Høgskulen på Vestlandet

BACHELOROPPGAVE

Lisenskontrollsystem

License Control System

Gruppe D09

Rune Nordanger Mæstad
Simen Aarø Mathisen

DAT191
Fakultet for ingeniør- og naturvitskap (FIN)
Institutt for datateknologi, elektroteknologi og realfag
Dataingeniør

Veileder: Violet Ka I Pun

Innleveringsdato: 23.05.2022

TITTELSIDE FOR HOVEDPROSJEKT

Rapportens tittel:	*Dato:*
Lisenskontrollsystem	23.05.2022
Forfatter(e):	*Antall sider u/vedlegg: 48*
Simen Aarø Mathisen, Rune Nordanger Mæstad	*Antall sider m/vedlegg: 55*
Studieretning:	*Antall disketter/CD-er:*
Dataingeniør	0
Kontaktperson ved studieretning:	*Gradering:*
Violet Ka I Pun	Begrenset til 23.05.22
Merknader:	

Oppdragsgiver:	*Oppdragsgivers referanse:*
Stacc Escali AS	
Oppdragsgivers kontaktperson:	*Telefon:*
Gerdt Sverre Vedeler	926 22 063

Sammendrag:

Målet med prosjektet var å utvikle en løsning som forenkler arbeidsflyten for selgere i Escali. Løsningen ble en webapplikasjon som gir selgere direkte tilgang til kunde- og avtaledata. Evalueringen av resultatet var en brukertest hvor ansatte i Escali fikk teste løsningen, og gi tilbakemeldinger. Gruppen konkluderer med at resultatet, med enda mer funksjonalitet, i stor grad kan benyttes som en løsning for problemstillingen.

Summary:

The project goal was to develop a solution that simplifies the workflow for Escalis sellers. The solution, a web application that allows the sellers to have direct access to the customer and agreement information, was evaluated with a user test in which multiple people from Escali tested and provided feedback. The group concludes that the result, with even more functionality, can be used as a solution to the problem description.

Stikkord:

Webapplikasjon	.NET	Blazor

Høgskulen på Vestlandet, Fakultet for ingeniør- og naturvitenskap
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

Prosjektet er gjennomført av Rune Nordanger Mæstad og Simen Aarø Mathisen, og er en bacheloroppgave tilknyttet studieretningen Dataingeniør ved Høgskulen på Vestlandet. Gjennomføringen av prosjektet foregikk i perioden 10.01.22 til 23.05.22. Oppgaven ble utformet av Stacc Escali AS, og denne rapporten er en beskrivelse av veien fra problembeskrivelse til løsning.

Ved valg av oppgave ble det vektlagt et ønske om å tilegne seg nye ferdigheter og kunnskaper om teknologier gruppen ikke hadde fra før. Stacc Escali AS hadde en interessant oppgavebeskrivelse, med mulighet for å lære nye teknologier. På grunnlag av dette falt valget på denne oppgaven.

Gruppen ønsker å rette en stor takk til Gerdt Sverre Vedeler og andre ved Escali for tett samarbeid, samt god veiledning gjennom hele prosjektet. Samtidig vil gruppen også takke veileder, Violet Ka I Pun, for tett oppfølging og gode, detaljerte og konstruktive tilbakemeldinger.

Innholdsfortegnelse

ORDLISTE

.NET	Utviklerplatform fra Microsoft.
Azure SQL	Tjeneste fra Microsoft for SQL-relasjonsdatabaser i “skyen”.
Blazor	Rammeverk fra Microsoft for å utvikle webapplikasjoner.
C#	Programmeringsspråk fra Microsoft.
CRM-system	Customer Relationship Management-system. System for å holde oversikt over kunderelasjoner.
Entity Framework Core	Database-rammeverk fra Microsoft.
Frontend-rammeverk	Rammeverk for å utvikle interaktive webapplikasjoner.
HTTP	Hypertext Transfer Protocol. Protokoll for kommunikasjon mellom server og klient. HTTP er tilstandsløs.
Microsoft-sertifisert	Sertifisering som Microsoft Partner.
MVC	Model-View-Controller. Designmønster i utvikling av programvare.
MVVM	Model-View-ViewModel. Designmønster i utvikling av programvare.
React	Frontend-rammeverk fra Meta for å utvikle webapplikasjoner.

