Høgskulen på Vestlandet

BACHELOROPPGAVE

Medlemsdatabase System

Member Database System

Ørjan Enes - 180337

Kjetil Moen Hunshammer - 182719

Dataingeniør

Fakultet for ingeniør- og naturvitenskap

Institutt for datateknologi, elektroteknologi og realfag

Veileder for oppgaven Tosin Daniel Oyetoyan

Innleveringsdato 02.06.2020

Rapportens tittel: Medlemsdatabase System	*Dato: 02.06.2020*
Forfatter(e): Kjetil Moen Hunshammer, Ørjan Enes	*Antall sider u/vedlegg: 58*
	Antall sider vedlegg: 6
Studieretning: Dataingeniør	*Antall disketter/CD-er:* 0
Kontaktperson ved studieretning: Tosin Daniel Oyetoyan	*Gradering:* Ingen
Merknader: Ingen	

Oppdragsgiver: Det Akademiske Kvarter	*Oppdragsgivers referanse:* Ingen
Oppdragsgivers kontaktperson: Khiem Tran, Oda Bjerkan	*Telefon:* 95804967

Sammendrag:

Målet for prosjektet er å utvikle et system som har alle medlemmene, gruppene og organisasjonene til "Det Akademiske Kvarter". Med dette systemet skal de kunne håndtere alle medlemmer sine ferdigheter, kontaktinformasjon og annen personaldata. Systemet skal være robust, fleksibelt og skalerbart.

The goal for our project is to develop a system containing the members, groups, and organizations within "Det Akademiske Kvarter". With this system, they should be able to manage all their member's abilities, contact information, and other personal data. The system should be robust, flexible, and scalable.

Stikkord:

React	.NET Core 3.1	MobX

Høgskulen på Vestlandet, Fakultet for ingeniør- og naturvitskap

Postadresse: Postboks 7030, 5020 BERGEN Besøksadresse: Inndalsveien 28, Bergen

Tlf. 55 58 75 00 Fax 55 58 77 90 E-post: post@hvl.no Hjemmeside: http://www.hvl.no

FORORD

Bacheloroppgaven “P20 - Personaldatabase” er skrevet av Kjetil Moen Hunshammer og Ørjan Enes våren 2020. Dette er den avsluttende oppgaven på bachelorstudiet Dataingeniør ved Høgskulen på Vestlandet, avdeling Bergen.

Oppgaven går ut på å lage et nytt “personaldatabase”-system for “Det Akademiske Kvarter”. Systemet skal holde styr på alle organisasjoner, grupper og frivillige under “Kvarteret”. Dette systemet skal holde orden på kompetansen og vervene til alle medlemmer under hver organisasjon, og medlemmene skal kunne oppdatere sin egen kontaktinformasjon og kompetanse.

Vi vil takke Tosin Daniel Oyetoyan for god veiledning og oppfølging gjennom hele prosjektet. Vi har fått mange gode tilbakemeldinger på både prosjektet og oppgaven, og lært mye om hvordan vi jobber i større prosjekt.

Vi vil også takke venner og bekjente som har vært behjelpelige som testobjekter i testfasen av applikasjonen.

Akronymer

CRUD	Create, Read, Update, Delete
API	Application Programming Interface
TFRP	Transparently applying Functional Reactive Programming
DOM	Document Object Model
HVL	Høgskolen på Vestlandet
JSX	JavaScript XML
HTTP	Hyper Text Transfer Protocol
MVC	Model View Controller
SPA	Single Page Application
REST	Representational State Transfer
JWT	JSON Web Token
HTML	Hyper Text Markup Language
CSS	Cascading Style Sheets
Js	JavaScript
Ts	TypeScript
IDE	Integrated Development Environment
UML	Unified Modeling Language
NPM	Node Package Manager

Ordliste

API	Definerte metoder for kommunikasjon mellom en service og andre programvarer
Backend	Databasen og serversiden av en applikasjon
Brukergrensesnitt/Frontend	Delen av applikasjonen som vises for sluttbrukeren
State	Object som er håndtert innad i React
REST	Et sett av definerte begrensninger som brukes for å utvikle web-tjenester, basert på HTTP
RESTful API	Web service basert på REST arkitektur
Applikasjon	Et server-klient program som kan kjøre i browseren
React	JavaScript rammeverk for utvikling av brukergrensesnitt
Rammeverk	En plattform med definerte funksjoner, utviklere kan bruke for å utvikle applikasjoner
Klient	Brukergrensesnittet av applikasjonen
Database	En organisert samling av data, strukturert i tabeller for lagring av data

Innholdsfortegnelse

1 INNLEDNING

I dette kapittelet vil det bli gitt en introduksjon for bakgrunnen til dette prosjektet. Mål og motivasjon, kontekst, begrensninger og ressurser er det som kommer til å bli diskutert.

1 Motivasjon og mål

Prosjektoppgaven ble tildelt gruppen gjennom HVL Bergen og ble valgt av gruppen på bakgrunn av at de ønsket å utvikle noe webbasert.

Prosjektoppgaven har Det Akademiske Kvarteret i Bergen som sin oppdragsgiver. Det Akademiske Kvarter er kulturhuset for studenter i Bergen og har som mål og gjøre studenthverdagen til studenter i byen bedre. Kvarteret ble åpnet av Kronprins Haakon 22.februar 1995, og feirer med det 25 år i år, 2020. Selve huset består av tre etasjer som huser i overkant av 2000 arrangement hvert år. I tillegg til kafe og pub er det konsertrom, møterom og en rekke oppholdssteder for de besøkende rundt om på huset. Huset drifters som regel av frivillige studenter i Bergen, hvor de som gjengjeld får et stort sosialt nettverk, interne goder og gratis tilgang til flere av arrangementene på huset.

1.1.2 Problemstilling

Oppdragsgiver for denne bacheloroppgaven har per dags dato et "personaldatabase"-system de ønsker å oppgradere med mer funksjonalitet samt gjøre systemet raskere. Systemet i dag har liten mulighet for å gruppere personer ut fra hvilken organisasjon og gruppe de tilhører, noe som medfører at systemet er tregt og uoversiktlig å bruke. Det er også begrenset med muligheter for å gjøre spesifikke søk i databasen og det ønskes at brukere skal kunne tagges med stikkord, man kan gjøre søk på. Det skal også være mulighet for å hente ut data fra databasen, slik at dette videre kan brukes i andre databehandlingsprogrammer, noe som ikke er mulig i dag.

Dagens system har ikke tilstrekkelig tilgangsstyring. Dette medfører at det er en omfattende jobb for administratorene å holde persondata oppdatert til enhver tid. Her ønskes det at det skal være mulig å opprette ulike nivåer med ulik tilgang til systemet.

Serversiden til dagens system er ikke fleksibelt nok, og det har ikke mulighet for fleksible API-kall. Dette er hovedårsaken til at oppdragsgiver opplever dagens system som tregt. Det er ønsket å løse dette ved å utvikle et nytt API som støtter fleksible kall til databasen.

Finne en løsning på problemene systemet til oppdragsgiver har i dag og implementere disse løsningene på en god måte i det nye systemet, er problemstillingen for denne bacheloroppgaven.

1.1.3 Mål

Målet for det tekniske arbeidet er å utvikle et nytt web-basert personalsystem for Det Akademiske Kvarteret og tilknyttede organisasjoner i Bergen. Det nye systemet skal løse de problemene dagens system har ved å implementere gode løsninger for problemene som har blitt diskutert i problemstillingen.

Delmålene for prosjektet blir definert ut fra kravspesifikasjonene som er omtalt i kapittel 2.2 i denne rapporten som omhandler funksjonalitet til applikasjonen samt krav den skal oppfylle.

1.2 Kontekst

1.2.1 Prosjektet

Det Akademiske Kvarter har i lengre tid sett behovet for å oppdatere sitt nåværende personalsystem. På grunn av dette har de i samarbeid med HVL Bergen valgt å gi dette som en bacheloroppgave for avgangsstudenter 2020.

Under prosjektperioden er det ønske om å få utvikle et system som har den samme funksjonaliteten som dagens system, i tillegg til en del annen funksjonalitet. Det skal ikke legges vekt på styling, siden dette er noe it-folkene innad i Kvarteret vil ta seg av i etterkant. Det har i samtale med kontaktperson på kvarteret, Khiem Tran, kommet frem at de viktigste oppgavene under prosjektet er å få til så mye funksjonalitet som mulig i den tidsrammen som er tilgjengelig.

Som nevnt over har Kvarteret sin egen it-avdeling, disse vil under prosjektperioden være tilgjengelig for spørsmål og hjelp om det vil være behov for det.

1.3 Avgrensning

Det er alltid begrensninger når det kommer et prosjekt av denne typen. I dette tilfelle er det spesielt aktuelt å legge merke til tidsrammen som går fra uke 11 til uke 23, våren 2020. Begrenset kunnskap innenfor rammeverkene som oppdragsgiver ønsker å bruke samt at gruppen kun er to personer er noen av begrensningene.

1.4 Ressurser

Dette kapittelet vil ta opp de ulike ressursene gruppen vil benytte under prosjektet, og ressursene de hadde til dispensasjon.

1.4.1 Oppdragsgivers ressurser

Kvarteret kunne stille med lokaler hvor gruppen kunne sitte og jobbe dersom det var ønske om det, de skulle også være tilgjengelig for møter og gi tilbakemelding på utført arbeid underveis. It avdelingen ville stå til dispensasjon og være tilgjengelig for å hjelpe dersom det ville være nødvendig.

1.4.2 Andre ressurser

Når det kom til å skaffe seg kunnskap ble det avtalt i planleggingsfasen at gruppemedlemmene skulle utføre et online kurs i .NET Core og React. Dette kurset har med det blitt grunnlaget for hvordan gruppe valgte å gjennomføre de oppgavene prosjektet hadde som mål. I tillegg til dette har dokumentasjonen online for både React og .NET Core vært til stor hjelp.

1.5 Organisering av rapporten

Kapittel 1	Presentasjon av mål og motivasjon for oppgaven. Det blir presentert begrensningene vi har for oppgaven. Til slutt blir det presentert de tilgjengelige ressursene.
Kapittel 2	Presentasjon av bakgrunnen for prosjektet. Det blir også presentert kravspesifikasjonen, og vårt tidlige forslag til løsning.
Kapittel 3	Presentasjon av ulike designvalg og verktøy, og hvorfor. Det blir også tatt opp alternative løsninger vi har sett på, og hvordan vi skal evaluere prosjektet.
Kapittel 4	Presentasjon av hvordan vi har utført prosjektet.
Kapittel 5	Presentasjon av hvordan vi har evaluert produktet, og hva resultatet av evalueringen har vært.
Kapittel 6	Presentasjon av diskusjon rundt fremgangsmåte, resultat og problemstillinger.
Kapittel 7	Presentasjon av konklusjon og videre arbeid.
Kapittel 8	Presentasjon av referanser.
Kapittel 9	Presentasjon av vedlegg.

2 PROSJEKTBESKRIVELSE

2.1 Praktisk bakgrunn

2.1.1 Prosjekteier

Det Akademiske Kvarter er som nevnt i kapittel en, det kulturelle studenthuset i Bergen. De drives av frivillige studenter fra de ulike student institusjonene rundt om i byen, og har om lag 300-400 aktive studenter knyttet til huset til enhver tid. Under prosjektet vil arbeidet med oppgaven bli fulgt opp av en gruppe personer i ulike nøkkelroller på Kvarteret samt personer fra Studentersamfunnet i Bergen. Ved endt prosjekt vil Det Akademiske Kvarter ha alle rettigheter til programmet og kildekode.

2.1.2 Tidligere arbeid

Det Akademiske Kvarter har en tidligere løsning for håndtering av studenter knyttet til huset, som de har utviklet selv, i perioden 2007 frem til i dag. Mye av funksjonaliteten i dette systemet ønskes det at man viderefører inn i et nytt system.

Med det systemet Kvarteret har i dag, følger det også en database. Dersom det er mulig å bruke denne databasen videre skal man gjøre det. Et diagram over hvordan denne ser ut er vist på bilde under.

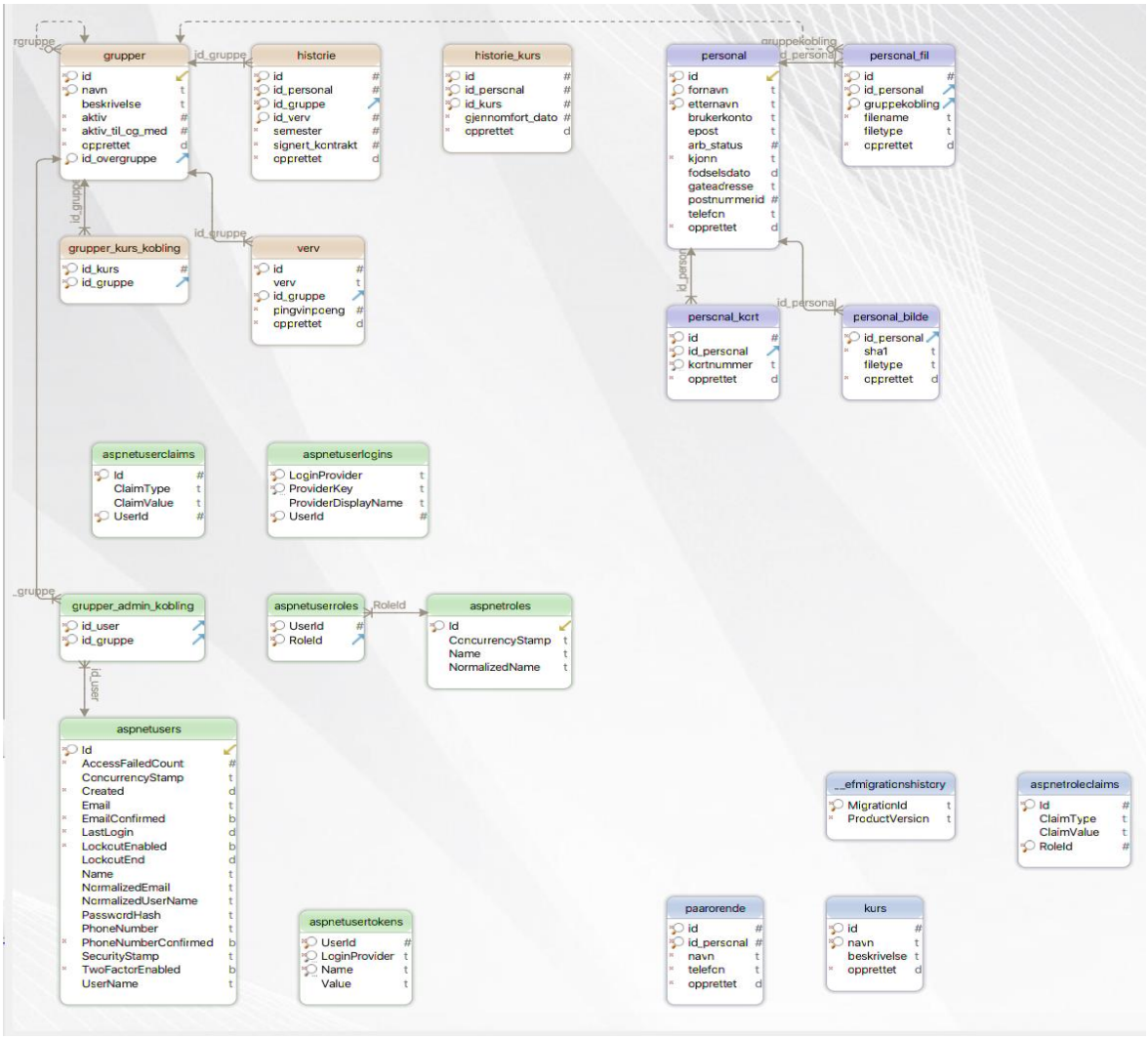


Figure 1. Databasemodell over dagens system .

Det er også et grafisk grensesnitt tilknyttet til denne databasen, men dette er noe gruppen ikke har fått tilgang til.

2.2 Kravspesifikasjon

Kravspesifikasjonene gitt av oppdragsgiver kan man dele disse inn i funksjonelle og ikke funksjonelle krav. Funksjonelle krav vil si krav om hva systemet er ment å gjøre, mens ikke-funksjonelle krav vil si hvordan systemet skal implementere de funksjonelle kravene. I punktene under beskrives disse ulike kravene for denne oppgaven.

2.2.1 Funksjonelle krav

Funksjonelle krav	Kort
RESTful Api	FK1
Søk og Filtrering	FK2
Generere nytt passord via epost	FK3
Generere nytt passord manuelt	FK4
2-Faktor autentisering	FK5
Eksport av data til CSV	FK6
Varslinger via epost	FK7
Fleksible API-kall	FK8
Tilgangstyring	FK9
CRUD: På alle entiteter	FK10
Profilside for brukere	FK11

Table 1, Funksjonelle krav.

2.2.2 Ikke funksjonelle krav

Ikke-funksjonelle krav	Kort
Enklere enn tidligere system	IFK1
Alle brukere skal ha tilgang	IFK2
Responsivt design	IFK3
Kjappere enn tidligere system	IFK4
Overholde GDPR regler	IFK5

Table 2. Ikke funksjonelle krav.

De funksjonelle kravene i kravspesifikasjonen er det gruppen har brukt for å definere delmålene underveis i prosjektet.

2.2.3 Løsningen

For å komme frem til et løsningsforslag har gruppen brukt de kravene som kom frem i kravspesifikasjonen samt en oversikt over hvordan databasen er strukturert i dag. Det blir viktig at så mange så mulig av disse kravene blir tilfredsstilt og at man kan bruke de dataene som ligger i databasen i dag videre.

Det er i utgangspunktet ønsket at backend utvikles i .NET Core 3.1. Videre står gruppen ganske fritt til å velge den teknologien de selv ønsker og jobbe med, dette blir diskutert nærmere i kapittel 3. Det har ikke blitt satt noe krav til styling av komponentene som brukes, dette grunnet at oppdragsgiver selv vil styles disse etter sine standarder i etterkant.

2.3 Løsningside

Forslaget til løsning er utarbeidet av gruppen på bakgrunn av møter og den informasjonen som har blitt sendt til gruppen, fra oppdragsgiver.

2.3.1 Funksjonalitet

Forslaget bygger på oppdragsgivers ønske om å lage en medlems database applikasjon som er raskere og mer brukervennlig enn hva de har i dag. Dagens database applikasjon har per dags dato ikke støtte for alle de funksjonen som de ønsker, for at det skal være enkelt å søke opp og holde data oppdatert. Kravene til funksjonalitet er med dette et resultat av at de ønsker en ny applikasjon som skal løse disse begrensningene.

Løsningsforslaget er derfor en helt ny front- og back-end, med innlogging og tilgangsstyring av hver enkelt bruker basert på hvilken rolle brukeren har. Alle brukere skal også ha en status som sier om brukeren er aktiv eller ikke. Om brukeren ikke er aktiv, skal den ha spesielle tilganger til å legge inn historisk data. Brukere skal også ha tilgang til å endre og oppdatere personalia om seg selv.

Når det kommer til struktur og organisering vil det finnes organisasjoner som har grupper relatert til seg. I disse gruppene vil man igjen ha mulighet til å legge til aktive brukere, hvor en organisasjon kan ha mange grupper som igjen har mange brukere under seg.

