Western Norway University of Applied Sciences

BACHELOROPPGAVE

Development of a project managment tool with accompanying database

Utvikling av prosjektstyringsverktøy med tilhørende database

Petter Tesdal

Kristoffer Fjeldstad Madsen

Eirik Sangiorgi Brakstad

Bachelor, Dataingeniør

Fakultet for teknologi, miljø- og samfunnsvitskap

Tosin Daniel Oyetoyan

12.06.2024

Document Control

Rapportens tittel	*Dato:*
Utvikling av prosjektstyringsverktøy med tilhørende database	13. mai 2024
Forfatter(e): Petter Tesdal Kristoffer Fjelstad Madsen Eirik Sangiorgi Brakstad	*Sider u/vedlegg:* 92 *Antall sider vedlegg* 57
Studieretning: Dataingeniør	*Antall disketter/CD-er* 0
Kontaktperson ved studieretning: Tosin Daniel Oyetoyan	*Gradering:* Open
Merknader: Ingen	

Oppdragsgiver:	*Oppdragsgivers referanse*
TechnipFMC	Ingen
Oppdragsgivers kontaktperson:	*Telefon*
Morten Wilhelmsen	+47 402 32 409

Sammendrag:
Dette bachelorprosjektet har som mål å utvikle et nytt prosjektstyringsverktøy for TechnipFMC som viderefører egenskapene til den gamle løsningen. Prosjektet har tatt utgangspunkt i den forrige løsningen som var et Excel-dokument utviklet av TechnipFMC. Som fokus har prosjektet gått ut på å gjøre løsningen mer brukervennlig men fortsatt kunne beholde kompleksiteten til den originale løsningen.

Stikkord:	*Stikkord:*	*Stikkord:*
Prosjektstyring	Transformasjon	Excel

Høgskulen på Vestlandet, Fakultet for teknologi, miljø- og samfunnsvitenskap
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

Dette prosjektet tar for seg arbeidet rundt bachelorprosjektet *Development of a project managment tool with accompanying database fra* (*HVL*). Dette prosjektet markerer vår treårige utdannelse på dataingeniør med spesialisering innen programvare utvikling ved Høgskulen på Veslandet (HVL).

Vi ønsker å takke alle som har vært med å bidra til gjennomføringen av dette prosjektet og da særlig vår veileder Tosin Daniel Oyetoyan som har kommet med verdifulle tilbakemeldinger, innspill og god oppfølging. Vi ønsker også å takke vår kontaktperson i TechnipFMC Morten Wilhelmsen som har ledet oss til det endelige produktet vi har i dag.

Ordliste

avhengighet Pakker som applikasjonen er avhengig av for å bygges. 53

back-end Delen av applikasjon som ikke er direkte synlig for brukeren, men som oppererer i bakgrunnen. 10

brukergrensesnitt Delen av et dataprogram eller et digitalt system som brukeren samhandler med, inkludert alle visuelle og interaktive elementer. 4, 8, 11

front-end Delen applikasjonen som brukeren direkte samhandler med. 10

gate Et utrykk brukt av TechnipFMC for å referere til en gruppe arbeidsoppgaver som hører sammen. Høyeste ordens underdeling av et prosjekt. 7, 9, 10, 43, 55, 69

JSON JavaScript Object Notation. 53

latency Den tid det tar å hente datapakker fra databasen til applikasjonen. 29, 31

layout Et omriss som inkluderes på tvers av en nettside, ofte bestående av navigasjonsfelt i ytterkantene av nettsiden. 53

PEM Project Execution Manager. Rolle gitt til den prosjektleder i TechnipFMC som er ansvarlig for et gitt prosjekt. 48, 69

PO-dato Purchase Order date. Dato kunde har satt som siste frist for overlevering av et prosjekt. 48, 69

programvare Lagring av data. 9, 10

programvare De binære filene som gjør det mulig å bruke en datamaskin eller annen elektronikk. 9

pull-request En gren sendes til GitHub Repository for gjennomgang og fletting med hovedgrenen. Det sjekkes for konflikter mellom grenen og hovedgrenen, samt hvilke endringer som er blitt gjort. 81

rammeverk Teknologi benyttet ved utvikling av programvareapplikasjoner som inneholder retningslinjer, biblioteker og verktøy. 10–13

SF-dato Scheduled Finish date. Dato TechnipFMC har planlagt at et prosjekt skal være ferdig, bestemt i henhold til PO-dato for det gitte prosjektet, balansert mot når verksted og test-kammer er ledig. 48, 69

single sign-on En autentiseringsprosess som lar brukere få tilgang til flere applikasjoner eller systemer med bare en innlogging. 18, 51, 68, 80

single-page applikasjon En single-page applikasjon er en applikasjon som kun benytter seg av et enkelt web-dokument, og oppdaterer dette heller enn å laste inn separate dokument når brukere navigerer rundt i applikasjonen. 9

skybasert Lagring, programvare eller databehandling som leveres over internett gjennom eksterne servere. 8

task En enkelt arbeidsoppgave, og den minste underdelingen av et prosjekt. 7, 9, 10, 43, 55, 69, 87

TechnipFMC Fransk-Amerikansk subsea-firma, og gruppens oppdragsgiver. . x, 2–7, 12, 15, 21, 23, 31, 34, 40, 67, 71, 72, 77, 80, 84, 86, 87

techstack Teknologien som får de underliggende verktøyene, rammeverk og bibliotekene som brukes til å bygge og kjøre. 19

UI/UX User-Interface/User-Experience. Læren om korrekt implementering av

brukergrensesnitt med hensyn til både det visuelle og det funksjonelle. 11

webapplikasjon En applikasjon som kan kjøres gjennom en nettleser. 9, 10, 23, 24

Innholdsliste

Figurliste

Tabelliste

Liste over kodeutdrag

Kapittel 1

Innledning

1.1 Kontekst

TechnipFMC sin avdeling på Ågotnes er en underdel av det Fransk-Amerikanske selskapet TechnipFMC, som utfører vedlikehold på ventilblokker ofte referert til som *Juletrær*, se figur 1.1-1. Disse blokkene er kuber på ca. 64 kubikkmeter som står på havbunnen i opptil 25 år. Ved behov gjennomføres vedlikehold av juletrær hvor de overhales og testes på et verksted, eksempelvis ved TechnipFMC sin avdeling på Ågotnes. Slike vedlikeholdsprosesser er en kombinasjon av repetitive og langtekkelige oppgaver hvor feiltrinn kan koste TechnipFMC og kundene deres milliarder av kroner. Hele den sammensatte prosessen ved overhaling og testing av juletrær tar normalt 1-2 år, og består av mange mindre prosesser som er ekstremt tidssensitive. Det er derfor kritisk for prosjektleder og andre ansatte å ha god oversikt slik at prosjektene fullføres i tidsrommet avtalt med kundene. Grunnet mange av disse arbeidsoppgavenes natur krever de også bruk av enten verktøy eller arbeidsstasjoner som fungerer som flaskehalser på verkstedet da de kun kan brukes til et juletre om gangen. Arbeidsstasjonene er dermed opptatt i et betydelig tidsrom. Dette legger enda et lag med mulige årsaker til forsinkelser, ettersom forsinkelser i ett prosjekt kan lede til forsinkelser i ett annet, som deretter starter en kjedereaksjon.

Figur 1.1-1: Bilde av en ventilblokk med Kilde: https://ndla.no/image/40794?

1.2 Motivasjon

Arbeidet TechnipFMC utfører på Ågotnes er en viktig del av oljeindustrien i Norge. Vedlikehold av dette utstyret som skal tåle å stå på havbunnen i flere tiår er essensielt, og det kan ikke være betydelige mangler underveis i disse prosjektene. Forsinkelser eller tabber kan føre til tap av flere milliarder kroner. Med tanke på dette er det nødvendig for

TechnipFMC å ha en detaljert oversikt over prosjektene de utfører, og oppgavene som inngår i disse. Dette er fokusområde for vår applikasjon.

Prosjektet er en spennende mulighet til å samarbeide med et så stort firma som TechnipFMC. Med tanke på oppgaver man kan komme over i arbeidslivet, er prosjektet relevant, og det ses frem til å ta med seg denne erfaringen videre.

1.3 Prosjekteier

Oppdragsgiver og prosjekteier vil være TechnipFMC sin avdeling Ågotnes. TechnipFMC er en av verdens ledende bedrifter innen energiprosjekter, teknologier, systemer og tjenster relatert til *subsea-* og *surfaceprosjekter* (TechnipFMC, 2024). Deres representant Morten Wilhelmsen er kontakt personen for prosjektet og vil ta delvis ansvar for å håndtere overlevering av prosjektet.

1.4 Problembeskrivelse og mål

For å holde oversikt over alle de individuelle arbeidsoppgavene vedlikeholdsprosessene består av, har TechnipFMC tatt i bruk et Excel-dokument utviklet av Morten Wilhelmsen, prosjektansvarlig i TechnipFMC Ågotnes. Dette dokumentet har blitt benyttet en betydelig mengde av ansatte siden det ble opprettet, og er i dag svekket som oversiktsverktøy grunnet overbelastning av data. Dette har da ledet TechnipFMC til å søke etter en alternativ løsning for dokumentet i samarbeid med Høgskulen på Vestlandet.

1.4.1 Problembeskrivelse

Excel-dokumentet TechnipFMC bruker i dag er preget av et dårlig brukergrensesnitt, og nå som det har vært i bruk over lengre tid er det blitt overbelastet med kodeblokker og utregninger. Grunnet dette har TechnipFMC gitt en oppgave som går ut på å utvikle et alternativt verktøy, som skal ha samme formål som Excel-dokumentet til TechnipFMC. Dette formålet er å skape overblikk av prosjektenes fremdrift, med større kapasitet og mer brukervennlighet enn det nåværende systemet. Brukere av den gamle løsningen har ifølge oppdragsgiver gjentatte ganger kommet med kommentarer på dokumentets brukervennlighet og hastighet. Dokumentet hang seg stadig opp ved navigering og ny data

brukte lang tid på å laste inn. De kom med kommentarer på brukstilfeller som var unødvendig komplisert og tidskonsumerende, og oppfordret til å implementere ideer som kan gjøre løsningen mer brukervennlig. Derfor har prosjektet et stort søkelys på hvordan applikasjonen oppfattes av brukeren i henhold til innlasting av data og enkelhet da dette virker å være de største problemene arbeidsgiver vil ha løst.

1.4.2 Bruksmål

En bruker sitt hovedbruksmål er å gå inn på løsningen og å få et kjapt overblikk av hvordan status på prosjektene ser ut. Disse brukerne kan være ansatte som skal se om et prosjekt skal bruke et arbeidsareale snart eller en leder som vil ha status på prosjektene. Redigering og oppdatering av prosjektene er det løsningen vil bli brukt minst til, men som også er viktigst at blir implementert riktig.

1.4.3 Mål

Målet med prosjektet er å utvikle et produkt som hjelper TechnipFMC med å få et overordnet blikk av deres prosjekter og tidsplan. Applikasjonen skal ha et brukervennlig grensesnitt som er enkelt og intuitivt å bruke for deres ansatte samtidig som man beholder kompleksiteten til det originale Excel-dokumentet.

1.4.4 Forskningsspørsmål

Ved behovene til TechnipFMC avklart er det mulig å formulere forskningsspørsmål som skal evaluere om prosjektet har tilfredsstilt arbeidsgiver sine krav. Forskningsspørsmålet skal gi svar på om prosjektet har klart å forbedre brukeropplevelsen til de ansatte, ivaretatt funksjonaliteten til det opprinnelige verktøyet og forbedret brukervennligheten.

Forskningsspørsmål:

Hvordan best transformere et Excel-dokument til en optimalisert og brukervennlig applikasjon?

1.5 Oppbygging av rapporten

Denne rapporten inneholder totalt sett 8 kapitler som sammen skal gjennomgå hele prosessen med å utvikle produktet XT-gates. De er strukturert som følger:

Kapittel 1 er en introduksjon til oppdragsgiver TechnipFMC og problemstillingen de står ovenfor, og hvorfor prosjektgruppen har valgt dette som sitt bachelorprosjekt.

Kapittel 2 er en beskrivelse av selve prosjektet som skal utføres, og rammene det skal gjøres innenfor.

Kapittel 3 er en dokumentasjon av utarbeidingen for arbeidsmetodikker, både utvikling og design som har blitt vurdert eller tatt ibruk.

Kapittel 4 er en oversikt over de forskjellige avgjørelsene som er tatt underveis i designprosessen, og drøfting av disse, samt planlegging av utviklingsløpet.

Kapittel 5 er en grundig beskrivelse av den endelige løsningen som har blitt implementere for å løse problemstillingen.

Kapittel 6 er resultatene produsert på slutten av prosjektet, presentert som de er uten drøfting.

Kapittel 7 er en diskusjon av resultatene produsert i kapittel 6.

Kapittel 8 er en helhetlig konklusjon for prosjektet, og det videre arbeidet som må utføres eller som anbefales utført.

Kapittel 2

Prosjektbeskrivelse

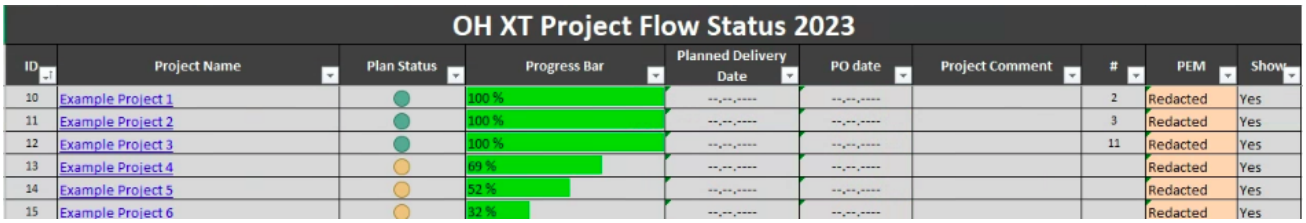
Dette kapittelet tar for seg den praktiske bakgrunnen samt initielle tanker om løsninger satt i starten av prosjektet. Sammen med de drøftede løsningene er foreløpige krav og begrensninger satt for prosjektet.

2.1 Tidligere arbeid

Morten Wilhelmsen har utformet et Excel-dokument som holder styring på TechnipFMC sine prosjekter. Dette dokumentet er godt utformet og har alt av relevant data som de trenger for å få et overblikk av prosjektenes fremdrift Det er også utgangspunktet for at de nå vil ha en mer permanent løsning. Derfor har Bachelorprosjektet bygget videre på dette dokumentet og til delvis utformingen for å skape et resultat som er til nytte for TechnipFMC. Selv om prosjektet ikke bygger direkte på eksisterende kode, er design og konseptuelle ideer influert av den tidligere løsningen.

Et nøkkelkonsept brukt i Excel-dokumentet er en gruppering av arbeidsoppgaver kalt en gate. Arbeidsoppgavene i en gate er arbeidsoppgaver som kan utføres parallelt med hverandre, og et prosjekt består av enten 14 eller 15 gates.

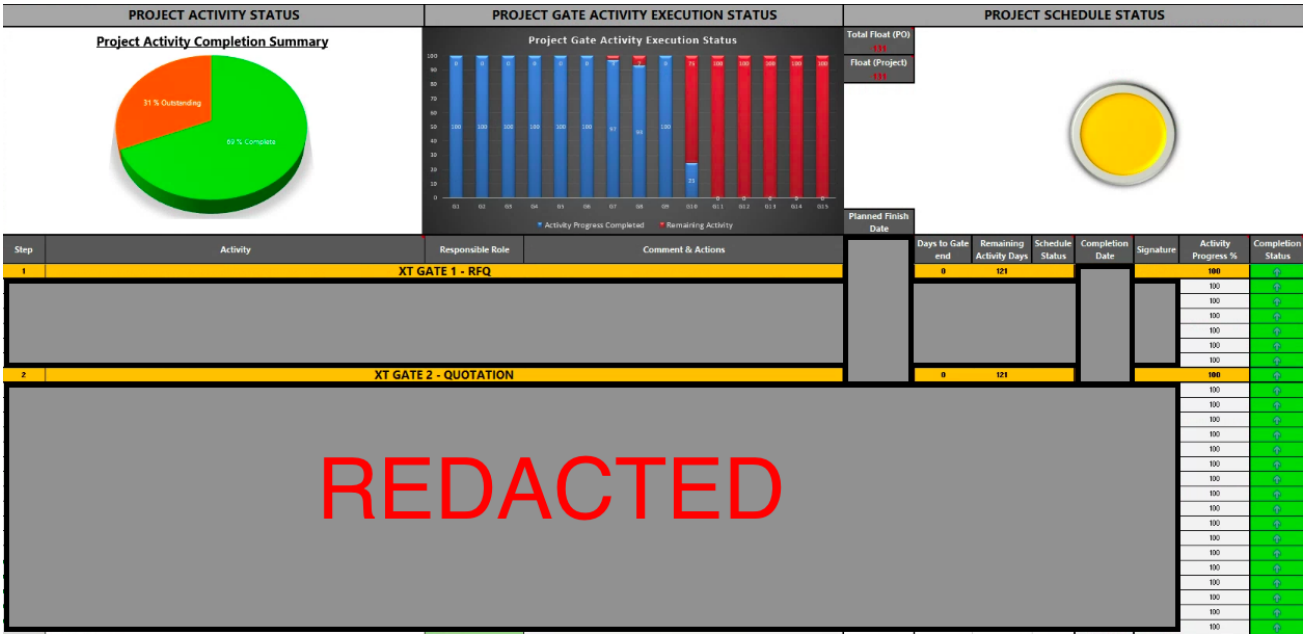
Enhver gate består igjen av spesifikke arbeidsoppgaver, omtalt i prosjektet som en task. Antall tasks i en gate varierer som regel i antall mellom 5 til 20.



OH XT Project Flow Status 2023

ID	Project Name	Plan Status	Progress Bar	Planned Delivery Date	PO date	Project Comment	#	PEM	Show
10	Example Project 1		100 %	--.--.----	--.--.----		2	Redacted	Yes
11	Example Project 2		100 %	--.--.----	--.--.----		3	Redacted	Yes
12	Example Project 3		100 %	--.--.----	--.--.----		11	Redacted	Yes
13	Example Project 4		69 %	--.--.----	--.--.----			Redacted	Yes
14	Example Project 5		52 %	--.--.----	--.--.----			Redacted	Yes
15	Example Project 6		32 %	--.--.----	--.--.----			Redacted	Yes

Figur 2.1-1: Et utsnitt av listen over prosjekter slik den var fremstilt i Excel, med info delvis redaktert. Listen fungerte som en hovedside for navigasjon internt i dokumentet.



Figur 2.1-2: Original side for visning av et prosjekt i Excel. Konfidensiell informasjon er sladdet ut under de grå boksene.

2.2 Initielle krav og løsningsidè

Denne seksjonen tar for seg prosjektets initielle krav basert på oppgavens utforming og samtaler med oppdragsgiver, samt en initiell løsningsidé basert på disse kravene.

2.2.1 Initielle krav

Kravene fra oppdragsgiver er både omfattende og veldig åpne. I design og arkitektur skal løsningen være skybasert, og ha et mer brukervennlig brukergrensesnitt enn den tidligere løsningen. Oppdragsgiver har også i senere tid spurt spesifikt om at databasen som tas i bruk

skal være en SQL-database basert på deres allerede eksisterende SQL Server programvare.

I form av funksjonelle krav har oppgaven et bredt spektrum av informasjon som skal regnes ut og fremstilles i løsningen. Dette henger igjen fra den tidligere løsningen i Excel-dokumentet nevnt i punkt 2.1. Dette inkluderer utregninger av tall som varighet og utførelsesdato på hver task, noe som vil ha utslag i andre tall og visuelle fremvisninger. Et eksempel på dette er at dersom varigheten til en task økes så må arbeidstiden på den tilhørende gate også økes. Dette vil så ha et utslag i datoene for når et mål skal være ferdig, og som kan forandre om prosjektet i sin helhet er i rute eller ikke. Disse funksjonene kan løses på mange måter, men må være til stede. Med andre ord skal prosjektstyringsplatformen være skreddersydd til deres arbeidsprosess, og skal følge deres tidligere arbeidsflyt.

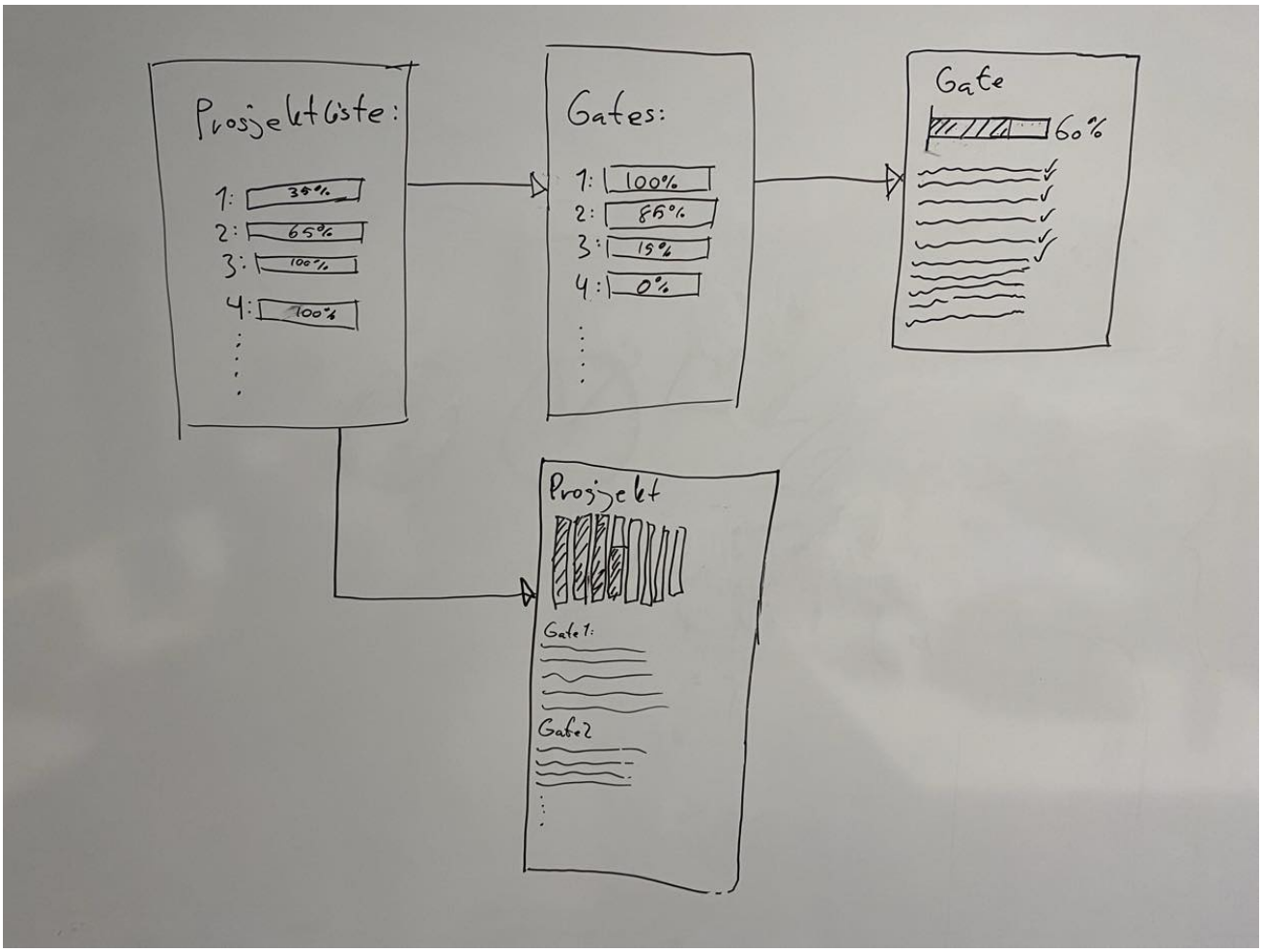
Oppsummert har arbeidsgiver satt spesifikke ønsker til funksjonelle krav og valg av database, mens design, ikke-databaserelatert teknologi, og utførelsesmetodikk har blitt overlatt til prosjektgruppens avgjørelser.

2.2.2 Initiell løsnings-idé

Den initielle løsningsideen var å utvikle en webapplikasjon som ivaretar den grafiske fremstillingen av informasjon relevant for det prosjektoverblikket som er hovedformålet med produktet. Samtidig skal informasjonen lagres i og hentes fra en database med raskest mulig lesing og skriving av data. Med hovedbruken definert i punkt 1.4.2 er tilgjengeligheten til applikasjonen en prioritet. Dermed har det blitt tatt utgangspunkt i en webapplikasjon for å unngå å skjule applikasjonen bak nedlasting og installasjon av programvare.

For å behandle kravet om brukervennlighet skal webapplikasjonen være en single-page applikasjon for å gjøre navigering av applikasjonen så responsiv som mulig (Docs, n.d.). Applikasjonen vil fortsatt være delt opp i sider, men på skift av en side vil kun relevant data hentes for å redusere mengden trafikk. Dette vil gjøre at nettsiden responderer raskt til navigering, og venting vil kun foregå i form av lesing fra databasen. Sidestrukturen vil være liknende en desktopapplikasjon i likhet med strukturen i Excel-arket 2.1. Det er tenkt å inkludere et dashbord, med generell informasjon om prosjekter og eventuell statistikk.