Scrum	Agil utviklingsmetodikk.
SignalR	Et bibliotek fra Microsoft som opprettholder tilkobling mellom server og klient ved hjelp av WebSockets
SQL	Structured Query Language. Språk for å kommunisere med relasjons-databaser.
Web API	Web Application Programming Interface. For å kommunisere på tvers av applikasjoner over internett.
Webapplikasjon	En nettside designet for interaksjon med bruker.
WebSockets	Protokoll for kommunikasjon mellom klient og server. Kommunikasjonstilkoblingen er vedvarende.

1 INNLEDNING

1.1 Kontekst

Oppdragsgiver for dette bachelorprosjektet er Stacc Escali AS, heretter referert til som Escali. Escali er et programvareselskap, eid av FinTech-konsernet Stacc AS, som utvikler løsninger innenfor finans-, treasury- og porteføljeforvaltning (Stacc Escali, u.å. a).

Kunder, typisk innen bank, forsikring, pensjon og investering, inngår avtaler med Escali og betaler en årlig sum for å få tilgang til deres tjenester. Escali bruker et CRM-system ("*Customer Relationship Management*", system for å holde oversikt over kunderelasjoner) for å lagre kundedata og avtaleinformasjon. Escalis fremtidige mål er å gjøre det slik at kunden selv kan administrere sine egne avtaler. Dette krever at avtaleinformasjon og lisenskoder flyttes fra CRM-systemet og ut i "skyen" (vedlegg 9.3). Å "flytte data til skyen" (*engelsk*: "Cloud migration") vil si å flytte data, applikasjoner og lignende til et miljø for skybasert databehandling (Semilof, Casey og Montgomery, 2021).

1.2 Prosjekteier

Escali er prosjekteier for bachelorprosjektet. Som nevnt i seksjon 1.1, er Escali et programvareselskap som utvikler løsninger innenfor finans-, treasury- og porteføljeforvaltning. Escali har vært aktiv i programvaremarkedet for finansielle løsninger siden 2002, og har i dag 18 ansatte og rundt 130 kunder. Bedriften er en Microsoft-sertifisert[1] partner (Microsoft, u.å. b), og drifter alle sine løsninger på Microsoft-plattformen.

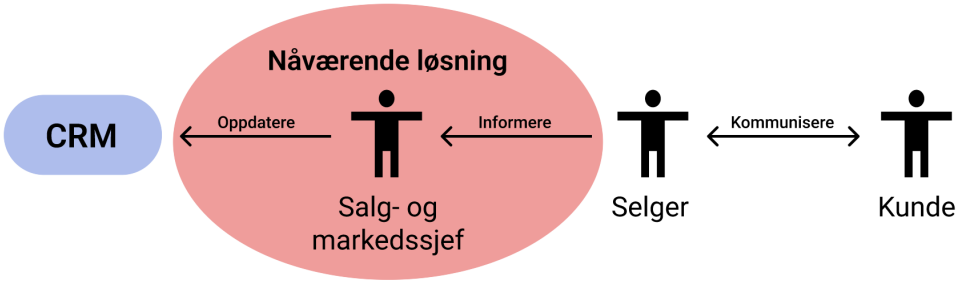
Escali tilbyr flere programvareløsninger, deriblant Escali Financials. Dette er en løsning for treasury- og porteføljeforvaltning, og gir kunder mulighet til å ha aksjer, fond, rentepapirer og lignende samlet på ett sted (Stacc Escali, u.å. a).

1.3 Motivasjon

Det er flere grunner til at Escali ønsker å flytte dataene sine fra CRM-systemet og ut i "skyen", blant annet for at kunder selv kan administrere egne avtaler.

[1] For å være Microsoft-sertifisert må de ansatte i bedriften bli kurset i Microsoft-teknologier.

Escalis nåværende løsning med kundedata og avtaler i et CRM-system, gjør det vanskelig å opprette og endre nye avtaler effektivt. Det er kun salg- og markedssjef i Escali som har tilgang til CRM-systemet. For å opprette eller endre en avtale, må kundens selger kontakte salg- og markedssjef, som igjen må gjøre endringer i CRM-systemet (figur 1.1). Dette kan effektiviseres gjennom å flytte kundedata og avtaleinformasjon til et nytt system i "skyen", og gi flere tilgang til denne dataen.



Figur 1.1: Nåværende løsning for oppretting og endring av avtaler

Datalagring hos Escali foregår i to ulike systemer. Kundedata og avtaleinformasjon, for både eksisterende og potensielt nye kunder, lagres i CRM-systemet, mens lisenskoder (koder som gir kunder tilgang til kjøpte produkter) lagres i et separat system hos Stacc. Det foreligger et ønske hos Escali om å samle all informasjon i ett system, som er, ifølge Escali, en god grunn til å fase ut CRM-systemet.

