-testing-library/intro.

[28] *ESLint*. [Internett] https://eslint.org/ [hentet 06. mai 2020]. Tilgjengelig på: https://eslint.org/docs/about/.

[29] *What is Babel*. [Internett] https://babeljs.io/ [hentet 06. mai 2020]. Tilgjengelig på: https://babeljs.io/docs/en/.

Vedlegg

Vedlegg A: Visjonsdokument

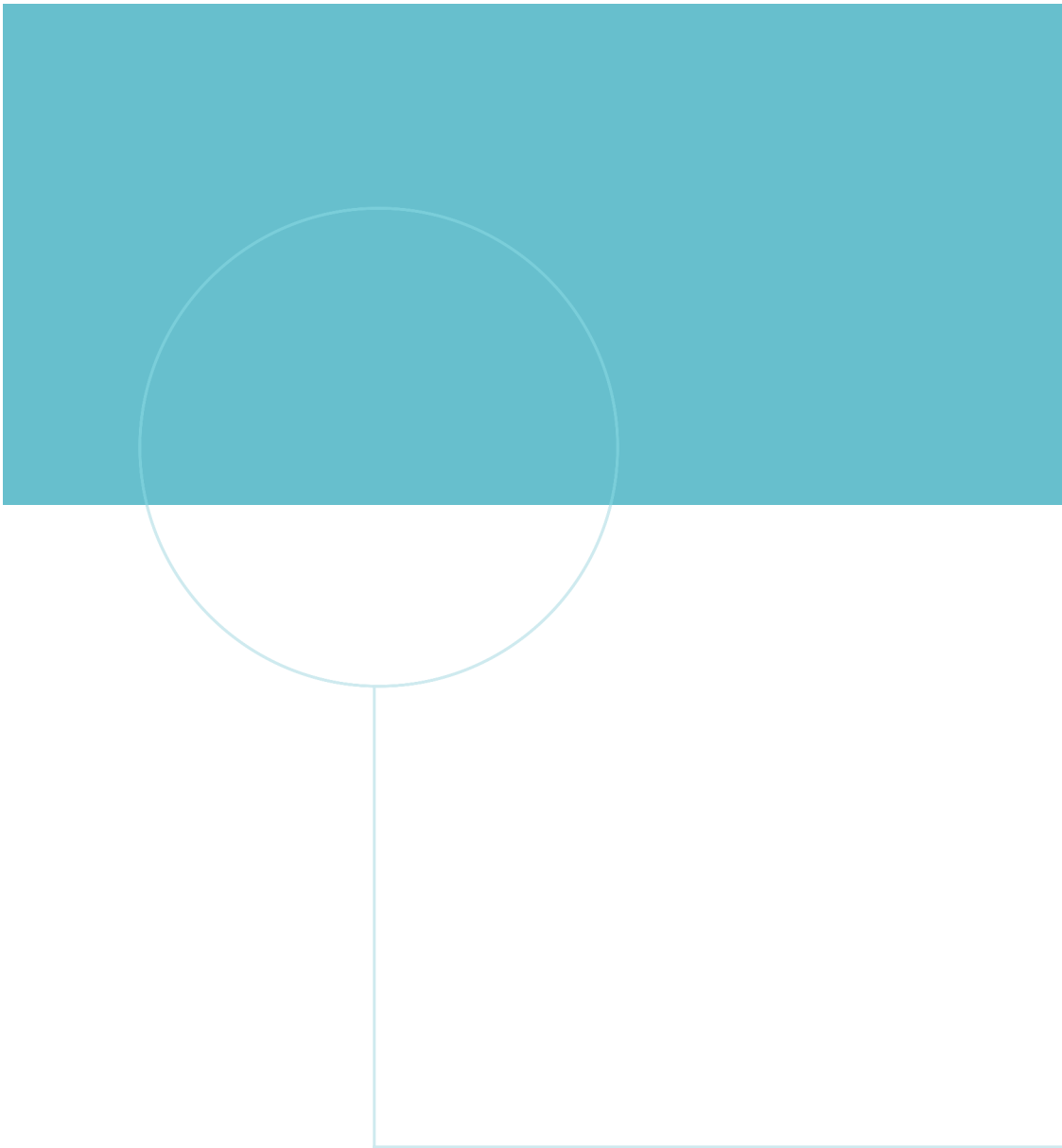
Vedlegg B: Kravdokumentasjon

Vedlegg C: Forstudierapport

Vedlegg D: Prosjekthåndbok

Vedlegg E: Systemdokumentasjon

Vedlegg F: Brukerhåndbok



NTNU

Kunnskap for en bedre verden