For å gjøre det lettere å søke på brukere vil systemet også ha et “tag”-system hvor man kan legge til tags på brukerne. Dette vil si at man kan tagge brukerne med stikkord og når man søker på dette ordet får man tilbake brukerne som har dette tagget koblet til seg. På bakgrunn av dette kommer søkefunksjonen til å støtte søk på flere enn en tag, slik at man får tilgang til de personene man er ute etter. Søkefunksjonen kommer også til

å støtte søk på andre data, som for eksempel navn, telefonnummer eller annet. I tillegg til dette vil det nye systemet støtte eksport av data i form av en CSV-fil.

Systemet vil kunne lagre historikk på brukere om hvilket semester de har vært aktiv, hvilken gruppe(r) de har vært medlem i og hvorvidt de har signert kontrakt for gjeldende semester samt verv de har hatt i gruppen. Denne funksjonaliteten er noe dagens system har, og det nye systemet vil ha noe som tilsvarer dette.

Lagring av personlige opplysninger skal følge GDPR og brukerne må godkjenne lagring av sine opplysninger etter en angitt avtaletekst. En administrator vil også ha mulighet til å slette brukere fullstendig fra databasen ved behov.

2.4 Litteratur om problemstillingen

Det har blitt gjort en del undersøkelse under planleggingsfasen for å kartlegge hva som er viktig å passe på under utviklingen av et web-basert databasesystem. I dette avsnittet diskuteres det nærmere hva gruppen fant av ressurser.

Mokhtar Ebrahim skrev i en artikkel for DZone i 2019 om de ni vanligste feilene som blir gjort når man skal utvikle en database. Noen av de tingene som trekkes fram i dette innlegge er ting som dårlig planlegging, dårlig navngiving, manglende dokumentasjon og utilstrekkelig testing. Dette samt de andre punktene Ebrahim beskrive er ting gruppen har hatt i tankene under prosjektet.

I en annen artikkel på DZone skrevet av Anna Makowska i 2018, tas det opp feil som ofte gjøres under API utvikling, som for sluttbrukeren kan være irriterende. En av de tingene som blir beskrevet er rapportering av feilmeldinger som ofte ikke gir noe nyttig tilbakemelding om hva som har gått galt. For å unngå dette i denne applikasjonen tenkes det å bruke Axios for å undersøke kallene mellom API og brukergrensesnitt. Dette gir mulighet for å se hva feilmeldinger som ligger i kallet og man kan ta forhåndsregler for å sikre at brukeren blir godt informert om hva som har gått galt. En annen ting som er verd å merke seg og som Makowska omtaler som en vanlig feil mange gjør, er API-dokumentasjon. Det blir anbefalt en rekke løsninger som forenkler denne prosessen, hvor en a de er swagger noe gruppen har valgt å bruke.

For å få en bedre forståelse for hva som er viktig å passe på når man skal utvikle et grafisk grensesnitt har det blitt sett på en artikkel på meetadeveloper.com (Order Group, 2016). Det nevnes i denne artikkelen at man ikke nødvendigvis trenger en stor test gruppe for brukertesting og det vises til en artikkel av Jakob Nielsen, som forteller hvorfor man kun trenger 5 bruker for å dekke 80% av feilene. Dette er noe gruppen vil ha i tankene når de planlegger for testing. En annen ting som også diskuteres i denne

artikkelen er viktigheten av å holde kompleksiteten så lav som mulig på skjermen brukeren skal benytte. Dette for å sikre en enkel og god brukeropplevelse når de skal håndtere data.

Resultatene fra undersøkelsene gjort i forbindelse med dette avsnittet har vært til stor hjelp under utviklingen av applikasjonen.

3 DESIGN AV PROSJEKTET

I dette kapittelet diskuteres designet for prosjektet. For å kunne vurdere og ta stilling til hvilke design som passer best for å utviklingen av applikasjonen har det blitt vurdert flere alternativ. Dette blir beskrevet mer i detalj i dette kapittelet.

3.1 Valgt prosjektdesign

3.1.1 Rammeverk og biblioteker for webapplikasjonen - React

React er et Javascript bibliotek som gjør det enkelt å lage interaktive brukergrensesnitt. React er state og komponentbasert, som vil si at React kun kaller en oppdatering av DOM-modellen dersom dataene i "state"-en endrer seg. Dette gir mulighet for å bygge store webapplikasjoner hvor det kan skje endringer i deler av applikasjonen uten at hele siden må lastes inn på nytt(reactjs.org).

I senere tid har React også fått støtte for TypeScript. TypeScript er et "open-source" programmeringsspråk utviklet og vedlikeholdt av Microsoft. Språket er et strengt syntaktisk supersett av JavaScript og tilfører språket valgfri statisk skriving (TypeScript).

3.1.2 Rammeverk og biblioteker for backend - .NET Core

.NET Core er et open-source rammeverk som brukes for å server applikasjoner på Windows, Linux og macOS. Dette rammeverket tilbyr høy ytelse og støtte for integrasjon mot de fleste moderne klientside-rammeverk. Når det kommer til hosting av serveren står man fritt til å hoste applikasjoner i skyen eller på egne servere. Dette gir frihet til å gjøre det som passer brukeren best(Microsoft). Man får også enkelt mulighet til å legge til dependencies gjennom den innebygde "package manager"-en NuGet (Microsoft, 2020).

3.1.3 Utviklingsmetode - Scrum

Margaret Rouse skrev i en artikkel fra 2017 om Scrum (Rouse M., 2017) som en metode for å styre prosjekter med vekt på samarbeid, ansvarlighet og en iterativ fremgang mot et veldefinert mål. Prosessen starter med et enkelt premiss, lag en hypotese på hvordan man tror noe fungerer, prøv det ut, reflekter over resultatet og gjør nødvendige justeringer i neste iterasjon. I en slik utviklingsprosess står åpenhet, undersøkelse og tilpassing sentralt forteller Rouse.

Rouse trekker frem tre rolletyper i scrum prosessen. Disse er produkteier, scrum master og scrum utviklingsteamet.

- Produkteier blir sett på som det medlemmet i teamet som har ansvar for å kommunisere og avtale kriteriene for sluttproduktet med kunden. Dette vil si at dette er bindeleddet mellom utviklingsteamet og kunden.
- Scrum master er ansvarlig for at man følger scrum filosofiens beste praksis, for å sikre fremgang i prosjektet.
- Utviklingsteamet er personene som sammen lager og tester iterasjonene mot sluttproduktet.

Scrum process:

- Daglig scrum: kort møte hvor det diskuteres arbeidet som har blitt utført dagen før og hva som skal gjøres denne dagen.
- Sprint planlegging: I starten av hver sprint setter teamet seg ned og bestemmer seg for hvilke oppgaver som skal gjennomføres i den kommende sprinten.
- Sprint gjennomgang: Gjennomføres ved endt sprint, der en gjennomgang av det som har blitt gjort blir presentert for produkteier og interessenter.
- Refleksjon: Etter endt sprint holdes det et møte hvor man kan reflektere over arbeidet som er gjort og komme frem til endringer som kan utføres i kommende sprinter.

Scrum består også av en rekke artefakter og innebærer:

- En produkt-backlog, med oppgaver som skal gjøres i prosjektet.
- En sprint-backlog, med oppgaver som skal utføres i en gitt sprint.
- Et produkt-increment, som er en samling av elementer fra sprint backlogen som oppfyller kravene som er ferdigstilte.

3.1.4 State Management - MobX

MobX er et bibliotek som gjør state management enkelt og skalerbart. Det er noen få hovedkonsepter i dette biblioteket det er verdt å legge merke til, Observable state, Computed values, reactions og actions. På MobX sin nettside forklares disse konseptene på følgende måte:

- Observable fungerer ved at den gjør en verdi om til en regnearkcelle, men med støtte for mer enn kun primitive verdier.
- Computed values, en annotering man kan sette på funksjoner og verdier for at de skal kjøres når relevante data endres.
- Reaction fungerer som computed values men i stedet for å lage en ny verdi, kan denne sett i gang andre prosesser.
- Actions som er annoteringer som settes på alt som gjør noe med “state”-en

(MobX, 2020)

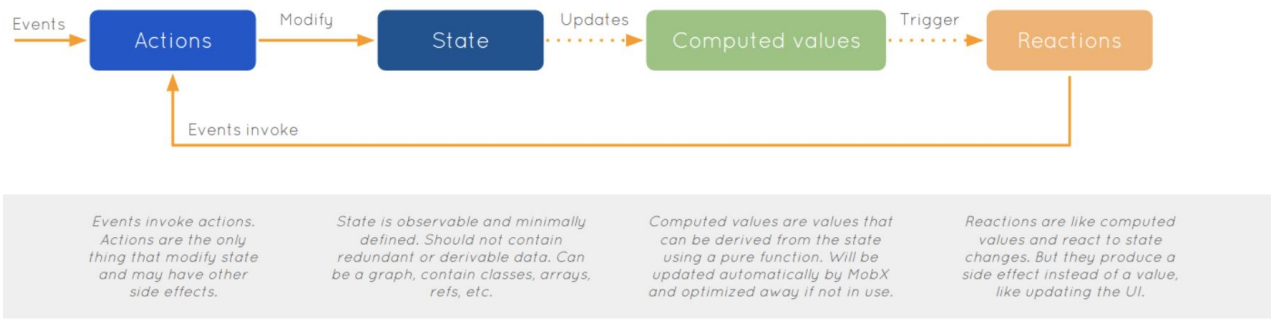


Figure 2. Mobx prinsipper, hentet fra Mobx.js.org.

3.1.5 Arkitektur

Applikasjonen som skal lages skal utvikles fra bunnen av. Dette vil si at at gruppen har stått fritt til valg av arkitekturen for de forskjellige delene i applikasjonen. Forslagene som diskuteres i punktene under er hva gruppen har kommet frem til.

Model View Controller (MVC)

Forslag til arkitektur skal bygge på prinsippet at hver del av applikasjonen kun skal ha ansvar for en oppgave. Ved å bruke MVC (Wikipedia), er målet å dele brukergrensesnittet fra dataene, slik at endringer i en del ikke skal ha noen innvirkning på den andre delen. I 4.7.1 blir det gått i dybden på hvordan prosjektet er bygget opp for å nå dette.

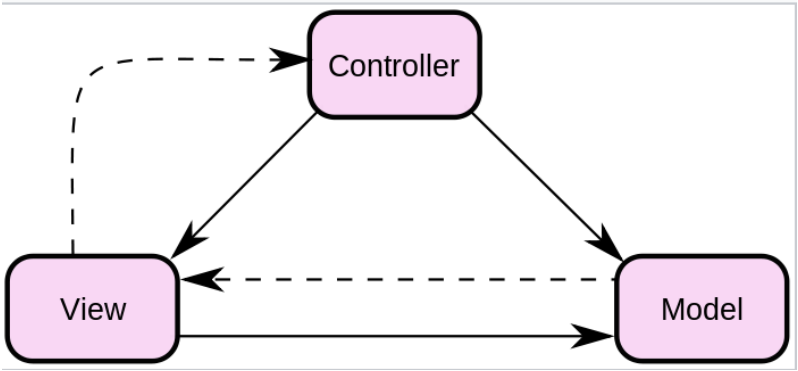


Figure 3. Model View Controller, hentet fra https://no.wikipedia.org/wiki/Model–view–controller.

Brukergrensesnitt

For å gjøre det enkelt og oversiktlig tenkes en mappestruktur som vist under som et godt utgangspunkt for utviklingen av brukergrensesnittet.

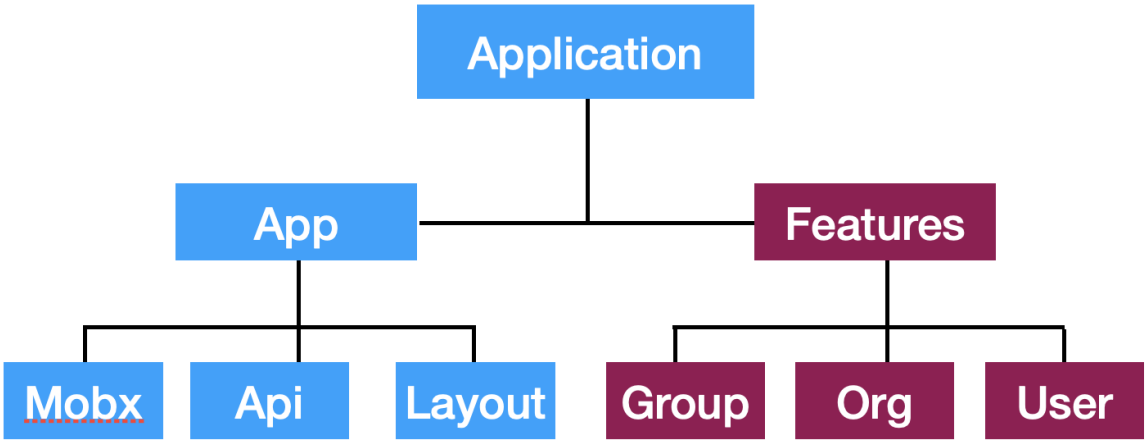


Figure 4. Mappestruktur for brukergrensesnitt.

3.1.6 Design

Lite er sagt fra oppdragsgiver om designet på applikasjonen, utover at de ønsket å gjøre stylingen selv, i etterkant av endt prosjekt. På bakgrunn av denne informasjonen utarbeidet gruppen noen enkle skisser for en tenkt utforming av de forskjellige sidene i applikasjonen. Det ble også bestemt å bruke "Semantic UI React" (Semantic UI React), som er et bibliotek for ferdigdesignet komponenter for React applikasjoner. Dette vil spare gruppen for mye arbeid med å designe komponentene selv, og er spesielt bra siden oppdragsgiver uansett vil legge på eget design i etterkant. Dersom "Semantic UI React" ikke støtter en eller flere komponenter som det er bruk for, vil disse komponentene bli laget ved hjelp av "styled-components". Dette er et bibliotek som bringer ideen om å bruke komponenter for å produsere en applikasjon sammen med CSS på en enkel og intuitiv måte.

3.1.7 UML

Under er det vist hvordan gruppen tenker å bygge applikasjonen. Det blir vist de ulike entitetene samt forholdene mellom disse. (UML, 2005)

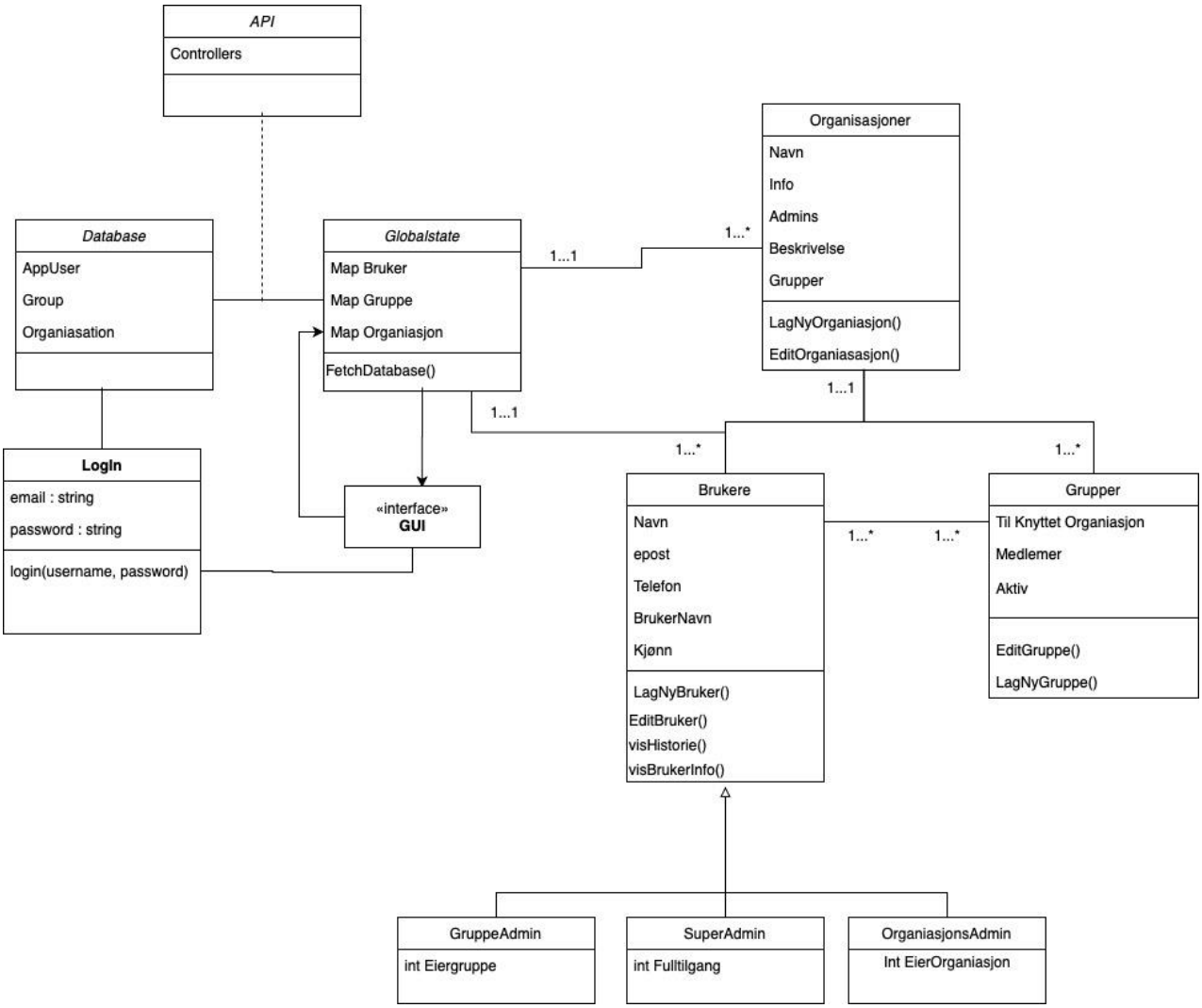


Figure 5. UML diagram av applikasjonen.

3.2 Alternative Prosjekt Design

Som utvikler i dag har man en rekke muligheter når det kommer til valg av teknologi i et utviklingsprosjekt. I dette avsnittet diskuteres noen alternativer som kunne vært brukt i stedet for det som ble valgt.

3.2.1 Alternative rammeverk og biblioteker for webapplikasjonen - AngularJS

AngularJS ville vært et godt alternativ i utviklingen av klient siden av applikasjonen. Det er et meget populært JavaScript rammeverk hvor det er enkelt å finne ressurser og hjelp online. Utviklerne av AngularJS definerer det som hva HTML ville vært, dersom det var designet for applikasjoner. Ved å bruke HTML som målspråk og utvidelse av syntaksen, gir rammeverket muligheten til å uttrykke komponenter tydelig og kortfattet. I tillegg kan man binde data til HTML gjennom uttrykk som gjøre det lettleselig og raskt å utvikle applikasjoner. (AngularJS, 2020)

3.2.2 Alternative rammeverk og biblioteker for backend - PHP

PHP er et godt alternativ til .NET Core. Det er et språk som er enkelt å skrive, og det finnes mye dokumentasjon, siden språket har vært ute lenge. En av de største fordelene gruppen kunne hatt ved å bruke PHP, er tidligere erfaring. Gruppen har brukt dette språket før, og bygget både API og full-stack løsninger. Hadde PHP vært brukt i backend, hadde utviklingen gått mye kjappere og det hadde blitt brukt mindre tid på feilsøking. Negative sider med å bruke PHP er at det ikke er like mange bibliotek eller moduler tilgjengelig for dette. .NET Core har fordelen ved at det er utviklet spesifikt til det formålet det er brukt for, og har mange hjelpefunksjoner til å gjøre API-bygging lettere. Sikkerhet er også et område .NET har fordeler, da vi i PHP med stor sannsynlighet måtte ha implementert dette selv (PHP).