Det er også viktig å nevne at avtalebegrensninger, som i dag lagres som lisenskoder (figur 1.2), vil kunne lagres på et mer effektivt format i et nytt system i "skyen". Dette vil kunne gi mulighet for å gjøre en automatisk sjekk på om kunder overstiger begrensningene (f.eks. om en kunde overstiger antall transaksjoner) for produktet de betaler for, samt kunne varsle kunden om at produktet må oppgraderes til neste nivå. Dette er funksjonalitet Escali ønsker seg, men som ikke lar seg gjøre i nåværende løsning.

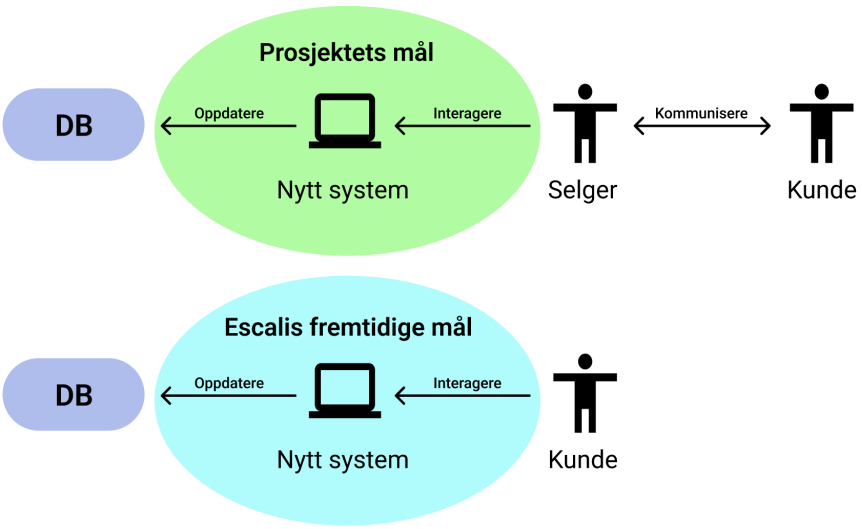
```
Qsa63-1sdCR-1sd51-RxsdE-1XgTQ
```

Figur 1.2: Eksempel på en lisenskode lagret i dagens Escali-system

1.4 Problembeskrivelse og mål

Escalis CRM-løsning for lagring og oppdatering av kundedata og avtaleinformasjon er ineffektiv, og krever unødvendig bruk av ressurser. De ønsker å flytte data ut i "skyen", til et nytt system. Dermed kan selgere, og i fremtiden kunder, administrere kundedata og avtaler (figur 1.3) uten å gå via salgs- og markedssjef. Et nytt system gir i tillegg mulighet for å legge til flere funksjoner knyttet til kunder og avtaler, som tidligere ikke har latt seg gjøre.

Dette bachelorprosjektet vil være et delmål mot Escalis endelige mål (se seksjon 1.1). I stedet for å la kunder kunne administrere avtalene sine direkte, vil prosjektets fokus være å gi selgere[2] tilgang til kundedata og tilhørende avtaler, samt administrere dette, i "skyen" (figur 1.3).



Figur 1.3: Prosjektets og Escalis fremtidige mål for oppretting og endring av avtaler

Problemstillingen for prosjektet ble dermed:

- Hvordan kan gruppen utvikle en løsning som gir selgere mulighet til å oppdatere kundedata og avtaleinformasjon, og dermed unngå unødvendig kommunikasjon via salg- og markedssjef?

Målet for prosjektet ble å utvikle et system som gir Escalis selgere en oversikt over produkter, kundedata og avtaleinformasjon, samt mulighet for å oppdatere og vedlikeholde dette.

[2] Kontaktpersoner til kunder

1.5 Oppbygging av rapporten

Rapporten er delt opp i 7 kapitler:

Kapittel to er en mer detaljert beskrivelse av prosjektet. Denne delen går i dybden av bakgrunn og initielle krav for prosjektet, men inneholder også relevant prosjektlitteratur.

Design av prosjektet, diskusjon av forskjellige fremgangsmåter og planlegging, blir dekket i det tredje kapittelet.

Fjerde kapittel inneholder detaljer om prosjektløsningen: Overordnet systemarkitektur, databasestruktur og oppbygging av webapplikasjon.

I kapittel fem presenteres resultat, evaluering av resultat, samt en beskrivelse av hvordan prosjektet har blitt gjennomført.

Kapittel seks inneholder diskusjon av prosjektgjennomføringen og tekniske løsninger.