Det er blitt utformet et utkast relatert til utseendet og struktur på applikasjonen i form av tegningen vist i figur 2.2.2-1. I denne vises to mulige strukturer, begge med utgangspunkt i en prosjektliste basert på den i Excel-dokumentet (slik som vist i figur 2.1-1). Den nederste



Figur 2.2.2-1: To initielle forslag for struktur på applikasjonen

strukturen er basert direkte på Excel-dokumentet (slik som vist i figur 2.1-2), og gjør ingen store visuelle endringer fra denne, mens den øvre er tenkt at skal ha tasks og gates på to forskjellige sider for å minimere unødvendig datainnlasting.

2.3 Kompetansebehov

For å utvikle en webapplikasjon kreves det ulik kompetanse for å dekke etterspurte krav, samt skape et vellykket produkt som arbeidsgiver kan være fornøyd med. Den nødvendige tekniske kompetansen for prosjektet er håndtering og implementering av programvare, back-end og front-end.

Videre kompetanse knyttet til spesifikke rammeverk og kodespråk vil være nødvendig

avhengig av hvilke fremgangsmåter og utviklerverktøy som blir brukt. Varierende mengder kompetanse kreves for forskjellige rammeverk, og dette må bli tatt hensyn til når det skal settes stilling til valg av disse.

Ettersom det også er nødvendig å designe et brukergrensesnitt for applikasjonen er det også et delvis behov for grunnleggende kunnskaper innenfor UI/UX.

2.4 Avgrensninger

Som nevnt i punkt 2.2.1 har oppdrasgiver gitt få avgrensinger når det kommer til design av applikasjonen. Det er derimot satt en god del funksjonelle krav. Disse kravene blir fremstilt senere i rapporten i punkt 4.11, da det blir gjort en gjennomgang av avgjørelser som er blitt tatt for løsningen, enten basert i krav fra oppdragsgiver eller interne krav fra prosjektgruppen for å minske prosjektets omfang.

2.4.1 Arkitekturbegrensninger

Oppdragsgiver har lagt frem et krav om at databaseløsningen som skal tas i bruk er deres interne SQL Server løsning, og at databasen dermed må implementeres med dette i tankene.

Prosjektet er åpent til alle øvrige arkitekturvalg etter eget skjønn ettersom oppdragsgiver ikke har preferanser eller begrensninger utover databasevalg.

2.4.2 Funksjonelle krav til programmet

Det er valgt å avgrense prosjektet til å kun inkludere de sidene som allerede er i Excel-dokumentet, med en ekstra arkiv-side for de prosjekter som er ferdig. Variablene som ble endret på av personalet i det tidligere Excel-dokumentet skal også kunne redigeres i den nye løsningen, samt at det vil påvirke andre fremstillinger av info i sanntid. En nærmere fremstilling av disse individuelle kravene blir gått gjennom i punkt 4.11.1 samt i A.2.

2.4.3 Vurderte funksjonaliteter

Optimalt sett ville applikasjonen hatt en form for sanntidsoppdatering på tvers av brukere. Ettersom dette er et komplisert tema som prosjektgruppen ikke har erfaring med er det bestemt at dette vil nedprioriteres. Dette vil resultere i en løsning som er mindre optimal

men mer gjennomførbar. Implementasjonen for sanntidsoppdatering kunne for. eks blitt gjennomført ved en web-socket, men ble vurdert som altfor tidskrevende.

Autentiseringmuligheter er nødvendige for en fullstendig separering av administratorbrukere og vanlige brukere. Det er dog ikke strengt tatt nødvendig ettersom det kan regnes med at alle som har tilgang til nettsiden er ansatte i TechnipFMC uten ondsinnede hensikter. Autentisering skal legges opp til, men dersom det viser seg å være for tidkrevende å implementere kan det nedprioriteres til fordel for mer nødvendige funksjonaliteter.

2.5 Ressurser

Som en veiledning for arkitekturen til applikasjonen har FMC gitt tilgang til en kopi av Excel-dokumentet som presenterer deres tidligere løsning. To Eksempler på hvordan Excel-dokumentet er utformet er gitt i figurer 2.1-1 og 2.1-2. Underveis i utviklingen skal Excel-dokumentet bli tatt i bruk som et visuelt hjelpemiddel for utformingen av produktet, samt kilde for de matematiske utregningene som ligger bak de ulike datoene og variablene som fremstilles for prosjekter.

Oppdragsgiver har lagt frem en serverløsning gjennom deres interne systemer i TechnipFMC Norge. Denne skyløsningen består av en SQL Server lokalisert på TechnipFMC Norge sine lokaler i Kongsberg Teknologipark, og inneholder all data TechnipFMC har lagret på databaser. I denne skyløsningen har det blitt dedikert en enkel database for produktet hvor det kan opprettes tabeller etter egen struktur.

På forespørsel fra prosjektgruppen har IT-avdelingen til TechnipFMC Ågotnes anskaffet en enkelt desktop til serverformål slik at applikasjonen kan hostes på denne. Sammen med denne er det gitt innloggings-informasjon til en enkelt bruker i deres interne system, slik at serveren kan logges på nettverket deres.

Underveis i utviklingen vil det antakeligvis være nødt til å ta ibruk rammeverk og bibliotek som er ukjente. Av denne grunn vil det da benyttes relevant dokumentasjon i forhold til korrekt implementering av disse. Etterhvert som disse tas i bruk vil de bli referert til underveis i rapporten og i litteraturlisten.

2.6 Litteratur om problemstillingen

For inspirasjon til oppgaven, har vi sett på flere tidligere oppgaver. Et prosjekt med et likt utgangspunkt til Bachelorprosjektet er *Web application for managing corporate resources* Elisabeth Marie Hovden (2019) som utviklet en Webapplikasjon for å erstatte et firma sitt Excel dokument. Denne gruppens løsning var en Webapplikasjon, med Python som backend og ren Javascript for å holde siden interaktiv. Utifra konklusjonen deres, og hvordan de ville gjennomført prosjektet dersom de kunne startet på nytt, virker det som at en Webapplikasjon var en god løsning. De påpekte også at Javascript tok for mye tid og at et rammeverk som Vue, React eller Angular hadde økt hastighten på utviklingen. Et annet prosjekt som det er hentet inspirasjon fra stammer fra NTNU, *System development of dashboard application - Visualizing customer energy data* Torbjørn Bakke (2022). Denne oppgaven godt skrevet og hadde mye god dokumentasjon som har vært inspirerende både for produktet og rapporten. Det viktigste som ble tatt fra rapporten var en bekreftelse på Nuxt/Vue og hvordan det hadde blitt brukt til å utvikle dashbordet til applikasjonen. Dette var noe som gjenspeilet gruppens initielle løsning beskrevet i punkt 2.2.2.

Kapittel 3

Prosjektmetodikk

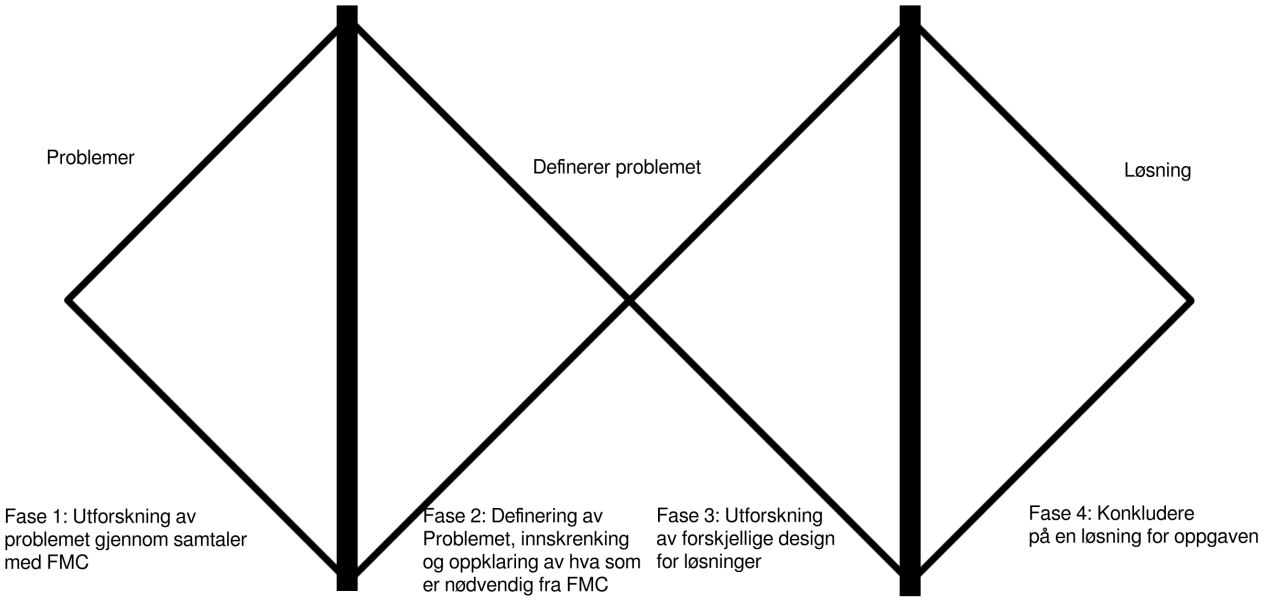
I dette kapittelet vil vi diskutere prosjektets arbeidsmetodikk og designprosess. Målet er å etablere en konkret og forhåndsbestemt tilnærming til prosjektarbeidet, både når det gjelder design og utførelse. Designprosessen vil bli drøftet ved å gjennomgå dobbeltdiamanten i punkt 3.1.1 og Design thinking i punkt 3.1.2. For utviklingsmetodikk vil en bred versjon av agil metodikk og fossefallsmetoden bli drøftet. Dette er ikke alt som ble diskutert, men det er heller valgt å diskutere egenskaper ved utviklingsprosessen en selve navngitteprosesser, da det var dette prosjektgruppen var mest komfortable med.

3.1 Designprosess

Ettersom arbeidsgiveren har gitt oss stor frihet, og samtidig har klare krav til funksjonalitet (se punkt 2.2.1), er det nødvendig med grundig designarbeid. Det er derfor satt av tid til å definere prosjektets designprosess ved å undersøke ulike metoder. Undersøkelsen er gjort ved å velge den eller de metodene som passer best for vårt formål ved å undersøke forskjellige prossesser og enkeltegenskaper ved andre. Målet etter undersøkingen er å komme fram til en egentilpasset prosess som er en blanding av det som var undersøkt. De to mest drøftede prosessene er dobbeltdiamanten i punkt 3.1.1 og Design Thinking i punkt 3.1.2. Google design sprints var noe som også ble sett på, men for rapporten sin del hadde det lite å si for sluttresultatet, som er grunnen til at det blir unnlatt da prosessen ikke virket like relevant som de to andre.

3.1.1 Dobbeltdiamanten

Designprosessen dobbeltdiamant går ut på å definere to helhetlige faser for design utviklingen. Den første fasen (diamanten) blir brukt på å definere problemstillingen som TechnipFMC står ovenfor, og hva de er ute etter i en løsning. I fase 2 (neste diamant) så skal et design for løsningen defineres. Grunnen til at disse to prosessene tenkes på som diamanter er formen som starter som et punkt og blir bredere før den samler seg til et punkt igjen. Tanken er at man starter snevert og er åpne for alle ideer og spørsmål, før man innskrenker til slutt for å komme til en konklusjon. Dette gjelder både del 1 og del 2 (Lyons-Kokkin, 2021). Under er et diagram som viser hvordan en slik prosess hadde sett ut med vår oppgave.

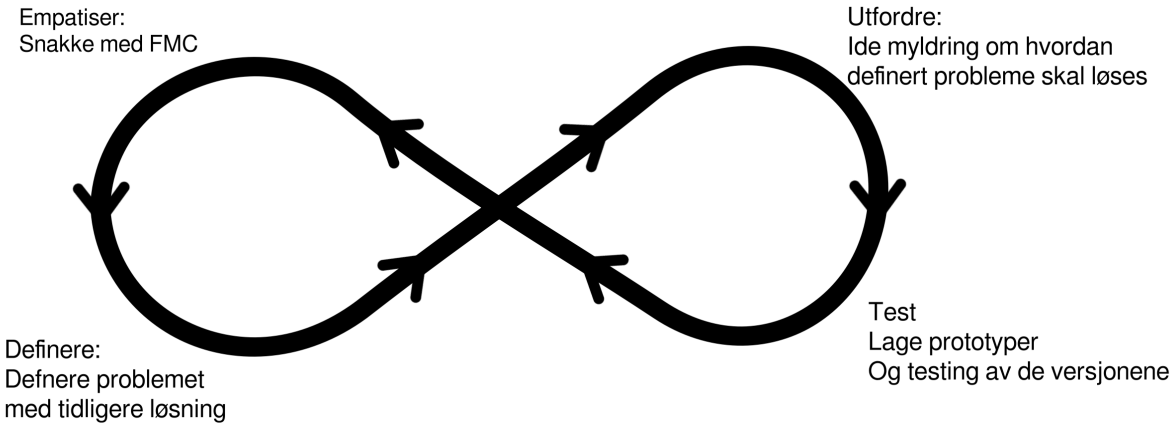


Figur 3.1.1-1: Diagram av et forslag for en dobbelt diamant design prosess for oppgaven

Diagramet viser til de fire fasene med forklarende tekst for hver av de. Det er viktig å påpeke at dette ikke var det endelig, men er mer hvordan denne prosessen skulle eventuelt ha sett ut i dette prosjektet. For prosjektets del virker dobbelt diamanten svært attraktivt og kommer derfor til å bli en stor inspirasjon til den endelige designprosessn.

3.1.2 Design Thinking

Design Thinking er særlig effektiv for å sikre at sluttproduktet er brukervennlig og møter de faktiske behovene til brukerne (Han, 2022). Prosessen er veldig lik den til dobbeltdiamanten, men har et større fokus på utviklingen av tester og prototyper og en prosess som skal gjentas flere ganger. Prosessen går ut på å utforske brukerens behov, for så å definere og utfordre de behovene etterfulgt av prototyping og testing med brukertilbakemeldinger (Han, 2022). Denne prosessen er da mer iterativ og kjøres i runder, med feedback fra brukeren .



Figur 3.1.2-1: Diagram av et forslag for en Design thinking prosess for oppgaven

3.1.3 Diskusjon rundt designprosess

Både dobbeltdiamanten og Design Thinking er gode alternativer, men prosjektet heller mer mot dobbeltdiamanten, da den passer godt prosjektgruppens preferanse for design, men også har et stort fokus på å se flere muligheter, for så å snevre inn scopet. Prosjektet er veldig åpent og behøver en større grad av utforskning av dens forskjellige deler, noe som gjør at dobbeltdiamanten virker mer relevant for denne oppgaven (Lyons-Kokkin, 2021). Konkluderende er det valgt en versjon av dobbeltdiamanten i starten av prosjektet for så å gå over til noe som likner mer den itererende Design Thinking under selve utviklingen. Her er tanken at selve strukturen og ideen blir definert tidlig, mens små designavgjørelser blir gjort underveis i utviklingen gjennom prototyper/iterasjoner og møter med arbeidsgiver.

3.2 Utviklingsmetodikk

En god utviklingsmetodikk er viktig for å sette rammene for hvordan prosjektet skal oppnå ønsket resultat innen tidsfristen. Det er mange ulike utviklingsmetoder som har fordeler og ulemper ovenfor hverandre. Derfor har vurdering av ulike utviklingsmetoder vært viktig for å finne ut hvilken metode som var best for prosjektet. Aspekter som var med å påvirke valget var blant annet inkludering av, og hyppig kommunikasjon med arbeidsgiver, samt tydelig organisering og oversikt av arbeidsoppgaver. Den planlagte utviklingsperioden var ikke lang, og det var derfor viktig med god struktur og jevnlige tilbakemeldinger fra arbeidsgiver for en effektiv utviklingsprosess.

3.2.1 Agile Metoder

Agile metoder er et rammeverk hvor gjennomføringen av utviklingen tar en iterativ tilnærming mot sluttproduktet. Arbeidet deles opp i korte tidsrammer som kalles for iterasjoner eller sprinter, og har vanligvis en varighet på opptil noen uker. En viktig del av agile metoder er hyppig kommunikasjon med arbeidsgiver for å tilpasse produktet i henhold til tilbakemeldinger og etterspørsel av krav. Agile metoder er åpen for endringer underveis i utviklingen. Fordelen med god kommunikasjon mellom kunden og andre involverte i prosjektet er at sjansen for at produktet utvikles feil minimeres. (Vilmur, 2020).

3.2.2 Fossefallsmetoden

Fossefallsmetoden, også kalt rigid, er en prosess som blir delt opp i flere faser, hvor det meste er definert på forhånd anngående *krav, design, utvikling, verifisering, lansering og vedlikehold*. Det jobbes med i bolker som skal gjøres ferdig og det er da ikke vanlig å bevege seg tilbake til en bolk etter at man er ferdig. Dette gjør det med andre ord vanskelig å gjøre endringer underveis som prosjektet progresserer. Det vil derimot være svært effektivt hvis alt er forhåndsdefinert og det ikke er grunn for å endre på krav underveis. (Sander, 2023).

3.2.3 Metodediskusjon

For utviklingsmetodikken er det blitt valgt en versjon av agil metodikk, med et kanbanbrett som blir tilbydd gjennom GitHub (Videre formidlet i 4.12.1). Fossefallsmetoden ansees som lite effektiv for prosjektet fordi det er så åpent. Det er stor sannsynlighet for at definisjoner av

krav satt tidlig i prosjektet må endres underveis i utviklingsprosessen. Under slike tilstander er det viktigere og mer verdsatt med en tilpasningsdyktig metodikk, som fører til at prosjektet passer bedre en agil utforming.

3.3 Design- og metodikkonklusjon

En designprosess med dobbeldiamantmetoden har blitt vurdert ut fra diskusjonen i kapittelet.

1. Starten av prosjektet skal preges av møter innad i prosjektgruppen for å idemyldre en løsning på oppgaven.

2. Et møte med oppdragsgiver må settes opp for så å snevre inn funksjonene og løsningene oppdragsgiver ønsker. Videreutvikling gjøres for å utarbeide detaljer av arkitekturen på oppgaven for så å snevre inn igjen etterpå for å få en mer konkluderende løsning.

3. Gruppen har også lagt opp til bruk av deler av Design Thinking ved å gjennomgå prototyper, detaljer av løsninger, og utforminger sammen med oppdragsgiver.

Grunnlaget for dette er at prosjektet er åpent og endringer må antakeligvis utføres underveis. Oppdragsgiver må også få tid til å finne fram informasjon, som f.eks om hvilken single sign-on og database de ønsker å benytte. I tillegg er det mange tall, variabler og spesifikke utregninger den gamle løsningen tar ibruk pga. sin natur som et Excel-skjema. For å sikre en kvalitativ fremdrift for prosjektet blir møter med oppdragsgiver viktig for å bekrefte sammenhengen mellom disse.

Med aspekter som har blitt omtalt tidligere i kapittelet, vil utviklingen av løsningen være en egendefinert versjon av agile metoder. Prosjektet vil følge en iterativ tilnærming mot sluttproduktet, men vil dele det opp i stedet for å ha en fungerende demo på hvert steg. Hver iterasjon starter med defineringer av issues som må gjøres før de utføres gjennom prosessen. Iterasjonene vil også ha testing og vurdering av iterasjonene på slutten innad i prosjektgruppen for å overholde status på prosjektet samt gjennomføre realistiske vurderinger på fremgangen.

Kapittel 4

Design av prosjektet

Kapittelet tar for seg de forskjellige designvalgene som har blitt vurdert og bestemt for den nye løsningen. Først vil temaene angående valg samt alternativer som kunne ha blitt gjort bli presentert, etterfulgt av en konkluderende avgjørelse og begrunnelse for hvert tema. Etter alle designvalg som anvendes i den endelige løsningen kommer et konkluderende sammendrag av webapplikasjonens helhetlige design. Med diskusjonen av designet vil utformingen av produktets utviklingsløp bli diskutert, etterfulgt av en evalueringsplan for å avslutte kapittelet.

4.1 Verktøy

Under utviklingen av designet, og selve arbeidsflyten under prosjektet, har flere verktøy blitt tatt ibruk. Formålet med verktøyene som blir spesifisert her er å oppnå en bedre fremstilling av status og arbeid på prosjekt, illustrative designfremstillinger og generelle arbeidsverktøy som ikke har direkte innflytelse på techstacken.

Figma Figma er et gratis online brukegrensesnitt for å prototype og kollaborere på visuelt design. I dette prosjektets tilfelle er Figma brukt for å lage en visuell prototype av applikasjonen samt for å få tilbakemelding av arbeidsgiver på brukervennligheten til funksjonalitetene de har bedt om.

Overleaf Overleaf er et online skriveverktøy brukt for å skrive i LaTeX, som igjen er en dokumentasjonsform tiltenkt enkel og profesjonell strukturering av store dokumenter(som f. eks. en bachelorrapport). Som studenter på et datastudie vil det å skrive en tekst i delvis kodeform ikke være så ukjent, og dermed virker dette som et godt valg for en så omfattende tekst.

GitHub GitHub er blitt brukt som en skyløsning for versjonskontroll, og i samkjør med GitHub er GitHub Projects tatt i bruk. GitHub Projects er en prosjektoversikt med innebygde funksjonaliteter som f. eks. kanban-board som er blitt tatt i bruk underveis i prosjektet.

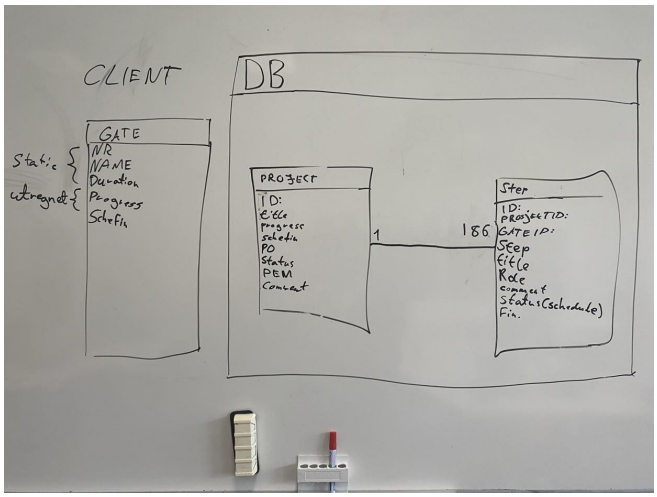
Vercel Vercel er en frontend-vert som kan integreres med GitHub for å enkelt sette opp nettsiden på et domene. Denne siden vil også oppdateres automatisk hver gang main-grenen til prosjektet oppdateres.

NuxtDevTools Nuxt DevTools gir insikt i applikasjonen gjennom mer synlig statistikk av komponenter og stores samt tall på tid brukt i henting av prosjektet.

IDE Visual Studio Code er den foretrukne IDE-en en majoritet av prosjektgruppen har valgt å bruke underveis i utviklingen, og har støtte for de aller fleste språk og rammeverk. VSCode er et Integrated Development Enviroment (IDE) og brukes for å mer effektivt utvikle kode i et miljø med hjelpefunksjonaliteter for enklere skriving av kode. Funksjonaliteter som tilbys er testing, språkservere, syntaksmarkering og feilretting. For anvending av kode er det svært nyttig med et slikt verktøy, som gjør at gruppen har valgt å ta det ibruk.

4.2 Designutføring

Designet av løsningen har blitt utført stort sett etter planene konkludert med i punkt 3.3. God tid har blitt brukt på å utforske muligheter for løsninger da oppgaven ble utdelt, og for design ble en generell arkitektur idemyldret, samtidig som at muligheter ble utredet slik at de kunne bli fremstilt til oppdragsgiver på det første oppdateringsmøtet.