3.2.3 Alternative utviklings metode - Extreme Programming

Extreme Programming er en utviklingsmetodikk som ble vurdert i starten av prosjektet. Denne går ut på ekstremt hyppig møtevirksomhet med kunde, og å lage enkle moduler hurtig. Disse vil bli vist fram til oppdragsgiver, og de kan komme med tilbakemeldinger. Denne utviklingsmetodikken krever møter med oppdragsgiver hver dag, eller annenhver dag, slik at de kan komme med tilbakemelding fortløpende. Kvarteret var tidlig ute og sa at de ønskte omtrent et møte hver uke, eller annenhver uke, så Extreme Programming hadde blitt vanskelig (Wells D., 2013).

3.2.4 Alternative state management - Redux

I dokumentasjon for Redux på deres hjemmeside, beskrives Redux som en forutsigbar “state-container” for JavaScript applikasjoner. Konseptet er at man legger “state”-en til applikasjonen i “stores”, der komponenter i applikasjonen igjen kan får tilgang til disse. Det at de forskjellige komponentene i applikasjonen bruker samme “state- data er med på å sikre at applikasjonen oppfører seg slik den skal. Redux har også en stor fordel når det kommer til “state debugging”. Feilsøking går gjennom “The Redux DevTools”, som gjør det enkelt å følge “state” endringer (Redux, 2020).

3.3 Diskusjon av alternativene

Når det kommer til valg av teknologi, har gruppen stått ganske fritt til å velge det de selv ønsket. Eneste kravet som har kommet fra oppdragsgiver har vært at de ønsket å bruke .NET Core i utviklingen av server siden av applikasjonen. Dette kravet kom likevel ikke frem i starten av prosjektet, så gruppen hadde planer om å bruke PHP i utviklingen av serversiden.

Ved å bruke React med TypeScript utnyttes kunnskap som gruppemedlemmene alt sitter på. Typescript ble også valgt over en ren JavaScript ES6. Dette grunnet at det gjør det enklere å plukke opp småfeil i syntaksen gjennom å definere typer og "interfaces" som kan oppføre seg som objekter, noe som JavaScript mangler. Skulle man valgt et annet rammeverk, som AngularJS eller lignende, ville et resultat av dette vært at for mye tid hadde gått til å lære seg nye ting. Spesielt med tanke på at gruppen alt måtte lære seg .NET Core, ville dette ført til at man hadde hatt mindre tid til å utvikle den ønskede funksjonaliteten for applikasjonen.

Når det kommer til "state management" ble også MobX valgt over Redux på grunn av at gruppen alt hadde erfaring i å bruke dette biblioteket. Det kunne likevel vært aktuelt å bruke noe annet dersom det hadde kommet noen krav om det.

Det ble tidlig bestemt at gruppen skulle gå for en agil utviklingsmetodikk, hvor metodikken som ble valg var Scrum. Gruppen består kun av to personer så det ble gjort noen endringer i måten Scrum filosofien ble utført. Møter ble ikke holdt hver dag, slik som det ofte gjøres med scrum, men et par ganger i uken. Andre deler, som sprinter, sprint planlegging, backlogs og refleksjon, har blitt fulgt opp. Denne arbeidsformen har også gjort gruppen veldig fleksibel for endringer under prosjektperioden.

3.4 Valg av verktøy, programmeringsspråk og rammeverk

3.4.1 Verktøy

IDE/Utviklingsplattform

For å effektivt skrive kode, og få teste denne hurtig, er det viktig å finne en god utviklingsplattform. Gruppemedlemmene har forskjellig bakgrunn, og har ulike preferanser, så dette var åpent blant gruppemedlemmene. Under hele utviklingsprosessen var det i hovedsak to plattformer brukt, Visual Studio Code (Visual Studio Code, 2020) og JetBrains Rider (JetBrains, 2020).

GitHub

Oppgaven var avhengig av å bruke et versjonskontrollsystem for å gjøre utviklingsprosessen dynamisk. GitHub er brukt mye på skolen og ellers i miljøet, så dette var et naturlig valg. GitHub gjør det lett for alle gruppemedlemmene å oppdatere kodebasen, og både Visual Studio Code og JetBrains Rider har muligheter for å bruke git-baserte løsninger. Prosjektet er åpent for alle å se, siden produktet ikke skal kommersialiseres, og kun kjøre for Kvarteret. Ved feil på programmet er det enkelt å gå tilbake å se hva som er gjort, og med mye ressurser på nett rundt GitHub, er det effektivt å bruke (GitHub).

Node.js/Node Package Manager

For å ha et lokalt utviklingsmiljø der vi kan kjøre React applikasjonen under utvikling, har vi gått for Node.js. Dette er også det mest vanlige verktøyet å bruke når man utvikler React applikasjoner. Node.js har en veldig god måte å håndtere pakker på, som heter Node Package Manager, eller npm. Denne gjør det enkelt å hente ned pakker med ferdigbygde moduler som du kan bruke i applikasjonen du utvikler. NPM håndterer også oppdatering av disse pakkene når det kommer nye versjoner (NodeJS).

NuGet

Slik som pakkehåndtering i frontend er styrt av NPM og Node.js, er pakkehåndtering i backend styrt av NuGet. NuGet inneholder over hundre tusen pakker, og disse er brukt i millioner av applikasjoner. Dette gjør at gruppen kan bruke ferdige bibliotek for håndtering av, for eksempel, databasen. Pakkene er Open Source og utprøvd, så kritiske deler av applikasjonen, som sikkerhet, har egne pakker og "best practices" (Microsoft, 2020).

3.4.2 Programmeringsspråk/Rammeverk

React

React er beskrevet i punkt 3.1.1

.NET Core

.NET Core er beskrevet i punkt 3.1.2

MobX

MobX er beskrevet i punkt 3.1.4

TypeScript

React er et JavaScript-bibliotek, og når en skriver React komponenter og kode, blir denne skrevet i JavaScript. Et alternativ til JavaScript, er TypeScript. TypeScript er et supersett av JavaScript. Det vil si at TypeScript-kode blir kompilert til vanlig JavaScript kode. Gruppen valgte å gå for TypeScript over JavaScript på grunn av simplisitet. TypeScript har en del fordeler i syntaksen og når det kommer til debugging. Siden TypeScript har typer på variabler, kan Visual Studio Code se gjennom applikasjonen, og finne feil der vi bruker typer som ikke er lov. Dette sparer mye tid, da debugging ikke er nødvendig så ofte (TypeScript).

C#

C# eller "C Sharp" er programmeringsspråket som er brukt i .NET Core, i backend. .NET Core kan kun skrives i C#, så her hadde vi ikke mye valg. C# er ikke noe bedre enn noen

andre programmeringsspråk, det handler mer om preferanser og hva du skal utvikle. I dette prosjektet var C# det eneste gruppen kunne velge, da Kvarteret hadde som krav at de ville ha .NET Core i backend.

3.5 Prosjektmetodikk

I dette kapittelet vil det bli lagt frem hvilken utviklingsmetodikk gruppen benytter under prosjektet. Det som vil bli diskutert er Scrum, prosjektplan samt risikovurdering.

3.5.1 Utviklingsmetodikk - SCRUM

Som det ble beskrevet i punkt 3.1.1 ble Scrum valgt som utviklingsmetode for dette prosjektet. Dette er en metode som har blitt undervist i på HVL Bergen og gruppemedlemmene er kjent med hvordan det fungerer. Rollene i denne prosessen har vært noe vanskelig å definere slik det blir gjort i scrum, spesielt med tanke på at gruppen kun består av to personer. Måten dette har blir løst på er at medlemmene i fellesskap har fylt ut backloggen for prosjektet på bakgrunn av kravspesifikasjoner og samtale med oppdragsgiver. Det samme har blitt gjort under sprint planleggingen, hvor gruppen i felleskap har definert mål for sprinten og hentet oppgaver fra backloggen. Hver sprint har blitt satt til en uke med oppstart mandag. Et møte har blitt holdt med begge medlemmene hvor man har gått gjennom sprinten som har vært og gjort justeringer om nødvendig, samt planlagt oppgavene for den neste sprinten. Utover dette har det vært holdt møter med veileder og oppdragsgiver annenhver uke for å vise hva som har blitt gjort, og for å få tilbakemeldinger (Rouse M., 2017).

3.5.2 Prosjektplan

En prosjektplan i form av et GANTT-skjema har blitt brukt for å holde oversikt over oppgavene i prosjektet. I tillegg til dette har Trello blitt benyttet for å holde oversikt over hva hvert medlem holder på med til enhver tid. Et eksempel på hvordan Gantt-skjemaet har sett ut under prosjektet er vist på bildet under (Gantt.com, 2020).

Medlems database Kvarteret

Det Akademiske Kvarter

Project Start Date 3/9/2020 (mandag) Display Week 1

Project Lead

08.02.2020	START	END	DAYS	% DONE	WORK DAYS
Plannlegging/oppstart		-			-
Hele Oppgaven	man. 3/09/20	ons. 6/10/20	94	50%	68
Kurs i .NET Core	man. 3/09/20	søn. 3/29/20	21	100%	15
Prosjekt plan	man. 3/09/20	fre. 4/03/20	26	100%	20
Statusmøte Referat	man. 3/16/20	fre. 3/27/20	12	100%	10
Mål og metode	man. 3/09/20	man. 3/09/20	1	100%	1
Nav flow diagram	man. 3/09/20	søn. 3/22/20	14	100%	10
Skisse ui	man. 3/09/20	søn. 3/22/20	14	100%	10
Risiko Vurdering	16.03.2020	ons. 4/01/20	17	100%	13
FP presentasjon	man. 4/20/20	tir. 4/21/20	2	100%	2
Statusrapport	tir. 4/28/20	tir. 4/28/20		0%	1
Sprint 1		-			-
enkel API for Users	man. 3/16/20	søn. 3/29/20	14	100%	10
basic front end for Users	man. 3/16/20	søn. 3/29/20	14	100%	10
API organisasjon	man. 3/16/20	søn. 3/29/20	14	100%	10
ProfilePageUser	man. 3/16/20	søn. 3/29/20	14	100%	10
API Grupper	man. 3/16/20	søn. 3/29/20	14	100%	
Frontend: Grupper	man. 3/16/20	søn. 3/29/20	14	100%	
Validering Server/client	man. 3/16/20	søn. 3/29/20	14	100%	10
Sprint 2		-			-
ER: Users og gruppe	man. 3/30/20	søn. 4/05/20	7	100%	5
Classes for kurs, kort + api for disse	man. 3/30/20	søn. 4/05/20	7	100%	5
Gjøre ferdig Contoller gruppe	man. 3/30/20	søn. 4/05/20	7	100%	5
Lage Kurs Controller	man. 3/30/20	søn. 4/05/20	7	100%	5
ER: Grupper og Organisasjon	man. 3/30/20	søn. 4/05/20	7	100%	5
Sprint 3		-			-
Planlegging for kommende sprint	man. 4/06/20	man. 4/06/20	1	100%	1
Test for Api	man. 4/06/20	søn. 4/12/20	7	100%	5
Frontend: SØK	man. 4/06/20	søn. 4/12/20	7	100%	5
Fronted: grupper slett/edit/add	man. 4/06/20	søn. 4/12/20	7	100%	5
API: legg til og slett admin	man. 4/06/20	søn. 4/12/20	7	100%	5
fronted: legg til og slett admin	man. 4/06/20	søn. 4/12/20	7	100%	5

Figure 6. Gantt diagram(under prosjektet).

3.5.3 Risikovurdering

Gruppen har benyttet seg av et risikoskjema i evaluering av hvilke risikoer som er tenkt kunne oppstå. Det er her brukt en skala relatert til sannsynlighet, hvor man har delt inn i tre grupper, lite sannsynlig, sannsynlig og meget sannsynlig som går i Y retning. I X retning har alvorligheten blitt definert som lav, middels og høy. Hvordan man skal forstå oppsettet er dersom en index lander i et grønt felt anses problemet som uproblematisk. Ender indexen i gult felt anses det som middels problematisk, og man bør legge vekt på å prøve og minske risikoen. Alt som ender i rød sektor anses som meget problematisk, og forholdsregler er høyst nødvendig for å sikre et vellykket prosjekt.

	Meget sannsynelig		4, 5 og 6	
Sannsynelig	Sannsynelig		3	7
	Lite sannynelig		2	1,8
		Lav	Middels	Høy
			Alvorlighet	

Figure 7. Risiko oversikt.

S = Sannsynlighet

A = Alvorlighet

RF = Fargen er et resultat av å bruke tabellen gitt over

	Risiko	S	A	RF	Tiltak
1	Et gruppemedlem blir syk over en lengre periode	LS	H		Dersom en av medlemmene blir syke over en lengre periode, vil det bli vanskelig og fullføre prosjektet slik det er tenkt i dag. Det vil da bli aktuelt å redusere omfanget av hva vi ønsker å få til i den gitte perioden vi har å jobbe med dette. Videre arbeid vil da bli avtalt med arbeidsgiver og veiledere dersom dette blir aktuelt.
2	Gruppemedlem møter ikke til avtalte møter	LS	M		Dersom man ikke kan møte til avtalte møter skal dette gis beskjed om. Om du ikke møter gjentatte ganger uten å si ifra, vil dette blir tatt videre til veileder.
3	Tildelte oppgaver er ikke fullført i slutten av en sprint	S	M		At avtalte ting ikke er ferdig ved slutten av en sprint er å regne med. Vi vil da på møte diskutere de aktuelle oppgavene og se på hva vi kan gjøre for å fullføre disse i den kommende sprinten.
4	Oppdragsgiver ønsker mer funksjonalitet enn der er tid til å implementere	MS	M		God samtale med arbeidsgiver og informere løpende om progresjonen i prosjektet. Dette skal være med på å gi de en realistisk ide om hva vi vil ha mulighet til å gjøre under prosjektperioden.
5	Problemer med å løse en eller flere oppgaver relatert til funksjonalitet	MS	M		Dersom det oppstår problemer med å implementere funksjonalitet, vil dette blir tatt opp med veiledere for å få input på hvordan vi kan gå frem for å løse dette.

6	Programvareoppdateringer	MS	M		Programvareoppdateringer for programmer vi bruker under prosjektet er svært sannsynlig og vil bli tatt stilling til når de kommer. Det kan bli aktuelt å ikke oppgradere med en gang, dersom dette har potensiale til å skape problemer for prosjektet.
7	Korona karantenen vedvarer	S	H		Slik det ser ut i dag vil korona-karantene fortsette. Vi føler vi har funnet en god måte for gruppen å samarbeide og planlegge gjennom bruken av Zoom. Det kan også bli aktuelt å jobbe i ukes sprinter i stedet for to ukers sprinter, for å oftere komme i lag og diskutere det vi jobber med.
8	Får ikke kontakt med arbeidsgiver	LS	H		Det kan hende at det blir vanskelig å ha kontakt med arbeidsgiver grunnet korona krisen. Dersom dette skulle skje vil vi høre med veileder hva vi skal gjøre videre.

Table 3. Risiko oversikt.

3.6 Evalueringsplan

I dette kapittelet vil det bli diskutert hvordan det er tenkt at prosjektet skal evalueres. Man vil da skille mellom det som skal evalueres og resultatene av denne evalueringen.

3.6.1 Det som skal evalueres

Planen er at applikasjonen som blir utviklet skal evalueres kontinuerlig gjennom hele prosjektet. Det blir naturlige å skille mellom intern- og ekstern evaluering når det kommer til hvem som utfører evalueringen. Det vil også være nødvendig å skille mellom hva som evalueres. Det blir da tenkt at denne inndelingen vil være aktivitetene for evalueringen, produktet som evalueres og ressursene som brukes.

Evaluerings aktiviteter

Metodene gruppen har valgt å bruke i evalueringsprosessen er i form av sprint reviews, og ulike former for testing av implementasjonen i applikasjonen. Underveis i prosjektet vil det sees på om dette er effektivt når det kommer til å få de tilbakemeldingene gruppen har behov for. Ut ifra hva gruppen sitter igjen med etter en evaluering, vil det

bli tatt valg om man skal fortsette med denne formen for evaluering videre, eller om man må gjøre noen endringer.

Produktet

Produktet er selvfølgelig også en viktig del av det som skal evalueres. Metodene gruppen har valgt for evaluering skal være med på å sikre at det som produseres samstemmer med kravspesifikasjonene som er gitt.

Ressursene

Det som inngår i ressurser, er i hovedsak personer som tar del i testingen og evalueringen. For å sikre at gruppen får de tilbakemeldingen de har behov for, er det derfor nødvendig å gjøre vurdering underveis. Tilbakemeldinger fra brukere er ikke alltid like nyttig. Dersom det skulle oppstå noen problemer med de involverte vil det bli tatt stilling til dette underveis.

En annen ressurs vil være plattformen evalueringen gjennom testing utføres på. Det er i utgangspunktet tenkt å gjøre det lokalt på pc-en under møter med testobjektene. Grunnet situasjonen våren 2020, må dette tas stilling til underveis i prosjektet, når det kommer til hvordan dette skal løses.

Intern evaluering

Den interne evaluering vil bli gjennomført av gruppen på det arbeidet som har blitt gjort i hver sprint, som en del av scrum syklusen. Det vil her bli tatt stilling til om det som har blitt produsert tilfredsstiller de kravene oppgaven er ute etter å løse. Det er også her man tar stilling til om de metodene man bruker fungere tilstrekkelig for å sikre god kvalitet i applikasjonen.

Ekstern evaluering

Når det kommer til ekstern evaluering er det planlagt møter med oppdragsgiver annen hver uke for å få tilbakemeldinger på det arbeidet som har blitt gjort. Måten dette vil fungere på er at gruppen forteller oppdragsgiver hva de ønsker at de skal teste, for så å observere og ta tilbakemeldinger på om funksjonene er intuitive å bruke samt om det samsvarer med det de var ute etter å få laget.

En mer omfattende brukertest er også planlagt etter hvert som applikasjonen begynner å få nok funksjonalitet. Denne vil bli utført ved å publisere applikasjonen på oppdragsgivers servere slik at brukergruppen kan teste den ut og komme med tilbakemeldinger. Det er forventet at denne fasen vil gi mye nyttig informasjon i form av tilbakemeldinger. Grunnet Covid-19 viruset, som har ført til karantener verden over, er det usikkert om denne fasen er gjennomførbar under prosjektperioden.

3.6.2 Resultatene fra evaluering

Resultatene fra evalueringen vil være en viktig del i utviklingsprosessen. Spesielt for gruppen er disse resultatene viktig, for å sikre seg om at det som blir produsert er det oppdragsgiver har ønske om. Resultater fra hver evaluering vil bli tatt med inn i neste sprint planlegging, dersom det var noe oppdragsgiver ikke var helt fornøyd med.

Resultatene vil også være en viktig del når det kommer til å vurdere arbeidet som har blitt gjort i denne bachelor oppgaven, hvor resultatene kan være med på å si noe om hvor vellykket prosjektet har vært

4 Produktdesign

I dette kapitlet vil det bli gitt en innføring i designet av prosjektet. Hver del som har vært med på å utforme løsningen samt fremgangsmåte, vil bli diskutert.