Syvende kapittel konkluderer oppgaven og forteller om videre arbeid med resultatet.

2 PROSJEKTBESKRIVELSE

2.1 Praktisk bakgrunn

2.1.1 Escalis produktstruktur

For videre forståelse av rapporten, er det viktig å forstå Escalis produktstruktur. Escalis produkter er delt inn i moduler, hvor hver modul har flere modulnivå. Figur 2.1 viser et eksempel på et produkt med modulen "Stocks and Funds", og tilhørende modulnivåer. Modulnivåene har ulike begrensninger, og er prissatt ut i fra disse. Eksempelvis har første modulnivå på figur 2.1 maks forvaltningsvolum på 250 MNOK, og totalt 100 transaksjoner til disponering.

PRICELIST ESCALI FINANCIALS 2021 – NOK per MONTH							
Stocks and Funds							
250MNOK/100	500MNOK/250	1MRD/500	2,5MRD/1000	5MRD/2500	5MRD/5000	10MRD/Unlim.	Unlimited
950	1900	2850	3800	4750	5700	6650	7600

Figur 2.1 Eksempel på modul og modulnivåer for Escali Financials (fra vedlegg 9.3)

Escali bruker prislister til å prissette modulnivåer. Prislisten blir oppdatert manuelt en gang i året. Kunder kan inngå avtaler for moduler, og prisen blir satt basert på hvilket modulnivå kunden velger.

2.1.2 Initielle krav

Det initielle kravet til prosjektet var at det skulle utvikles en webapplikasjon med tilhørende database. Dette skulle være en prototype på fremtidig løsning. Minimumskravet ble å lage en webapplikasjon og database med følgende funksjonalitet:

- Innlegging og vedlikehold av kunder og kundedata.
- Innlegging og vedlikehold av produkter, moduler og modulnivåer.
- Innlegging og vedlikehold av prislister.
- Innlegging og vedlikehold av nye og eksisterende avtaler.

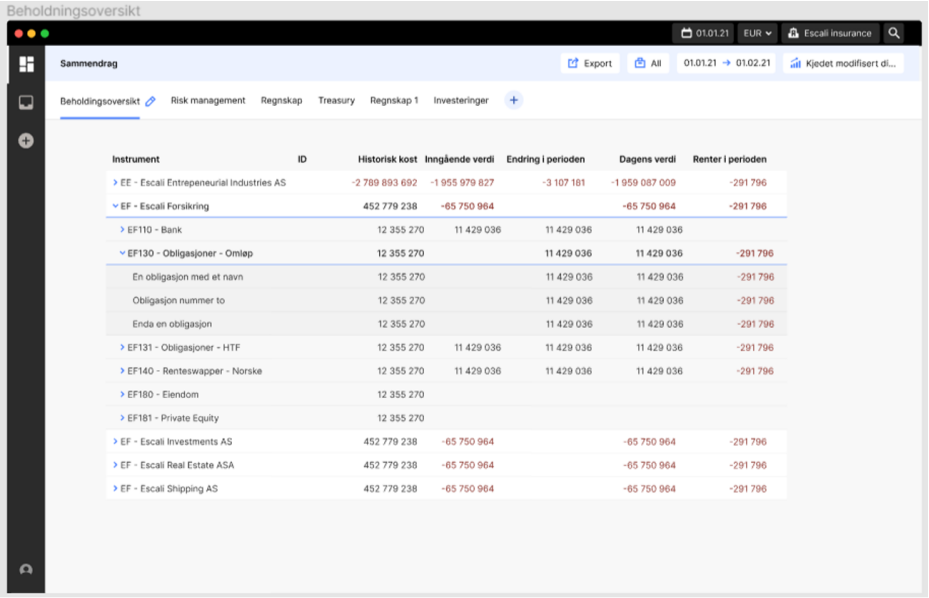
Videre følger funksjonalitet som Escali ønsket seg i webapplikasjonen, og som skulle implementeres om det ble tid til overs etter at minimumskravene var fullført:

- Automatisk sjekk på om kunders forbruk av Escalis tjenester overstiger begrensningen gitt i avtalen.
- Muligheter for å lage et tilbud på en avtale for en kunde.
- Lage fakturagrunnlag:
    - Årlig fakturering
    - Lisensøkninger
- Integrasjon mot regnskapssystem.
- Tilleggsinformasjon på kunder.
- Mobilvennlighet.