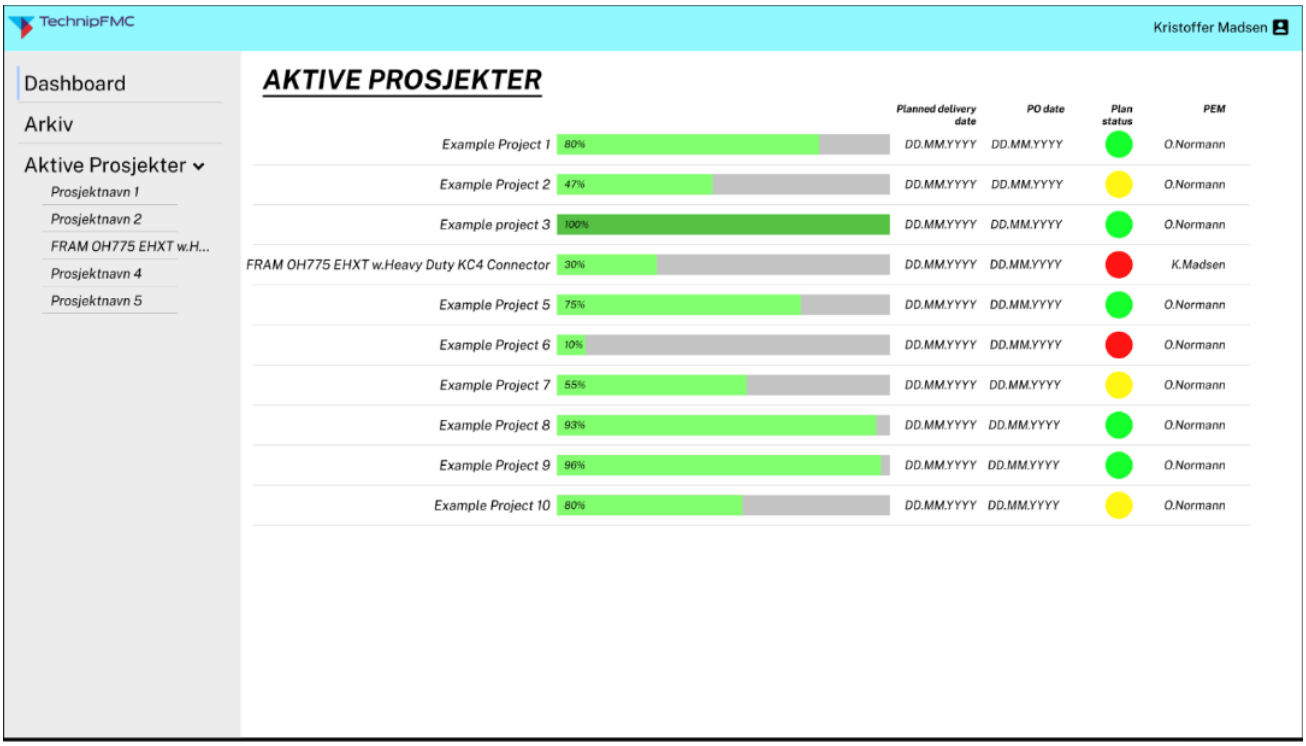


Figur 4.2-1: En generell databasestruktur som ble utarbeidet før møte med oppdragsgiver

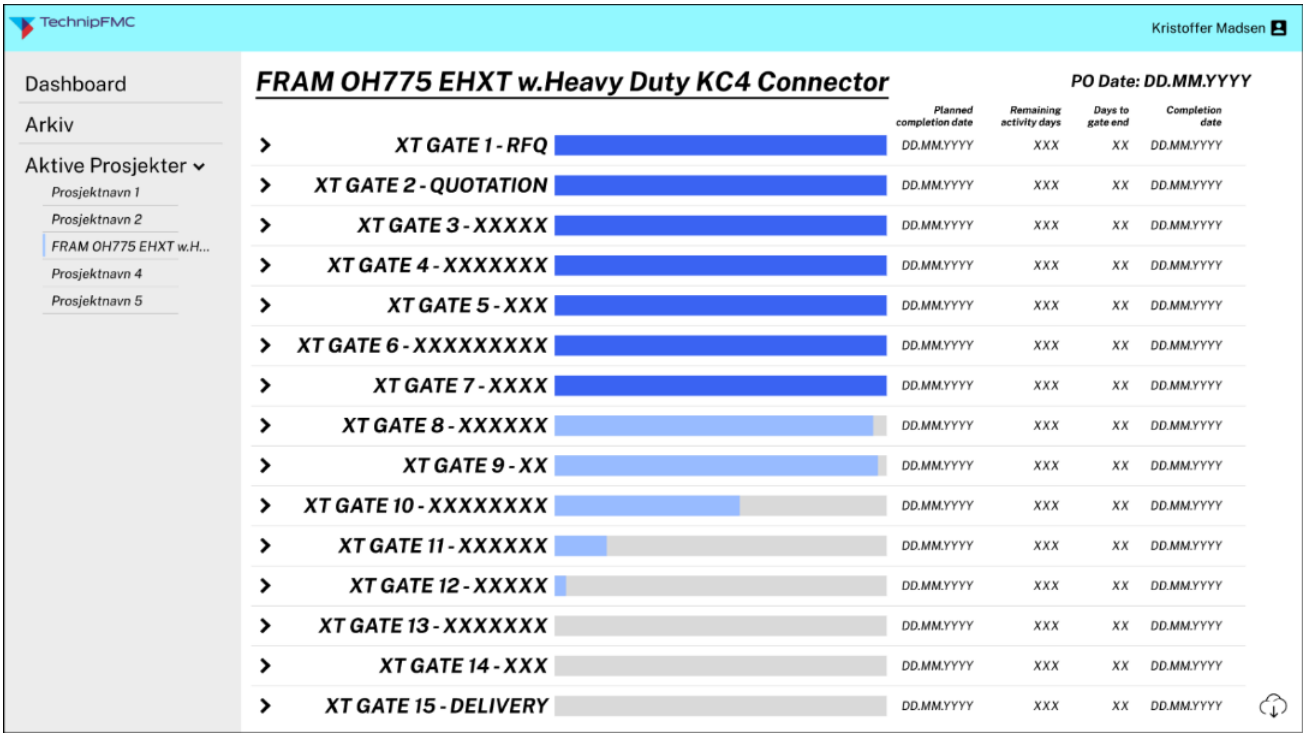
Ved første møte ble detaljerte forklaringer av funksjoner på oppgaven gjennomgått, samtidig som at et mer detaljert krav for arkitekturen ble avklart (se 2.2.1). Som et resultat av dette møtet har løsninger blitt idemyldret for en mer definert problemstilling, før den ble strammet inn med en prototype.

4.2.1 Protoype

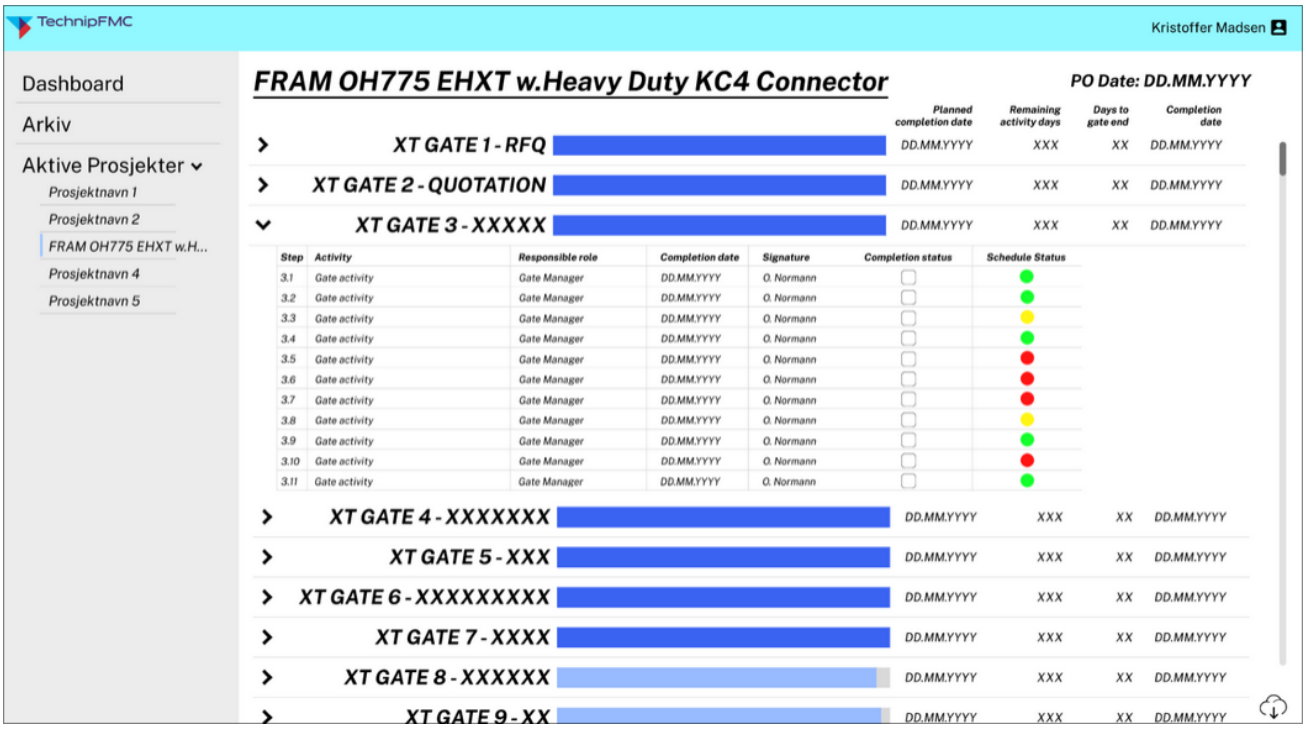
Basert på Excel-dokumentet fra TechnipFMC ble det laget en digital papirprototype som vist i figurer 4.2.1-1, 4.2.1-2 og 4.2.1-3. Prototypen ble laget etter møtet med oppdragsgiver for å kunne stramme inn løsningen og hvordan de ville ha den utformet. Etter et til møte med oppdragsgiver for å fremvise prototypen var det meste av designet på plass.



Figur 4.2.1-1: Dashbordet til prototypen



Figur 4.2.1-2: Prosjektsiden



Figur 4.2.1-3: Åpning av et prosjekt på prosjektsiden

4.3 Web- eller desktopapplikasjon

Den initielle løsningsidè som nevnt i punkt 2.2.2 var en web-applikasjon. Grunnet oppgavens natur virker dette som den mer naturlige løsningen på oppgaven. Til tross for dette er det fortsatt vurdert muligheten for å utvikle en desktopapplikasjon for å kunne gjennomgå begge alternativene på første statusmøtet med oppdragsgiver.

4.3.1 Webapplikasjon

Hovedgrunnen til at en webapplikasjon ble foreslått som den initielle løsningen (omtalt i punkt 2.2.2), var tilgjengeligheten det tilbyr. Enhver ansatt i TechnipFMC som har tilgang til deres interne nettverk knyttet til kontoret på Ågotnes kan enkelt skrive inn en lokaladresse i nettleseren, og få tilgang til webapplikasjonen.

Det er også en nevneverdig fordel at alle medlemmene på prosjektgruppen har delvis erfaring med webutvikling, selvom mengden erfaring varierer. Ved webutvikling må det antakeligvis læres nye rammeverk og bruk av biblioteker som er ukjente, men i den kontekst av at det

bygger på en utviklingsteknikk som er kjent fra før.

4.3.2 Desktopapplikasjon

En desktopapplikasjon vil ha fordel ved at den vil unngå utrulling til en server på samme måte som en webapplikasjon. Den ville også vært mer optimalisert enn en webapplikasjon, men flaskehalsen i begge tilfeller vil være tilkoblingen til databasen, og dermed er ikke dette en betydelig fordel med en desktopapplikasjonsløsning. Det samme gjelder en desktopapplikasjons mulighet til å kjøres uten nettilgang, ettersom databasen er avhengig av internett (Gomez, 2023).

I motsetning til en webapplikasjon må derimot en desktopapplikasjon lastes ned, installeres og deretter kjøres av brukeren (Gomez, 2023). Skulle applikasjonen oppdateres i fremtiden vil denne prosessen bli gjentatt. Dette bryter med tilgjengelighetsbehovet slik det er beskrevet i punkt 1.4.2 på en måte en webapplikasjon ikke gjør.

Til slutt bør det og nevnes at prosjektgruppen ikke har erfaring med utvikling av desktopapplikasjoner, som leder til enda et lag av manglende kompetanse som må tilegnes underveis i prosjektutviklingen

4.3.3 Konklusjon

En webapplikasjon vil gi en mye høyere grad av tilgjengelighet. Fordi produktet grunnet dens natur som koblet opp mot en database og dermed allerede et nettverk, virker dette som den beste løsningen. Det er også nevneverdig at det forventes mindre, om enn fortsatt betydelige, mangler i den nødvendige kompetansen ved utvikling av en webapplikasjon.

Med det er et valg tatt for å gå med valget fra den initielle løsningen i punkt 2.2.2, en webapplikasjon.

4.4 Sideoppdeling

Løsningsdesignet er delt opp i 4 sider:

1. **Dashbord** En side som skal fremvise mer generelle informasjon om nettsiden.

2. **Prosjektliste** En liste over alle de aktive prosjektene samt litt mer info om hver av de enn dashborder.

3. **Prosjektside** Fremstilling av et helt prosjekt med alle dens detaljer.

4. **Arkiv** En liste over alle prosjekt som ikke er aktive.

4.5 Atomisert webdesign

Med en stor webapplikasjon som inneholder mange tall, utregninger og mulige komponenter som må gjenbrukes på flere sider, skal løsningen utvikles på en atomisert måte. De minste komponentene som knapper, lister progressivebarer og dato fremstillinger, skal ha egne komponenter som er isolerte og enkle å gjenbruke flere steder i webapplikasjonen uten at de påvirker andre deler av koden. En større del av applikasjonen, som prosjektliste skal så ta ibruk disse komponentene, og passe på at tallene som blir matet til de er riktige. Denne komponenten vil så være pakket inn i en komponent med et enda større omfang o.s.v. Dette vil skape mer stabilitet under utviklingen og enklere implementering av de samme komponentene på flere steder under utviklingen. Hver komponent skal være uten påvirkninger fra andre komponenter, for å være så isolert og gjenbrukbar som mulig. Nedsider ved et slikt design har en lengre oppstarts tid fordi utviklingen starter med de minste komponentene og må sørge for at de kan fungere i isolasjon (Davidson, 2023).

4.5.1 Vue

En av de første beslutningene prosjektgruppen ble enig om, var å bruke Vue i løsningen. Et medlem av gruppen hadde en del erfaring med rammeverket, og utifra dokumentasjon og erfaring virket det som et godt verktøy for utviklingen av prosjektet. Vue er et frontend Javascript framework med en komponentstruktur og tilknyttet HTML-fragmenter som gjør det svært effektivt. En av dets hovedfunksjoner er muligheten for å deklarere verdier i HTML-template og tracking av Javascript states for å oppdatere DOM (*The Progressive JavaScript Framework*, 2024). For prosjektet virker disse funksjonene gode for å effektivisere utviklingen og for å reaktivt oppdatere variabler ettersom de endres av brukeren. En stor egenskap av Vue er sin egenskap til å ha single-file-komponenter som vil da i praksis være atomiserte 4.5. Vue har mange likheter med React, og kan ansees som et relativt likt rammeverk med små forskjeller. Vue er dermed sett på som et veldig godt valg, med delvis

erfaring fra tidligere, enkel implementering av atomisert webdesign og enkel gjenbruk av kode for effektivitet i prosjektet. (*The Progressive JavaScript Framework*, 2024).

4.6 Reaktivitet

Med Vue som et verktøy er det muligheter å ta ibruk dens reaktivtets funksjon. Reaktivitet gir muligheten til å behandle variabler tilnærmet et Excel-skjema. Det er utmerket for dette prosjektet som er grunnen til at det har vært en stor del av designet og løsningen av denne oppgaven, og vil dermed utdypes videre i denne seksjonen.(*Reactivity in Depth*, 2024).

4.6.1 Single Page Application (SPA)

En Single Page Application (SPA) er en type webapplikasjon som laster inn en enkelt HTML-side, og dynamisk oppdaterer innholdet på den samme siden etter hvert som brukeren navigerer gjennom applikasjonen. I XT-Gates applikasjonen har dette vært tatt i bruk for å gjøre at nettsiden er rask og responsiv.

Med Vue.js håndteres SPA-rutingen internt i applikasjonen, og URL-ruting til ulike deler av applikasjonen blir administrert automatisk. Dette betyr at når brukeren navigerer mellom ulike visninger eller komponenter, blir bare den nødvendige dataen lastet dynamisk i bakgrunnen, og hele siden trenger ikke laste på nytt. Dette reduserer nettverkstrafikken betydelig og forbedrer responstiden ettersom det ikke kreves en ny forespørsel til serveren for hver sideendring.

4.6.2 Excel-eksempel

Det klassiske eksempelet for å beskrive reaktivitet i Vue er å sammenligne det med et Excel-ark. I et Excel-ark kan du skrive inn variabler og definere de i forskjellige celler, for å så ta ibruk disse variablene i andre celler gjennom formler. Så langt er dette veldig likt vanlig koding, men der det skiller seg er hva som skjer hvis du endrer en av variablene. I et Excel-skjema vil resultatet forandre seg utifra endringen i variablen. Et eksempel på et slikt ark vises i 4.6.2-1

Tabell 4.6.2-1 : Eksempel på reaktivitet i Excel

	A	B
1	3	Verdi 3
2	2	Verdi 2
3	5	Formel er =1A + 2A

I Javascript vil ikke variablene definert tidligere ha en effekt på andre variabler som er definert med den variablen. De må istedet initialiseres på nytt, eller oppdateres gjennom en metode.

```
let A0 = 1
let A1 = 2
let A2 = A0 + A1

console.log(A2) // 3

A0 = 2
console.log(A2) // Fortsatt 3
```

kodeutdrag 4.1: Et eksempel på hvordan ting ikke er reaktivt i vanlig JavaScript

Som sett i punkt 4.1 vil ikke Javascript kunne behandle reaktiviteten til et Excel.ark. Vi kan derimot imitere dette ved å definere en funksjon som oppdaterer denne variablen. Funksjonen må høre etter en endring i enten A1 eller A2, og så kjøres hvis en av variablene blir endret. I Vue blir dette gjort ved å definere et spesielt objekt som har en *getter* og *setter* som oppdaterer all bruk av verdien til objektet (*Reactivity in Depth*, 2024). Dette vil være en veldig enkel forklaring på hvordan det fungerer med reaktivitet i Vue. En variabel blir definert til et objekt gjennom *ref*(), og for å bruke disse i en kalkulasjon som skal høre etter endringer pakkes det inn i *computed*. Et kjapt eksempel på dette blir vist i listing 4.2.

```
const A0 = ref(1)
const A1 = ref(2)

const A2 = computed(() => A0.value + A1.value)
console.log(A2) // Gir ut 3

```

```
A0.value = 2
console.log(A2) // Gir ut 4
```

kodeutdrag 4.2: Et praktisk eksempel på hvordan reaktivitet oppfører seg i Vue

4.7 Nuxt

Etter avgjørelsen om å ta ibruk Vue, var det naturlig å vurdere Nuxt. Nuxt er en videreutvikling av Vue, med mer fokus på å kombinere backend og frontend for å utvikle en full-stack web-applikasjon. Funksjoner som Nuxt gir tilgang til er enklere implementering server-side rendering, enklere implementering av routing, god integrasjon av server, og implementasjon av middleware. Det tilsvarende rammeverket i React er Next, som går ut på å implementere mye av det samme. Dermed er det vurdert å bruke Nuxt for å ha en homogen kodebase med god og integrert dokumentasjon samt dens nyttige funksjoner (*Introduction*, n.d.). Både front-end og backend deler av prosjektet vil innholde Javascript kode, og vil bli godt knyttet sammen på en måte som gir mening. Det vil også bli mindre whiplash da det ikke er grunn til å hoppe mellom språk, hvis både frontend og backend er skrevet i Javascript. Et alternativ som ble vurdert, var Express.JS som en backendløsning. Med Nuxt sin dokumentasjon og funksjoner vil derimot utviklings prosessen være mer effektiv, løsningen enklere å implementer, samt at de ekstra funksjonene er nyttige.

4.7.1 Server engine

Med Nuxt kommer en inkludert serverengine kalt Nitro. Denne er integrert sammen med Vue og er laget spesifikt til dette. For prosjektets del har vi brukt Nitro for samme grunn som Nuxt, se punkt 4.7. I prosjektet er Nitro tenkt å bli brukt for autentisering gjennom middleware og henting/oppdatering av databasen som en API. (*Server Engine*, 2024)

4.8 Statemanagment

Nuxt tilbyr kraftige state bibliotek og alternativer for å utvikle en reaktiv nettside som deler state mellom kompenenter. En knapp i en komponent skal kunne ha en effekt på et tall i en annen komponent. Med en felles state kan dette implementeres enkelt da variablen som behandles i begge komponentene kan bli sett på som den samme på tvers av komponenter. I

punkt 4.6, ble en forkalring på hvordan reaktivitet fungerer, med å ta ibruk statemanagement kan denne reaktive variablen bli delt på tvers av komponenter, som vil gi prosjektet større mulighet til å ta full fordel av dens egenskap. (*State Management*, 2024)

4.8.1 Pinia

For å forenkle statemanagement har vi tatt i bruk et offisielt tilbydd bibliotek av Vue kalt Pinia. Pinia er et *store* bibliotek som er en videreutviklet versjon av det tidligere brukte biblioteket *Vuex*. Med *Pinia* kan prosjektet definere en *store* som kan bli tatt ibruk like mye i en komponent som i en API routing, og på den måten kan reaktivtet enklere bli brukt og oppholdt gjennom applikasjonen. Prosjektet skal ta i bruk en store for hver *module* av prosjektet, f.eks prosjektStore, gateStore og taskStore, som er de forskjellige delene av et prosjekt. Hvis en i gruppen vil manipulere eller hente data fra en gate, så må de importere gateStore, og så kjøre metoden det trenger fra det importerte store objektet. Gruppen ser for seg at dette vil gi komponentene et knyttepunkt for reaktivitet, og en enkel måte å gjenbruke metoder for reaktivitet på tvers av komponenter.

4.9 Renderingstrategi

For brukervennlighet sin skyld har det blitt vurdert hvilken type framvisning vi vil ha av nettsiden. Målet er å velge den typen teknologi som vil gi lavest latency og innlastingstid for brukeren. Renderingstrategi går i grunn ut på å konvertere kode til et visuelt brukergrensesnitt på brukeren sin skjerm. Som alternativ har tre forskjellige teknologier blitt vurdert.

4.9.1 Client-Side Rendering (CSR)

En klientsiderendret applikasjon vil gi klienten HTML-kode og Javascriptkode for så at klienten framkaller den ferdige nettsiden ved å kjøre Javascriptekoden. Serveren sin oppgave er da kun å sende ut filene, mens klienten må laste og fremstille den på egenhånd. Dette er løsningen som er minst krevende for serveren og også minst krevende å utvikle da det er utgangspunktet til de fleste web-applikasjoner.

4.9.2 Server-Side Rendering(SSR)

Serversiderendring har fordelen av å kjøre Javascriptkoden på serveren for så å gi ut en statisk side til klienten. Dette er mindre krevende for klienten fører til at fremkallingen av nettsiden vil være mye kjappere. Baksiden ved SSR er en mindre mengde interaktivitet, fordi Javascriptkoden allerede er kjørt på serversiden. Dette gjør utviklingen av interaktivitet enten vanskelig eller rett og slett umulig uten å ta ibruk UR som gjennomgås i 4.3.3.

4.9.3 Universal Rendering (UR)

Universalrendring er en form for tjenersidefremvisning som legger mer vekt på tjeneren, men mindre vekt på klienten. Nettsiden vil da framvises raskere av at tjeneren oppgir en ferdig fremvist HTML-kode og laster ned Javascriptkoden etterpå for å gjøre ting interaktivt, i en prosess kalt hydrering. For brukeren vil da nettsiden tilsynelatende lastes raskt inn, mens interaktiviteten til komponenter vil komme etterpå. Hensikten er at nettsiden gir en illusjon av mer hyppig og raskere henting av siden da det visuelle kommer først, og deretter interaktiviteten.

4.9.4 Konkluderende rendring

Rendring er en vanskeligere avgjørelse da det er vanskelig å si hvor utfordrende det er å implementere de forskjellige teknologiene. CSR er den tradisjonelt brukte rendringstrategien og er sjeldent et feil valg. Men med tanke på prosjektets mål om brukervennlighet så ser en ganske stor verdi i SSR som gir løsningen en raskere fremkalling av siden. Ulempen med SSR er da en mindre mengde interaktivitet da javakoden allerede er fremkalt på serversiden. Derfor virker UR som ett veldig godt alternativ med både rask fremkalling og interaktivitet.

4.10 Database

Som gjennomgått i punkt 2.4 har oppdragsgiver satt en avgrensning på at applikasjonen må benytte seg av deres SQL Server-database. Dette var derimot ikke spesifisert i utgangspunktet, og det er dermed undersøkte alternative databaseløsninger for å effektivisere uthenting av data. Anbefaling ble senere overstyrt av oppdragsgiver, men prosjektgruppen har valgt å beholde drøftingen ettersom den er relevant i forhold til optimalisering av prosjektet.

4.10.1 Relasjonsdatabase

En løsning er å utforme en relasjonsdatabase og strukturere en frontend med denne som utgangspunkt. I en relasjonsdatabase opprettes det tabeller hvor dataene er lagret som attributter i rader. Dersom en vil opprette relasjoner mellom tabellene, settes det primær- og fremmednøkler til attributtene som skal knytte tabellene sammen. Applikasjonen vil da kunne hente ut data på relasjonsbasis, og enklere velge og matche det som skal hentes ut. På den måten kan datatrafikken reduseres og kun hente ut relevant data til brukeren. Dette er den databaseløsningen TechnipFMC hadde et ønske om å bruke til prosjektet og dermed vektlegges dette høyt som et alternativ. (IBM, n.d.)

4.10.2 Grafdatabase

Som en alternativ løsning har er en grafdatabase utforsket. En grafdatabase er strukturert i form av noder og kanter mellom nodene, istedenfor tabeller. Nodene inneholder ulike egenskaper og attributter, og kantene representerer direkte relasjonene mellom nodene. Forholdet er dermed lagret i databasen, og vil ikke endres. Fordelen med å ha relasjonen mellom nodene lagret i kantene mellom dem, er at forholdet mellom nodene ikke må beregnes i form av komplekse operasjoner når spørringer til databasen utføres. Det kan resultere i raskere spørringer og at uthentingen av informasjon vil ha lavere latency. Ulempen er at prosjektet mister muligheten til å bruke et "selective query language". Latency i sammenheng med prosjektet er ventetiden det tar å hente datapakker fra databasen til applikasjonen. Resultatet av en grafdatabaseløsning ville potensielt føre til økt optimalisering, som er et av hovedfokusene til prosjektet. (Service, n.d.)