4.1 Innledende fase

Ved prosjektstart ble gruppen introdusert for de personene som skulle ha ansvar for prosjektet sett fra oppdragsgivers side. Dette var Khiem Tran, Administrerende Leder ved Kvarteret og Oda Bjerkan, Ceremonimester ved samfunnet i Bergen. På oppstartsmøte ble det bestemt at Khiem ville være den personen gruppen skulle forholde seg til under prosjektet, grunnet hans tekniske innsikt i hvordan ting fungerer i dag. Oda ville være behjelpelig med input på design, og hvilke funksjoner som de ønsket prosjektet skulle fokusere på å lage.

I denne fasen ble det også lagt mye vekt på å tilegne seg ny kunnskap innenfor rammeverkene som skulle brukes. Valg og oppsett av utviklingsmiljø var også en del av denne fasen.

4.2 Planlegging

For at prosjektet skulle ha en jevn progresjon gjennom hele prosjektperioden ble det utformet en overordnet plan for hvordan man skulle nå målene for prosjektet. I denne fasen ble det utarbeidet en arkitektur for applikasjonen, samt en rekke skisser for hvordan gruppen tenkte brukergrensesnittet ville se ut.

Underveis i prosjektet har man jobbet i sprinter, hvor man på hvert oppstartsmøte har tatt tak i noen av målene for prosjektet. Målene har så blitt analysert, og oppgaver opprettet for å fullføre disse målene. De oppgavene som ble opprettet under disse møtene har blitt lagt inn i Trello, med et prioriteringssystem på hva som skulle gjøres først og sist (Trello). Et eksempel er vist på bilde under.

Figure 8. Eksempel fra Trello.

4.3 Brukergrensesnitt

I dette kapittelet gis det en innføring i hvordan det grafiske grensesnittet er bygget opp.

4.3.1 Komponenter i React

Oppbyggingen av det grafiske brukergrensesnittet har blitt bygget ved hjelp av React-komponenter. Når man bygger React-komponenter oppfordres det til å generalisere så mye som mulig, for at komponentene skal bli gjenbrukbare og enkle å bruke. Målet med disse er å lage en applikasjon med gjenbrukbare deler som kan kjøre isolert fra hverandre, hvor hver del kan byttes ut om nødvendig.

I React skiller man mellom klasse- og funksjonelle-komponenter. I dette prosjektet er det utelukket brukt funksjonelle komponenter og man kan anse at det er dette som menes når komponenter omtales. Det at en komponent er funksjonell betyr at den er brygget opp som en JavaScript-funksjon med en “return statement” som returner JSX, som React bruker for å vise innhold på skjermen.

Man deler også komponenten i “stateless” og “stateful” komponenter. Forskjellen på disse om komponenten er direkte klar over den globale “state”-en til applikasjonen eller ikke. I dette prosjektet er det blitt benyttet begge deler, hvor som regel komponenter som har i oppgave å vise data får denne sendt via props fra “parent”-komponenten, mens steder hvor den globale “state”-en blir endret har blitt koblet direkte til MobX.

For oppbygging av komponenter bruker prosjektet samme ide gjennom hele applikasjonen. Denne går ut på at det i hver modul er en "parent"-komponent som knytter sammen en rekke underkomponenter for å skape helheten.

Modal er også en stor del av det grafiske grensesnitt. Dette er når man viser en komponent på toppen av en annen komponent, og bare "dimmer" den som er i bakgrunnen. For å gjøre denne enkel å bruke samt høyst gjenbrukbare har det blitt laget en felles "Modal" som kan ta hva som helst som argument og vise det på skjermen. Et eksempel på dette er vist under.

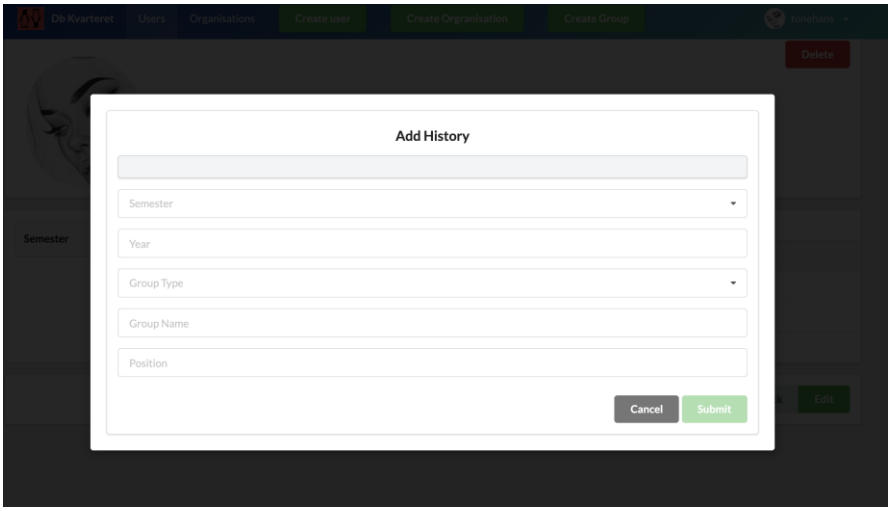


Figure 9. Modal eksempel.

I styling av komponenter har det blitt valgt å bruke "Semantic UI React", som er et bibliotek med ferdig stylede komponenter. Grunnen til dette valget kom av at oppdragsgiver var veldig klar på at de ville style applikasjonen selv etter sine standarder. Ved å bruke dette bibliotek har gruppen raskt og enkelt kunne lage nye komponenter og implementere de i brukergrensesnittet. Det vil også være enkelt for oppdragsgiver i etterkant å implementere sin styling ved å legge til en style variabel i de forskjellige komponentene, som vil overstyre det som kom fra "Semantic UI".

4.3.2 Navigasjon

Navigeringen i applikasjonen blir utført ved hjelp av React Router. Dette er et React bibliotek som gjør det mulig å navigere mellom komponentene i en "single-page" applikasjon, uten at siden lastes inn på nytt.

Oppsettet for dette i denne applikasjonen er vist på bilde under.

```
<Fragment>
    <ModalContainer/>
    <ToastContainer position={"bottom-right"}/>
    <Route path='/' exact component={HomePage}/>
    <Route path={'/(.+)'} render={() => (
        <Fragment>
            {LogiedInuser && <NavBar/>}
            <Container style={{marginTop: '7em'}}>
                <Switch>
                    <AdminRoute path='/admincontroller' exact component={AdminCtrl}/>
                    <AdminRoute path='/users' exact component={PesonellDashBoard}/>
                    <MyProfilePrivateRoute path='/users/:id' exact component={UserProfile}/>
                    <PrivateRoute path='/group/:id' exact component={GroupDetails}/>
                    <AdminRoute path='/organisation' exact component={OrganisationDashBoard}/>
                    <AdminRoute path='/organisation/:id' exact component={OrganisationDetails}/>
                    <AdminRoute key={location.key} path={['/creategroup', '/managegroup/:id']} exact
                           component={GroupForm}/>
                    <AdminRoute key={location.key} path={['/createorganisation', '/manageorganisation/:id']} exact
                           component={OrganisationForm}/>
                    <PrivateRoute key={location.key} path={['/createUser', '/manage/:id']} exact
                           component={PersonelForm}/>
                    <Route path='/login' component={LoginForm}/>
                    <PrivateRoute path='/unauth' component={UnAuth}/>
                    <Route component={NotFound}/>
                </Switch>
            </Container>
        </Fragment>
    )}/>
</Fragment>
```

Figure 10. React router oppsett.

I tillegg til å bruke Route-komponenten som kommer med React Router, har det også blir skrevet noen variasjoner av denne for å sikre tilgangen til ulike komponenter. Dette har blitt gjort for å unngå at brukere skal kunne navigere direkte til en komponent de ikke skal ha tilgang til. Et eksempel på hvordan dette er gjort er gitt ved AdminRoute på bildet under (React Router).

```
const AdminRoute: React.FC<IProps> = ({component: Component, ...rest}) => {
    const rootStore = useContext(RootStoreContext);
    const {isLoggedIn, LogiedInuser} = rootStore.userStore
    return (
        <Route
            {...rest}
            render={(props => (!isLoggedIn && LogiedInuser!.roles[0].name ==='Superuser') ?
                <Component {...props}/> : <Redirect to={'/unauth'}/>)}
        />
    );
}
```

Figure 11. AdminRoute eksempel.

4.4 Responsivitet

For at det skal være så enkelt som mulig for brukerne å gå inn og oppdatere sin egen informasjon har et krav til applikasjonen vært at den skal være responsiv. Dette vil si at den skal fungere like godt uavhengig av om man er på en pc, nettbrett eller mobil. Dette

er løst ved bruk av Grid-, Column- og Responsive komponentene som kommer med "Semantic UI". Dette har gjort det enkelt å vise elementer avhengig av vidden på den skjermen man er på. Muligheten til å spesifisere hvor mye av skjermen en kolonne skal ta opp, basert på hvilken skjermstørrelse man bruker, har også vært en del av det responsive designet. Under vises det hvordan dette fungerer på navigasjon baren.

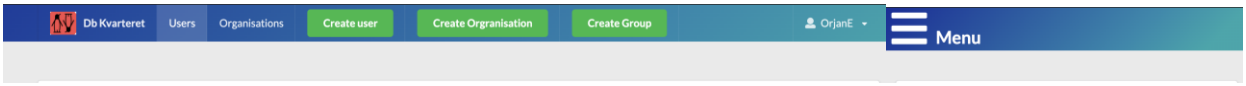


Figure 12. og 13, Responsivitet eksempel.

4.5 Mobx

For å styre den globale staten i applikasjonen har det blitt valgt å bruke MobX. I dette kapittelet blir det sett nærmere på hvordan dette har blitt implementert.

4.5.1 Global state

Det har blitt valgt å bruke en global state i applikasjonen av den grunn at det gjør det enkelt å få tilgang til "state"-data, uten å måtte sende dataene gjennom alle nivåene i komponent treet som props. Det gjør at en komponent kan spørre den globale staten direkte om å få tilgang til de dataene den trenger. Dette minsker kompleksitet når det kommer til håndtering av "state", og gjør koden mer ryddig og oversiktlig å lese. Dette er mye enklere enn hvordan data normalt deles i en react app, "top-down".

Figure 14. Forhold mellom global state og brukergrensesnittet.

Bildet over viser forholdet mellom komponentene i brukergrensesnittet og den globale staten. For å gjøre ting mer oversiktlig har man delt opp staten i forskjellige deler ut ifra hvilken del av applikasjonen den omfatter. Et typisk eksempel på hvordan dette har blitt gjort i denne applikasjonen, er at man har separert staten som omhandler bruker, organisasjon og gruppe. Det har også blitt opprettet et mellomledd mellom den globale staten og brukergrensesnittet som man har kalt root-Store. Root-store er en JavaScript-klasse med referanse til de forskjellige delene i state-Store, som ved opprettelse benytter seg av Reacts context for å gjøre deling av state mulig.

4.6 Kobling mellom brukergrensesnittet og serversiden

Dette kapittelet forklarer hvordan koblingen mellom brukergrensesnittet og serveren er håndtert i denne applikasjonen.

4.6.1 Axios

Axios har blitt brukt for å gjøre HTTP requests mot APIet over JavaScript interfacet fetch. Valget av Axios har blitt gjort på bakgrunn av at det støtter et større utvalg nettlesere, samt at det støtter en enkel måte å avskjære HTTP pakker, for å undersøke og gjøre endringer på disse (Axios).

4.6.2 Oppsett

I prosessen med å hente data fra serveren, og vise disse i brukergrensesnittet, er det tre deler som binder det hele sammen. Disse er API kontrolleren på serversiden, agent filen på klientsiden og MobX.

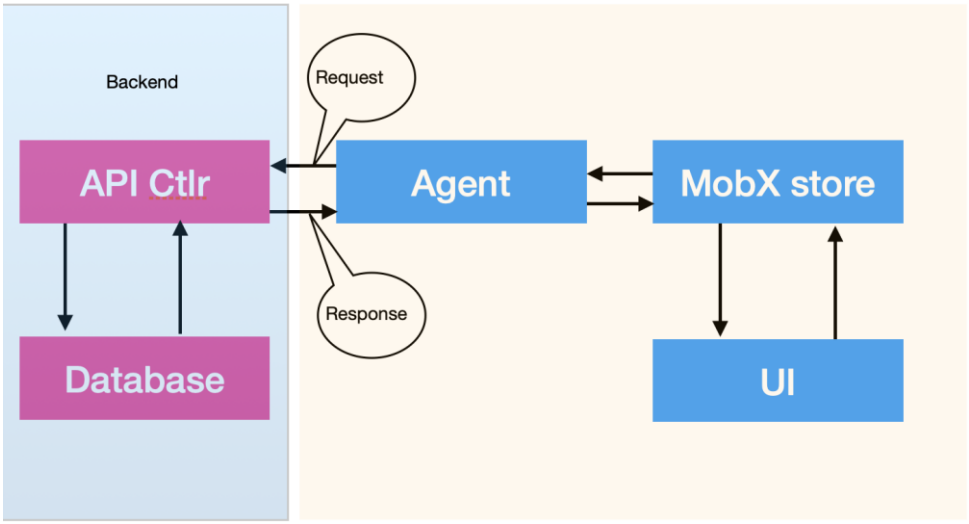


Figure 15. Kobling mellom brukergrensesnitt og server side.

API kontrolleren ("API Ctlr") sin oppgaver og motta "request"-er fra klientsiden, for så å delegere "request"-ene til den logikken på serversiden som kan håndtere disse. Den er også ansvarlig for å sende respons opp til MobX store når requestene er håndtert. Agenten sitter mellom serveren til applikasjonen og MobX storen hvor den globale staten blir lagret og er en samling av alle Axios kallene som blir gjort mellom klienten og serveren. Måten det hele fungerer på er at dersom brukergrensesnittet prøver å hente data i MobX store som ikke ligger i den, så går Mobx gjennom agenten og gjøre et kall til serveren om å få tilsendt den dataen man ønsker. Dataene blir så lagret i den globale staten slik at man slipper å gå gjennom samme prosess neste gang brukergrensesnittet ønsker denne dataen. Dette med forbehold at man ikke laster applikasjonen på nytt i mellomtiden.

4.7 Backend

I dette kapittelet blir det forklart hvordan serversiden av applikasjonen er satt opp. Det vil forklares hva som inngår i de forskjellige delene, samt hvordan disse henger sammen.

4.7.1 Arkitektur

Som det ble lagt frem i punkt 3.1.5 så skulle arkitekturen til applikasjoner bygge på prinsippet "Model View Controller", og det er dette som har vært i tankene når applikasjonen har blitt bygget. Under vises det hvordan oppsettet har blitt, med de forskjellige delene, og sammenhengen mellom disse.

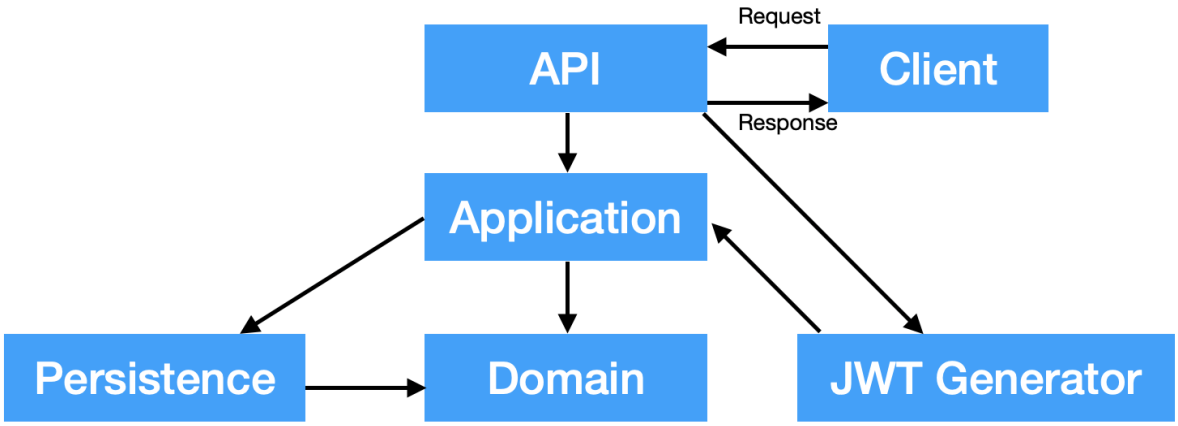


Figure 16. Server arkitektur.

Videre i dette avsnittet blir det gått nærmere inn på hva som inngår i de forskjellige delene. Det har vært viktig med oppdeling av prosjektet, slik at hver del av backend har sin egen rolle. Dataen blir da sendt mellom modulene, og hver modul håndterer sin del.

API

API-prosjektet er kontrolleren på serversiden, og denne har ansvar for å motta HTTP forespørsel fra klient siden, og sende HTTP respons tilbake. Dette er "dum" kontroller, med ingen logikk skrevet inn i seg. Denne må dermed videredelegere forespørsler fra klienten til Application-prosjektet, hvor logikken ligger. Hvert API-kall har en [Http] tag på toppen, som sier om det er en "Get", "Post", "Delete" eller "Put" forespørsel. Her blir selve stien deklarert. I eksempelet under er {id} satt inn, og det vil si at det som kommer inn i denne forespørselen, blir gjort om til en id variabel.

```
        [HttpDelete("{id}")]
```

```
public async Task<ActionResult<Unit>> Delete(Guid id)
{
    return await Mediator.Send(new Delete.Command { Id = id });
}
```

Table 4. Kodeeksempel API siden.

Application

Application prosjektet inneholder all logikken på serversiden og er den delene som håndterer alle forespørsler som mottas i API-kontrolleren. Denne delen av applikasjonen inneholder mye kode og for å holde det oversiktlig, er den blitt delt opp i undergrupper basert på hvilke oppgaver som utføres. Dette vil si at logikk angående organisasjonen, gruppe og bruker er fordelt i ulike mapper.

Her er det implementert full CRUD for:

- User: Styring av bruker, og brukerdata.
- AccessGroup: Styring av rollene til hver bruker.
- Card: Styring av internkort.
- Course: Styring av kurs, med mulighet for å legge til og fjerne brukere på kurs
- Dependent: Legge til pårørende på brukere.
- Group: Styring av grupper.
- Organisation: Styring av organisasjoner, og organisasjons-administrator.
- Photos: Styring av alle entiteter med bilde koblet mot seg.
- Tags: Styring av “tagger”, og legge/fjerne tag på brukere.

Hver modul er bygd opp med standard CRUD funksjonalitet. “Create”, “Read”, “Update” og “Delete”. Under er det lagt ved et skjermbilde av gruppe delen av Application. Her ligger det en fil for hver funksjon som kan gjøres på en gruppe, og hver klasse har kun et formål.

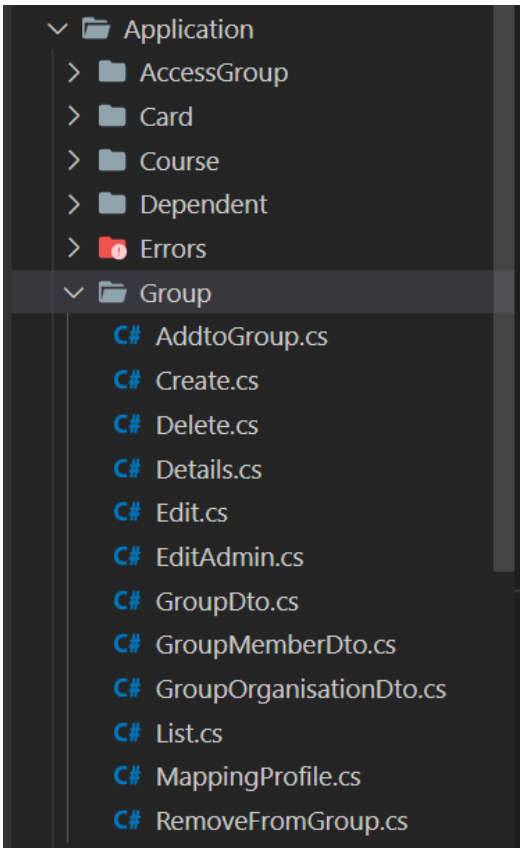


Figure 17. mappe og filstruktur Application prosjektet.