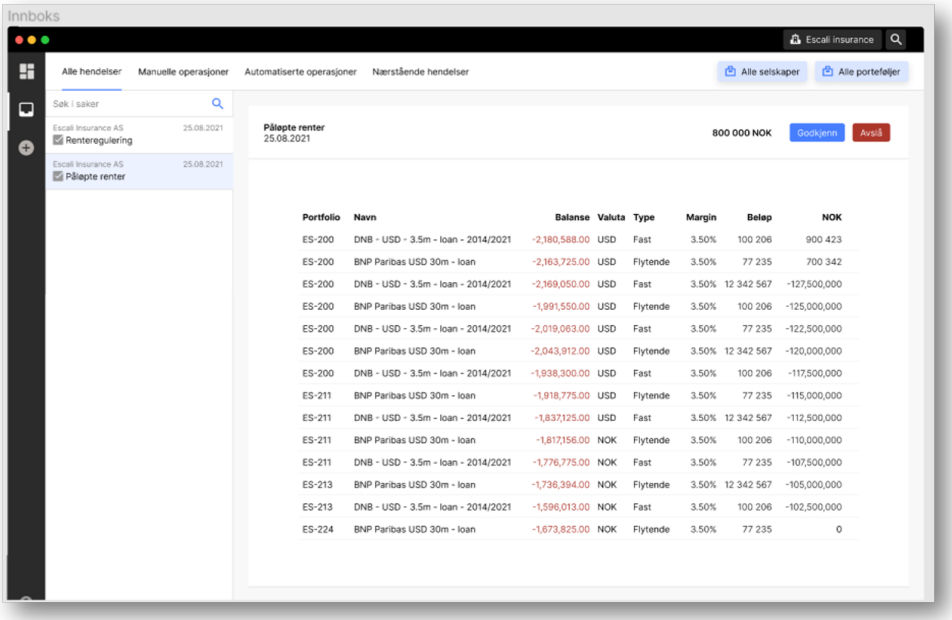
Ettersom Escali er en Microsoft-sertifisert partner ble det foreslått å bruke Microsoft-teknologier for utvikling.

2.1.3 Initiell løsnings-idé

Løsningen for å oppnå prosjektmålet ble å implementere systemet som en webapplikasjon med database. Retningslinjer og kravspesifikasjon er i stor grad blitt utarbeidet av produkteier. Forslag til databasemodell, basert på Escalis produktstruktur, ble gitt av Gerdt Sverre Vedeler, noe som var til stor hjelp for gruppen. Dette la et godt grunnlag for utviklingen senere i prosjektet.



Figur 2.2: Prototype på design



Figur 2.3: Prototype på design (2)

Skissene i figur 2.2 og 2.3 er prototyper fra et annet pågående prosjekt internt i Escali. Disse ga et godt utgangspunkt for hvordan designet på løsningen ville bli.

2.1.4 Tidligere arbeid

Essensen av prosjektet kan ikke klassifiseres som nyskapende i den forstand at noe lignende aldri har blitt gjort før. Det å opprette en database med informasjon, lage en kobling til en webapplikasjon, for så å presentere dataene til brukere, har blitt gjort mange ganger

tidligere. Det finnes utallige slike systemer, eksempelvis fra Facebook og Twitter. I Facebooks tilfelle brukes en arkitektur kalt TAO (The Associations and Objects) (Bronson, 2013). Kort fortalt bruker TAO en server/database og klient, med et kommunikasjonsledd i midten. Denne oppbyggingen ligner gruppens tiltenkte system.

2.2 Avgrensninger

For dette bachelorprosjektet ble det beregnet 4-5 måneder til planlegging og implementasjon av løsningen. Denne korte tidsperioden betraktet gruppen som en avgrensning for prosjektet. Om gruppen hadde fått for eksempel 12 måneder på prosjektet, ville det naturligvis vært mulig å implementere en webapplikasjon med mer funksjonalitet.

Når det kommer til valg av teknologier for webapplikasjonen, anbefalte Escali å bruke Microsoft-teknologier da de selv bruker dette i sine eksisterende systemer. Om gruppen derimot ønsket å bruke andre teknologier, var Escali åpen for dette. Valg av database var fastsatt av Escali på forhånd. Her skulle det brukes en relasjonell database (forklares i 2.4.1). Som følge av dette, mente gruppen at teknologivalg var en delvis avgrensning i prosjektet.

2.3 Ressurser

Det ble brukt flere ressurser i prosjektet. Veileder har bidratt med mye hjelp og tilbakemeldinger på rapport og støttedokumenter. På selve utviklingsprosjektet fikk gruppen god hjelp fra prosjekteier til utforming av database, design og felles forståelse av ønsket funksjonalitet. Ved problemer tilknyttet programmeringen fikk gruppen hjelp fra Escalis interne utviklere.