Grafdatabase kan være et godt alternativ ettersom prosjektene til TechnipFMC er sortert i en type trestruktur og kan enkelt oversettes til en graf. Datastrukturen som er ønsket er også ganske statisk, så det å miste tilgangen til SQL er overkommelig.

4.10.3 Konkluderende database

Etter en nøye analyse av begge de to databasestrukturene analysert i punkt 4.10.2 har gruppen kommet frem til at en grafstruktur er den strukturen som passer best til prosjektets data, grunnet dataens naturlige trestruktur, og det er dermed gitt en anbefaling til oppdragsgiver om at dette er databasestrukturen som burde velges. Oppdragsgiver har derimot, når denne anbefalingen ble lagt frem, ble det bedt spesifikt om å tilpasse seg deres

allerede eksisterende database i form av en SQL Server-sky. Ettersom en relasjonsdatabase er fullt mulig, om ikke like optimalisert som en grafdatabase, det er valgt å tilpasse seg til dette kravet.

4.11 Krav

For å skape en felles forståelse av hvordan webapplikasjonen skal fungere som et ferdig produkt, er det planlagt både funksjonelle og ikke-funksjonelle egenskaper. Det kreves at disse egenskapene er en del av det fullførte produktet for å sørge for et vellykket prosjekt. Listene over krav gjør det enklere å sette mål for produktet og sørger for en mer retningsbevisst utvikling. Når produktet er fullført og klar for evaluering, vil listene over krav bli tatt i bruk for å måle suksessen på produktet.

4.11.1 Produktets funksjonelle krav

Produktets funksjonelle egenskaper er tekniske interaksjoner mellom bruker og applikasjon og er definert i følgende tabell:

Tabell 4.11.1-1 : Produktets funksjonelle krav

Krav	**Beskrivelse**
Opprette bruker	Bruker skal kunne opprette bruker
Innlogging	Bruker skal kunne logge inn på brukeren sin
Logg ut	Bruker skal kunne logge ut av brukeren sin
Gi brukere tilgang (admin)	Administrator skal kunne frata brukertilgang hvis en ansatt som har tilgang ikke skal ha det
Opprette prosjekt (admin)	Administrator skal kunne opprette nye prosjekter i prosjektlisten på hjemmesiden.
Arkivering av prosjekt	Prosjekter som er ferdig og ikke lenger er relevante skal kunne arkiveres for å rydde opp nettsiden
Se arkiverte prosjekt	Kunne se gjennom tidligere arkiverte prosjekter
Undo av arkivering	Kunne aktivere prosjekt som har blitt arkivert ved et uhell
Se prosjektstatus	Bruker skal enkelt kunne se status på et prosjekt på hjemmesiden (Prosentstatus, PO-dato osv.)
Se helhetlig status på prosjektene	
Helhetelig status skal linke til prosjekt som ligger bak skjema	
Redigere PO dato på prosjekt	
Oppdatere status på prosjekt	
Mulighet til å legge til alternativt GATE 6	Administrator skal kunne legge til alternative GATE 6 hvis prosjektet trenger modifikasjon utenom normen
Åpne prosjekt	Bruker skal kunne åpne de forskjellige prosjektene
Se Gate-status	Bruker skal kunne se status på gates fra siden med oversikt over gatene
Åpne gates	Bruker skal kunne åpne de individuelle gates
Se underdelinger	Bruker skal kunne se de individuelle underpunkter som tilhører en gate
Endre status på underdelinger	Bruker skal kunne endre status mellom ferdig og ikke for de individuelle arbeidsoppgaver tilhørende en gate
Signere på en task	Når en task settes til fullført skal en bruker kunne signere med sitt brukernavn

4.11.2 Produktets ikke-funksjonelle krav

Produktets ikke-funksjonelle krav er de krav som omhandler de underliggende egenskapene nødvendig for applikasjonen og er definert i følgende tabell:

Tabell 4.11.2-1 : Produktets ikke-funksjonelle krav

Krav	Beskrivelse
Brukervennlighet	Produktet må være lett navigerbart, oversiktlig for brukere og responsivt
Sikkerhet	Applikasjonen må være sikker, og ikke ha informasjon tilgjengelig for offentligheten
Styrbarhet	Applikasjonen må kunne administreres fullstendig av TechnipFMC sine ansatte uten hjelp fra utviklerne

4.12 Prosjektutføring

Denne seksjonen vil ta for seg prsojektts planlagte utviklings metode og plan, og er utformet med tanke på punkt 3.3. Etter arbeidsmetoden og prosjektplanen skal en gjennomgang av prosjektets risikovurdering samt evalueringsplan fremvises.

4.12.1 GitHub Flow

Som nevnt innledningsvis er det ønsket en god organisering av arbeidsoppgaver. En arbeidsstrategi som oppfyller dette, er GitHub Flow. GitHub Flow er en arbeidsstrategi som gruppen planlegger å ta i bruk under utviklingen av prosjektet, ettersom den skaper en oversiktlig arbeidsplan som er enkel å ta i bruk, og inneholder blant annet kanban-board, branching, pull-requests og backlog.

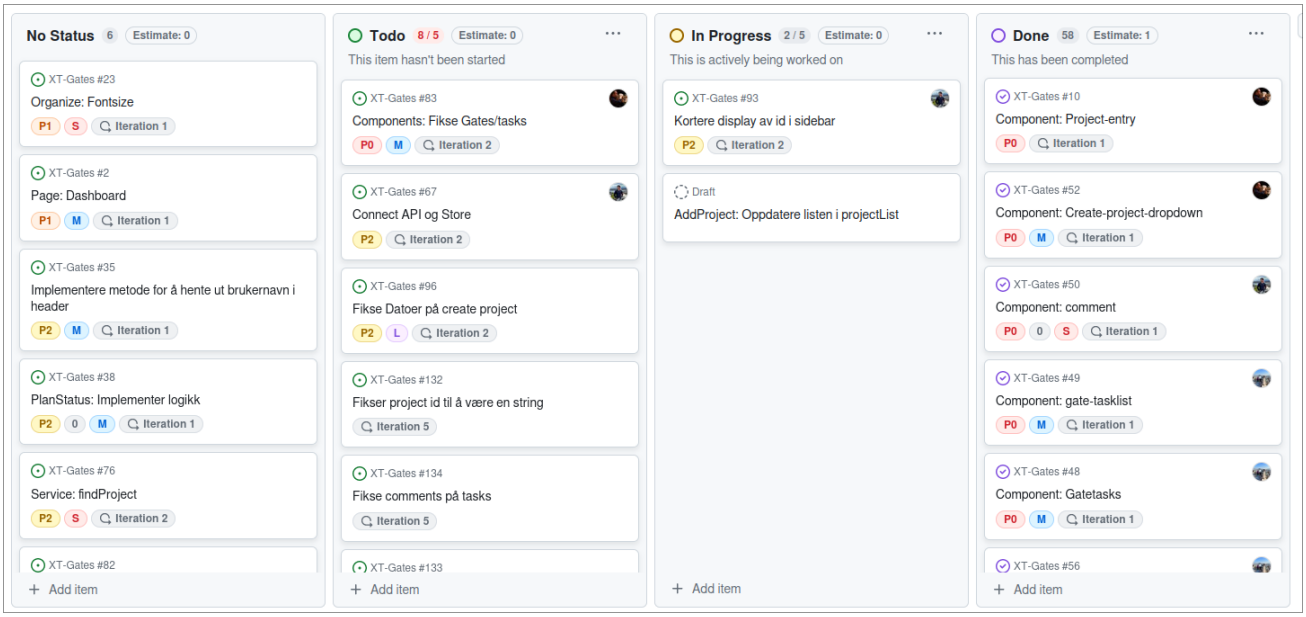
Kanban er en form for Agil metodologi som skaper en visualisering av arbeidsflyten ved å benytte seg av et kanban-board. I et kanban-board finner mann oversikt over steg i prosjektet, og innenfor hvert steg befinner det seg ulike arbeidsoppgaver. Et steg kan være en form for iterasjon eller sprint. Arbeidsoppgavene klassifiseres i forhold til hvor de befinner seg i sprinten.

Ved bruk av GitHub sitt kanban-board kan man opprette iterasjoner, arbeidsoppgaver med beskrivelser og prioritering, holde oversikt over oppgaver som må gjøres, arbeides med og som er fullført i iterasjonen. I prosjektet skal oppgavene kategoriseres i Todo", "In

Progress”og Done”. Prosjektet har en hovedgren kalt ”Main”og man kan utfra denne lage grener hvor en trygt kan gjøre endringer og skrive kode, også kalt branching. Grenene opprettes fra og tilsvarer arbeidsoppgaver gruppen oppretter i kanban-boardet. For eksempel hvis en arbeidsoppgave i iterasjonen er ”20-opprette-template-for-side-bar”, kan et gruppemedlem trykke inn på arbeidsoppgaven, lese beskrivelse av arbeidet, tilegne oppgaven til seg selv og trykke ”Create Branch”.

Etter at utvikleren har utført endringer i grenen sin, kan det sendes en pull-request til Main. Med en pull-request får de andre utviklerne oversikt over gjennomførte endringer i grenen og mulighet til å stille spørsmål og gi tilbakemelding på koden via kommentarer. Endringer gjennomført i grenen blir ikke merget med Main før det blir godkjent av en annen person.

I backlogen blir de vist en oversikt over alle arbeidsoppgaver som må gjennomføres, blir arbeidet med eller er fullført. Dette muliggjør at prosjektgruppen kan ved nødvendighet gå til en tidligere arbeidsoppgave og se hvilke endringer som ble gjennomført. For eksempel dersom en gren som merges med Main skaper problemer for koden uten at det blir oppdaget i forkant. Backlogen vil også gjøre det enkelt å se oppgaver fra tidligere iterasjoner som ikke blir gjennomført.



Figur 4.12.1-1: Product Backlog

Lengden på iterasjonene er satt til 1 uke. Hensikten med ukentlige iterasjoner er å sørge for

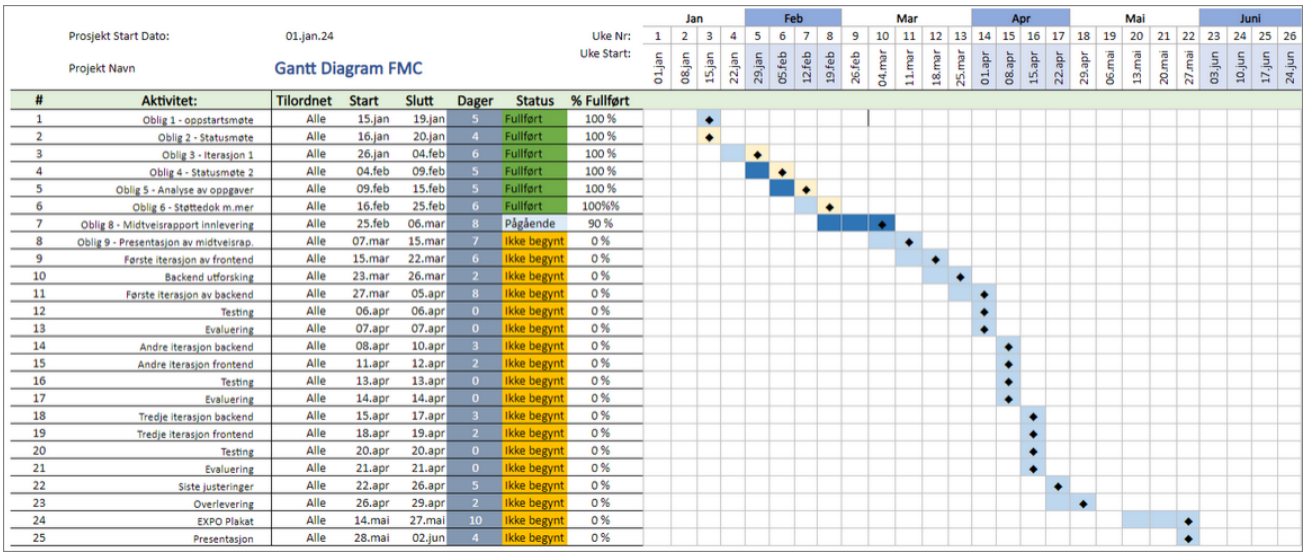
at utviklingen får en stadig fremgang. Korte iterasjoner gir bedre mulighet til å respondere på tilbakemeldinger fra de involverte i prosjektet, og mulighet til å enkelt utføre endringer ved behov. Dette anses som essensielt for å skape en effektiv utvikling og unngå unødvendig arbeid. Ettersom iterasjonene er korte, kan arbeidsoppgaver som ikke blir fullført innen fristen bli flyttet til neste iterasjon.

Figur 4.12.1-2: Viser endringer gjort i en pull-request

4.12.2 Prosjektplan

God planlegging og utforming av en fremdriftsplan er et godt hjelpemiddel for å holde oversikt på hvor man befinner seg i prosjektet, og enkelt holde kontroll på hva som er oppgavene i det nåværende stadiet av prosessen. En fremdriftsplan vil gjøre det enklere for å holde kontroll på frister, planlegge iterasjoner og raskere iverksette nødvendige arbeidsoppgaver. I tillegg hjelper det med å sikre at alle er på samme side og skaper en bedre flyt i arbeidsmiljøet mot hovedmålet.

Verktøyet brukt for å opprette en fremdriftsplan er et GANT-Diagram utviklet i Excel. Gant diagrammet holder blant annet kontroll på navnet av aktiviteten, samt en utregning mellom start og slutt dato som viser antall arbeidsdager tilgjengelig for å fullføre oppgaven. Det er også en meny for å endre status på oppgaven mellom Ikke påbegynt, Pågående og Fullført, og fargen i kolonnen vil endre seg deretter. I GANT-barene er det markert med en diamant i hvilken uke fristen for oppgaven befinner seg. Kolonnen diamanten befinner seg i er lyseblå når den ikke er begynt, mørkeblå når den er pågående og gull når den er fullført.



Prosjekt Start Dato: 01.jan.24

Projekt Navn: Gantt Diagram FMC

#	Aktivitet:	Tilordnet	Start	Slutt	Dager	Status	% Fullført
1	Oblig 1 - oppstartsmøte	Alle	15.jan	19.jan	5	Fullført	100 %
2	Oblig 2 - Statusmøte	Alle	16.jan	20.jan	4	Fullført	100 %
3	Oblig 3 - Iterasjon 1	Alle	26.jan	04.feb	6	Fullført	100 %
4	Oblig 4 - Statusmøte 2	Alle	04.feb	09.feb	5	Fullført	100 %
5	Oblig 5 - Analyse av oppgaver	Alle	09.feb	15.feb	5	Fullført	100 %
6	Oblig 6 - Støttedok m.mer	Alle	16.feb	25.feb	6	Fullført	100%%
7	Oblig 8 - Midtveisrapport innlevering	Alle	25.feb	06.mar	8	Pågående	90 %
8	Oblig 9 - Presentasjon av midtveisrap.	Alle	07.mar	15.mar	7	Ikke begynt	0 %
9	Første iterasjon av frontend	Alle	15.mar	22.mar	6	Ikke begynt	0 %
10	Backend utforsking	Alle	23.mar	26.mar	2	Ikke begynt	0 %
11	Første iterasjon av backend	Alle	27.mar	05.apr	8	Ikke begynt	0 %
12	Testing	Alle	06.apr	06.apr	0	Ikke begynt	0 %
13	Evaluering	Alle	07.apr	07.apr	0	Ikke begynt	0 %
14	Andre iterasjon backend	Alle	08.apr	10.apr	3	Ikke begynt	0 %
15	Andre iterasjon frontend	Alle	11.apr	12.apr	2	Ikke begynt	0 %
16	Testing	Alle	13.apr	13.apr	0	Ikke begynt	0 %
17	Evaluering	Alle	14.apr	14.apr	0	Ikke begynt	0 %
18	Tredje iterasjon backend	Alle	15.apr	17.apr	3	Ikke begynt	0 %
19	Tredje iterasjon frontend	Alle	18.apr	19.apr	2	Ikke begynt	0 %
20	Testing	Alle	20.apr	20.apr	0	Ikke begynt	0 %
21	Evaluering	Alle	21.apr	21.apr	0	Ikke begynt	0 %
22	Siste justeringer	Alle	22.apr	26.apr	5	Ikke begynt	0 %
23	Overlevering	Alle	26.apr	29.apr	2	Ikke begynt	0 %
24	EXPO Plakat	Alle	14.mai	27.mai	10	Ikke begynt	0 %
25	Presentasjon	Alle	28.mai	02.jun	4	Ikke begynt	0 %

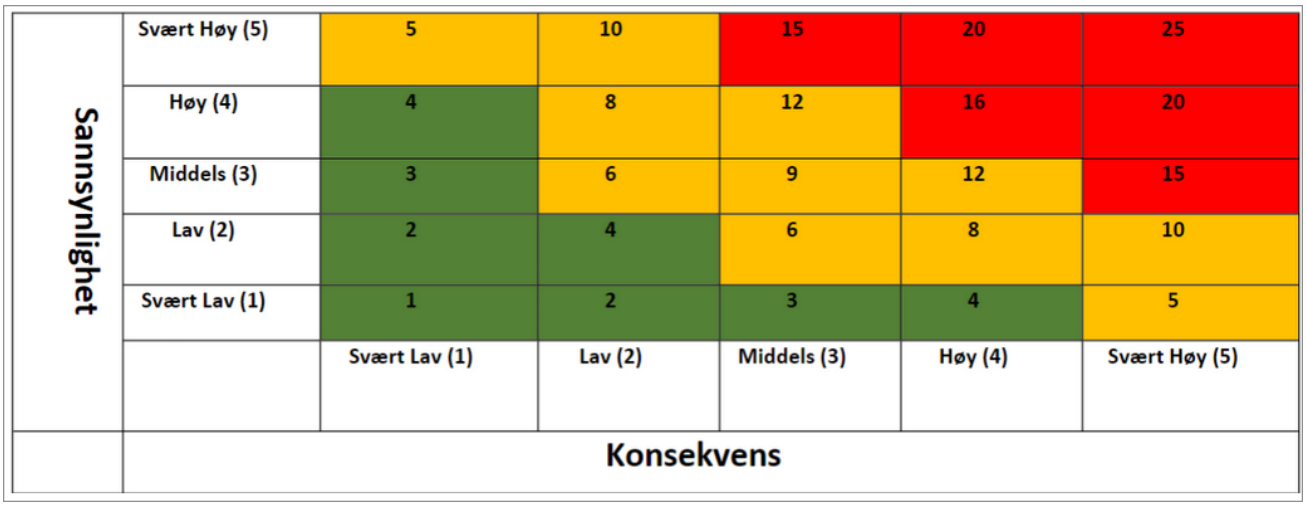
Figur 4.12.2-1: Prosjektets Gant-diagram

4.12.3 Risikovurdering

Gjennomføringen av prosjekter, særlig prosjekter på størrelse med en Bachelor oppgave, innebærer potensielle problemer. Slike problemer kan hemme og hindre fremgang i prosjektet og er en trussel for å opprettholde tidsfrister. Det er derfor viktig at alle prosjektmedlemmer er klar over risikobildet og tiltak som kan bli gjort for å unngå eller håndtere risikoene.

Risikovurderingen er presentert i form av en tabell utformet av en mal supplementært av HVL. Risikoene blir vurdert på en skala av risikoproduktet fra Svært 1 til 25. Risikoproduktet er produktet av sannsynligheten for at problemet oppstår multiplisert med konsekvensen av problemet (Sannsynlighet x Konsekvens = Risikoprodukt). Både sannsynlighet og konsekvens har en skala fra Svært Lav (1) til Svært Høy (5). Risikoprodukt med verdi 1-4 er markert med grønn, 5-12 er markert med oransje, og 15-25 er markert med rødt.u

Risikoproduktet er satt gjennom vurderinger gjennomført i starten av prosjektet. Først ble det satt en hendelse/risiko som kan oppstå, også referert til som problemet. Dette er altså farene gruppemedlemmene ser for seg kan oppstå underveis i prosessen. Deretter er det satt årsak for at problemet kan oppstå, som kan variere fra en til flere årsake. Utfra hendelse/risiko



Figur 4.12.3-1: En forklaring av risikovurderingens grunnlag

og årsak er det satt en vurdering fra 1 til 5 på sannsynligheten for at problemet oppstår, og konsekvensen dersom det skulle oppstå.

Beskrivelser av de ulike risikoene

	Hendelse/ Risiko	Årsak	Sannsynlighet	Konsekvens	Risikoprodukt	Tiltak
1	Produktet blir ikke ferdig i tide	Feilprioritering av arbeidsmengde.	Lav (2)	Svært Høy (5)	10	Sette tydelige milepæler Møte frister
2	Produktet oppfyller ikke krav.	Mangel på kommunikasjon.	Høy (4)	Middels (3)	12	God kontakt og dialog med arbeidsgiver
3	Mangle kjennskap til utviklingsverktøy	Bruk av ny teknologi/ ukjente rammeverk	Svært Høy (5)	Middels (3)	15	Utforske og bruke tid på å forstå og sette seg inn i ny teknologi.
4	Dårlig kommunikasjon mellom arbeidstaker og arbeidsgiver	En av sidene tar ikke kontakt. Misforståelser mellom partene.	Høy (4)	Middels (3)	12	Sørge for å ta jevnlig kontakt. Være tydelige og ærlige med arbeidsgiver.
5	Mangel på arbeidskraft	Sykdom/ vekkreist	Høy (4)	Lav (2)	8	

Figur 4.12.3-2: Åpning av et prosjekt på prosjektsiden

4.13 Evalueringsplan

Evalueringsplanen sin hensikt er å tilrettelegge målinger for gruppen og oppdragsgiver sin vurdering av prosjektet underveis og ved slutten av utviklingen. For evalueringen underveis skal komponenter testes etterhvert som de blir lagt til i webapplikasjonen for sikre at de oppfører seg slik som planlagt. Testingen vil bli utført med å skrive ut data og meldinger til console, håndtere feilmeldinger som oppstår ved implementering og sørge for at komponentene oppfyller sin hensikt. I tillegg skal prosjektgruppen ha møter med oppdragsgiver via Teams for å få tilbakemeldinger. Tilbakemeldingene skal sørge for at produktet utvikler seg i riktig retning og unngå misforståelser i henhold til logikk av funksjonelle egenskaper. Møtene skal også gi mulighet for å stille spørsmål mellom partene involvert.

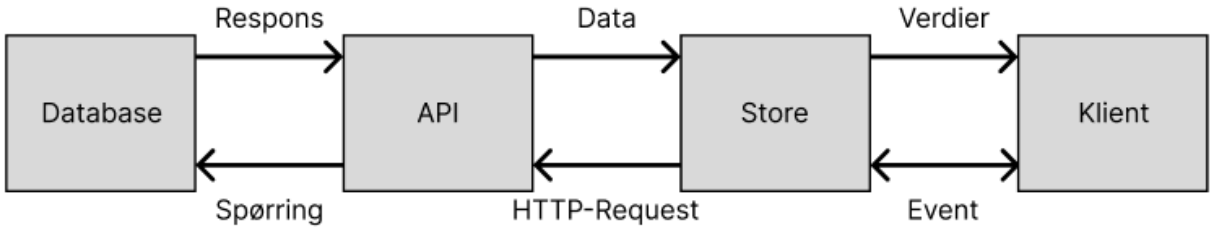
For evalueringen ved slutten av prosjektets utviklingsperiode skal det sammenlignes funksjonaliteter i webapplikasjonen med lister av funksjonelle og ikke-funksjonelle egenskaper hentet fra visjonsdokumentet. Sammenligningen vil gi en enklere evaluering av resultat og oversikt over produktet sin helhet.

I tillegg vil det bli utført brukertesting av oppdragsgiver fra TechnipFMC, Morten Wilhelmsen. Det er han som utviklet den nåværende Excel-løsningen, som skal gi tilbakemeldinger og stille til rådighet underveis, og er en av hovedpersonene som skal ta i bruk produktet. Ved brukertestingen skal prosjektgruppen få tilbakemeldinger på det visuelle og funksjonelle i webapplikasjonen, samt stille til rådighet dersom det oppstår forvirringer. Etter brukertesting skal det fylles ut et spørreskjema som skal hjelpe å vurdere brukervennlighet, oversiktlighet og optimalisering. Spørsmålene skal inneholde svaralternativer fra 1 (veldig uenig) , til 5 (veldig enig). I kapittel 6.1 vil resultatet vurderes og gi en indikasjon på hvor vellykket prosjektet var i henhold til forskningsspørsmålet. Ved å utføre testingen på Wilhelmsen som ønsket en forbedret løsning fra Excel, vil det bli enklere å få svar basert på sammenlikning av løsningene.

Kapittel 5

Detaljert løsning

I dette kapittelet skal løsningen som har blitt valgt utdypes ved en gjennomgang av de forskjellige strukturene i programvarearkitekturen. Avsnittet vil utdype komponenter, sider og grafisk brukergrensesnitt for klienten, oppkobling og tabellstruktur for databasen, samt gjennomgå hvordan serveren brukes og hvordan arkitekturen har anvendt den. Til slutt vil et mer helhetlig bilde på konteksten fremstilles for å demonstrere hvordan disse kobles sammen. Observer figur 5.0-1 for å få et generelt overblikk over den grunnleggende strukturen i arkitekturen.