Persistence

Persistence har ansvaret for alt som har med lagring av data i databasen. Her defineres det hvordan tabellene i databasen skal være strukturert samt forholdene mellom de forskjellige tabellene. Her blir altså tabellene opprettet, og Foreign Keys mellom tabeller blir satt inn ved hjelp av kode. Denne koden blir kjørt, og da blir databasen oppdatert med de endringene som er kodet inn. Ved bruk av Persistence og Database Context, er det lett å migrere data mellom SQLite, SQL Server eller Mysql, og Persistence modulen kan opprette migreringer mellom disse.

Domain

Domain inneholder entitetene som brukes i applikasjonen. Det som defineres her er det som blir til tabeller og verdier i databasen når den opprettes ved hjelp av persistence prosjektet. På bildet under er det gitt et eksempel på en entitet i prosjektet, med de variablene som inngår i denne.

```
public class History
{
    public Guid Id { get; set; }
    public string GroupName { get; set; }
```

```
        public string Position { get; set; }
        public string GroupType { get; set; }
        public int Year { get; set; }
        public string Semester { get; set; }
        public virtual ICollection<UserHistory> UserHistory { get; set; }
    }
}
```

Table 5. Kodeeksempel History.

JWT Generator

Dersom en bruker logger inn i applikasjonen blir det produsert en JSON Web Token, JWT for kort. Denne "token"-en lagres lokalt i nettleseren til brukeren, i form av en streng, helt til brukeren logger ut, eller sesjonen utløper. Denne strengen blir brukt til å sjekke om brukeren fortsatt er pålogget, og at identiteten til brukeren er korrekt. JWT er en streng som kan se slik ut:

```
eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJuYW1laWQiOiJvZGEiLCJuYmYiOjE1ODk0NzM2NzAsImV4cCI6MTU5MDA3ODQ3MCwiaWF0IjoxNTg5NDczNjcwfQ.6iNzFJYXVddnRjGGcqKmegwFchBMxKt8X0okxGkJ5S4q5l1qvRMfrbWbImjwTSdxGga1cuS9NpO8Rextu_0oZg
```

Når denne koden blir sendt til API-et, kan den sjekke om denne er lagret på noen av brukerne, og dermed verifisere at den tilhører denne gitte brukeren. Dette gjøres istedenfor at brukernavn og passord blir sendt til API-et hver gang brukeren skal identifisere seg (JWT).

4.8 Feilhåndtering

I dette kapittelet sees det nærmere på hvordan feilhåndtering behandles i applikasjonen.

4.8.1 React toastify

Toastify er en pakke som installeres ved hjelp av npm og gjøre det enkelt å vise varsler på klient siden. Et eksempel på hvordan det ser ut er vist under. Om det kommer feilmeldinger fra frontend eller backend, blir de som er relevante for brukeren vist i en "toast"-melding (React Toastify, 2020).

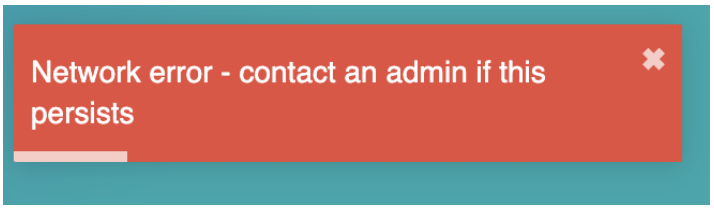


Figure 18. React toastify eksempel.

4.8.2 Axios interceptors

Feil som oppstår i kommunikasjonen mellom klient og server blir fanget opp i agent filen ved hjelp av axios interceptors. Her sjekker man status på forespørselen fra klienten, eller svaret fra serveren, før det sendes videre (Axios).

Interceptor request

I forespørslene mot serveren sjekkes det om brukeren har en JWT Token lagret lokalt i nettleseren. Dersom han har det, blir denne lagt til i "Authorize"-headeren som blir sendt opp til serveren. Har brukeren ikke en token lagret, vil det bli returnert en "reject error".

Interceptor response

I responsen fra serveren er det en del flere feilmeldinger vi kan gjøre sjekker på. Det denne applikasjonen sjekker per dags dato er Network, 404, 400 og 500 feilmeldinger og responsen på disse er enten å sende bruker til en "not found"-komponent, eller skrive ut en toast med hva som har gått galt. Slik det er satt opp i dag er det veldig enkelt å legge til flere sjekker for andre feilmeldinger også, men grunnet ingen tilbakemelding fra oppdragsgiver på hva de ønsker at skal håndteres, er det kun disse applikasjonen har støtte for ved prosjektslutt.

4.8.3 Feilmeldinger i MobX store

En annen plass feil blir håndtert er i MobX storene, hvor feil som oppstår blir håndtert ved hjelp av try/catch og vist til brukeren ved hjelp av Toastify. Typiske feil som kan oppstå og bli fanget opp her er relatert til forms og bruker input.

4.9 Testing

Testing av produktet er en viktig del i prosjektprosessen for å sikre seg god kvalitet på det som blir produsert samt at det samsvarer med kravspesifikasjonen over det oppdragsgiver ønsker seg.

4.9.1 Brukertesting

Det å skrive tester til dette prosjektet har blitt nedprioritert grunnet tidsrammen og arbeidsmengden som skulle fordeles på to personer. For brukergrensesnittet har all testing vært basert på brukertesting, hvor brukerne som regel har vært venner og bekjente av gruppemedlemmene. Det ideelle vil så klart vært å bruke medlemmer fra Kvarteret i denne form for testing, men kvarteret har valgt å ikke følge opp det som har blitt avtalt, så gruppen har måtte finne alternative måter å gjøre testingen på.

I sluttfasen av prosjektet har en talsperson fra samfunnet i Bergen blitt mer involvert i prosjektet. Dette har ført til at gruppen har kunnet vist frem og fått tilbakemeldinger på arbeidet som har blitt utført, men grunnet at dette ikke skjedde før i sluttfasen av prosjektet har det vært vanskelig å gjøre store endringer basert på tilbakemeldingene.

4.9.2 Postman

Teste av serversiden har det blitt gjort ved forhåndsdefinerte forespørsler i Postman, dette for å gjøre det raskere å teste implementasjon på serversiden uten å manuelt måtte skrive kall inn hver gang. Disse kallene har autorisering implementert, ved å kjøre et kall mot '/api/users/login', for å generere en JWT. Denne blir så lagret i Postman, slik at den kan brukes til autorisering mot andre API-kall.

4.10 Arbeidsfordeling

Planen for arbeidsfordelingen i prosjektet var i utgangspunktet slik at en av medlemmene skulle ha hovedansvar for serversiden mens den andre skulle ha ansvaret for brukergrensesnittet. I praksis har det ikke blitt helt slik, hvor det heller har vært en mer løs arbeidsfordeling hvor oppgavene har blitt fordelt underveis ut fra hvilken fase prosjektet var i. Dette har fungert veldig godt, og har en av gruppemedlemmene stått fast på en oppgave har den andre vært behjelpelig for å løse problemet.

5 EVALUERING

I dette kapittelet blir det tatt opp hvordan evalueringen i dette prosjektet har blitt gjennomført. Det blir beskrevet hvilke metoder som er brukt, hva som er blitt evaluert samt resultatene av evalueringen.

5.1 Evalueringsmetode

5.1.1 Intern evaluering

Produktet

For å sikre at produktet har blitt utviklet riktig har man brukt ulike former for testing for å sikre dette. Det som inngår i testing har vært interaksjon testing, unit testing, testing av sikkerhet og brukertesting. Resultatene fra disse har blitt evaluert av gruppen og endringer i applikasjonen har blitt utført på bakgrunn av det man har konkludert med etter de ulike testene.

Metodene

Som en del av utviklingsmetodikken scrum har det vært utført en evaluering av arbeidet som har blitt gjort etter hver sprint. Dette for å sikre kvaliteten på arbeidet som ble utført under sprinten samt fange opp problemer medlemmene har støtt på. Måten dette har blitt gjort på er gjennom møte mellom gruppemedlemmene, hvor man systematisk har gått gjennom oppgavene som har blitt utført under sprinten og diskutert rundt arbeidet. Dersom noe ikke tilfredsstiller kravene som har blitt satt har dette blitt notert og en oppgave blitt opprettet i kommende sprint, hvor man skulle rette opp i dette. Oppgaver som har blitt opprettet under denne evalueringen har fått førsteprioritet i kommende sprint, slik at man blir ferdig med det man har startet på og ikke begynner på noe nytt før man oppfyller kravene.

Ressurser

En del av den interne evalueringen som har blitt gjennomført har bestått av å kartlegge nyttigheten av de ressursene gruppen har valgt å bruke i denne prosessen. Det gruppen har sett på her er hvorvidt de eksterne personene gruppen har valgt å bruke i denne prosessen har gitt nyttige tilbakemeldinger på arbeidet som har blitt gjort.

5.1.2 Evaluering gjennom testing

Brukertesting

Det ble tidlig bestemt at testingen i dette prosjektet skulle gjennomføres som brukertesting, etter hvert som funksjonalitet ble implementert. Gjennom prosjektperioden var planen å bruke personer knyttet til oppdragsgiver som testobjekt. Dette har ikke vært gjennomførbart før i sluttfasen av prosjektet og gruppen har derfor måttet bruke eksterne personer for testing, underveis i prosjektet.

Etter at utviklingsfasen ble avsluttet publiserte gruppen applikasjonen på Microsoft Azure, som er en skyløsning for publisering av applikasjoner. Det ble også laget en guide sammen med et spørreskjema som ble sendt til oppdragsgiver, med forespørsel om de kunne gå inn og testet applikasjonen og svarte på spørsmålene. Gruppen fikk da for første gang konkrete tilbakemeldinger fra oppdragsgiver, men disse har ikke vært mulig å gjøre noe med, siden utviklingsfasen da alt var avsluttet.

Integrasjon- og Unit Testing

Integrasjonstester er blitt gjort med Postman, som er et verktøy for å dokumentere og teste API. Med dette verktøyet kan du kjøre kall mot API-et, og bekrefte at du får tilbake den dataen du trenger. Med Postman kan du også lage egne tester for hvert kall, så du kan automatisk sjekke om verdiene stemmer. På bildet under ser du et eksempel på hvordan du sjekker at responsen har statuskode 200 OK.

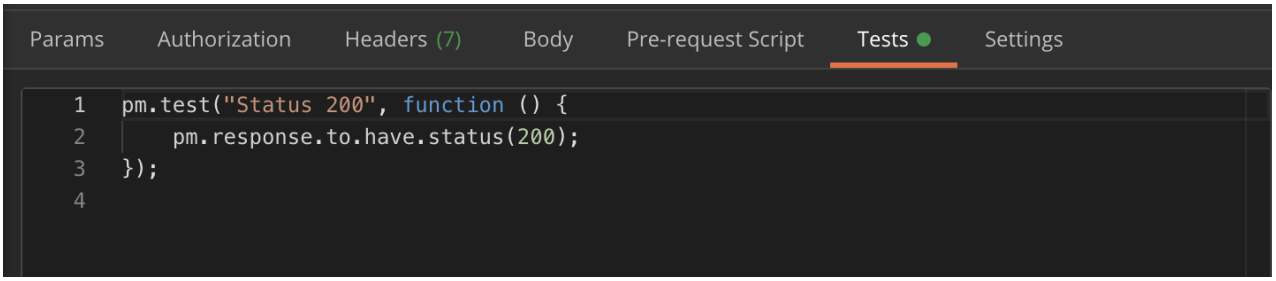


Figure 19. Postman test eksempel.

Det er skrevet tester for samtlige API-kall, og disse er brukt for å kontrollere at API-et fungerer slik det skal (Postman, 2020).

Sikkerhet

Sikkerhet er høyt prioritert for å sikre personvern og data på avveie. Siden starten av prosjektet, har vi tenkt på sikkerhet, og hvordan vi skal teste for dette. Det har blitt brukt en guide med mange tips til hvordan lage applikasjonen sikker, og dette har vært til stor nytte når det kommer til sikkerhet. I .NET Core finnes det pakker som håndterer en del av sikkerhetsmekanisme ved en web applikasjon. For frontend, er det viktig at de riktige "HTTP header"-ene er lagt inn, slik at Chrome og andre nettlesere vet hvordan

applikasjonen skal fungere. Dette er gjort med en "middleware" som setter "Content-Security-Policy" (Security Headers).

5.2 Evalueringsresultat

5.2.1 Resultat fra internevaluering

Produkt

Når det kommer til resultatene fra de ulike formene for testing som har blitt utført har gruppen stått alene når de har vurdert disse resultatene. Ideelt hadde gruppen ønsket at kontaktperson hos oppdragsgiver skulle tatt del under denne evalueringen. Dette for å sikre at valgene gruppen tok var med på å dytte prosjektet i riktig retning, med tanke på hva oppdragsgiver var ute etter å få utviklet. Dessverre har ikke dette vært tilfelle og gruppen har måtte ta valget om å fortsette utviklingen basert på egne meninger om hva applikasjonen skulle være under mesteparten av prosjektet. Resultatet av dette er at man nå står igjen med en applikasjon som er gruppemedlemmenes tolkning av kravspesifikasjonene og problemstillingen.

Metodene

Når det kommer til metodene som her blitt brukt i evalueringen har gruppen hatt stor nytte av de valgene man har tatt.

Sprint reviews etter hver sprint har vært noe gruppen har gjennomført hver uke, sammen. I utgangspunktet hadde man sett for seg at kontaktpersonene hos oppdragsgiver skulle ta del i denne prosessen. I starten av prosjektet var det gjort forsøk på å møte oppdragsgiver annen hver uke for å gå gjennom arbeidet som var gjort. Under disse møte var det minimalt med tilbakemeldinger gruppen satt igjen med, og det virket som om kontaktperson var lite interessert i arbeidet som hadde blitt utført. Etter en liten stund sluttet oppdragsgiver å møte på disse møtene, og gruppen valgte å gjøre evalueringen av arbeidet på egenhånd for å opprettholde fremgangen i prosjektet.

Ressurser

Det er tidligere nevnt at det gruppen har sett på som ressursen i evalueringsfasen har omfattet personene som har vært involvert, samt plattformen man har brukt i testfasen. Under planleggingen var det tenkt å bruke personer fra oppdragsgiver som testobjekter, men dette har blitt vanskelig grunnet oppdragsgivers mangel på oppfølging. Dette førte til at gruppen ganske tidlig i prosjektet måtte ta en evaluering på om man skulle fortsette med den opprinnelige planen angående testobjektene, eller gjøre en endring. Det som da ble bestemt var at man skulle bruke venner og bekjente som testobjekt istedenfor, frem til oppdragsgiver valgte å ta mer del i prosjektet igjen. Noe gruppen

lykkes med og gjorde at man fikk tilbakemeldinger på arbeidet som ble utført gjennom hele prosjektet.

En annen ting man har måtte vurdere og gjort endringer på, er plattformen brukertestingen har blitt utført på. Det var her i utgangspunktet tenkt å arrangere møter, og utføre testingen lokalt sammen med testobjektene. Dette var noe man måtte gå vekk fra når man så at landet ble stengt ned grunnet covid-19. Resultatet av dette er at man har utført dette over Zoom (Zoom Video Communications Inc, 2020) ved hjelp av skjermdeling. Gruppen har også publisert applikasjonen på nett, hvor testobjektene kunne logge på og teste.

Konklusjon av intern evaluering

Resultatene fra den interne evaluering gjort under prosjektet har vært med på å håndtere problemene som har oppstått underveis, spesielt med tanke på valgene gruppen gjorde når kommunikasjonen med oppdragsgiver ikke fungerte. At man hadde planlagt å ta stilling til hvor godt evalueringsplanen fungerte, var med på å gjøre det veldig enkelt å håndtere de endringene som oppsto underveis.

Måten ting har blir gjort på når det kommer til evaluering av hverandres arbeid har også fungert veldig bra. Det å jobbe i iterasjoner hvor man har mulighet til å gå tilbake og endre på ting som ikke har vært helt optimalt, har vært noe prosjektet har nytt godt av.

5.2.2 Resultat etter testing

Brukertesting

Resultatene gruppen har fått etter brukertesting har vært til stor hjelp når det kommer til utviklingen av brukergrensesnittet i applikasjonen. Spesielt med tanke på brukervennlighet og sikkerhet har dette hatt mye å si for hvordan applikasjonen har blitt ved prosjektslutt. Brukertestingen har dessverre ikke gått opp mot Kvarteret, men mot venner og bekjente. Disse har da prøvd ut applikasjonen, og klikket rundt på de forskjellige elementene. De har kommet med kritikk på både brukergrensesnitt og funksjoner.

Helt på slutten av prosjektet fikk vi hatt en brukertest med Kvarteret, der de fikk prøve systemet selv. Oda fikk en link til en Google Forms, og denne kunne hun dele med andre på Kvarteret. Oppgavene og spørsmålene oppdragsgiver fikk, var en kombinasjon av rangering 1-5 og kommentarer. Resultatet fra dette er vist under.

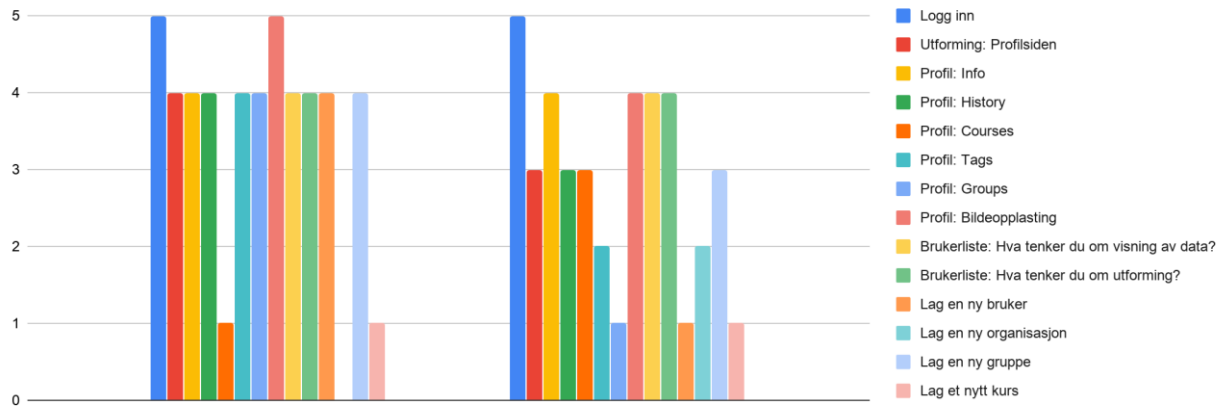


Figure 20. Resultat fra brukertest.

Oversikten over benytter en rangering fra 1-5. Man kan av dette se at det som regel er gode tilbakemeldinger på det som har blitt testet. Det kommer likevel frem at ikke alt har fungert på en intuitiv måte, hvor man kan se at "Lag et nytt kurs" har fått en dårlig tilbakemelding.