Escali har stilt med arbeidsplass til disposisjon fra start til slutt, noe gruppen har nytt godt av. Dette skapte et godt og trygt arbeidsmiljø, hvor terskelen for å diskutere og stille spørsmål ble lav.

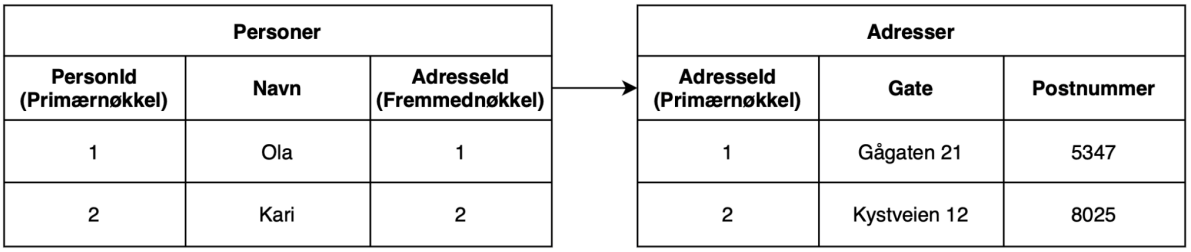
2.4 Litteratur om problemstillingen

2.4.1 Relasjonell database

I prosjektet skulle det implementeres en database for lagring og oppdatering av informasjon. En database er en samling av strukturert data, designet og utviklet for et spesifikt formål (Elmasri og Navathe, 2016, s. 4-5).

Det finnes flere typer databaser: En relasjonell database er en database med en samling av strukturert og relatert data (Elmasri og Navathe, 2016, s. 150). Et annet eksempel er NoSQL-databaser. Disse tillater lagring og manipulasjon av ustrukturert og semistrukturert data. Andre databasetyper er objektorienterte databaser og distribuerte databaser. En objektorientert database representerer informasjon i form av objekter. Distribuerte databaser består av to eller flere filer lokalisert på forskjellige steder, som kan være lagret på flere datamaskiner, eller spredt over forskjellige nettverk (Oracle, u.å. a).

Gruppen skulle bruke en relasjonell database. Dette er en effektiv og fleksibel måte å behandle strukturert informasjon (Oracle, u.å b). Data i relasjonelle databaser lagres i tabeller (også kalt entiteter), hvor en tabell kan inneholde et enormt antall rader med data. En rad i tabellen kan ha flere kolonner med ulike data og datatyper, og identifiseres med en primærnøkkel. Kolonner i tabellrader kan også inneholde referanser til andre tabellrader, og det er nettopp dette som gjør databasestrukturen relasjonell. (Oracle, u.å. b).



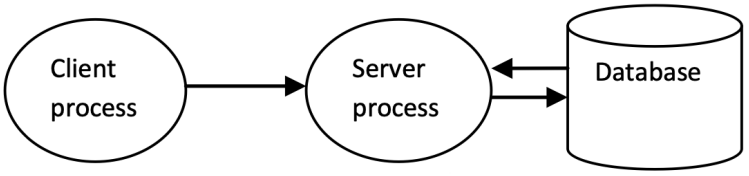
Figur 2.4: Enkelt eksempel på en relasjonell database-tabeller

Figur 2.4 viser et enkelt eksempel på tabeller i en relasjonell database. Person-tabellen har tre kolonner: PersonId, Navn og AdresseId. PersonId er primærnøkkelen som identifiserer person-rader i tabellen. AdresseId er en referanse, en fremmednøkkel, til Adresse-tabellen. Dvs. at første tabellrad i Person-tabellen, personen med navn Ola, har adressen med AdresseId lik 1, altså Gågaten 21.

For å aksessere data i en relasjonell database brukes SQL. SQL er et programmeringsspråk for å definere, hente og gjøre operasjoner på data i en relasjonell database (IBM, 2021a).

2.4.2 Kommunikasjon mellom server og klient

I webapplikasjoner er det vanlig å bruke en klient-server-arkitektur (Oluwatosin, 2014, s. 67). I prosjektet skulle det utvikles en webapplikasjon, og det er derfor relevant å vite noe om ulike måter klient og server kan kommunisere på.