Figur 5.0-1: Dataflyten i applikasjonen

5.1 Komponenter

I henhold til punkt 4.5 er det blitt tatt utgangspunkt i et atomisert webdesign. Dette betyr da at nettsiden er delt opp i komponenter med forskjellige størrelser; *atomer, molekyler, organismer* og *sider*.

5.1.1 Atomer

Atomer er navnet på de minste komponentene i arkitekturen, som f. eks. progressBar, datoDisplay og statuslys. Disse komponentene er laget for å ta imot- og fremstille data på en konkret måte. Komponentene skal være mest mulig isolert med lite innflytelse på resten av applikasjonen. Et atom vil aldri kalle på andre komponenter, men blir heller kalt på av andre.

Et eksempel på en atomkomponent kan bli sett i listing 5.1.

```
<script setup>
  // Henter ut progressnumber fra kallet av komponenten
  const props = defineProps({
    progressNumber: Number
  });

</script>



<template>
    <!--Dette er bakgrunnsbaren under progress-->
  <div class="bar">
    <!--Dette er den dynamiske baren for progress-->
    <div class="bar progress" :style="{ width: props.progressNumber + '%' }">
    </div>
  </div>
</template>



<style scoped>
.bar {
  width: 100%;
  height: 20px;
  background-color: white;
  border:1px solid black;
  border-radius: 13px; /*legg denne til for runde kanter*/
```

```
}

.bar.progress {
  border: 0px;
  width: var(--progress);
  background-color: green;
}
</style>
```

kodeutdrag 5.1: ProgressBar.vue

I listing 5.1 ble progresjonsbaren først definert med en scriptseksjon som henter ut den data som skal mates til komponenten, og denne dataene skal deretter fremstilles visuelt ved hjelp av CSS i stilseksjonen. Emnetaggen scopedbrukes her for å spesifisere at stilen kun skal benyttes internt i denne komponenten, og dermed ikke påvirke andre komponenter med et uhell.

5.1.2 Molekyler

Akkurat som molekyler i kjemien er grupperinger av atomer, er molekylkomponenter i utviklingsmetodikk de komponenter som tar ansvar for å bruke og gruppere atomkomponenter. I dette prosjektet har molekyler hovedsakelig vært oppføringer av tasks, gates og prosjekter. Disse komponentene er også blitt implementert atomisert i så stor grad som det har vært praktisk mulig å gjøre, men grunnet bruk av stores, se punkt 4.8.1, har det til tider har vært upraktisk å følge atomiseringsprinsippet slavisk. Molekylene ble da i utgangspunktet implementert som komplett atomiserte komponenter, men for å kunne samhandle med databasen og verdier i storen på korrekt måte ble det da gjort kall til store der verdien skulle behandles for enklere organisering. For å fremstille en slik komponent i rapporten har et forenklet utklipp av en slik komponent blitt vist i listing 5.2.

```
  <template>
    <div class="gate-card" @click.capture="openCollapsable">
      <div class="list" >
        <div class="gateNR">
          <span>{{ props.gateNR }}</span>
        </div>
```

```
        <div class="progress">
          <ProgressBar :progressNumber=gateProgress />
        </div>
        <div class="plannedDate">
          <DateEntry :dateString = plannedDate.value />
        </div>
        <div class="daysToEnd">
          <span>{{daysToEnd}}</span>
        </div>
        <div class="completion">
          <DateEntry :dateString = props.completionDate />
        </div>
      </div>
      <CollapseTransition class ="collapsable">
        <div v-show="isOpen">
          <hr class="solid">
          <GateContent :gateID = props.gateID />
        </div>
      </CollapseTransition>
    </div>
<script setup>
  // Henter gate storen
  import {useGatesStore} from '@/stores/gates';
  // Henter de ulike variablene fra gaten
  const gateStore = useGatesStore();
  const props = defineProps({
    gateID: {
      type: String,
      required: true
    },
    gateNR: {
      type: Number,
      required: true
    },
```

```
41      projectId: {
42        type: String,
43        required: true
44      },
45      completionDate: {
46        type: String,
47        required: true
48      },
49      daysToEnd: {
50        type: Number,
51        required: false
52      }
53    });
54  // Henter oppdaterte variabler
55    const plannedDate =
56      computed(() => gateStore.calculateDate(props.projectId, props.gateNR));
57    const daysToEnd =
58      computed(() => gateStore.calculateDaysToEnd(plannedDate.value));
59    const gateProgress = gateStore.getGateProgress(props.gateID);
60    const isOpen = ref(false);
61    function openCollapsable(event) {
62      if (!event.target.closest('.collapsable')
63          && !event.target.closest('.title')
64          && !event.target.closest('.delete')) {
65              isOpen.value = !isOpen.value;
66      }
67    }
68  </script>
```

kodeutdrag 5.2: GateEntry.vue

I listing 5.2 blir flere verdier matet inn via komponentkallet i foreldrekomponenten gjennom funksjonen defineProps. Dette blir gjort i tråd med atomisert design, og har vært en god praksis gjennom prosjektet. For å oppdatere disse variablene på en ryddig måte har

gateStoren blitt hentet. Denne brukes for å lytte etter endringer i variabelen ved computed(() =>) og oppdatere variablene. Atomkomponenter er tatt i bruk i template-seksjonen og blir matet verdier gjennom defineProps og foreldrekomponenten, eller reaktive variabler (se 4.6). Vi ser f.eks at progressbar blir matet en inputverdi i koden <ProgressBar :progressNumber=gateProgress />, som så blir tatt videre i listingen 5.2. Dette eksempelet var for GateEntry men en tilsvarede komponent er laget for task- og prosjektoppføringer.

5.1.3 Organismer

Etter molekyler er neste komponentklassifisering i atomisert design organismer. Organismer er mer komplekse komponenter som består av både molekyler og atomer. I dette programmet har de stort sett vært formatert som lister av molekylkomponenter. Som eksempel på dette er GateList sin template-seksjon vedlagt i listing 5.3.

```
<template>
  <div class="descWrapper">
    <GateDesc/>
  </div>
  <hr class="solid"/>
  <draggable class="gatelist" v-model="gates"
  @end="onEndDrag" group="gates" item-key="ID"
  handle=".handle" animation="300">
  <template #item="{ element, index }">
    <div :key="element.ID">
      <GateEntry :gateID="element.ID"
      :gateNR="element.gateNR"
      :title="element.title"
      :projectId="props.projectId"
      :completionDate="element.completionDate"
      responsiblePerson="element.responsiblePerson || ''" />
      <div
        v-if="index < gates.length - 1"
        @click="addGateBetween(element.gateNR)"
        class="gate-divider">
```

```
21        </div>
22      </div>
23    </template>
24  </draggable>
25    <div class="emptylist"v-if="gates.length === 0">
26      No gates found for this project
27        <div class="gate-empty" @click="addGateBetween(1)">
28        </div>
29    </div>
30    <Modal @close="toggleModal" :modalActive="modalActive">
31      <h1>New Gate</h1>
32      <form @submit.prevent="submitForm">
33        <label>Gate title: </label>
34        <input type="text" id="title" v-model="formData.title" required><br>
35        <button type="submit" class="addButton">Create Gate</button>
36      </form>
37      <button class="closeButton" @click="toggleModal">Cancel</button>
38    </Modal>
39  </template>
```

kodeutdrag 5.3: GateList.vue

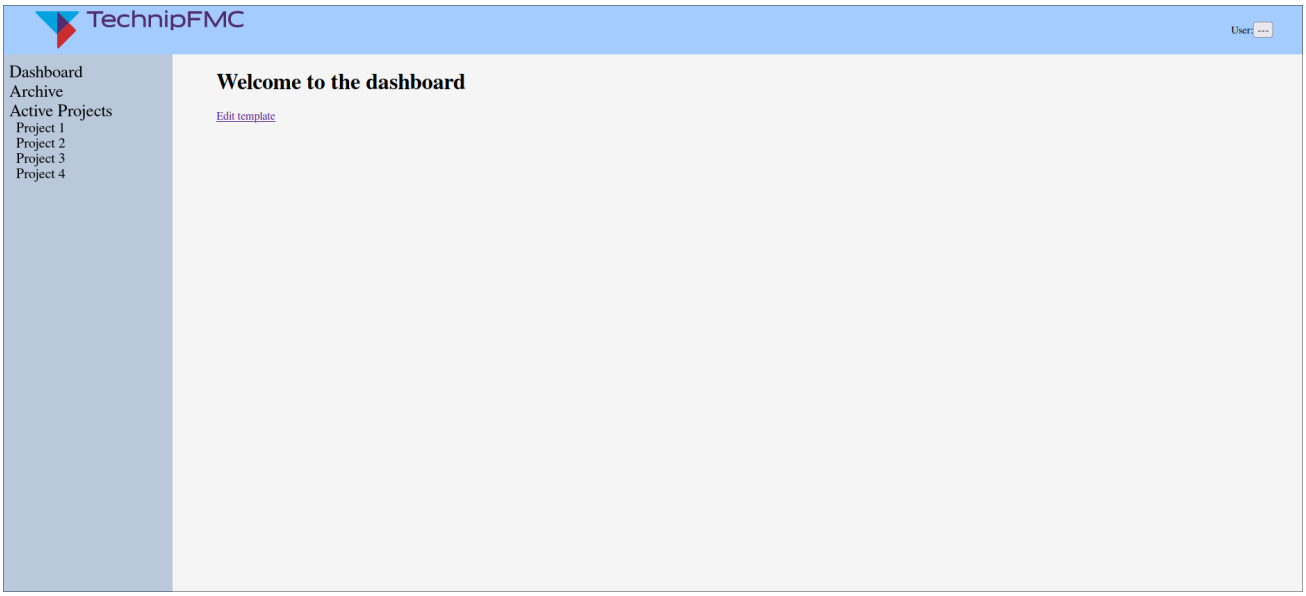
I listing 5.3 hentes en liste over alle gatene i gateStoren og fremstiller de i template ved en for-løkke <draggable class=”gatelist” v−model=”gates” som looper gjennom gatelisten. Deretter opprettes et GateEntry for hver gate i gatelisten med initielle verdier og tilhørende id. Disse brukes da som utgangspunkt i listing 5.2.

5.2 Sider

Denne seksjonen tar for seg hvilke sider som er opprettet i prosjektet og hvordan de er strukturert.

5.2.1 Dashbord

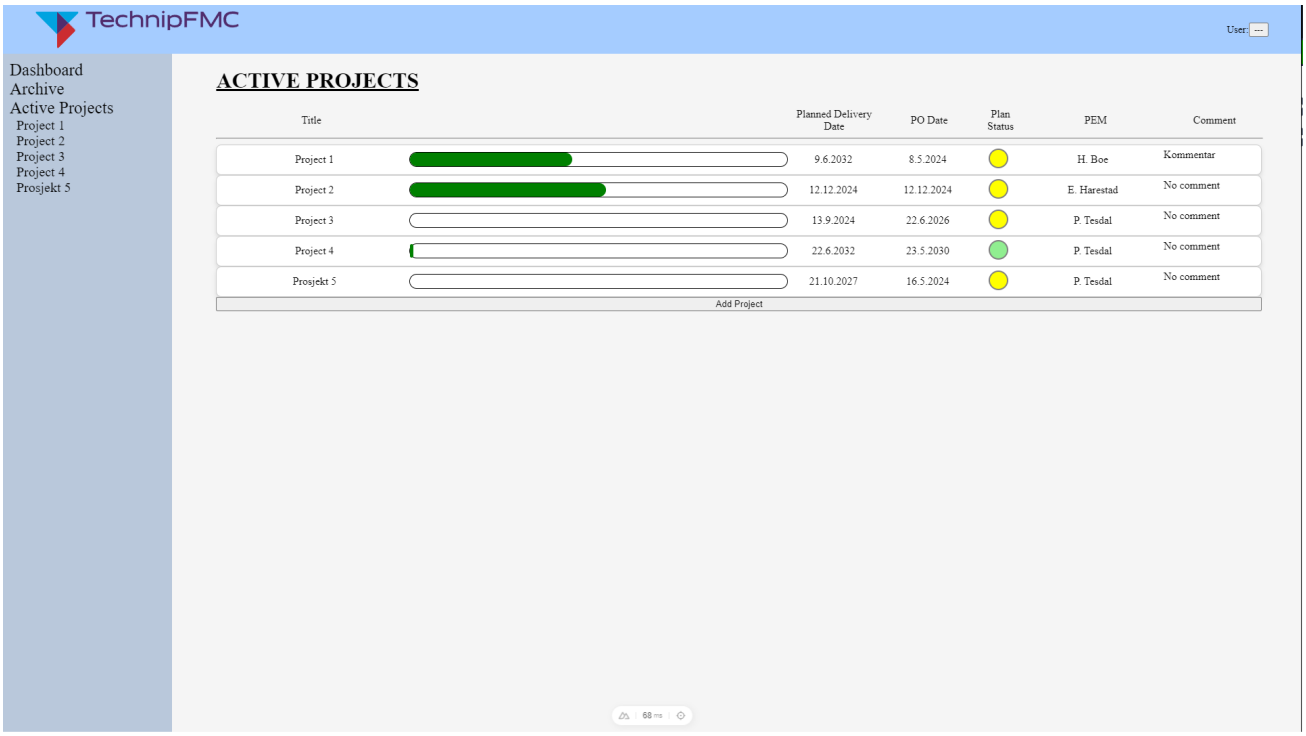
Under utviklingen av designet ble det planlagt å utvikle en form for dashbord (se punkt 4.4). Spesifikt hva som skulle fremstilles her ble ikke fastsatt, men muligheten for å fremstille generell statistikk virket praktisk, og var planlagt å utvikle etter hvert. Prosjektet som det står nå har derimot underutilisert denne siden da den kun inneholder muligheten til å gå siden for visning og redigering av prosjektmalen. Det er likevel ikke planen per nå å fjerne siden ettersom det fortsatt kan settes opp enkle visninger av informasjonen i databasen, og evt. sitte opp linker til de aktuelle prosjektene. For den visuelle representasjonen av sidens nåværende tilstand se figur 5.2.1-1.



Figur 5.2.1-1: Et utsnitt av Dashbord siden av løsningen

5.2.2 Prosjektliste

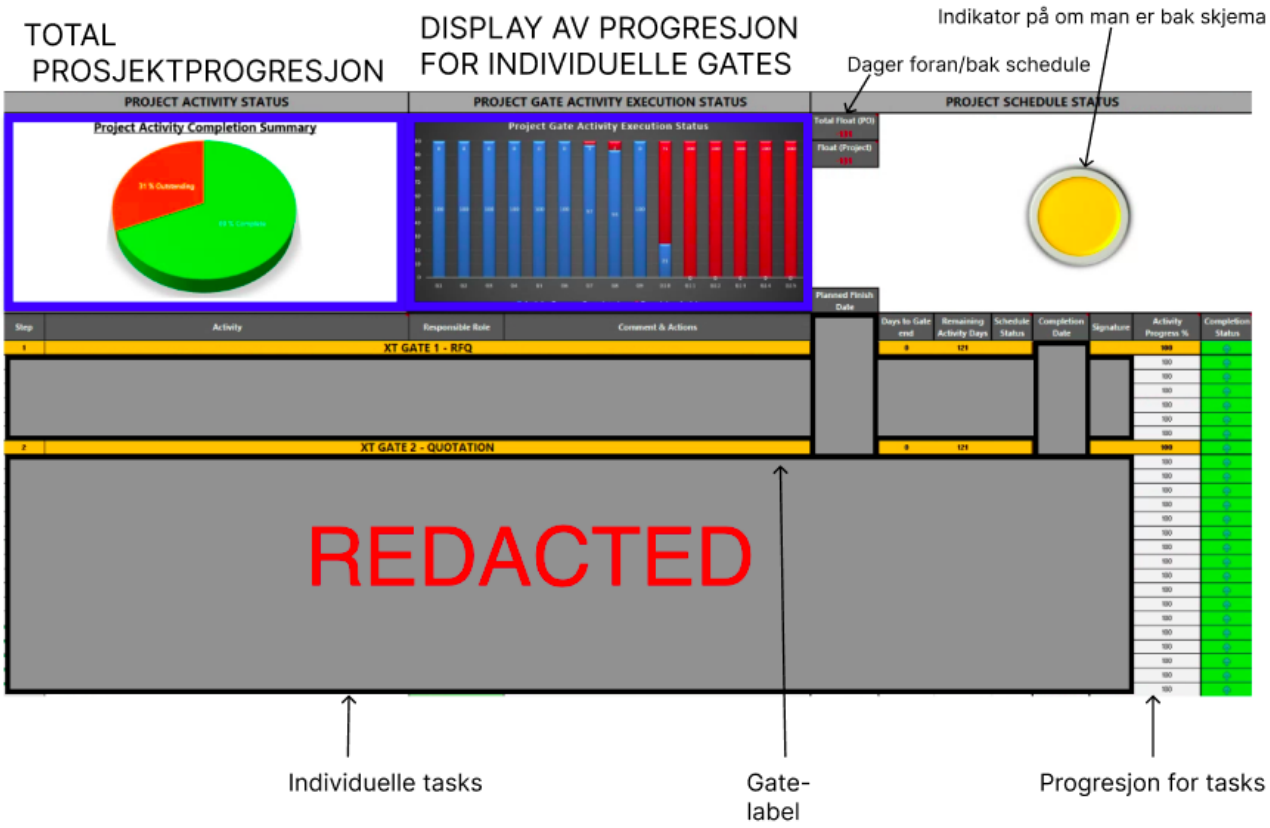
Aktive prosjekter er siden hvor det enkelt skal vises en oversikt over prosjektene som er i arbeid, og deres status. Her vil PO-dato, SF-dato, progresjon, tidsmessig status, kommentar og PEM fremvises for ethvert prosjekt. Progresjon vises i form av en bar slik som i det originale dokumentet, og kommentar skal tillate PEM å legge ved en forklaring på hvorfor et prosjekt evt. er bak tidskjema, og om dette er under kontroll. Resten av dataene kan enkelt fremvises som de er. Se figur 5.2.2-1 for nåværende løsning.



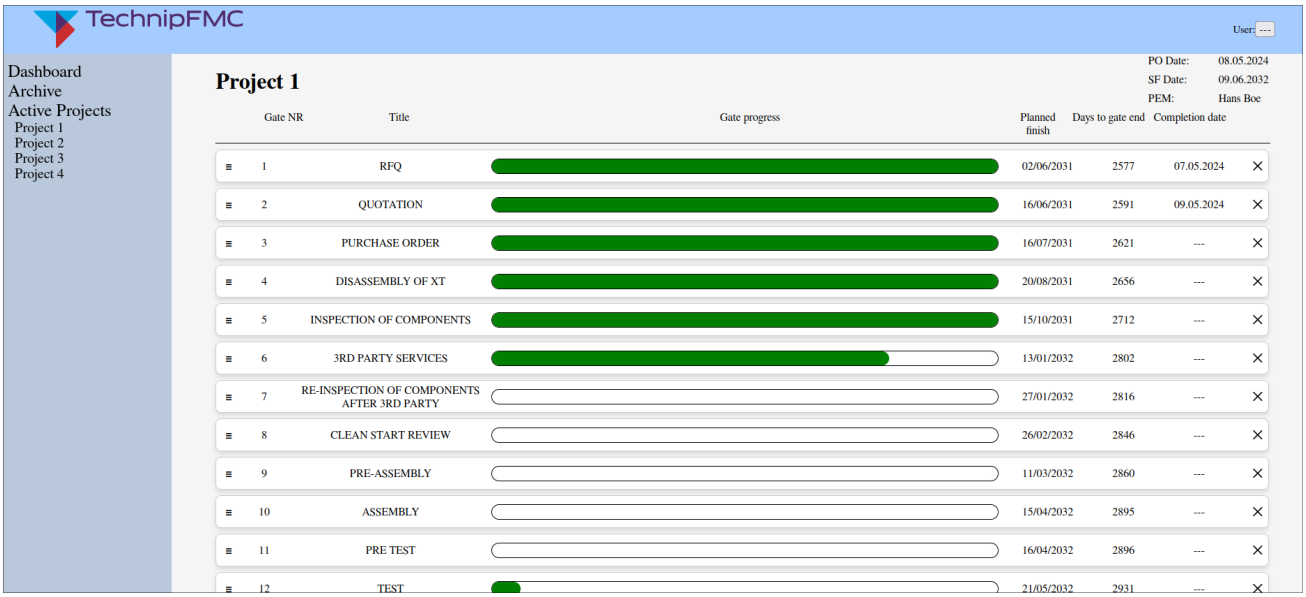
Figur 5.2.2-1: Prosjektoversikten/fremsiden

5.2.3 Prosjektside

Selve prosjektsiden er endret veldig fra det originale Excel-dokumentet. Observer figur 5.2.3-1 og 5.2.3-2. For å redusere mengden plass brukt på informasjonen synlig i øvre halvdel av dokumentet er det vurdert at kakediagrammet til venstre, og statuslyset til høyre skal fjernes. Begge deler informasjon kan finnes på den helhetlige prosjektoversikten, og tar for mye plass i forhold til verdien av den informasjonen som vises. Selv om informasjonen i det konkrete arket hentet fra Excel-dokumentet ikke kan vises, kan det opplyses at informasjonen stort sett korresponderer til det prosjektgruppen har døpt tasks. De to gule labelsene er de to første gatene. Ettersom Excel-dokumentet er ekstremt langt å bla gjennom pga. alle taskene, er det valgt å konvertere gatene til dropdown-knapper, med de tilhørende taskene i den tilsvarende dropdown-menyen. Gatenes progresjonsbarer i toppen er blitt flyttet fra toppen av skjermen der de står for seg selv, til å være direkte på de korresponderende gate-knappene.



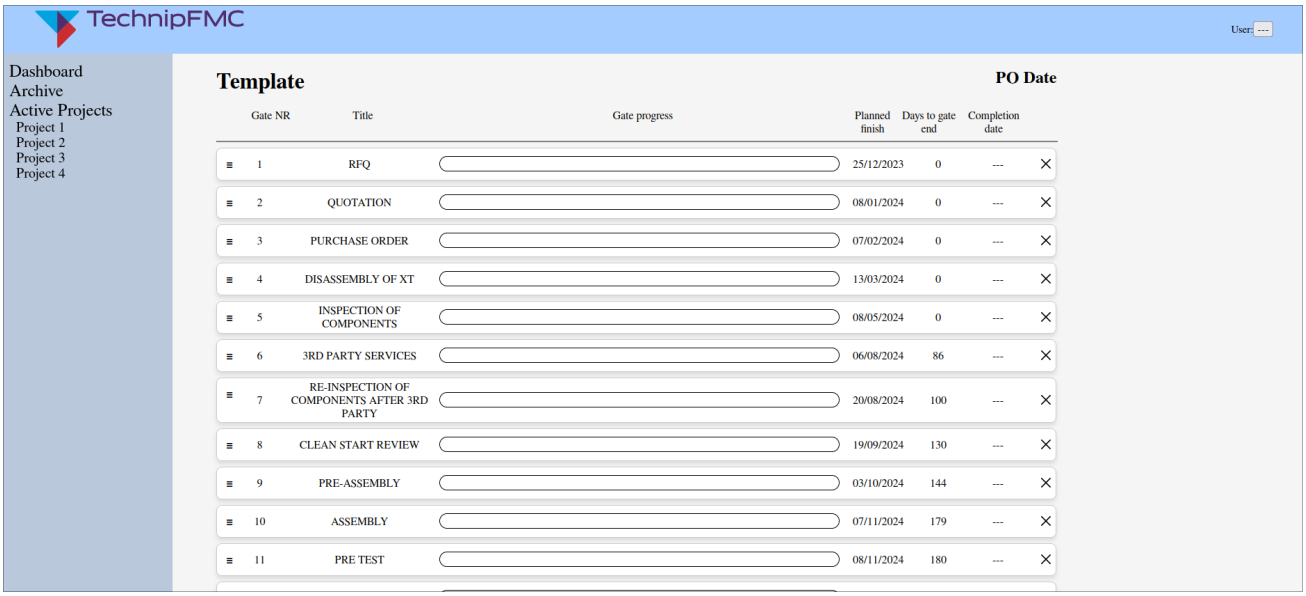
Figur 5.2.3-1: Prosjektside i Excel-dokumentet med beskrivelse av informasjon



Figur 5.2.3-2: Prosjektfremstilling på prosjektsiden

5.2.4 Template

Templatesiden er en prosjektside laget for å kunne redigere et spesielt prosjekt kalt template. Dette prosjektet er lagt til i databasen og brukes for å ha en fellesstruktur på prosjektene som opprettes samtidig som at denne strukturen kan endres. Template siden er stort sett en kopi av prosjektsiden 5.2.3 med enkelte alterasjoner for å passe på at den f.eks ikke slettes. For prosjektets del har denne siden blitt implementert raskt, og mangler noen få funksjoner som burde bli implementert. Dette gir f.eks muligheten til å ha en backup versjon i tilfelle noe skulle gå galt, en bedre gjennomgang av sikkerhet og passe på at personen som redigerer har autorisasjon til denne funksjonen. Disse funksjonene mangler pga. mangel på tid og utfordringer i implementeringen i oppdragsgivers single sign-on.