Integrasjon- og Unit testing

Integrasjon og unit testene er under sortert inn i de ulike kravene fra oppdragsgiver. Her er det forklart hvordan det er testet, for å sjekke funksjonene, og eventuelt om funksjonene ikke er implementert. Disse kan du finne igjen i 2.2.1 i rapporten, under "Funksjonelle krav". Hvilke krav hver test har som mål å teste, er beskrevet ved hjelp av FKxx.

FK1 & FK10

Hele backend av applikasjonen er et rent API, slik at det ikke er tilknytning mellom frontend og backend, annet enn att frontend henter data fra APIet til backend.

Testdata blir opprettet i Persistence prosjektet, under SeedData funksjonen. I forklaringen under tar vi utgangspunkt i "Cards" modulen, men dette er gjort på alle de forskjellige modulene.

```
if (!context.Cards.Any())
            {
                var cards = new List<Card>()
                {
                    new Card
                    {
                        CardNumber = "1198",
                        Created = DateTime.Now.AddMonths(-2),
                        AppUser = context.Users.FirstOrDefault()
                    },
                    new Card
                    {
                        CardNumber = "1060",
                        Created = DateTime.Now.AddMonths(-4),
                        AppUser = context.Users.FirstOrDefault()
                    },
                };

                context.Cards.AddRange(cards);
                context.SaveChanges();
            }
```

Table 6. Kodeeksempel SeedData.

Denne dataen blir da lagt inn i databasen når prosjektet blir kjørt, om den ikke finnes der fra før.

```
SELECT * FROM `Cards`;
```

#	Id	CardNumber	Created	AppUserId
1	EDB9CCAD-AFEB-4255-99E4-78BA8FE1BC68	1060	2020-01-11 21:12:08.7118331	c
2	91D43D9D-5EE2-47A9-BAFD-3C6FBF121293	1047	2020-02-11 21:12:08.7101129	c
3	DBC9381B-5E1E-49F0-B1F0-81D13E457B09	14	2020-04-11 21:12:08.7084017	c
4	A8ADF778-89D4-46E3-87A8-4C7D035717DD	1198	2020-03-11 21:12:08.7011822	c

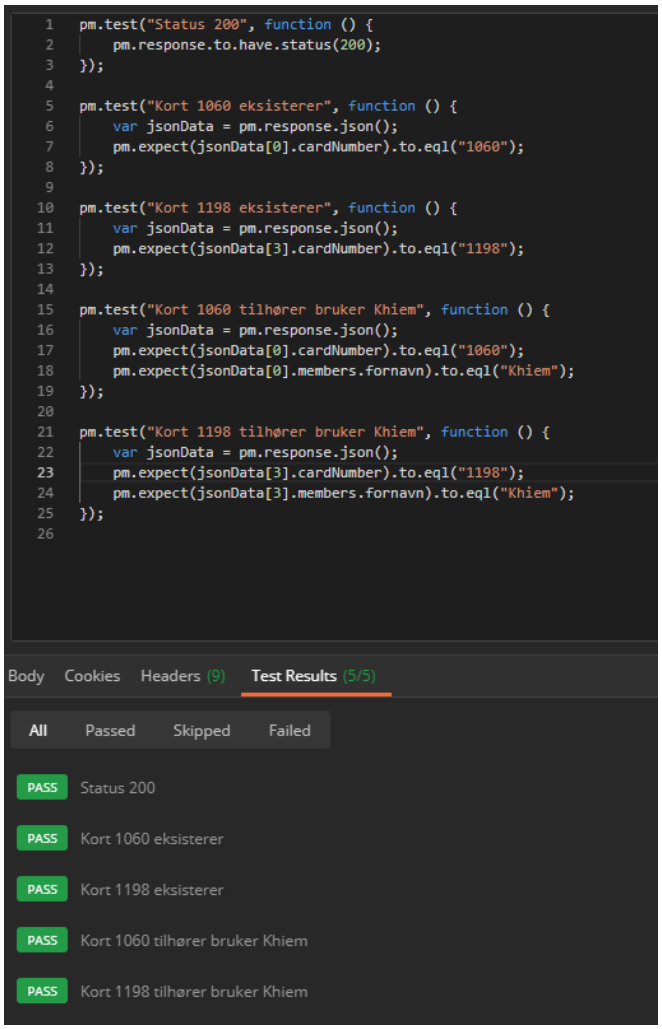
Figure 21. Eksempel av data lagret i databasen.

Denne dataen blir da tilgjengelig i APIet, under "http://localhost/api/cards", som er et API-kall for å hente ut en liste over alle kort.

```
[
    {
        "id": "edb9ccad-afeb-4255-99e4-78ba8fe1bc68",
        "cardNumber": "1060",
        "created": "2020-01-11T21:12:08.7118331",
        "members": {
            "id": "c",
            "username": "khiem",
            "isAdmin": false,
            "fornavn": "Khiem",
            "etternavn": "Bruker",
            "email": "epost@epost.test",
            "phoneNumber": "90807060",
            "kjonn": "man",
            "workstatus": "active",
            "created": "2020-05-11T21:12:08.4030137",
            "dateOfBirth": "2018-05-11T21:12:08.4053598",
            "streetAddress": "Olav Kyrres gate 1",
            "areaCode": "5004"
        }
    },
    {
        "id": "91d43d9d-5ee2-47a9-bafd-3c6fbf121293",
        "cardNumber": "1047",
        "created": "2020-02-11T21:12:08.7101129",
        "members": {
            "id": "c",
            "username": "khiem",
            "isAdmin": false,
            "fornavn": "Khiem",
            "etternavn": "Bruker",
            "email": "epost@epost.test",
            "phoneNumber": "90807060",
            "kjonn": "man",
            "workstatus": "active",
            "created": "2020-05-11T21:12:08.4030137",
            "dateOfBirth": "2018-05-11T21:12:08.4053598",
            "streetAddress": "Olav Kyrres gate 1",
            "areaCode": "5004"
        }
    },
```


Figure 22. Resultat fra et api kall til databasen.

Deretter blir det testet om responsen er lik det som ligger i Seed dataen.



Figur 22.1 Test resultat api kall

FK2 & FK6

Test av søk og filtrering er gjort manuelt, ved å implementere, og deretter teste. Vi tester søk og filtrering i lag med eksportering av CSV. Vi tester ved å søke etter en bruker, og ser om vi får tilbake riktig bruker.

I bildet under er det søkt etter “olav kyrre”. Siden du ser at alle bor i Olav Kyrres Gate, vil dette søke treffe alle.

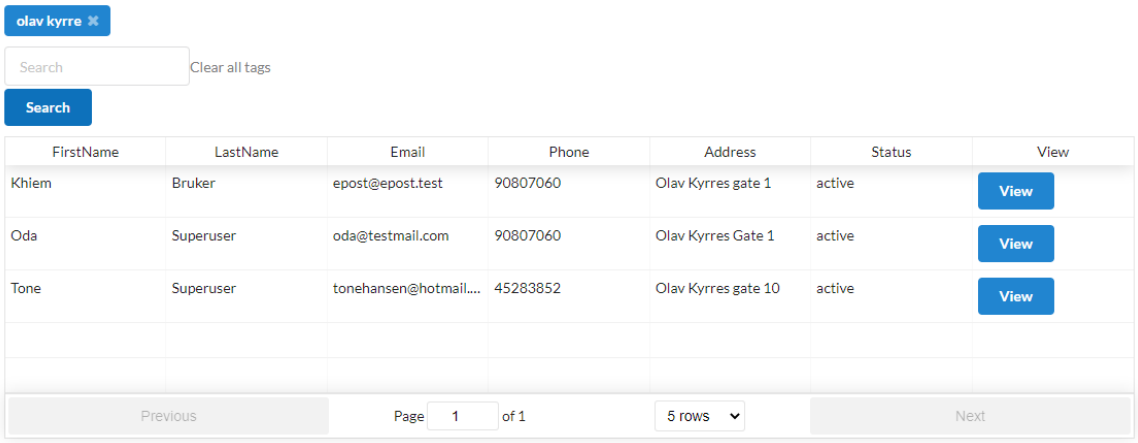


Figure 23 Eksempel fra applikasjon søk 1.

Videre kan vi legge til en søkestreng til, slik at søket blir mindre. Under har vi lagt på søkestrengen "superuser". Her ser vi at "Khiem Bruker" forsvinner, siden han ikke treffer på etternavnet "Superuser". Slik har vi testet søkefunksjonen, og prøvd ut ulike søkestrenger, og sett at vi får tilbake de dataene vi ønsker.

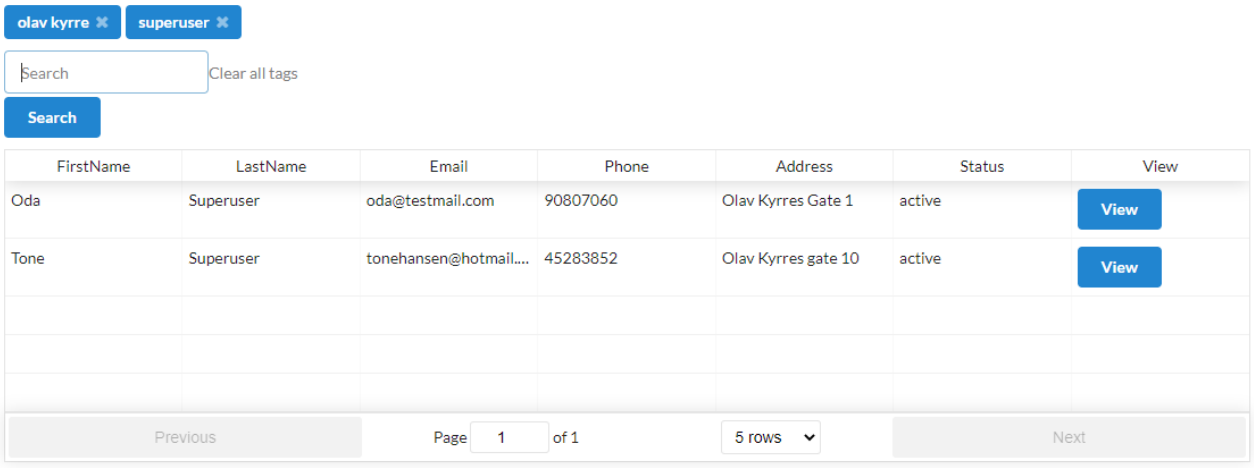


Figure 24, Eksempel fra applikasjon søk 2.

Under søkestrengen har vi også en eksportering funksjon, som henter ut dataen vi har søkt på, og laster den ned i CSV format.

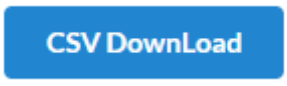


Figure 25 Eksempel fra applikasjonen CSV Download.

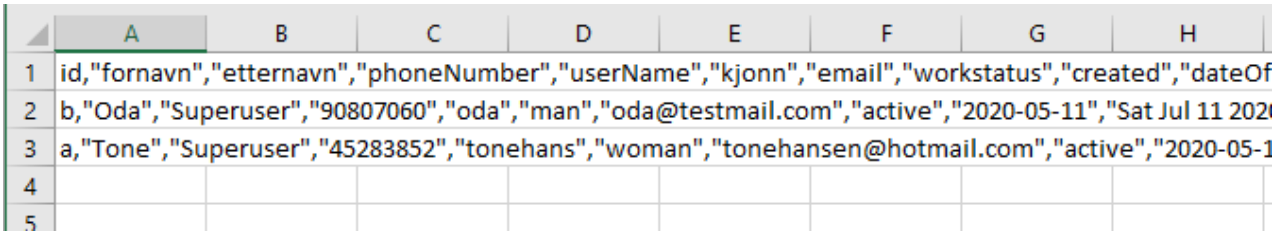


Figure 26, Data eksportert fra applikasjonen 1.

Det er deretter mulig å gjøre dataen om fra CSV til XLSX, som er Excels vanlige format, og da vil dataene være mere lesbare.

	A	B	C	D	E	F	G
1	id	fornavn	etternavn	phoneNumber	userName	kjonn	email
2	b	Oda	Superuser	90807060	oda	man	oda@testmail.com
3	a	Tone	Superuser	45283852	tonehans	woman	tonehansen@hotmail.com
4							
5							

Figure 27, Data eksportert fra applikasjonen 2.

FK3 & FK4

Generering av passord er ikkje gjort. Planen har alltid vært å gjøre dette før prosjektet vart levert til Kvarteret, og gjort etter deres krav. Dette har vært vanskelig, og vi har ikke helt fått klart hvordan vi skal gjøre det med opprettelse av passord til nye brukere. Du kan per i dag kun opprette passord i SeedData funksjonen.

Generering av passord er da heller ikke gjort via e-post.

FK5

2-faktor autentisering er ikke implementert, da vi ikke fikk tid til det. Denne var også litt avhengig av passord generering.

FK7

Varsling via e post krever en form for epostklient. Dette er ikke implementert da tiden ikke strakk til.

FK8

API-kallene vi gjør er fleksible, i den form at du kan hente ut data ved å kalle hver funksjon. Det eneste som kunne vært mer fleksibelt, er antallet vi henter ut. Om det nå ligger 300 brukere i systemet, så henter api-kallet ut en liste over alle brukere. Vi kunne lagt ved en begrenser, som begrenser antallet vi får tilbake.

FK9

Tilgangsstyring er oppfylt, og alle brukere i systemet får en rolle. Når en bruker blir opprettet, får den "Bruker" rollen. På frontend er dette låst ned, så de tre ulike rollene har tilgang til ulike sider. Der vil du få feilmeldinger om du ikke har tilgang til funksjonen.

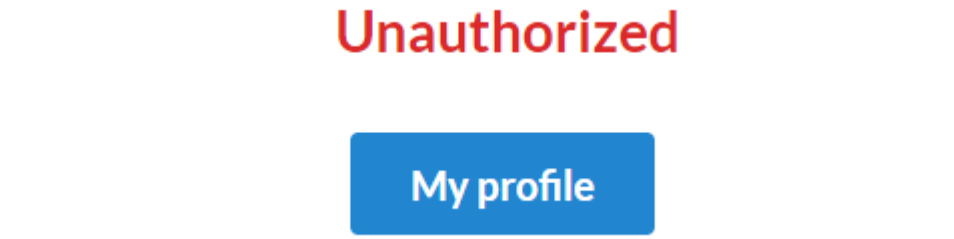


Figure 28, Eksempel, tilgang nektet.

På backend er det også satt opp nedsperringer ved bruk av Authorize atributten i controlleren. Denne er ikke ferdig implementert, og det mangler å sette riktig atributt på riktig funksjon. Det er noen tilfeller der denne ikke fungerer optimalt. Ved å nevne et eksempel, så fungerer ikke denne atributten når en bruker kun skal ha mulighet til å endre sin egen informasjon. Da er det ikke rollespesifikke tilganger, men brukerspesifikke tilganger som trengs. Denne funksjonen ble startet på, men det ble ikke helt ferdig.

```
[Authorize(Roles = "Bruker")]
[HttpDelete("{id}")]
0 references
public async Task<ActionResult<Unit>> Delete(Guid id)
{
    return await Mediator.Send(new DeleteUser.Command { Id = id });
}
```

Figure 29, kodeeksempel fra API.

FK11

Profilsiden for brukere ble testet ved å enkelt navigere rundt på siden. Alle de øvrige funksjonene er testet i FK10 testen.

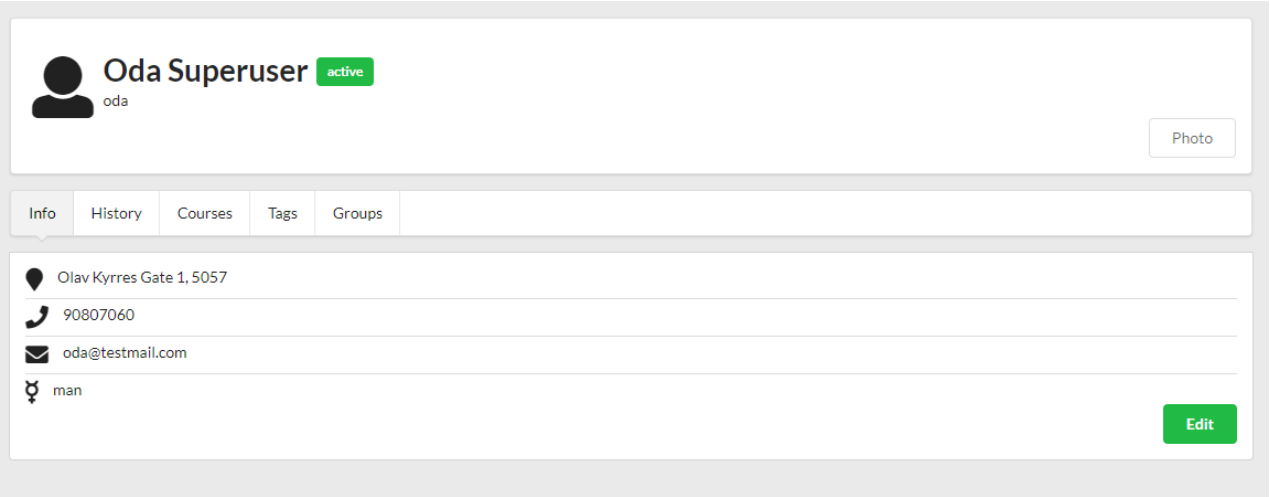


Figure 30, Eksempel fra applikasjonens profilside 1.

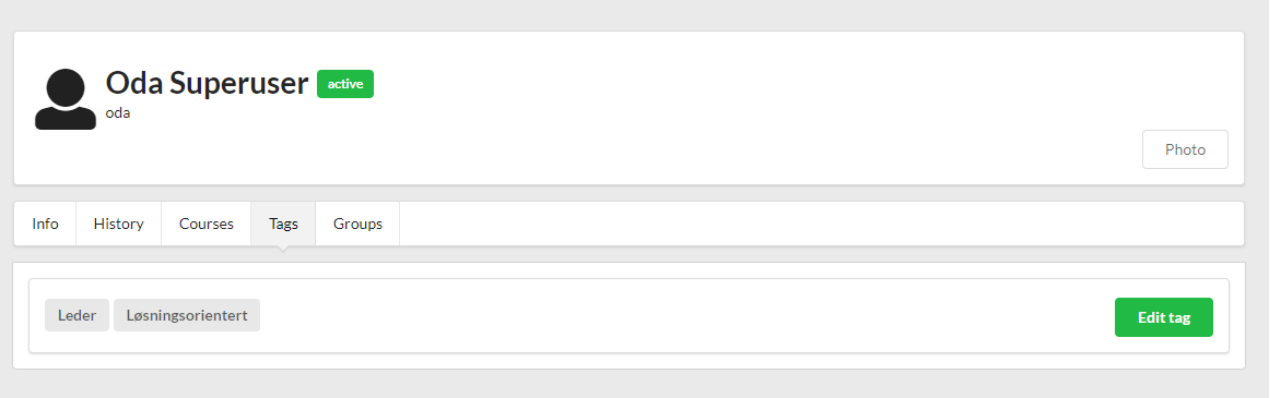


Figure 31, Eksempel fra applikasjonens profilside 2.

Sikkerhet

Testing av sikkerhet er noe som har blitt fokusert på gjennom hele prosessen. Testingen har blitt gjennomført manuelt, og koden har blitt endret på de steder man har funnet sikkerhetshull. Etter endt prosjekt var det også gjennomført manuelle tester av APIet, for å se om det var noe vi hadde oversett. Dette var gjort ved å kjøre API-kall som ikke ville fungere, for å sjekke at man ikke fikk tilgang til informasjon som det ikke skulle være tilgang til.