Figur 2.5: Klient-server system (Oluwatosin, 2014, s. 67)

Klient-server-arkitekturen (figur 2.5) går tradisjonelt sett ut på at klienten sender forespørsler til server, som igjen håndterer forespørslene. En slik arkitektur kan bestå av mange klienter, men også flere servere. Serveren har ansvar for dataprosesseringen og sender eventuelle resultater tilbake til klienten. Denne arkitekturen har flere fordeler. Blant annet blir deling av ressurser mellom klienter enklere ettersom all data er lagret på serveren. I tillegg blir klient-applikasjonen mindre ressurskrevende å kjøre da server er ansvarlig for dataprosessering (Oluwatosin, 2014, s. 67).

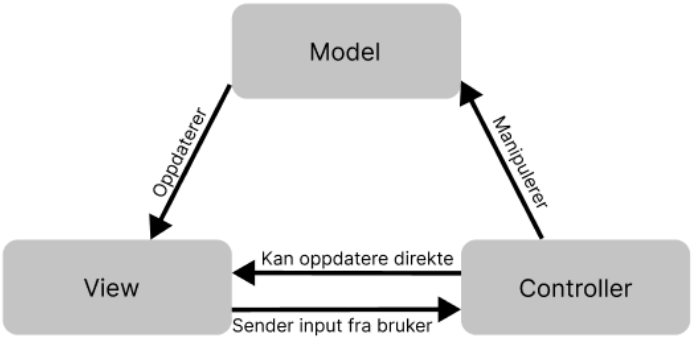
Det finnes flere ulike protokoller for kommunikasjon mellom server og klient. HTTP ("Hypertext Transfer Protocol") er et eksempel på en protokoll som tydelig implementerer klient-server-arkitekturen. Serveren mottar forespørsler fra klienten gjennom verb-metoder, som "Get" og "Post". Disse brukes henholdsvis når klienten skal hente og oppdatere data hos serveren. Etter at dataprosessering er fullført sender server et svar med relevant data, og statuskode for forespørselen, tilbake til klienten (MDN Web Docs, 2022a).

WebSockets er et annet eksempel på en protokoll for kommunikasjon mellom klient og server. I motsetning til HTTP, sørger WebSockets for at det alltid er en vedvarende tilkobling mellom klient og server. Klienten kan alltid sende data til serveren, og motsatt, uavhengig av om det sendes en forespørsel i forkant. Dette legger til rette for rask sanntidskommunikasjon (Internet Engineering Task Force, 2011).

2.4.3 Designmønster

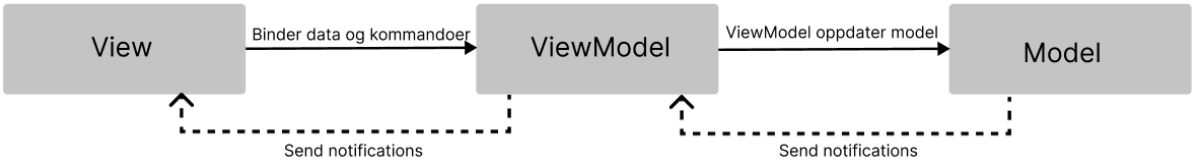
I programvareutvikling er et designmønster en generell, gjenbrukbar løsning for å løse et gjentakende problem. Løsningen beskriver komponentene som utgjør designet, forholdet mellom de, ansvarsområder og samarbeid. Konsekvensene av å bruke et designmønster er at det kan påvirke resultatet, og at det må tas hensyn til avveininger (E. Gamma m.fl., 1994, s. 2).

Det finnes flere designmønstre for webapplikasjoner, deriblant MVC (Model-View-Controller) og MVVM (Model-View-Viewmodel).



Figur 2.6: MVC

MVC er et designmønster som deler opp løsningen i tre deler: Model[3], View[4] og Controller[5] (figur 2.6). Mønsterets hovedfokus er oppdeling av programvarelogikk og brukergrensesnitt. Modellen definerer hvilke data applikasjonen skal inneholde. Hvis tilstanden til dataen endres, vil modellen varsle visningen slik at brukergrensesnittet oppdateres. Visningen definerer hvordan applikasjonens data skal presenteres. Kontrolleren inneholder logikken som oppdaterer modellen ved input fra brukere (MDN Web Docs, 2022b).