Figur 5.2.4-1: Utsnitt av templatesiden, som er funksjonelt en kopi av prosjektsiden

5.2.5 Archive

Arkivet er en side som er lagt til i løsningen men ikke implementert. Det er derimot lagt opp for at det skal enkelt kunne implementeres, og at det ikke skal ta så veldig mye tid. Hvert prosjekt har fått en archive bool som skal si om prosjektet er arkivert eller ikke. For å implementere archive så er det kun å kopiere prosjekt listen, hente ut kun prosjekter som er arkivert, og så fremstille de på siden. Det vil være noe jobb å separere arkiverte prosjekter og aktive prosjekter i store funksjonene, men det skal være overkommelig og ikke ta alt for mye tid. Det var ønskelig at dette skulle implementeres i dette prosjektet men tiden strakk desverre ikke til.

5.3 Grafisk brukergrensesnitt

Utseendet til nettsiden er stort sett tilsvarende det som ble tenkt i prototypen, se punkt 4.2.1, da oppdragsgiver var fornøyd med denne. Nevneverdige endringer er bare fargekoding og inklusjonen av et kommentarfelt på prosjekt-/taskvisninger. Det er også litt forskjell i hvordan Gatene er fremstilt, da de nå vises i kort i stedet for i klassiske rader.

5.3.1 Layout-design

Nettsidens layout består av en navigasjonsbar på siden, og en mindre header-bar langs toppen. Header-baren er tiltenkt en TechnipFMC-logo i venstre hjørne, og brukernavn/-info i høyre. Headerens farge er valgt til en lys blå i stil med TechnipFMC sin logo ettersom denne fargen er veldig lett på øynene. Navbaren på siden er tiltenkt generell navigasjon mellom hovedsidene, og enkel tilgang til de aktive prosjektene.

5.4 Databaseløsning

5.4.1 Databasevalg

På grunnlag av drøftingen presentert i punkt 4.10 er det blitt lagt frem anbefaling til oppdragsgiver om at en grafdatabase best vil passe strukturen på data som skal lagres i databasen. Oppdragsgiver har derimot lagt frem spørsmål om å legge opp til en relasjonsdatabase, ettersom de vil inkorporere dataene i firmaets allerede eksisterende Microsoft SQL Server, noe som da er blitt gjort.

5.4.2 Database oppkobling

Tilkobling fra webapplikasjonens server til den tilhørende SQL-serveren er gjort kun ved å importere Microsofts offisielle pakke MSSQL. Dette er for å minke applikasjonens avhengigheter, i tillegg til at den er mer sannsynlig til å videreutvikles, noe som fører til mindre feilkilder i fremtiden dersom applikasjonen videreutvikles. Tilkoblingen til den tilhørende SQL-server gjøres gjennom to enkle filer. Den ene er en databasekonfigurasjon fil, som setter sammen tilkoblingsinformasjonen fra .env filen og legger de til i en JavaScript Object Notation (JSON) struktur som eksporteres, dette er da koden 5.4.

```
1 const dbConfig = {
2   server: process.env.DB_SERVER,
3   user: process.env.DB_USER,
4   password: process.env.DB_PASS,
5   database: process.env.DB_NAME,
6   port: process.env.DB_PORT,
7   authentication: {
8     type: 'default'
```

```
  },
  options: {
    encrypt: true
  }
};

export default dbConfig;
}
```

kodeutdrag 5.4: dbConfig

Spørringer blir gjort i Microsoft SQL stil og utføres gjennom en hjelpemetode kalt connectAndQuery i dbConnect filen. Denne metoden er ansvarlig for å opprette en tilkoblingspool til SQL-databasen ved bruk av MSSQL-modulen i Node.js. Ved å benytte seg av en tilkoblingspool, kan flere spørringer utføres samtidig, som resulterer i bedre ytelse og ressursutnyttelse. (Custer, 2021). Når en spørring blir sendt gjennom connectAndQuery, blir den behandlet av SQL-serveren, og resultatene returneres tilbake til kallet for videre behandling eller visning i applikasjonen. Koden er ført i listing 5.5.

```
import sql from 'mssql';
import dbConfig from './dbConfig.js';

// Oppretter en SQL-tilkoblingspool, gjennom dbConfig.
const pool = new sql.ConnectionPool(dbConfig);
// Oppretter en forbindelse til SQL-tilkoblingspoolen.
const poolConnect = pool.connect();

// Feilhaandtering
pool.on('error', err => {
    console.error('SQL Database Connection Pool Error:', err);
});

// Sikrer at tilkoblingspoolen er opprettet.
export async function connectAndQuery(queryString, params = []) {
  await poolConnect;
```

```
17
18    try {
19      const request = pool.request();
20
21      // Legger til parametere til requesten om det er noen.
22      params.forEach(param => {
23        request.input(param.name, param.type, param.value);
24      });
25
26      const resultSet = await request.query(queryString);
27      return resultSet.recordset;
28    } catch (err) {
29      console.error('Failed to execute query:', err);
30      throw err;
31    }
32  }
```

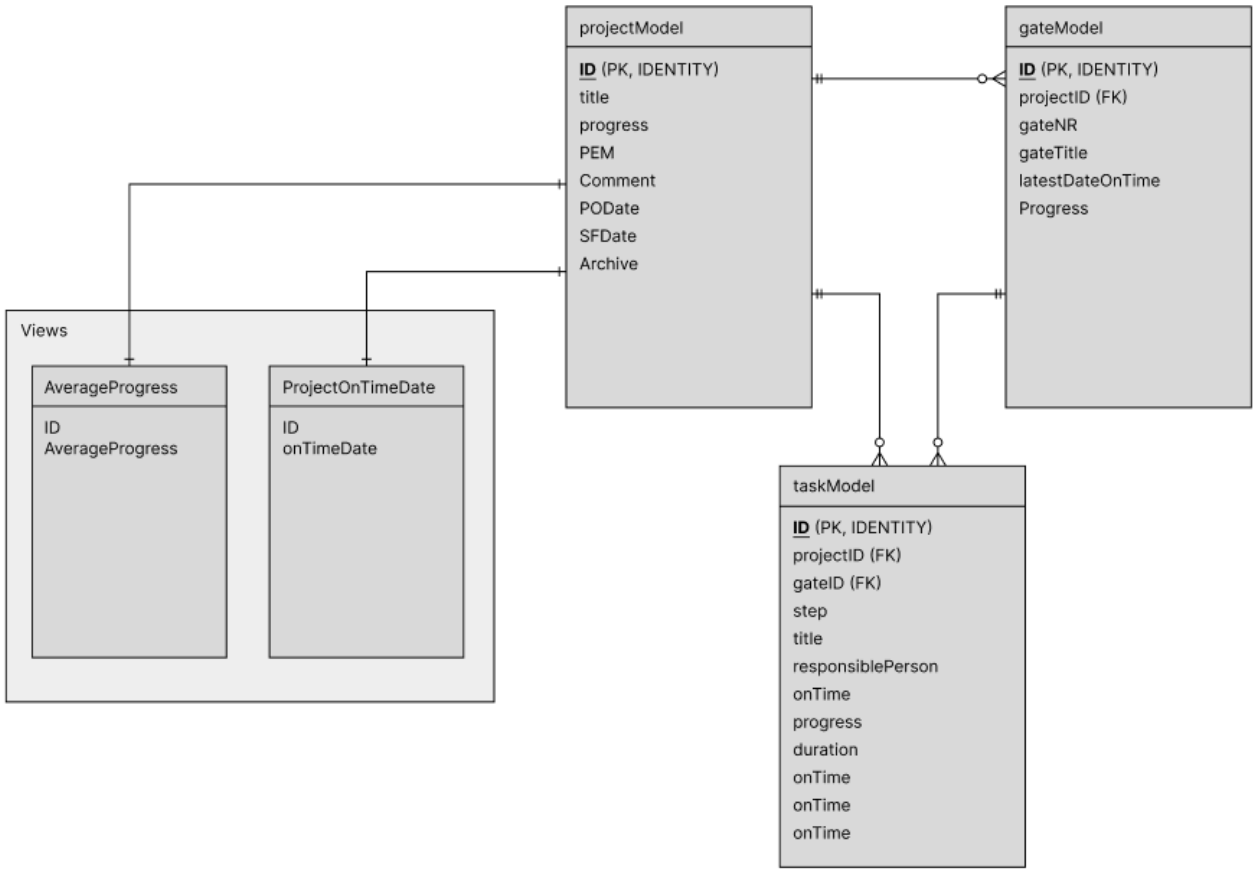
kodeutdrag 5.5: dbConnect

Oppdragsgiver har grunnet sikkerhetshensyn ikke mulighet til å gi tilgang til deres SQL-server, men det er blitt opprettet en utviklingsserver via Microsoft Azure som har fungert som surrogat underveis i arbeidet. Dette fører til en del mer arbeid under overleveringen av produktet da databasen og dens struktur må opprettes på deres egen database, samt forandre tilkoblingsinformasjonen og muligens autentisering-type. Dette har oppdragsgiver gått med på, da det var vanskelig å opprette god kontakt med IT-personalet i Oslo som styrer databasen.

5.5 Databasestruktur

Prosjektdataene er strukturert i tre tabeller som illustrert i figur 5.5-1.

Datastrukturen slik den er i templaten i det originale Excel-dokumentet er lagt til som et prosjekt i databasen. Dette er gjort ved at dataene er lagret i to tabeller, en som inneholder informasjonen om gates, og en som inneholder tabeller tilsvarende gates som igjen inneholder informasjon om tasks. Når et prosjekt opprettes så kopieres dette template



Figur 5.5-1: Kråkefotdiagram til databasen

prosjektet og tilhørende gates og tasks med ny ID. Dette betyr i praksis at når at et prosjekt opprettes vil det følge datastrukturen til templateprosjektet, som så kan endres i etterkant. Dette er gjort i henhold til den originale oppgavebeskrivelsen gitt av oppdragsgiver. Template prosjektet i databasen kan også redigeres gjennom template siden som ble utdypet i punkt 5.2.4.

5.5.1 Id

For Id på de forskjellige radene i tabellene har flere muligheter blitt testet ut, b.l.a. en egen formasjon på ID, og det å opprette en UUID i frontend. Det ble derimot raskt avgjort at det ryddigste var å opprette id i databasen gjennom en *IDENTITY* tag. Denne taggen vil passe på at hver rad ID rad i databasen er unik og genereres automatisk når en spørring legger til en rad.

5.5.2 Views

Views i databasen blir brukt for å regne ut interessante data som må gjøres på en større skala, disse er som regel data som hadde vært litt krevende å få gjennomført innad i applikasjonen og er derfor valgt å gjøres på databasesiden i stedet. Et eksempel på en slik view er avgProjectprogress, som går ut på å få en helhetlig ide av prosjektets fremgang. Dette regnes ut hvergang noen av disse tabellene oppdateres i databasen, men må hentes igjen i klienten for at det skal oppdateres, dette gjøres ikke med mindre brukeren henter nettsiden på nytt, men med viewets skala så er det vurdert at disse endringene ikke vil ha nok utslag til at det trenger å hente denne informasjonen på nytt.

5.5.3 Procedures

I databasen har det blitt tatt ibruk noen prosedyrer for å gjøre oppretting av rader lettere. Et godt eksempel på dette er opprettelsen av et prosjekt som kopierer et template objekt for så å fylle det inn med relevante detaljer. En spørring vil da mer se ut som i listing 5.6.

```
1       EXEC DuplicateProject
2       @OldProjectID = 1,
3       @NewProjectTitle = '${title}',
4       @PEMName = '${PEM}',
5       @PODate = '${PODate}',
```

```
6       @SFDate = '${plannedDate}';
```

kodeutdrag 5.6: Addproject query

Prosedyren vil kopiere template prosjektet med tilhørende tasks og gates. Etter det så oppdateres Id så Foreign key og Primary Key viser til de nye prosjektet. Det som da faktisk kjøres på databasen vil være det som blir definert i listing 5.7.

```
1  DECLARE @NewProjectID INT;
2
3  -- Setter inn the nye prosjektet
4  INSERT INTO dbo.projectModel
5  (title, progress, onTime, PEM, COMMENT, POdate, SFdate, archive)
6  VALUES ('Project 1', 0, 0, 'Petter Tesdal', '', '2024-12-12', '2024-12-12', 0);
7
8  -- Henter ny prosjekt ID
9  SELECT @NewProjectID = SCOPE_IDENTITY();
10
11 -- Seter gates for nytt prosjekt
12 INSERT INTO dbo.gateModel (prosjektID, gateNR, gateTitle)
13 SELECT @NewProjectID, gateNR, gateTitle
14 FROM dbo.gateModel
15 WHERE prosjektID = 1;
16
17 -- Endrer Id til prosjektet i gate til ny ID
18 DECLARE @GateMap TABLE (OldGateID INT, NewGateID INT);
19
20 INSERT INTO @GateMap (OldGateID, NewGateID)
21 SELECT old.ID, new.ID
22 FROM dbo.gateModel old
23 JOIN dbo.gateModel new ON old.gateNR = new.gateNR AND old.prosjektID = 1
24 AND new.prosjektID = @NewProjectID;
25
26 -- Setter inn tasks til nye gates
27 INSERT INTO dbo.taskModel
```

```
(prosjektID, gateID, title, responsiblePerson,
duration, step, onTime, progress)
SELECT @NewProjectID, gm.NewGateID, tm.title, tm.responsiblePerson,
tm.duration, tm.step, tm.onTime, tm.progress
FROM dbo.taskModel tm
JOIN @GateMap gm ON tm.gateID = gm.OldGateID
WHERE tm.prosjektID = 1;
```

kodeutdrag 5.7: Duplicateproject procedure

5.6 Stores

For state-management er kodebiblioteket Pinia tatt i bruk se punkt 4.8.1. Med Pinia er det blitt laget fire forskjellige stores. En for prosjekter, en for gates, en for tasker og en for authentifikasjonsvariabler. Disse storene blir aktivt brukt i hele applikasjonen, som gjør de til et slags knytte punkt for utregninger og henting av tall.

5.6.1 Komponent bruk

Gjennom applikasjonen har komponentene hentet og behandlet reaktive komponenter i storen. For å få de utregningene og reaktiviteten som er ønsket så er dette et ganske omfattende system som går mye frem og tilbake. Det enkleste og mest sentrale eksempelet blir anvendingen av variablen duration. Et lite utklipp av GateTask hvor taskDuration blir definert er vist i listing 5.8. Koden er simpel og definerer en taskDuration fra currentTask, som er hentet fra taskStore hvis den eksisterer. Ellers vil updateDuration kjøres hvis variabelen blir endret i templaten som oppdaterer taskduration i taskStore.

```
  const taskStore = useTasksStore();
  const currentTask = tasksStore.tasks.find(t => t.ID === props.taskID);

  const taskDuration = ref(currentTask ? currentTask.duration : 0);
  function updateDuration() {
    taskStore.updateTaskDuration(props.taskID, parseInt(taskDuration.value))
    editMode.value = false;
  }
```

kodeutdrag 5.8: Et utsnitt av gateTask

Denne listingen definerte noe veldig enkelt, det som kanskje er interresant er hvordan dette også påvirker progressbaren automatisk gjennom metoden definert i listing. Det som får denne metoden til å kjøres er computed. metoden getGateProgress er kalt i gateEntry og hentes på nytt igjen pga. computed som brukes i metoden. I praksis vil dette bety at forandringen i taskDuration sprer seg gjennom applikasjon og har en påvirkning på metoder som f.eks getGateProgress se 5.9.

```
1    function getGateProgress(gateID) {
2        const taskStore = useTasksStore();
3        return computed(() => {
4            const tasks = taskStore.getGateTasks(gateID)
5            if (!tasks.length) return 0;
6            const totalWeightedProgress = tasks.reduce((total, task) => {
7                return total + (task.progress / 100) * task.duration
8            }, 0);
9            const totalDuration = tasks.reduce((total, task) => {
10               return total + task.duration
11           }, 0);
12           return totalDuration > 0 ?
13           (totalWeightedProgress / totalDuration) * 100 : 0;
14       });
15   }
```

kodeutdrag 5.9: gateStore metodet getGateProgress

5.6.2 Bruk av API

Stores er også tatt aktivt i bruk for å knytte frontend og backend. Når en variabel oppdateres i en store, vil det ofte gjøres gjennom en metode som samtidig plasserer denne koden på databasen. Dette gjøres gjennom $fetch for å sende informasjonen til server gjennom serverroute basert på id, serveraddresse og Http metode. En slik sentral metode fra projectStore vises i listing 5.10 som ogsp knyttes til tidligere fremstilit listing 5.6 i punkt 5.5.3.

```
async function addProject(
ID, title, progress, plannedDate, PODate, status, PEM, comment) {
    const requestBody = {
        ID: ID,
        title: title,
        progress: progress,
        plannedDate: plannedDate,
        PODate: PODate,
        status: status,
        PEM: PEM,
        comment: comment
    };
    try {
        const response = await $fetch('/projects', {
            method: 'POST',
            headers: {
                'Content-type': 'application/json'
            },
            body: JSON.stringify(requestBody)
        });

        this.fetchProjects();

        index.value++;
    } catch (error) {
        return createError({
            statusCode: 500,
            statusMessage: 'Internal Server Error',
            data: 'Failed to create project'
        });
    }
}
```

kodeutdrag 5.10: projectStore addProject metoden

5.7 API

For API har det blitt brukt Nitro som bestemt i punkt 4.7.1, og har blitt brukt for å knytte bakenden til databasen. Hvert endepunkt i API'en er definert gjennom filstrukturen hvor navnet bestemmer om rute parametere skal bli gitt og hvilken metode som skal brukes. Alle vendepunkt tar ibrukt metoden fremvist i punkt 5.4.2 og bruker den ved å først definere en query som så gis videre til metoden *connectAndQuery*().

5.7.1 Routing og id

Routing og Id er sterkt knyttet opp til filstrukturen, dermed ses det nødvendig å vise det en figur av filstrukturen og forklaring av hvordan den brukes. Når det kalles på /projects med metoden POST så videreføres de til filen projects.post.ts. Hvis det heller er endepunktet /projects/5 med metoden put så kalles filen projects/[id].put.ts som kjøres. Filstrukturen kan du se i figur 5.7.1-1.

```
/
└── server
    ├── api
    ├── router
    │   ├── projects
    │   │   ├── [id].put.ts
    │   │   ├── [id].get.ts
    │   │   └── [id].post.ts
    │   ├── gates
    │   │   ├── [id].put.ts
    │   │   ├── [id].get.ts
    │   │   └── [id].post.ts
    │   └── tasks
    │       ├── [id].put.ts
    │       ├── [id].get.ts
    │       └── [id].post.ts
    ├── projects.post.ts
    └── projects.get.ts
```

Figur 5.7.1-1: En liten del av strukturen på API filene for å visualisere bere hvordan den fungerer

For å videreføre eksempelenes struktur tas det utgansgspunkt i project/post.ts metoden som tas ibruk for å opprette et posjekt og kalles på i listing 5.10. Listingen 5.11 henter verdiene fra en

body som er sendt med requesten til endepunktet. Deretter settes de inn i connectAndQuery metoden som sender spørringen til databasen.

```
export default defineEventHandler(async event => {
  let projects;
  try {
    const body = await readBody(event);
    const { title, progress, plannedDate, PODate, status, PEM, comment }=body;

    //Oppretter prosjektet
    projects = await connectAndQuery(`
      EXEC DuplicateProject
      @OldProjectID = 1,
      @NewProjectTitle = '${title}',
      @PEMName = '${PEM}',
      @PODate = '${PODate}',
      @SFDate = '${plannedDate}';`);

    return { updated: true };
  } catch (error) {
    console.log("error: " + error)
    return createError({
      statusCode: 500,
      statusMessage: 'Internal Server Error',
      data: 'Failed to retrieve records',
    });
  }
});
```

kodeutdrag 5.11: API, projects.post.ts

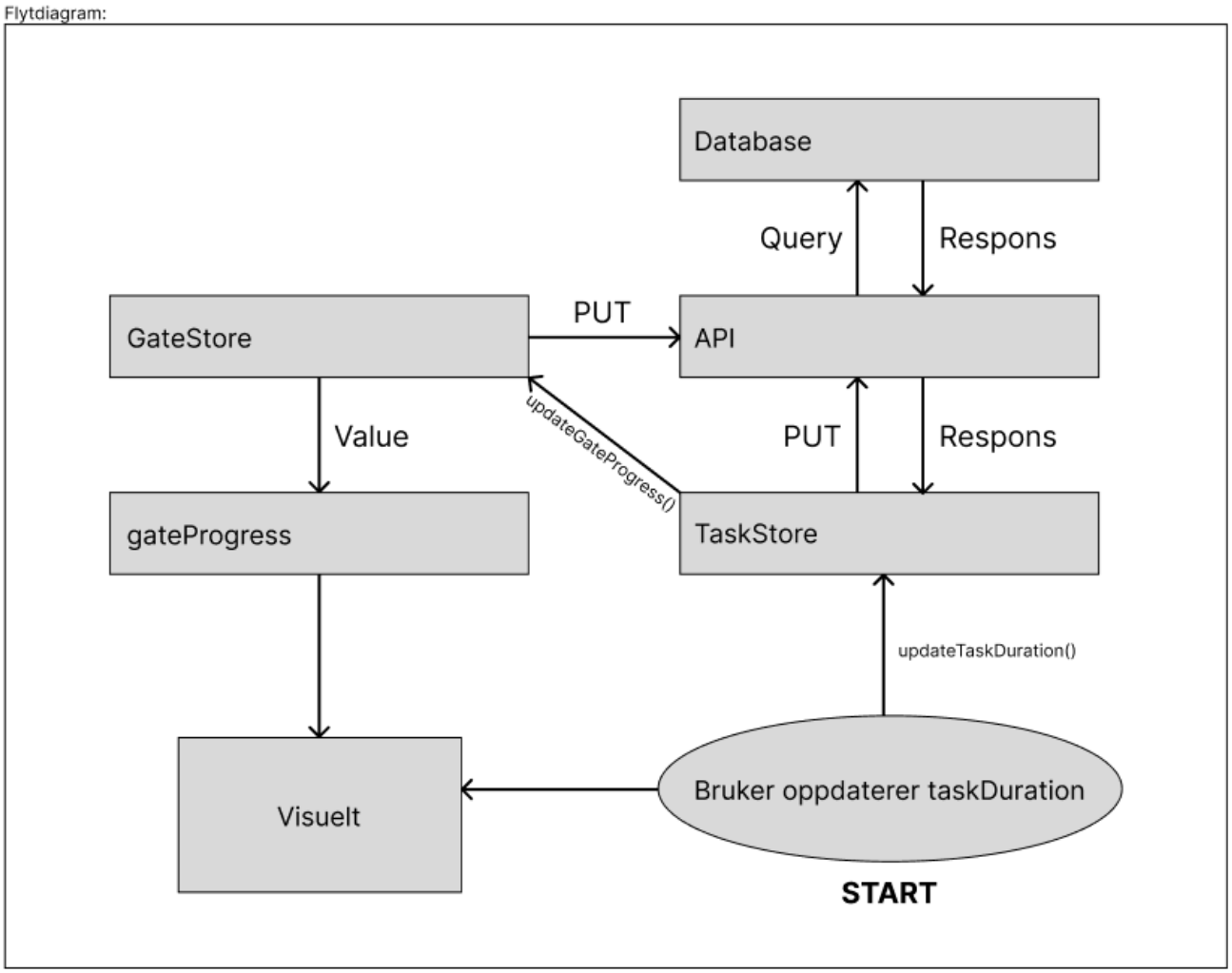
5.8 Arkitektur og dataflyt

Etter dette kapittelet skal en mer helhetlig struktur av prosjektet være litt mer klart, men for å samle de sammen, vil koblingen mellom de fremstilles her. For å forstå hvordan det hele

fungerer gås det gjennom eksemplene som er brukt gjennom hele kapitellet på en bredere måte.