Etter endt prosjekt vart også alle “request headers” testet. Dette ble gjort med Security Headers. Denne siden sjekker hvilken headers som ligger i responsen fra frontend, og gir en karakter fra F til A. Da vi testet første gang, fikk applikasjonen F. Basert på rapporten vi fikk fra denne siden, ble nye headers lagt til. Ved en ny kjøring fikk applikasjonen B (Security Headers).

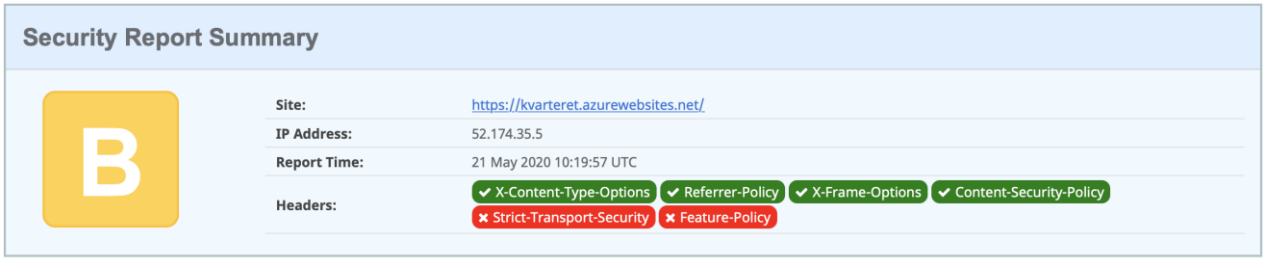


Figure 32. Resultat fra sikkerhetstest av headers.

Konklusjon av testresultatene

Testene som er brukt gjennom hele prosjektet har hjulpet veldig med å holde orden på alle modulene. Med disse testene kunne man enkelt sjekke om hvert API-kall fungerte både under og etter testing. Det hjalp veldig med å holde kontroll på at hele prosjektet fungerte tilfredsstillende etter endringer. Her ble det oppdaget flere feil, og disse ble da utbedret. Det var et ønske om å kjøre interne unit-tester for hver modul, men gruppen tok en beslutning om at tiden ikke strakk til. Kombinert med de andre problemene i prosjektperioden, var nok dette en god beslutning. Integrasjonstestingen av hele applikasjonen var viktigst.

6 DISKUSJON

I dette kapittelet vil det bli diskutert rundt de problemene gruppen har støtt på under prosjektperioden. Det blir også tatt opp litt om hvordan prosessen har vært og hvordan valgene gruppen har tatt, har påvirket resultatet.

6.1 Oppdragsgivers rolle

Da gruppen valgte oppgaven gitt av Kvarteret var det forventet at de skulle være delaktige og behjelpelige med informasjon og data relatert til det som skulle utvikles under prosjektperioden. Dette har i praksis ikke vært tilfelle da det heller har vært svært vanskelig å få data, informasjon og tilbakemeldinger på arbeidet som har blitt utført. Det var også over en lengre periode ikke mulig å få til noe møte med kontaktperson ved Kvarteret, hvor han ved gjentatte anledninger gikk med på å møtes, men lot være å møte opp. Det har ikke blitt gitt noen grunn til hvorfor Det Akademiske Kvarter ikke har overholdt sine oppgaver relatert til prosjektet, men det er tenkelig at covid-19 har vært en faktor, slik som for resten av verden. Gruppen mener likevel at dette ikke kan forsvare mangelen på oppfølging som har vært til tider.
Et resultat av dette er at gruppen internt har måttet evaluere arbeidet som har blitt gjort for så å godkjenne det, for å komme seg videre med utviklingen etter hver sprint. Dette

betyr at applikasjonen som har blitt utviklet er gruppens tolkning av problemstillingen og ikke nødvendigvis akkurat det oppdragsgiver var ute etter.

6.2 Ved prosjektstart

Ved oppstart av prosjektet, ble det avholdt møte med oppdragsgiver for å få en forståelse av hva de ønsket å oppnå med prosjektet. Det som kom frem på dette møtet var at det ikke var et mål å bli ferdig med hele prosjektet under prosjektperioden, men heller at gruppen skulle fokusere på å få til så mye funksjonalitet som mulig. Oppdragsgiver informerte også om at det ikke var nødvendig å fokusere noe særlig på styling av komponentene fordi dette var noe de ønsket å gjøre selv. Det ble også avtalt at gruppe skulle få tilsendt kravspesifikasjoner over hva funksjoner den nye applikasjonen skulle ha samt en oversikt over det systemet som de har i dag. Ut fra første møte virket det som om gruppe sto meget fritt når det kom til valg av teknologi i utviklingsfasen. Gruppen hadde med dette i utgangspunktet planlagt å bruke kunnskapene de hadde innenfor PHP og React for å utvikle applikasjonen.

Etter oppstartsmøtet satte gruppen i gang med å planlegge arbeidet som skulle utføres basert på det som ble sagt under oppstartsmøtet, selv om en oversikt over dagens system og kravspesifikasjon lot vente på seg. Først etter å ha purret på oppdragsgivere fikk gruppen tilsendt kravspesifikasjonene, men ingen oversikt over hvordan det grafiske grensesnittet så ut i dag. Hoved tingen man fikk ut fra kravspesifikasjonen var at oppdragsgiver ikke ønsket å bruke PHP, grunnet at de følte dette var en utdatert teknologi som ville bli vanskelig å vedlikeholde etter endt prosjekt. Dette betydde at gruppen ble nødt til å bruke en del tid, som kunne blitt brukt til utvikling av funksjonalitet, på å sette seg inn i et rammeverk man ikke hadde vært innom før.

6.3 Under prosjektperioden

Under prosjektperioden har det oppstått en del utfordringer som er verdt å nevne, siden disse har hatt store konsekvenser for sluttresultatet.

6.3.1 Covid-19

Våren 2020 ble spesiell for alle grunnet Covid-19 viruset. Dette har ført til nedstigninger verden over, i et forsøk på å hindre smitten fra å komme ut av kontroll. Dette førte til at gruppen, som i utgangspunktet hadde planlagt sprinter på to uker, valgte å endre dette til en ukes sprinter. Valget har sin bakgrunn i at gruppen ønsket å få tildelt mindre arbeidsmengde per sprint. Dette fordi det kunne bli vanskelig å opprettholde produktiviteten over to hele uker, slik som først planla. Det blir vanskeligere når man

sitter hjemme alene og jobber. Gruppen erfarte at dette var en god løsning som fungerte godt under prosjektet.

6.3.2 Teknologi

Det er nevnt tidligere i denne rapporten at gruppen i utgangspunktet ønsket å utvikle backenden for applikasjonen i PHP, men måtte gå vekke fra dette når oppdragsgiver hadde et annet ønske. Dette førte til at oppgaven ble mer omfattende enn hva gruppen først hadde antatt, grunnet at man måtte bruke mer tid på kunnskapsbygging. Et resultat av dette er at gruppen ikke har rukket å utføre alle de oppgavene de ønsket å utføre ved prosjektstart, siden det rett og slett ikke har vært nok tid.

6.4.3 Oppdragsgiver

I punkt 6.1 nevnes det at gruppen har hatt store problemer når det kommer til kommunikasjon med oppdragsgiver. Disse problemene startet tidlig i prosjektet og det har i alt kun vært holdt to møter mellom opprinnelig kontaktperson ved kvarteret og gruppen. I etterkant av disse to møtene var det en lengre periode ikke mulig å få til noe møte med oppdragsgiver, siden de ikke møtte opp på avtalte møter. Dette var et stort problem da det var blitt lagt opp til at oppdragsgiver skulle godkjenne implementerte funksjoner etter hver sprint, dette for å sikre at det som ble utviklet var det de ville ha.

Etter i overkant av en måned uten oppfølging fra Kvarteret fikk gruppen tildelt en ny kontaktperson som skulle ha ansvar for kommunikasjonen mellom oppdragsgiver og gruppen. Dette var en stor forbedring i forhold til hvordan det hadde vært over en lengre periode. Problemet var at dette ikke skjedde før prosjektet var inne i sluttfasen. Den nye kontaktpersonen hadde heller ikke noen teknisk kunnskap som kunne hjelpe gruppen, og gruppen var fortsatt avhengig av at den tekniske avdelingen ved Kvarteret var behjelpelige med info og data.

Først i mai fikk gruppen tilgang til oppdragsgivers server, med mulighet for å laste opp applikasjonen og teste denne. Dette var også første gang man fikk sett ordentlig på hvordan dagens system fungerte, hvor gruppen fikk en innlogging og kunne utforske dette på egenhånd. Grunnet at denne tilgangen kom så seint, har gruppen måttet utvikle en ny database og publisert applikasjonen for testing på egen server i prosjektperioden.

6.4.4 Konklusjon

For å oppsummere prosjektperioden har denne vært vanskelig. Dette skyldes i hovedsak kommunikasjonsproblemene med oppdragsgiver, noe som har forårsaket mye venting og frustrasjon i gruppen. Denne frustrasjonen har da vært rettet mot de teknisk

ansvarlige ved Kvarteret som ikke har fulgt opp og gitt gruppen det de trenger for å utføre prosjektet.

Under de omstendighetene som har vært har gruppen likevel klart og utvikle en del funksjoner i applikasjonen. Dette mye grunnet god kommunikasjon mellom medlemmene og innstillingen om at man bare må gjøre så godt man kan utfra det man har.

Når det kommer til funksjoner i applikasjonen, hadde det vært ønskelig og blitt ferdig med mer enn hva som er ferdig nå. Det hadde også vært til stor hjelp dersom tilbakemeldinger hadde blitt gitt etter hvert som det ble implementert, noe som hadde gitt bedre flyt i prosjekter og sikret et bedre resultat. Slik situasjonen har vært har gruppen gjort så godt de kan ut fra hvordan de har tolket kravspesifikasjonene og problemstillingen.

6.5 Drøfting av resultatet

Slik applikasjonen er i dag er det en løsning som krever at oppdragsgiver fortsetter å videreutvikle den, dersom de ønsker å ta den ut i produksjon. Produktet er også et resultat av at oppdragsgiver under planleggingsfasen ga uttrykk for at de ønsket å gjøre en del arbeid selv, etter gruppen var ferdig med prosjektperioden. Dette har medført at gruppen underveis ikke har fokusert på at hele applikasjonen skal være fungerende, men heller implementert så mye funksjonalitet som man har rukket. Det at oppdragsgiver skiftet kontaktperson i sluttfasen av prosjektet til en person som helst ville ha et ferdig produkt er ikke noe gruppen har hatt noen mulighet til å ta stilling til.

En viktig ting å få frem når man drøfter resultatet er at gruppen har fått veldig lite hjelp fra oppdragsgiver når det kommer til den dataen man har hatt brukt for i planleggingsfasen. Eksempler på dette er at oppdragsgiver hadde ønsker om at den nye applikasjonen skulle ha mye samme funksjonalitet som det systemet de har i dag. Likevel valgte de å ikke gi gruppen tilgang til, eller noen for av bilder, av dagens system før tre uker før bacheloroppgaven skulle leveres. Oppdragsgiver har også valgt å ikke ta del i testingen eller evalueringen av det som har blitt utviklet før etter at gruppen hadde avsluttet utviklingsperioden. Dette betyr selvfølgelig at det produktet som overleveres fra gruppen til oppdragsgiver ved endt prosjekt er gruppens egen tolkning av oppgaven som ble gitt. Oppdragsgiver skal ikke ha noen forventninger om at funksjonalitetene gruppen har utviklet, fungerer akkurat slik de ønsket seg, med tanke på at de ikke har vært behjelpelig med tilbakemeldinger under utviklingsprosessen.

Når det kommer til hvordan gruppen ser på resultatet, så er man fornøyd med det som har blitt utviklet, under de omstendighetene som har vært.

6.6 Refleksjon

Når man ser tilbake på prosjektperioden er det flere ting man kunne gjort annerledes, som kunne vært med på å gjøre sluttproduktet bedre.

Omfanget av oppgaven har vært større enn hva gruppen først antok. Dette kommer ikke som en overraskelse når man tenker over at medlemmene har lite erfaring med å planlegge prosjekter av denne størrelsen. Dersom man skulle utført prosjektet på nytt, hadde det vært ønskelig og vært minst en person til på gruppen. Dette av den grunn at arbeidsmengden på to personer har blitt for stor, og man har ikke rukket å fullføre alt man hadde satt seg som mål. Det hadde også vært ønskelig og tildelt enda mer tid til utvikling av modeller som skulle brukes i utviklingsfasen, som muligens hadde spart mye tid.

Det kunne også blitt gjennomført grundigere kode gjennomganger og modul testing underveis for å sikre kvaliteten på implementasjonen. Dette kunne vært med på å hindre at man måtte endre på kode i etterkant.

7 KONKLUSJON OG VIDERE ARBEID

I dette kapittelet vil det bli gitt en oppsummering av målene i prosjektet. Det vil bli diskutert hva som ikke har blitt fullført, hva resultatet kan brukes til, videre arbeid dersom oppdragsgiver ønsker å fullføre applikasjonen samt en konklusjon for hele prosjektet.

7.1 Oppsummering av prosjekt mål

Hovedmålet med dette prosjektet har vært å utvikle en personaldatabase for det Akademiske kvarter. Applikasjonen har i dag en fungerende backend og brukergrensesnitt. Det er likevel en del funksjonalitet gruppen ikke har hatt tid til å implementerer og en oversikt over arbeidet som har blitt ferdigstilt og ikke, er gitt i tabellen under. Punktene i denne tabellen er utarbeidet fra kravspesifikasjonene gitt sammen med denne oppgaven og har blitt sett på som delmålene underveis i prosjektet.

Hvordan man skal lese tabellen: Ruter med grønt er fullførte oppgaver, oransje er under testing og rødt er ikke implementert.

Funksjonelle krav	Kort	Ikke-funksjonelle krav	Kort
RESTful Api	FK1	Enklere enn tidligere system	IFK1
Søk og Filtrering	FK2	Alle brukere skal ha tilgang	IFK2
Generere nytt passord via epost	FK3	Responsivt design	IFK3
Generere nytt passord manuelt	FK4	Kjappere enn tidligere system	IFK4
2-Faktor autentisering	FK5	Overholde GDPR regler	IFK5
Eksport av data til CSV	FK6		
Varslinger via epost	FK7		
Fleksible API-kall	FK8		
Tilgangstyring	FK9		
CRUD: På alle entiteter	FK10		
Profilside for brukere	FK11		

Table 7. Oppsummering av prosjektmål.

7.1.1 Ikke oppnådde mål

Det er noen deler av oppgaven det ikke var tid til å fullføre, og dette er markert med rødt i tabellen over. Det som krever mest arbeid er fleksible API-kall, hvor det er behov for en del endringer i logikken på serversiden, for å minske hva man får tilbake fra databasen. Dette må også gjenspeiles på frontend, da denne håndterer større API-kall.

De andre punktene er, varsling via e-post, 2-faktor autentisering og generere passord via e-post/manuelt, og disse er relativt enkle å implementere. Testing av systemet på større datamengder er noe som ikke har vært mulig under prosjektet. Noe som gjøre det vanskelig å si hvordan implementasjonen vil fungere på mer data, og optimalisering av funksjoner kan være aktuelt.

Hovedårsakene til at målene ikke er nådd har vært at prosjekt omfanget har blitt for stort for to personer. I tillegg har kommunikasjonsproblemene og vanskene med å få det man trengte i planleggingsfasen, vært av betydning for resultatet.

7.1.2 Hva kan resultatet brukes til

Slik som applikasjonen, både frontend og backend er i dag, må den ha litt jobb før den er helt ferdigstilt. Hoved funksjonaliteten, som brukere, organisasjoner og grupper er klar, men det gjenstår enda en del arbeid når det kommer til tilgangsstyring og lagring av persondata.

Kvarteret var interessert i et system som skulle overta for et eksisterende system, men det har vært vanskelig å få vite hva dagen system faktisk kan gjøre, da det ikke ble gitt tilgang til dette før i sluttfasen av prosjektet. Det er derfor vanskelig å vite om systemet kan brukes til de formål den tidligere applikasjonen har hatt. Resultatet fra dette prosjektet må derfor sees på som en demo som trenger videreutvikling for å kunne tas i bruk.

7.2 Videre arbeid

Med tanke på at gruppen ikke har hatt tilstrekkelig med tid til å fullføre alt man satt seg som mål ved prosjektstart, er den en del ting oppdragsgiver må fullføre før de kan sette applikasjonen ut i produksjon. Det gruppen mener man må fullføre er nevnt i dette kapittelet.

GDPR var et opprinnelig krav i kravspesifikasjonene når denne oppgaven ble gitt. Gruppen har ikke hatt tid til å se nærmere på dette under prosjektet og dette er med det en av de viktigste tingene som må utføres dersom man skal ta i bruk applikasjonen. Oppdragsgiver har gitt uttrykk for at de ønsker å bruke Microsoft Azure som har mange funksjoner og guider som hjelper med å følge kravene til GDPR.

Slik applikasjonen er i dag så “hash”-er den og lagrer passord i databasen. Men for at ting skulle være enkelt under utvikling har gruppen valgt å bare lage samme passord på alle nye brukere. Det er viktig her at man fjerner dette og legger til en annen form for passord generering på nye brukere, slik at alle nye brukere ikke

ender opp med samme passord. Det er også nødvendig å legge til muligheten for å forandre på sitt eget passord samt en glemt passord funksjon på innloggingssiden. Oppdragsgiver hadde også et ønske om 2-faktor autentisering, men gruppen har ikke hatt tid til å se på dette.

Tilgangsstyring er noe som er blitt påbegynt og applikasjonen har i dag superbruker og vanlig bruker. Gruppen har i implementasjonen av dette møtt på noen problemer som har medført at dette ikke er helt ferdig og noe som må fullføres før applikasjonen kan tas i bruk.

API-kallene mot databasen har ikke blitt optimalisert slik at man kan begrense mengden med data man henter ut. Dersom oppdragsgiver ønsker å gjøre kallene raskere enn de er i dag, er dette en naturlig plass å begynne. Det har vært vanskelig for gruppen å teste dette, grunnet at man ikke har hatt tilstrekkelig med testdata under prosjektet.

Databasen gruppen har utviklet i dette prosjektet er ikke lik den databasen oppdragsgiver har i dag. Det er derfor nødvendig å migrere data over til den nye databasen, dersom den skal tas i bruk.

Oppdragsgiver har selv ønsket å style komponentene i applikasjonen. Med dette har gruppen ikke tildelt så mye tid til styling og kun laget et brukergrensesnitt som kan vise funksjonaliteten som har blitt lagt til. Dette betyr at oppdragsgiver selv må håndtere den universelle utformingen for å dekke de krav som omhandler dette.

Under prosjektet har gruppen ikke hatt tilgang til tilstrekkelig med testdata slik at man kan teste opp mot den mengden data oppdragsgiver har i dag. Med dette kan det godt tenkes at søkemetodene gruppen har implementert ikke er raske nok, og trenger endringer. Dette er noe som må testes.

Historikk på bruker når det kommer til verv og hvilke grupper de er medlem i har ikke blitt fullført av gruppen. Det var her usikkerhet om hvordan oppdragsgiver gjorde det i sitt system, og gruppen fikk ikke noe svar når de spurte om oppklaring i dette. Dette må oppdragsgiver selv implementere av den grunn.

Hvordan oppdragsgiver håndterer bildeopplastning i sitt system i dag har vært noe gruppen har etterlyst hele prosjektperioden, men ikke fått svar på før i sluttfasen. På grunn av dette har det blitt brukt Cloudinary i dette prosjektet. Det har i sluttfasen kommet frem at oppdragsgiver har brukt Azure for dette og er noe de må legge til den nye applikasjonen selv dersom de ønsker å fortsette med dette. De kan alternativt opprette en Cloudinary-konto, og fortsette å bruke bilde-funksjonen gruppen har laget.

For å sikre at alle brukerne holder sin egen informasjon oppdatert er det tenkt en form for varsling via e-post med jevne mellomrom. Dette er ikke blitt påbegynt under dette prosjektet.

Til slutt er den noen ting som er implementert som trenger endringer. Dette omhandler feilmeldinger i brukergrensesnittet. Det er her lagt opp til å håndtere feil og gi tilbakemelding til sluttbrukeren, men disse må endres på slik at de følger de standarder oppdragsgiver ønsker å bruke. Gruppen har ikke hatt informasjon om hva dette er under prosjektet. Det er også blitt gitt uttrykk for at en bruker skal kobles til en organisasjon, noe den ikke er i dag, men enkelt å endre på.

Den siste tingen er dataene som blir vist. Utfra brukertestene med oppdragsgiver virker det som om de ønsker at endre hvordan data vises i tabellene. Dette må derfor endres på slik at det blir slik de ønsker. Det har vært vanskelig for gruppen å vite hva som skulle vises grunnet mangelen på tilbakemeldinger underveis.

7.3 Konklusjon

Det gruppen sitter igjen med etter dette prosjektet er verdifulle kunnskaper innenfor full-stack utvikling. Gruppen har måttet tilegne seg kunnskaper innenfor React, .NET Core og publisering av applikasjoner på Microsoft Azure, alle kunnskaper som er veldig bra å ta med seg videre. Planlegging av et prosjekt helt fra starten av, for så å utvikle det er også en god erfaring medlemmene tar med seg fra dette prosjektet. Gruppen har også fått tatt i bruk en god del av det man har lært gjennom bachelor-løpet ved Høgskulen på Vestlandet, spesielt når det kom til planleggingen av det som skulle gjøres.

Applikasjonen som har blitt utviklet har som nevnt tidligere et fungerende brukergrensesnitt sammen med en backend som gjør lagring av data mulig. Under de omstendighetene som har vært under prosjektet ser gruppen seg fornøyd med det arbeidet som har blitt gjort. Det er likevel en god del fra kravspesifikasjonen gitt i starten av prosjektet som ikke er ferdig. Dette gjør at oppdragsgiver ikke kan ta applikasjon i bruk med en gang, men heller er nødt til å videreutvikle den om de ønsker å ta den i bruk.

Når det kommer til kommunikasjonsproblemene som har vært med oppdragsgiver har gruppen sett på det som uforståelig. Da spesielt med tanke på at det har fått høyskolen til å gi oppgaven til noen av sine avgangsstudenter, for så å ikke følge opp oppgaven tilstrekkelig under prosjektperioden. Dette kommer i grunn ikke som en overraskelse når man tenker over i etterkant, siden oppdragsgiver er en frivillig studentorganisasjon som har hatt de samme utfordringer som resten av verden grunnet covid-19 viruset under

prosjektperioden. Det at gruppen fikk tildelt en ny kontaktperson under prosjektet medførte også utfordringer, spesielt med tanke på at denne personen ikke hadde tilstrekkelige tekniske kunnskaper som kunne hjelpe gruppen. Dette medførte at informasjonsflyten måtte gjennom unødvendig mange ledd.

Slik kommunikasjonen ble mellom oppdragsgiver og gruppen kunne dette blitt håndtert på en bedre måte. Gruppen kunne vært flinkere på skriftlig avtaler, spesielt med tanke på hva informasjon gruppen hadde behov for og tidsfrist for når man trengte tilgang til dette. Denne skriftliggjøringen ville gjort det tydligere for utenforstående å følge gangen i prosjektet samt en bedre forståelse for hvorfor sluttproduktet har blitt slik det er. Noe annet gruppen kunne gjort bedre, er oppfølging på telefon når kommunikasjon på e-post ikke fungerte. Dette kunne gitt gruppen tidligere tilgang til nødvendig informasjon. Grunnen til at man ikke har gjort dette er fordi man har prøvd å ta hensyn til at oppdragsgiver har vært en frivillig organisasjon. Man har opplevd at det har vært vanskelig å balansere det med å være på tilbudssiden mot det som oppleves som å være kravstor eller masete. En ting som hadde vært til stor hjelp for å håndtere denne situasjonen ville vært klarere retningslinjer fra høyskolen, om hvordan kommunikasjonen mellom gruppen og oppdragsgiver skal foregå.

Oppsummering av prosjektet er at det har vært en lærerik prosess for medlemmene av gruppen. Dessverre har man ikke rukket å bli ferdig med alle prosjektmålene man hadde ved oppstart, men håper på at arbeidet som har blitt gjort kan være starten på noe som kan ha nytteverdi for oppdragsgiver i fremtiden.

8 REFERANSER OG BILDE/TABELL LISTE

1. 2FA (u.å), *What is two factor authentication*, [Internett] Tilgjengelig fra <https://authy.com/what-is-2fa/> [Lest 23.3. 2020]
2. Age S. (2017), *How to write a search component with suggestions in react*, [Internett] Tilgjengelig fra <https://dev.to/sage911/how-to-write-a-search-component-with-suggestions-in-react-d20> [Lest 10.4.2020]
3. AngularJS (2020) [Internett] *What Is AngularJS?* Tilgjengelig fra <https://docs.angularjs.org/guide/introduction> [Lest 30.05.2020]
4. Axios [Internett](u.å), Tilgjengelig fra: <https://github.com/axios/axios> [Lest 23. Mars 2020]
5. Codecademy (u.å) *What is CRUD?* [Internett] Tilgjengelig fra <https://www.codecademy.com/articles/what-is-crud> [Lest 3.3.2020]
6. Cummings N. [Internett](2020) *Complete guide to building an app with .Net Core and React* , Tilgjengelig fra: <https://www.udemy.com/course/complete-guide-to-building-an-app-with-net-core-and-react/> [Lest 26. Mai 2020]
7. Det Akademiske Kvarter [Internett](u.å) Tilgjengelig fra : <https://kvarteret.no>[27.Mai 2020]
8. Ebrahim M. (2019) *9 of the most common mistakes in database design* [Internett], DZone, tilgjengelig fra: <https://dzone.com/articles/9-of-the-most-common-mistakes-in-database-design>[Lest 1.Mars 2020]
9. Eirik Rossen (2019)[Internett] *SNL, Brukergrensesnitt* Tilgjengelig fra <https://snl.no/brukergrensesnitt> [Lest 30.5.2020]
10. Gantt.com (2020) [Internet], *Hva er et Gantt-diagram?* Tilgjengelig fra <https://www.gantt.com/no/> [Lest 30.05.2020]
11. GitHub [Internett] *What is GitHub?*(u.å), Tilgjenelig fra <https://www.w3schools.com/whatis/whatis_github.asp> [Lest 10. Januar 2020]
12. Jetbrains [Internett](2020), *Raider*, Tilgjengelig fra: <https://www.jetbrains.com/rider/> [Lest 20. Januar 2020]
13. JWT (u.å) [Internett] Tilgjengelig fra <https://jwt.io> [Lest 5.4.2020]
14. Makowska A. (2018), *5 common mistakes that may irritate your REST API Customers*, Tilgjengelig fra: <https://dzone.com/articles/5-common-mistakes-that-may-irritate-your-rest-api> [Lest 1. Mars 2020]
15. Microsoft (2020) [Internett] *Nuget dokumentasjon* Tilgjengelig fra <https://docs.microsoft.com/en-us/nuget/> [Lest 10.04.2020]
16. Microsoft [Internett] *.NET Core dokumentasjon* (u.å), Tilgjengelig fra: <https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-3.1>[Lest 25. Mai 2020]
17. MobX [Internett](u.å), Tilgjengelig fra: <https://mobx.js.org/README.html> [Lest 11. April 2020]
18. Nielsen J. (2000), *Why you only need to test with 5 users,* [Internett] Tilgjengelig fra :<https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/>[Lest 5.mars]
19. Node.js [Internett](u.å) *About Node.js®* Tilgjengelig fra: <https://nodejs.org/en/about/> [Lest 15. Januar 2020]
20. Order Group (2016), *Common mistakes in UX/UI- Meet A Developer,* [internett] Tilgengelig fra <https://meetadeveloper.com/common-mistakes-in-ux-ui-fd314d0b3a9f>[Lest 5.mars 2020]
21. Personopplysningsloven (2018), *Lov om behandling av personopplysninger*,[internett] Tilgjenglig fra <https://lovdata.no/dokument/NL/lov/2018-06-15-38>[Lest 17.02.2020]
22. PHP (u.å) [Internet] *What Is PHP?* Tilgjengelig fra <https://www.php.net/manual/en/intro-whatis.php> [Lest 25.05.2020]
23. Postman (2020) [Internett] Tilgjengelig fra <https://www.postman.com> [Lest 5.3.2020]
24. React [Internett] *React Context*(u.å), Tilgjengelig fra: <https://reactjs.org/docs/context.html> [Lest 15. April 2020]
25. React [Internett](u.å) *What Is React?*, Tilgjengelig fra: <https://reactjs.org/tutorial/tutorial.html#what-is-react>, [Lest 25. Januar.2020]

26. React Router (u.å) [Internet], *Quick Start*, Tilgjengelig fra <https://reacttraining.com/react-router/web/guides/quick-start> [Lest 9.03.2020]
27. React toastify (2020) [Internet], *React toastify*, Tilgjengelig fra <https://www.npmjs.com/package/react-toastify> [Lest 12.04.2020]
28. Redux [Internett](u.å) *Usage with React* Tilgjengelig fra: <https://redux.js.org/basics/usage-with-react>, [Lest 5. Mars 2020]
29. RESTful [Internett] *What is REST?* (u.å), Tilgjenelig fra: <https://restfulapi.net>[Lest 13. Februar 2020]
30. Rouse M. (2017) *What is Scrum?* [Internett], Tilgjengelig fra: <https://searchsoftwarequality.techtarget.com/definition/Scrum>[Lest 3. mars 2020]
31. Security Headers [Internett](u.å), Tilgjengelig fra: <https://securityheaders.com/.> [Lest 23. Mars 2020]
32. Semantic ui React [Intenett] Tilgjengelig fra <https://react.semantic-ui.com> [Lest 1.2.2020]
33. Sjøberg D. [Internett](2014) *Funksjonelle krav* ,Tilgjengelig fra: <https://www.uio.no/studier/emner/matnat/ifi/INF1050/v14/timeplan/inf1050.krav.29.1.2014.pdf> [Lest 5. Mai 2020]
34. Trello [Internett] *Hva er Trello?* (u.å) , Tilgjenelig fra <https://trello.com/about> [Lest 25. Mai 2020]
35. TypeScript [Internett] (u.å) *What is TypeScript?* Tilgjengelig fra: <https://www.typescriptlang.org> [Lest 25. Mai 2020]
36. UML (2005) [Internett] *What Is UML?* Tilgjengelig fra <https://www.uml.org/what-is-uml.htm> [Lest 25.05.2020]
37. Visual Studio Code [Internett](u.å), Tilgjengelig fra: <https://code.visualstudio.com> [Lest 26. Mai 2020]
38. W3school.com (2020), *Javascript HTML DOM* [Internett] Tilgjengelig fra <https://www.w3schools.com/js/js_htmldom.asp> [Lest 15.01.2020]
39. Watmore J. (2019), *Role based Authorization with Example API*, [Internett] Tilgjengelig fra <https://jasonwatmore.com/post/2019/10/16/aspnet-core-3-role-based-authorization-tutorial-with-example-api> [Lest 2.4.2020]
40. Wells D. (2013)[Internett](u.å) *Extreme Programming: A gentle introduction*, Tilgjengelig fra: <http://www.extremeprogramming.org/)> [Lest 26. Mai 2020]
41. Wikipedia [Internett] (2019) *Model View Controller*, Tilgjengelig fra:<https://no.wikipedia.org/wiki/Model–view–controller> [Lest 25. februar 2020]
42. Zoom Video Communications, Inc(2020)[Internet], *About Zoom*, Tilgjengelig fra <https://zoom.us/about> [Lest 30.05.2020]

List Of Images

List Of Tables

9 APPENDIX

9.1 Risikoliste

Beskrevet i punkt 3.5.3

9.2 GANTT diagram

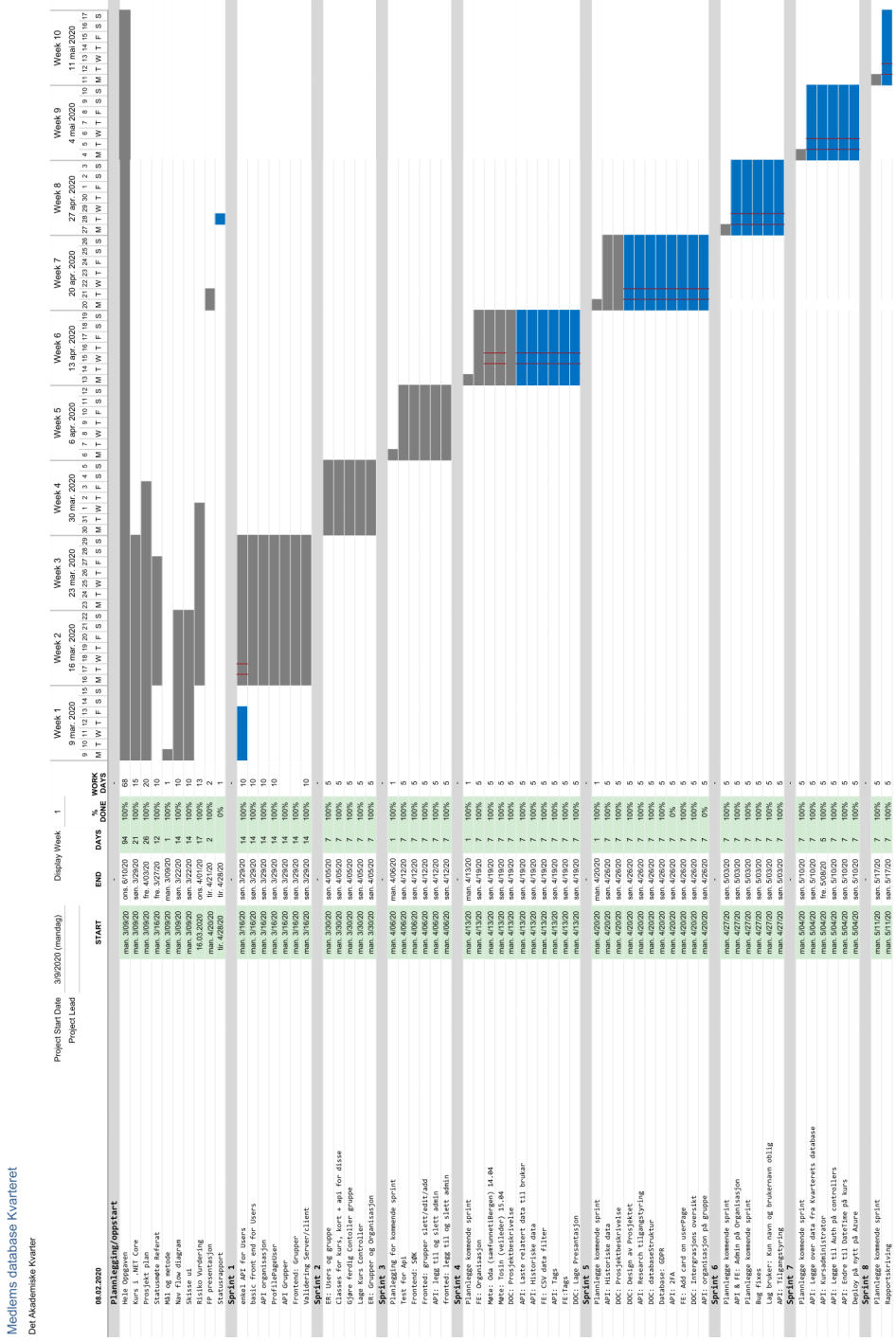


Figure 33. Gantt diagram(Fullstendig).

9.3 Oppgavebeskrivelse

P20 - Personaldatabase – «Kvarteret»

Det Akademiske Kvarter (Kvarteret) er studentkulturhuset i Bergen, drevet på frivillig basis av ca. 300-400 studenter fra alle de ulike studieinstitusjonene i Bergen. Huset samler et hundretalls studentorganisasjoner som arrangerer over 2150 arrangement i året. Studentersamfunnet i Bergen er en av byens eldste studentorganisasjoner, og den største som arrangerer fast på Kvarteret. Medregnet alle faste samarbeids-partnere engasjerer Kvarteret omlag 800 frivillige i året for å skape studentkultur og bedre studentvelferd.

www.kvarteret.no / www.samfunnetibergen.no

Dette prosjektet følges opp av en gruppe personer i ulike nøkkelroller på Kvarteret, samt en representant fra Studentersamfunnet i Bergen.

Oppgave

Målet med prosjektet er å utarbeide et nytt web-basert personalsystem, både front- og backend, som kan brukes av Kvarteret og fast tilknyttede organisasjoner. Det er selvsagt responsivt. Systemet må ivareta alle personopplysninger på en sikker måte, og oppfylle kravene til vern og behandling av personopplysninger ihht GDPR. Kildekoden må være fullt tilgjengelig overfor oppdragsgiver etter fullført prosjekt. Prosjektet må selvsagt være godt dokumentert.

Løsningen må støtte innlogging, slik at alle som legges inn som brukere i systemet automatisk får en innlogging knyttet til feks epost og med 2FA (feks. SMS med engangskode på mobilen). Den må videre støtte brukertilganger, slik at tilgang til informasjon og handlinger kan begrenses ut i fra organisasjon og gruppe. Alle brukerhandlinger i plattformen må loggføres.

I sin enkleste form skal systemet ha en profilside for den innloggede brukeren, hvor vedkommende kan vedlikeholde egen personalia. Utover det må brukerne være organisert i et hierarki, basert på hvilken organisasjon de tilhører og hvilken gruppe innenfor organisasjonen. Videre må systemet støtte fortløpende opprettelse av nye datafelt, samt nye kategorier av informasjon som skal registreres på den enkelte. F.eks. kurs og utmerkelser.

Dette må kunne vise en oversikt over historikk: hvilke grupper vedkommende har vært med i, og hvilke verv den har hatt i de ulike gruppene. Denne informasjonen må kunne hentes ut i form av rapporter, og kunne eksporteres i Excel-format. Systemet må også ha et RESTful API, som igjen gir grunnlag for å integrere systemet opp mot annen software. Relevante teknologier er Vue/React/Angular, GraphQL, SQL, HTML, JS og CSS.

Dere velger dette prosjektet dersom dere ønsker å utvikle en skalerbar løsning for administrering av frivillige og ansatte, hvor det på markedet per dags dato ikke eksisterer gode løsninger. Det gir godt grunnlag for å etablere egen bedrift og drive produktutviklingen videre, med oppdragsgiver som kunder. I tillegg kan Kvarteret tilby interngoder til prosjektteamet, for hele semesteret prosjektet pågår.

Figure 34. Oppgavebeskrivelse.