5.8.1 TaskDuration eksempel

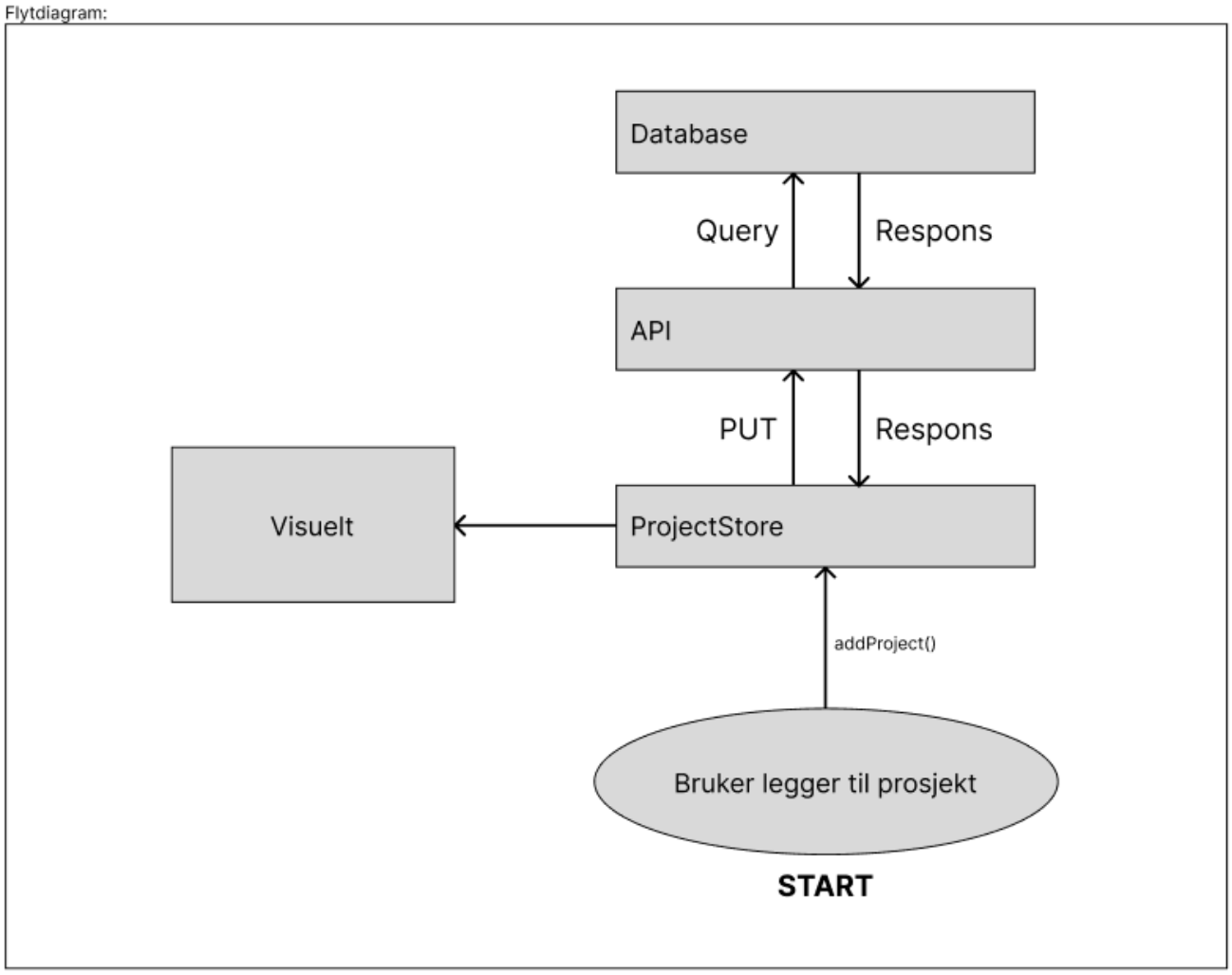
Når en bruker oppdaterer taskduration, skal dette fortelles om i taskstore. Taskstore vil så behandle dette reaktivt og oppdatere de komponentene som er avhengig av taskDuration. Et eksempel på dette er progress for en task. Når progress for en task oppdateres vil det både oppdateres visuelt, men asynkronisert vil en put request sendes til API som så oppdaterer databasen. Vi kan se dette i figur.



Figur 5.8.1-1: Flytdiagram for oppdatering av task-duration

5.8.2 AddProject eksempel

Når et prosjekt legges det til informasjonen i et form som sendes fra prosjektside komponenten til prosjektstore. Denne storen vil så pakke inn denne informasjonen og sende det videre til API endepunktet proejects med metoden POST. API'en vil deretter ta imot denne informasjonen og pakker det inn i en query som sendes videre til databasen. Denne databasen vil kjøre prosedyren for å duplisere et prosjekt og fyller ut det nye prosjektet med den nye informasjonen. Når dette er ferdig, hentes prosjektene på nytt og deretter visualiseres for brukeren.



Figur 5.8.2-1: Flytdiagram for opprettelse av prosjekt

Kapittel 6

Resultater

Dette kapittelet presenterer resultatene av evalueringen og utfallet til prosjektet i sin helhet. Gjennom en systematisk gjennomgang av hvert evalueringsresultat blir leseren ledet gjennom prosessene og metodene som ble brukt for å vurdere prosjektets fremgang og suksess.

6.1 Evalueringsresultat

Gjennom delkapittelet skal resultatene fra evalueringsdelen av prosjektet fremstilles. Resultatet av prosjektevalueringen er delt inn i flere underkapitler basert på evalueringsplanen slik som den er definert i punkt 4.13.

6.1.1 Implementering og feilhåndtering

Underveis i prosjektet var det planlagt en form for testing. Ettersom det ikke var komplisert logikk som skulle implementeres ble det ikke brukt UnitTesting. Det ble heller valgt en annen fremgangsmåte for å sørge for at implementasjonene oppførte seg som ønsket. Etterhvert som det ble implementert kode, ble det brukt mye feilhåndtering ilag med utskrift til konsoll for å holde kontroll over dataflyten. Dette innebar blant annet:

- Sørge for at data ble lagt til på riktig sted og skrive til terminal når det ble gjort.
- Sørge for at data ble overført mellom komponenter og sider, og skrive verdier til konsoll etterhvert som dette ble gjort.

- Bruk av Try/Catch for å fange eventuelle feilmeldinger og skrive ut diverse meldinger dersom en feilmelding skulle oppstå.

```
137  onMounted(async () => {
138    const projectId = route.params.id; // parser routen til en int og spesifiserer base 10
139    if (!projectId) {
140      console.error('Project ID is not provided');
141      return;
142    }
143
144    try {
145      // Henter prosjekt info
146      const fetchedProject = await store.getProjectById(projectId);
147      if (fetchedProject) {
148        project.value = fetchedProject;
149        console.log('Project found:', project.value);
150      } else {
151        console.error('Project not found:', route.params.id);
152        return;
153      }
154
155    } catch (error) {
156      console.error('Error fetching project:', error);
157    }
```

Figur 6.1.1-1: Utklipp av try/catch eksempel fra koden

```
Project found:                                                    [id].vue:149
▼ Proxy(Object) {id: '10', title: 'Project 1', progress: 44, onTime
  Date: '1970-01-01T00:00:00.000Z', PEM: 'Hans Boe', …} i
  ▶ [[Handler]]: MutableReactiveHandler
  ▼ [[Target]]: Object
      PEM: "Hans Boe"
      POdate: "2024-05-08T00:00:00.000Z"
      SFdate: "2032-06-09T00:00:00.000Z"
      archive: false
      comment: "\nKommentar\n"
    ▶ gates: {}
      id: "10"
      onTimeDate: "1970-01-01T00:00:00.000Z"
      progress: 44
      title: "Project 1"
    ▶ [[Prototype]]: Object
    [[IsRevoked]]: false
```

Figur 6.1.1-2: Utklipp av konsoll når programmet kjører og en prøver å gå inn i et prosjekt

6.1.2 Møter med oppdragsgiver

Totalt i utviklingsperioden hadde prosjektgruppen 5 møter med oppdragsgiver fra TechnipFMC. Møtene var en viktig del av utviklingsprosessen ettersom det ble mulighet til å vise frem produktet, få tilbakemeldinger, og stille spørsmål. Ettersom det ble tidlig lagd en

prototype som oppdragsgiver var fornøyd med, og prosjektet ble utviklet i henhold til denne og deres Excel-versjon, oppsto det få misforståelser og produktet ble utviklet korrekt. Som regel ble det diskutert hvordan datoene i prosjektet skulle samhandle og påvirke hverandre, slik at dette var tydelig før implementasjonen begynte og etter hvert som det ble implementert. Dette resulterte i at produktet ble utviklet tilnærmet slik oppdragsgiver hadde sett for seg og at prosjektetgruppen ikke kastet bort tid på feilimplementering.

6.1.3 Funksjonelle egenskaper

I evalueringsplanen ble det nevnt at funksjonelle egenskaper i prosjektet skulle sammenlignes med listen *Produktets funksjonelle krav* fra visjonsdokumentet. Produktet inneholder alle funksjonelle egenskaper i listen utenom punkt 1-4:

1. Opprette bruker
2. Innlogging
3. Logg ut
4. Gi brukere tilgang

Ved single sign-on er det ikke være nødvendig å opprette en bruker, ettersom brukerne av applikasjonen allerede skal ha klare brukere via Windows Authentication.

Dersom en bruker setter en task til fullført var det planlagt at signaturen skulle bli hentet fra brukernavnet til den innloggede brukeren. Som et resultat av manglende implementasjon av single sign-on i webapplikasjonen, er det ikke mulig å hente ut brukernavnet som ønsket. Dermed har det blitt implementert en alternativ løsning. En bruker kan fysisk sette et brukernavn, og ved signering settes signaturen til brukernavnet. Verdien som holder brukernavnet er derimot ikke lagret i databasen og vil derfor måtte skrives inn hver gang en bruker går inn på nettsiden.

De funksjonelle egenskapene produktet har implementert er dermed de nødvendige CRUD-opperasjonene nødvendig for at skal fungere etter sin hensikt:

- **Create:** Bruker har mulighet for å opprette et prosjekt, som dermed automatisk oppretter tilhørende gates og tasks for prosjektet. Om det er ønskelig, kan en bruker opprette en

ny gate i et prosjekt. Brukeren har også mulighet til å opprette en task i en gate.

- **Read:** Bruker har tilgang til å lese den nødvendige informasjonen for produktet sin hensikt og etter oppdragsgiver sine krav. Dette impliserer at brukeren kan se en liste av alle prosjekter med den ønskede informasjonen knyttet til hvert prosjekt. Det er også mulig å trykke seg inn i et spesifikt prosjekt for å få mer detaljer tilhørende prosjektet. I et prosjekt vises gates som hører til prosjektet med den etterspurte informasjon om hver gate. Brukeren kan også vise tasks som hører til en gate med nødvendig informasjon knyttet til hver task.

- **Update:** Bruker har mulighet til å endre informasjon angående prosjekt. Dette inkluderer endring av tittel, PEM, PO-dato, SF-dato og kommentar. For en gate kan bruker endre tittel og gate nummer. I en task kan bruker endre task nummer, tittel, ansvarlig rolle, progresjon (som vil påvirke progresjonsbaren i en gate), varighet for oppgaven og kommentar. Dersom brukeren setter en task til fullført kreves en signatur som signeres når bruker trykker *JA*. Signaturen tilsvarer navnet til brukeren, navnet har brukeren mulighet til å endre.

  Brukeren har også mulighet til å endre template. Alle prosjekter i etterkant av denne endringen vil følge strukturen til templaten.

- **Delete:**Bruker har mulighet til slette en task. Bruker har mulighet til å slette en gate. Dersom en gate slettes slettes tilhørende tasks. En bruker har mulighet til å slette et prosjekt. Dersom en bruker sletter et prosjekt, slettes alle tilhørende gates og tasks.

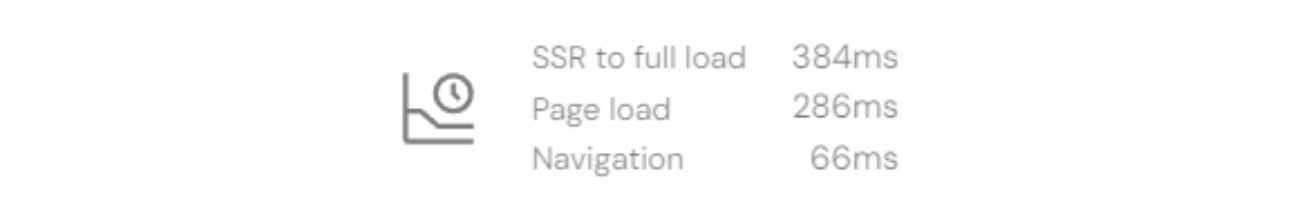
6.1.4 Ikke-funksjonelle egenskaper

Første punktet i Produktets ikke-funksjonelle krav er *Brukervennlighet* som blant annet betyr at webapplikasjonen skal være responsiv. For å måle hvor responsiv webapplikasjonen er har det blitt benyttet seg av NuxtDevTools se punkt 4.1, et visuelt verktøy som gir innsikt i webapplikasjonen(Fu, 2023). Verktøyet gir blant annet beregninger på:

- SSR to full load: Tiden det tar for *Server-Side Rendering*(*SSR*) å fullføre og tiden siden bruker på å laste inn i klientens nettleser. Denne beregningen indikerer dermed hvor lang tid det tar fra den første forespørselen til siden er fullstendig lastet inn og klar for interaksjon for brukeren.

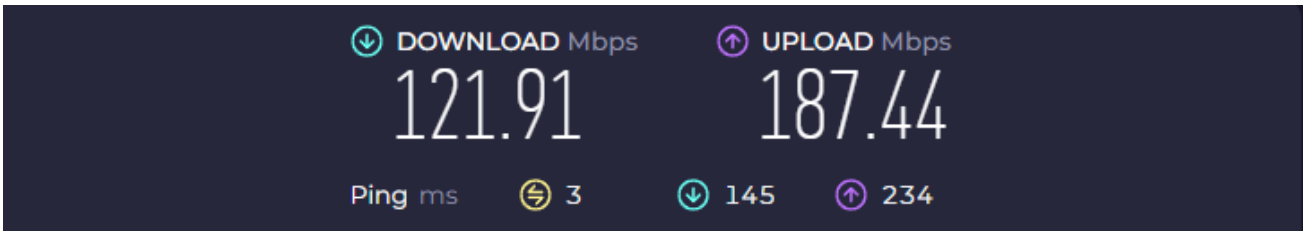
- Page Load: Den totale tiden det tar for hele siden å laste, inkludert ekstra ressurser som javascript og stildefinisjoner. Raskere page load bidrar til en mer responsiv brukeropplevelse fordi brukerne kan begynne å samhandle med siden etter hvor lite tid det tar for page load å fullføre. (*What Is Page Load Time on a Website and Why It Matters*, n.d.)

- Navigation: Måler hvor raskt webapplikasjonen svarer når brukeren forsøker å sende navigeringsforspørsler, for eksempel ved å klikke på lenker i sidebar eller på et prosjekt for å navigere gjennom ulike ruter i webapplikasjonen. En lavere navigasjonstid indikerer at webapplikasjonen skifter raskt mellom sider, noe som forbedrer hvor responsiv navigeringen og hastigheten av webapplikasjonen oppfattes av brukeren. (Scholz, 2023)



Figur 6.1.4-1: Resultat av beregninger fra NuxtDevTools

Ved beregningene navigerte nettleseren fra en annen annen nettside til webapplikasjonen for å sørge for at det ga et realistisk innblikk i en bruker som navigerer seg til webapplikasjonen og skal laste inn siden. Beregningene ble repetert 10 ganger for å sette et gjennomsnitt som målingen kunne ta et utgangspunkt ifra ettersom det var litt variasjon i resultatene. Det ble også utført en test på HVL av deres trådløse nettverk for å sjekke nedlasting og opplastingshastighet via Speedtest ettersom dette vil ha en påvirkning for resultatet med hvor raskt spørringer til databasen blir gjennomført (2024). Nedlastingshastigheten var 121,91 Mbps og opplastingshastigheten 187,44 Mbps. Resultatet for testen av internett hastigheten og snittet for beregningene fra NuxtDevTools vises under:



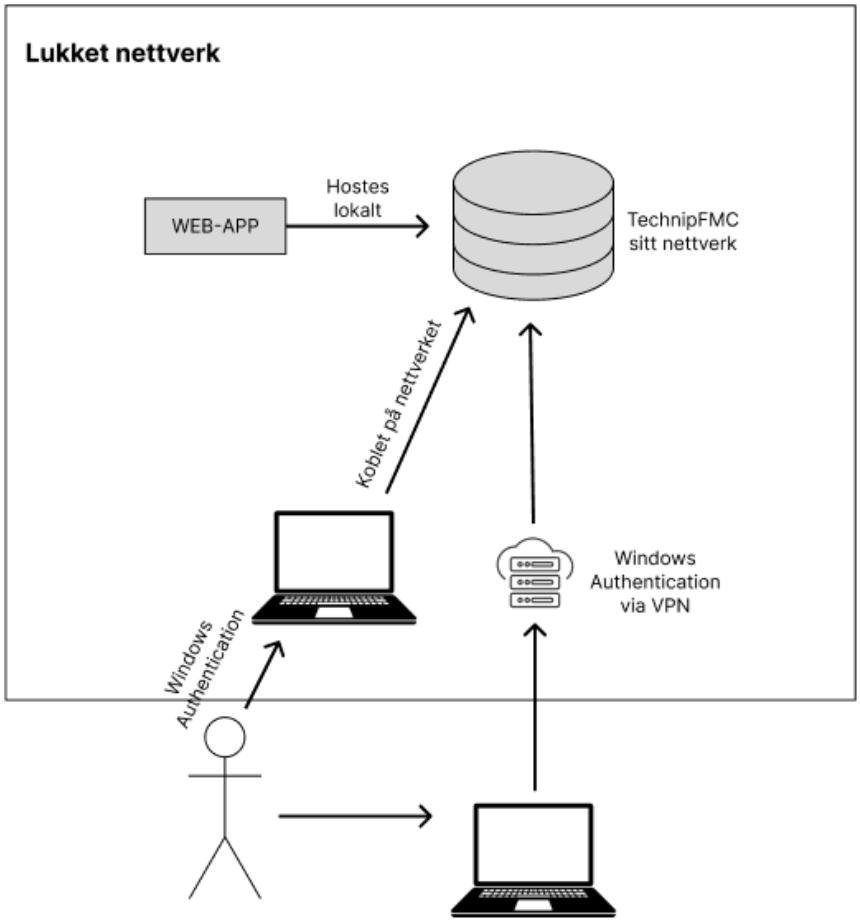
Figur 6.1.4-2: Nedlasting- og opplastinghastighet ved Høgskulen på Vestlandet sitt trådløse nettverk for studenter

	SSR	Page Load	Navigation
	280	216	224
	477	411	80
	288	185	141
	357	222	172
	248	184	82
	289	201	154
	275	202	85
	306	205	95
	340	261	162
	379	234	143
Sum:	3239	2321	1338
Average:	323.9	232.1	133.8

Figur 6.1.4-3: Gjennomsnittet av NuxtDevTools sine beregninger

Henting av data fra databasen er ikke fullstendig optimalisert og har et tydelig forbedringspotensiale. På grunn av manglende tid ble ikke produktet like effektivt som planlagt knyttet til prosessen rundt oppretting av prosjekter. Når brukeren legger til et prosjekt henter webapplikasjonen ut igjen alle prosjektene som eksisterer fra databasen. Den tenkte løsningen var at prosjektet heller sjekker hvilket prosjekt som mangler, og henter det spesifikke prosjektet fra databasen ved opprettelse.

Andre punkt i *Produktets ikke-funksjonelle krav*, *Sikkerhet* omhandler å ikke ha informasjon tilgjengelig for offentligheten. Fordi applikasjonen vil kjøre på TechnipFMC sin server, får man ikke tilgang til webapplikasjonen med mindre man benytter seg av en Windows PC som er godkjent på deres lukkede nettverk. For hjemmekontor gjelder samme kravet om å benytte seg av en godkjent Windows PC, samt at brukeren må benytte seg av VPN.



Figur 6.1.4-4: Nettverket til TechnipFMC

Beskyttelse mot SQL-injeksjoner og usikre spørringer henger sammen med sikkerhet. men er derimot konsepter koden i webapplikasjonen ikke har hatt som prioritering. SQL-injeksjoner innebærer at en angriper manipulerer spørringer sendt til databasen og får tilgang til sensitiv data som angriperen ikke skulle hatt tilgang på. Grunnen for at dette ikke er blitt ordentlig prioritert er manglende kunnskap innad i prosjektgruppen for å håndtere beskytte mot angrep og lite tid for å undersøke forbedringer.

6.1.5 Brukertesting

Siste møte med TechnipFMC og prosjektgruppen ble holdt på TechnipFMC sitt arbeidslokale på Ågotnes og det ble samtidig gjennomført brukertesting av prosjektets kontaktperson. Brukeren samhandlet lenge med applikasjonen, og fikk testet alle

brukstilefeller innen CRUD (create, read, update og delete) og navigering. Personen som brukertestingen ble utført på er også utvikleren av Excel-dokumentet, og skal ta i bruk applikasjonen ved overlevering. Dette førte til at brukeren enkelt kunne sammenlikne den gamle og nye løsningen, og gi en vurdering på prosjektet som vil være en viktig faktor i den konkluderende fasen av prosjektet.

6.1.6 Resultat av spørreskjema

I etterkant av brukertestingen ble det tildelt et spørreskjema med 21 spørsmål og påstander. Hensikten var å få et resultat av brukeren sin vurdering angående brukervennlighet, oversiktlighet og optimalisering i den nye løsningen sammenliknet med den gamle. Spørsmålene og svarene fra spørreskjema vises i tabellen under:

Tabell 6.1.6-1 : Svar fra spørreskjema

Spørsmål	**Svar**
Fikk du veiledning underveis i demonstrasjonen?	Ja
Vurder i hvor høy grad du mener applikasjonen utfyller de følgende kvalitetene. 1 er veldig dårlig utfyllelse, og 5 er veldig høy.	
Applikasjonen er visuelt oversiktlig	5
Applikasjonen er mer visuelt oversiktlig enn den forrige løsningen.	4
Applikasjonen er lett å navigere.	5
Applikasjonen er lett å navigere i forhold til den forrige løsningen.	4
Det er enkelt å opprette et prosjekt i applikasjonen.	5
Det er enkelt å opprette et prosjekt i applikasjonen i forhold til den forrige løsningen.	5
Det er enkelt å slette et prosjekt i applikasjonen.	5
Det er enkelt å slette et prosjekt i applikasjonen i forhold til den forrige løsningen.	5
Det er enkelt å få oversikt over prosjektene i applikasjonen.	5
Det er enkelt å få oversikt over prosjektene i applikasjonen i forhold til den forrige løsningen.	5
Det er enkelt å endre progresjon i applikasjonen.	5
Det er enkelt å endre progresjon i applikasjonen i forhold til den forrige løsningen.	5
Det er enkelt å endre på et prosjekt i applikasjonen.	5
Det er enkelt å endre på et prosjekt i applikasjonen i forhold til den forrige løsningen.	5
Applikasjonen er brukervennlig (Generelt)	5
Applikasjonen er mer brukervennlig enn forrige løsning (Generelt)	5
Applikasjonen er oversiktlig (Generelt)	5
Applikasjonen er mer oversiktlig enn forrige løsning (Generelt)	4
Applikasjonen er rask (Generelt)	5
Applikasjonen er raskere enn forrige løsning (Generelt)	5

6.2 Prosjektresultat

Prosjektet helhetlig har resultert i en web-app som utfyller de funksjonaliteter som eksisterte i det tidligere Excel-dokumentet, og forbedret det både i henhold til brukervennlighet og optimalisering slik som oppdragsgiver forespurte. Oppdragsgiver har gitt tilbakemelding på at produktet utfyller alle krav og forventninger de hadde, i tillegg til å overraske positivt med ekstra funksjonaliteter de ikke hadde sett for seg. Prosjektet har dermed resultert i både et produkt som utfyller de krav som var satt, og denne rapporten som dokumenterer prosessen bak produktutviklingen, og resultatene av dette.

Kapittel 7

Diskusjon

Dette kapittelet utgjør en viktig del av bachelorprosjektet, for her skal det reflektere over utfordringer, resultater og implikasjoner av arbeidet som har blitt utført. Dette kapittelet skal utforske og analysere de sentrale funnene, begrensningene og de potensielle konsekvensene av prosjektet. Gjennom en vurdering av resultatene og beslutninger søkes en forståelse av prosjektets styrker og svakheter, samt identifisering av områder for videre forbedring og arbeid.

Ved å fremstille de ulike aspektene av prosjektet, fra designvalg og metodikk til implementering og resultater, skal det fremlegges en helhetlig forståelse av det utførte arbeidet. Gjennom en kombinasjon av kvantitative og kvalitative analyser, skal det utforskes både de målbare resultatene og de subjektive erfaringene som har oppstått underveis i prosjektet.

7.1 Resultat

For å gi en måling på suksessen av prosjektet er det viktig å sette lys på en kritisk vurdering av resultatet. Her vil det bli gitt en detaljert vurdering angående resultatene som ble fremstilt i ReferanserEvalueringsresultat og ReferanserProsjektresultat.

7.1.1 Møter

Prosjektgruppen ønsket å ha flere møter med oppdragsgiver for å få nødvendige tilbakemeldinger rundt prosjektet, som omtalt i *Agile Metoder*. Dette gikk etter planen ettersom det var mulighet for å kalle inn til, og planlegge møter, via mail. slik at informasjon angående SSO kunne bli brukt for å knytte verifisering til brukerne av webapplikasjonen. Det ble sendt flere forespørsler og diskutert i om å få satt opp et møte med ansatte fra IT-avdelingen til oppdragsgiver. Et møte ble derimot aldri satt opp og konsekvensen av dette vil bli mer omtalt senere i delkapittelet.

Siste møte med TechnipFMC ble utført fysisk på deres arbeidslokaler hvor prosjektgruppen fikk de siste tilbakemeldingene for prosjektet. Tilbakemeldingene var veldig nyttige ettersom de stort sett omhandlet stylingen av prosjektet rettet mot størrelse av prosjekter, gates og tasks, samt utseende av etiketter og knapper. De etterspurte endringene ble lagt til i siste iterasjon og implementert for å tilfredstille oppdragsgiver sitt behov og ferdigstille utseende av produktet.

7.1.2 Mangel på UnitTesting

Gjennom utviklingsprossen ble det ikke benyttet Unittesting. Det ble derimot brukt en alternativ ordning ved å bruke utskrift til konsoll for å holde kontroll på dataflyten, samt få ut suksessfulle meldinger og feilmeldinger. På denne måten kunne utviklerne ha god oversikt hva som ble gjort rundt i applikasjonen og finne eventuelle feil og mangler ved implementasjoner. Selv om Unittesting eller en annen form for testing av koden hadde vært viktig for å oppdage problemer og bugs, særlig i de tidlige stegene av utviklingsperioden og i henhold til Agil metodikk, var denne fremgangsmåten tilstrekkelig for å teste koden og oppdage feil. Gruppen er klar over at mangel på UnitTesting kan ha en negativ påvirking rundt de kvalitative aspektene av produktet, testing er identifisert som et tydelig forbedringsområde for webapplikasjonen. (Boog, n.d.)

7.1.3 Tolkning av beregninger fra NuxtDevTools

Ved bruk av Resultat av beregninger fra NuxtDevTools ble en generell indikasjon av webapplikasjonen sin hastighet forstått. Hastigheten er vesentlig for brukeropplevelsen fordi det er knyttet opp mot hvordan brukeren får samhandle med webapplikasjonen.

SSR to full load resulterte i et gjennomsnitt på 323,9 ms. Dette er som nevnt i 6.1 hvor lenge

brukeren må vente samhandling med webapplikasjon kan begynne. Det gir dermed et mål på hvor raskt nettstedet kan bli visuelt og funksjonelt responsivt for brukeren etter den initielle forespørselen. Disse beregningene indikerer at i forhold til innlasting vil brukeren ha en god brukeropplevelse ettersom den ikke må vente lenge på å laste inn siden.

Page Load resulterte med et gjennomsnitt på 232,1 ms. Ettersom Page Load inkluderer alle ressurser som JavaScript-filer og CSS-stiler, virker dette som en veldig rask lastetid. En god innlastingstid for en nettside er som regel under 3 sekunder, mens en ønsker å oppnå en rasker Page Load Time på 1-2 sekunder (*Page Load Time vs. Response Time – What Is the Difference?*, 2024). Utifra dette tyder resultatet på at nettstedet er optimalisert godt, og ressursene lastes raskt inn i nettleseren, særlig iforhold til krav som stilles til en nettside i 2024.

Navigation hadde en gjennomsnittstid på 133,8 ms. Dette indikerer at nettstedet svarer raskt på brukerens navigasjonsforespørsler, for eksempel ved klikk på lenker eller bytte mellom sider. En lav navigasjonstid bidrar til en mer responsiv opplevelse for brukeren, og 133 ms er en veldig rask responstid. Når brukeren benytter seg av navigeringsamhandlinger i webapplikasjonen vil omdirigeringen knapt være merkbar.

7.1.4 Bedre implementasjon rundt opprettelse av prosjektet

Gruppen har ikke gjennomført målinger på hvor lang tid opprettelse av prosjekter tar. Opprettelsen skjer derimot ganske momentant fra brukeren sitt perspektiv. Forbedringsområde som gruppen har identifisert er relatert til det å lage god kode og effektiv kode. Ettersom det opprettes i underkant av ett prosjekt i måneden er ikke dette et problem som anses som kritisk for produktet.

7.1.5 Resultatene fra spørreskjema

Etter *Brukertesting* og *Svar fra spørreskjema* fikk gruppen en kommentar angående hvorfor det var svart *4* på *Applikasjonen er mer visuelt oversiktlig enn den forrige løsningen* og *Applikasjonen er mer oversiktlig enn forrige løsning* (*Generelt*). I den forrige løsningen hadde oppdragsgiver et kakediagram øverst på Excel-arket som viste en generell oversikt om progresjonen til prosjektene og et diagram som viste utført aktivitet av gates. Dette ble derimot ikke implementert i den nye løsningen da partene kom til enighet om at det var unødvendig å implementere ettersom hensikten med diagrammene var en motiverende visualisering men

som ikke tilbrakte mer informasjon og oversikt iforhold til oppsettet av den nye løsningen. Oppdragsgiver utdypet at dette ikke var gruppen sin skyld, og at den nye løsningen helhetlig var mer oversiktlig.

Den gamle løsningen hadde god oversikt av prosjekter, gates og tasks, samt statusen angående progresjonen. At den nye løsningen likevel er litt mer oversiktlig enn den gamle løsningen er derfor veldig positivt ettersom målet var å ivareta oversiktligheten og forbedre dersom det var mulig. I påstandene *Applikasjonen er visuelt oversiktlig* og *Applikasjonen er oversiktlig* (*Generelt*) var oppdragsgiver veldig enig.

Utenom oversiktligheten av produktet var påstandene i spørreskjema definert for å enklere måle brukervennlighet og optimalisering, ettersom dette var hovedkriteriene oppdragsgiver ønsket å få ordnet, definert i Problembeskrivelse og mål. Påstandene omhandlet enkeltheten relatert til oppretting og sletting av prosjekt, endringringer og navigering. Disse skulle gi en indikasjon til senere besvarelse på de spesifikke påstander om den nye løsningen oppfattes som brukervennlig og rask. Hensikten med denne fremgangsmåten var å enkleregjøre identifisering av forbedringsområder i webapplikasjonen dersom oppdragsgiver ikke var fornøyd. Her var besvarelsen derimot *Veldig Høy* på alle påstander utenom *Applikasjonen er lett å navigere i forhold til den forrige løsningen* ettersom Excel-løsningen ikke hadde komplisert navigering, da bruker kunne navigere til ulike Excel-ark i bunnen av applikasjonen. Besvarelsen indikerer at prosjektet har resultert i de forventninger oppdragsgiver har stilt til gruppen.

7.1.6 Manglende implementasjon av Single Sign-On (SSO)

SSO-implementering var et forslag fra oppdragsgiver og ble ikke implementert som et resultat av en svikt fra deres side angående å skaffe den nødvendige informasjon til gruppen for å implementere dette. Dermed var den manglende implementasjonen ikke noe oppdragsgiver klagde på. Mangel på kommunikasjon var en av risikoene definert i risikoanalysen, se 4.12.3-2, og her ser vi tydelig ett eksempel av konsekvensene hvor kommunikasjonen svikter.

Som et resultat av at det ikke er implementert SSO til applikasjonen vil ikke en bruker være direkte knyttet til applikasjonen. Dette betyr at det er verifisering knyttet til webapplikasjonen men ikke autentisering. Både en ”adminbruker”og en ”ordinær bruker”vil ha den samme tilgangen til manipulering av webapplikasjonen. Autentisering er dermed identifisert som et

tydelig forbedringsområde for webapplikasjonen for å kunne begrense tilgang til ulike type brukere og er et avvik fra punkt 4 i Produktets funksjonelle krav, *Gi brukere tilgang* (*admin*), og punkt 5, *Opprette prosjekt* (*admin*).

Gruppen kan ikke påstå med sikkerhet hvorvidt informasjonen stemmer ettersom det er muligheter for feilinformasjon fra oppdragsgiver. Grunnlaget for dette er at informasjon ikke har blitt forkynnet av ansatte som har direkte ansvar for TechnipFMC sine systemer, men av ansatte som har delvis forståelse angående et komplisert system.

7.1.7 Sikkerhet

Anngående sikkerhet er det uvisst hvor mye single sign-on og TechnipFMC sitt lukkede system beskytter mot SQL-injeksjoner. Temaet er derfor identifisert som et område som krever mer undersøkelse for å potensielt øke sikkerheten og beskytte mot uønskede angrep mot databasen. Gruppen vil heller ikke komme med noen påstander om hvordan TechnipFMC sine systemer gir beskyttelse mot SQL-injeksjoner.

7.2 Metodikk

7.2.1 Designmetodikk

Design prosessen under prosjektet har for det meste gått slikt som planlagt. Delen som ble utdypet rundt dobbeltdiamanten se punkt 3.1.1, ble godt gjennomført med idemyldring i de relevante fasene for så å konkreisere de sammen med oppdragsgiver. Det er mulig designløsningen kunne ha blitt mer detaljert i løsning tidligere, men med noe av informasjonen om database og arkitektur til oppdragsgiver som kom sent, så ble det løst godt. Etter dobbeltdiamant prosessen, gikk så designprosessen over til noe som mer liknet designthinking se punkt 3.3. Denne delen av prosessen kunne ha blitt mer hyppigt brukt da, det ikke var veldig nok møter mellom gruppen og oppdragsgiver til å ta dra nok nytte av det.

7.2.2 Utviklingsmetodikk

I henhold til konseptene knyttet til Agil utvikling skal gruppen reflektere hvorvidt disse har blitt oppfylt. Gjennom utviklingsperioden har prosjektet vært delt inn i iterasjoner og fulgt

planen til Ganttdiagrammet. Det er derimot ikke vært et konstant skille mellom backend- og frontenditerasjoner slik som planlagt i Prosjektplan. Dette kommer som et resultat av at gruppen tidlig i utviklingsperioden fikk ferdigstilt det meste av frontend i henhold til prototypen. I tillegg er produktet en full-stack applikasjon, hvor det ofte måtte gjøres små endringer i front-end etter hvert som kode knyttet til back-end ble implementert, noe gruppen burde tatt mer hensyn til da de valgte å dele prosjektet opp i backend- og frontendinterasjoner. Mot slutten av utviklingsperioden var nesten alle oppgaver knyttet til back-end. Likevel føler gruppen at prinsippene rundt å arbeide i iterasjoner og med oppgaver har blitt oppfylt, selv om det ikke har vært en tydelig skille mellom iterasjonene lengre ut i utviklingen. Det ble derimot diskutert ved bytte og underveis i iterasjonene hva som var hovedfokuset for planlagt arbeid.

Gruppen søkte også hyppig kommunikasjon med oppdragsgiver. Ved behov av besvarelse angående omfattende spørsmål eller generell tilbakemelding på prosjektet sin status underveis i utviklingsfasen, kalte gruppen inn oppdragsgiver til møte. Mindre omfattende spørsmål som ikke krevde like detaljert forklaring ble avklart over Mail. Dette førte til at gruppen kontinuerlig holdt oppdragsgiver involvert i prosjektet, samt skaffet nødvendig kritikk for å utvikle et vellykket produkt. Uten involvering av oppdragsgiver hadde det ikke vært mulig å utvikle et produkt som dekker deres behov og som ivaretar kompleksiteten til deres Excel-løsning.

Kontinuerlig integrasjon

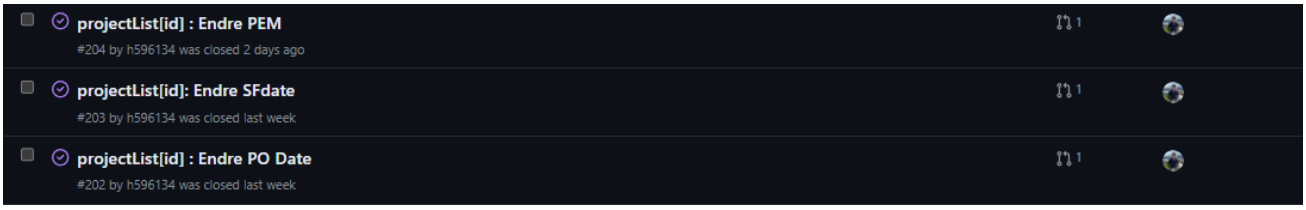
GitHub Flow hadde en sentral rolle i utviklingsprosessen. I GitHub Flow sitt kanban-brett ble det konstant opprettet oppgaver for den gjeldende eller fremtidige iterasjoner. Hvilken iterasjon oppgaven ble lagt til i var avhengig av hvor kritisk oppgaven var for fremgangen til produktet.

Når en utvikler ønsket å utføre endringer i koden, måtte endringene utføres i en gren for en eksisterende oppgave som tok for seg de nødvendige endringene. Dersom det ikke eksisterte en oppgave, måtte utvikleren opprette en ny oppgave for å så lage en gren for oppgaven og tilegne den oppgaven til seg selv. Gruppen hadde dermed god oversikt over hvem som arbeidet med hva under hele prosessen.

Etter at arbeidet i grenen var ferdig, ble det sendt en pull-request til hovedgrenen, hvor endringer i koden ble validert før den ble flettet inn. Valideringen innebar automatisk sjekk

av konflikter mot hovedgrenen, samt en manuell sjekk utført av en annen utvikler enn den som skrev den relevante koden. Om koden passerte begge disse to valideringstestene kunne den så bli flettet inn i hovedgrenen.

I sin helhet sørget arbeidet i grener for at det ikke direkte ble gjort endringer i hovedgrenen, samt at det var mulig å se over tidligere flettet inn kode dersom det oppsto problemer med deler av koden som hadde blitt implementert. Det sørget også for god kvalitet av koden som ble lagt til.



Figur 7.2.2-1: Utklipp av grener som har blitt lagt til hovedgrenen

7.3 Problemstilling

Hovedfokuset i prosjektet har vært knyttet til problemene som ble fremstilt i *Problembeskrivelse og mål* angående hastighet og simplisitet da Excel-løsningen ikke var tilstrekkelig når det gjaldt brukevennlighet og brukeropplevelse og løsningen ble omtalt som treg og at det var tungvint å utføre endringer.

Punktene tidligere i kapittelet som har adressert problemene kan trekkes ut fra *Resultatene fra spørreskjema* og *Tolkning av beregninger fra NuxtDevTools*. Resultatet presentert er et prodult som både har en god målbar hastighet i forhold til det som er forventet av dagens webapplikasjoner, og fra oppdragsgiver sitt synspunkt etter samhandling med produktet. Oppdragsgiver ga også tilbakemelding på at produktet lastet inn data fortere og var mer responsivt enn den gamle løsningen. Enkelhet er målbart i forhold til hvordan brukeren som skal benytte seg av produktet oppfatter det. I resultatene fra spørreskjemaet kommer det tydelig frem at oppdragsgiver er fornøyd med simplisiteten gruppen har integrert i løsningen.

Gruppen har derimot ikke fått et målbart resultat angående hvor lang tid det tar før et prosjekt legges til i databasen og til brukeren får oppdateringen på skjermen. Det er tydelig

et forbedringspotensiale som vil øke hastigheten knyttet til denne prosessen. Selvom den generelle hastigheten i produktet er en betraktelig forbedring, ønsker utviklerne å påpeke at dette burde undersøkes når det gjelder å skrive god kode.

Kapittel 8

Konklusjon og videre arbeid

Dette kapittelet vil ta for seg rapportens konklusjon om arbeidet, en vurdering om hvorvidt resultatet dekker problemstilling og forskningsspørsmålet. Videre vil det diskuteres hvilke deler av prosjektet som ikke ble ferdigstilt og som kan forbedres i fremtiden.

8.1 Konklusjon

Bachelorprosjektet hadde som formål å utvikle et prosjektstyringsverktøy for TechnipFMC sterkt knyttet til deres arbeidsprosess med mulighet til å gi dem et overordnet blikk over deres prosjekter og tidsplan. Prosjektet tok sikte på å utvikle et bedre alternativ enn den tidligere løsningen, med et fokus på å være mer brukervennlig og optimalisert. Samtidig skulle ikke disse egenskapene gå utover kompleksiteten som var tilgjengelig i den tidligere løsningen. I den anledningen var det ønskelig å besvare spørsmålet:

Hvordan best transformere et Excel-dokument til en optimalisert og brukervennlig webapplikasjon?

For å kunne svare på dette var det nødvendig med grundig planlegging av hvordan produktet skulle utvikles. Produktet måtte ivareta:

1. Den logiske funksjonaliteten
2. en enkel navigering
3. Enklere og mindre tidskrevende brukergrensesnitt

4. Automatisere prosesser

5. Responsivitet

Det var viktig å ivareta den logiske funksjonaliteten med tanke på utregninger. Her var god kommunikasjon med arbeidsgiver viktig for å skaffe forståelse rundt hvordan utregningen ble gjort, se 3.3. Hvilke utregninger som skulle skje automatisk og hvilke variabler bruker skulle ha mulighet til å endre på. Her har det blitt ivaretatt alle utregninger som den gamle løsningen utførte, samt gitt mulighet for å endre de variablene som oppdragsgiver ønsket, se punkt 6.1.3. Dersom en variabel er avhengig av en annen, vil denne oppdateres dersom avhengigheten endres på.

I et Excel-dokument er navigeringen normalt sett overkommelig. Brukeren har ulike ark den kan navigere seg til, og navigeringen gjøres via å klikke på ønsket ark i bunnen av dokumentet. Istedenfor å ha navigering til ulike sider i bunnen av produktet, er det implementer en sidebar med lenker til dashboard, arkiv, prosjektlisten og hvert individuelle aktive prosjekt. Det er også mulig å navigere til et individuelt prosjekt ved å klikke på prosjektet i prosjektliste-siden, se 4.11.1. Dette var for å øke brukeropplevelsen rundt navigeringen til et spesifikt prosjekt med å gi flere muligheter. Oppdragsgiver var veldig fornøyd med denne ekstra funksjonaliteten, *se Prosjektresultat.*

Enklere og raskere brukergrensesnitt var også essensielt for å øke brukeropplevelsen. Problemet med den gamle løsningen var at det tok lang tid å opprette nye prosjekter, gates og tasks, se *1.4*. Derfor ble denne prosessen delvis automatisert slik at en bare trengte å fylle inn tittel for prosjekt, PO-date, SF-date og PEM. Deretter ble prosjektet opprettet med alle tilhørende gates og tasks. I tillegg påpekte oppdragsgiver at det var tungvint å utføre endringer i Excel-dokumentet, og ønsket knapper som gjorde prosessen raskere. Dette er blant annet implementert med en *slider* for å endre på progresjonen til en task. Brukergrensesnittet har vært et stort fokus siden begynnelsen av prosjektet, og gruppen føler nødvendige og gode løsninger har blitt gjort for å skape en økt brukeropplevelse for oppdragsgiver. De funksjonelle prosessene er både raskere å gjennomføre og enklere.

Sist er responsiviteten knyttet til brukersamhandlinger. Med bruk av reaktive komponenter i Vue, oppfattes produktet som veldig responsivt fra brukeren sitt perspektiv i front-end, se punkt *7.1.5* og *4.6*. Dette fører til at brukeren raskt ser endringer på skjermen sin når den samhandler med webapplikasjonen. I back-end derimot tar prosessene lengre tid og

spørringer kan forbedres for å ytterligere optimalisere produktet.

Med dette konkluderes det med at produktet har vært en suksess i forhold til forventningene og kravene som var satt av oppdragsgiver, samt at valgene som har blitt gjort i utviklingprosessen har ført til en vellykket transformasjon av et Excel-dokument om til en webapplikasjon.

8.2 Videre arbeid

Denne delen av rapporten vil omhandle det videre arbeid som både skal gjøres og gruppen anbefaler blir gjort. I punkt 8.2.1 vil det arbeid gruppen har igjen før applikasjonen er deployet til TechnipFMC sine servere omhandles, mens punkt 8.2.2 vil gå over det arbeidet gruppen ser for seg blir nødvendig etter at applikasjonen er overlevert.

8.2.1 Deployment og siste rest

Gruppen har avtalt med oppdragsgiver at deployment til deres servere skal foregå på et fysisk møte blant partene en gang ila. de neste par ukene. I forbindelse med dette har det da også blitt avtalt at gruppen skal se på de følgende problemer:
Må løses:

- Arkiv må implementeres
- Knapper som ikke har fått styling må få dette
- Float-tall må implementeres for prosjektside
- Template må være gjemt bak flere advarsler for å unngå uhell

Skal vurderes:

- Utvide sidebar om bruker skulle overgå forventet maksantall av aktive prosjekter som skal vises her
- Å implementere SSO–systemet til TechnipFMC i en form hadde vært praktisk, men er per nå vanskelig

- Signering av en utført arbeidsoppgave er per nå veldig simpelt. Det er upraktisk å måtte skrive inn brukernavn hver gang du går inn på nettsiden, så dette burde løses på en alternativ måte

- Oppdragsgiver spurte i slutten av prosjektet om det var mulig å få en oversikt over bare tasks som ikke ble utført i tide underveis i tidligere prosjekt. Gruppen skal se på dette, men har ikke lovet implementering mtp. at det er utenfor oppgavens opprinnelige omfang.

8.2.2 Fremtidig arbeid

Post-launch support: Ettersom gruppen har utviklet applikasjonen i løpet av et enkelt semester, er det fullt mulig at feil eller mangler ved produktet har gått ubemerket hen, og må fikses i fremtiden. I en vanlig produktutviklingsprosess ville dette blitt håndtert med jevnlig vedlikehold etter hvert som slike feil blir oppdaget, men ettersom gruppen som en offisiell enhet vil slutte å oppstå etter prosjektslutt er ikke dette noe som kan tilbys.

Optimalisering: Selv om applikasjonen er tilstrekkelig optimalisert per nå, kan det alltids forbedres. Slik applikasjonen er ved dags dato er mange av SQL-spørringene internt i programmet veldig enkle, og kan antakeligvis skrives om slik at de krever mindre av databasen, som og er den største flaskehalsen i forbindelse til optimalisering av latency.

Andre avdelinger: Oppdragsgiver har nevnt at de ser for seg at andre avdelinger internt i TechnipFMC vil fatte interesse i produktet, og ha det tilpasset deres behov. Dette kan gjøres i form av deployment av flere servere og databaser, med forskjellig prosjektmal enn den nåværende XT Gates-spesifikke, eller en implementering av forskjellige maler og prosjektunderdelinger i den nåværende applikasjonen.

Sikkerhetstesting: Gruppen har grunnet det korte tidsrommet applikasjonen har blitt utviklet og manglende kompetanse innenfor temaet, ikke hatt anledning til å utføre penetrasjonstesting og andre sikkerhets-valideringer. Grunnet den sensitive naturen til informasjonen som skal lagres og håndteres i programmet er dette noe som optimalt sett burde gjennomføres i fremtiden.

Bibliografi

(2024).
URL: *https://www.speedtest.net/*

Boog, J. (n.d.), 'Unit testing: Advantages & disadvantages'.
URL: *https://theqalead.com/test-management/unit-testing/*

Custer, C. (2021), 'What is connection pooling, and why should you care'.
URL: *https://www.cockroachlabs.com/blog/what-is-connection-pooling/*

Davidson, T. (2023), 'Atomic web design: A quick rundown', *CleanCommit* .
URL: *https://cleancommit.io/blog/enhancing-web-design-efficiency-and-consistency-with-atomic-web-design/*

Docs, M. W. (n.d.).
URL: *https://developer.mozilla.org/en-US/docs/Glossary/SPA*

Elisabeth Marie Hovden, Anders Benjamin Grinde, E. T. B. (2019), 'Web application for managing corporate resources', *HVL* .
URL: *https://hvlopen.brage.unit.no/hvlopen-xmlui/bitstream/handle/11250/2606507/Hovden-Grinde-Botnen.pdf?sequence=1*

Fu, A. (2023), 'Introducing nuxt devtools', *Nuxt* .
URL: *https://nuxt.com/blog/introducing-nuxt-devtools*

Gomez, J. (2023), 'Web apps vs. desktop apps: Understanding the differences'.
URL: *https://www.koombea.com/blog/web-apps-vs-desktop-apps/*

Han, E. (2022), 'What is design thinking & why is it important?', *Harvard Business School Online*

.
URL: *https://online.hbs.edu/blog/post/what-is-design-thinking*

IBM (n.d.), 'What is a relational database'.
URL: *https://developer.mozilla.org/en-US/docs/Web/API/Performance_API/Navigation_timing*

Introduction (n.d.). Docs.
URL: *https://nuxt.com/docs/getting-started/introduction*

Lyons-Kokkin, S. J. (2021), 'Hvordan den doble diamanten kan hjelpe deg til å løse små og store problemer', *Kristiania* .
URL: *https://www.kristiania.no/kunnskap-kristiania/2021/01/hvordan-den-doble-diamanten-kan-hjelpe-deg-til-a-lose-sma-og-store-problemer/*

Page Load Time vs. Response Time – What Is the Difference? (2024).
URL: *https://www.pingdom.com/blog/page-load-time-vs-response-time-what-is-the-difference/*

Reactivity in Depth (2024). Docs.
URL: *https://vuejs.org/guide/extras/reactivity-in-depth.html*

Sander, K. (2023), 'Prosjektmodell', *estudie* .
URL: *https://estudie.no/prosjektmodell/*

Scholz, F. (2023), 'Navigation timing'.
URL: *https://developer.mozilla.org/en-US/docs/Web/API/Performance_API/Navigation_timing*

Server Engine (2024). Docs.
URL: *https://nuxt.com/docs/guide/concepts/server-engine*

Service, A. W. (n.d.), 'What is a graph database?'.
URL: *https://aws.amazon.com/nosql/graph/*

State Management (2024). Docs.
URL: *https://nuxt.com/docs/getting-started/state-management*

TechnipFMC (2024).
URL: *https://www.technipfmc.com/*

The Progressive JavaScript Framework (2024). Docs.
URL: *https://vuejs.org/*

Torbjørn Bakke, Sindre August Strøm, H. T. H. (2022), 'System development of dashboard application - visualizing customer energy data', *NTNU* .
URL: *https://hdl.handle.net/11250/3004174*

Vilmur, A. (2020), 'What is agile web development? everything you need to know', *Marcel Digital* .
URL: *https://www.marceldigital.com/blog/what-is-agile-web-development-everything-you-need-to-know*

What Is Page Load Time on a Website and Why It Matters (n.d.).
URL: *https://sematext.com/glossary/page-load-time/*

Tillegg A

Prosjektdokumentasjon

A.1 Visjonsdokument

A.2 Kravdokumentasjon

A.3 Systemdokumentasjon

A.4 Prosjekthåndbok

Tillegg B

Kode

B.1 Repository

Github repository: https://github.com/h584903/XT-Gates