Figur 2.7: MVVM

[3] Model (*engelsk*) refereres til som modell
[4] View (*engelsk*) refereres til som visning
[5] Controller (*engelsk*) refereres til som kontroller

MVVM er et designmønster basert på MVC, og består i hovedsak av en modell, en visning og en visningsmodell[6] (figur 2.7). Her har modell og visning samme roller som i MVC. I MVVM har visningsmodellen ansvar for applikasjonslogikk, og henter data fra modellen som igjen brukes av visningen i brukergrensesnittet. Visningen kan også kalle på metoder i visningsmodellen for å oppdatere data i modellen, eksempelvis om brukeren klikker på en knapp i grensesnittet som oppretter ny data. Fordelen ved MVVM er at utviklere og designere kan jobbe mer individuelt og samtidig på komponenter. Det er også mulig for utviklere å lage enhetstester for visningsmodellen og modellen uten å bruke visningen (Microsoft, 2012).

[6] ViewModel (*engelsk*) refereres til som visningsmodell

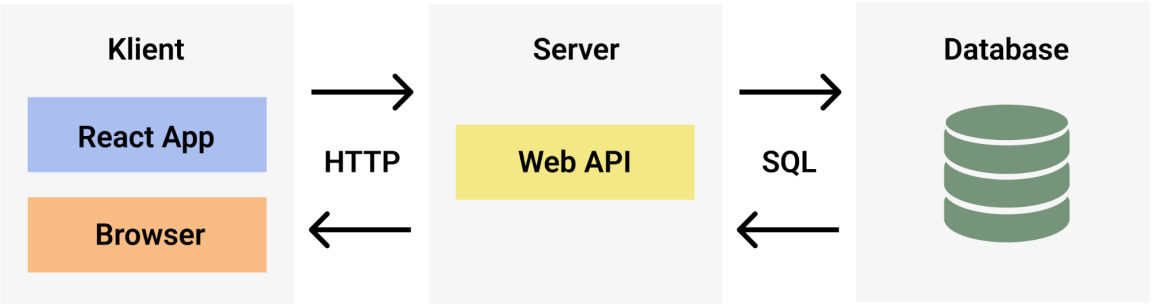
3 DESIGN AV PROSJEKTET

3.1 Forslag til løsning

3.1.1 React og Web API

En aktuell løsning til problemstillingen var å bruke klient-side React, et frontend-rammeverk fra Meta (Meta Platforms, u.å), for å utvikle webapplikasjonen. Fordelen ved å bruke klient-side React var at gruppen hadde tidligere erfaring med dette, noe som ville minsket behovet for å lære verktøy, og dermed frigjort tid til implementasjon. React er i tillegg et av de mest populære frontend-rammeverkene (State of JS, 2020), noe som fører til at det er mye dokumentasjon tilgjengelig.

Ettersom gruppen ville brukt React på klient-siden, og ikke server-side, hadde det vært behov for et Web API (Web Application Programming Interface) for kommunikasjon mellom klient og server (figur 3.1). Web APIet ville kjørt på serveren og kommunisert med klienten gjennom HTTP-protokollen. Å utvikle Web APIet ville tatt tid, samt ført til behov for mer vedlikehold.



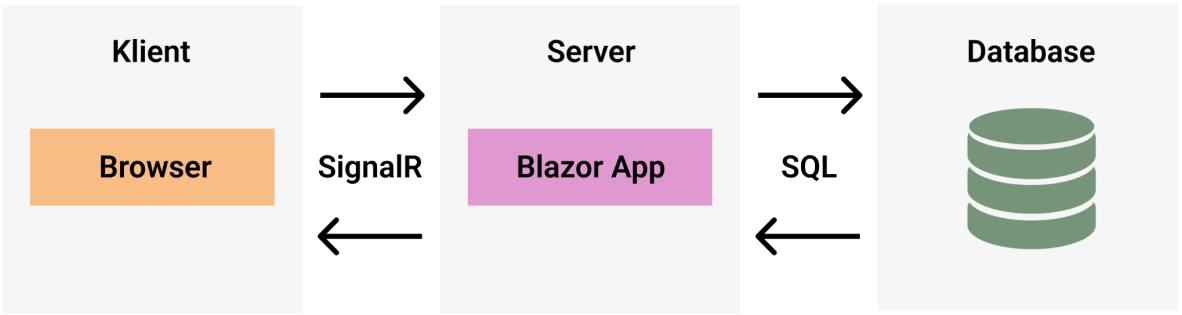
Figur 3.1: Systemarkitektur for løsning med React og Web API

3.1.2 Blazor Server-Side

Et annet alternativ til løsningen var å utvikle webapplikasjonen ved å bruke Blazor (Microsoft, 2022a). Blazor er et frontend-rammeverk fra Microsoft som bruker programmeringsspråket C#, og inngår i .NET-miljøet (utviklerplatform fra Microsoft) (Microsoft, 2022b). Dette alternativet ble foreslått og anbefalt av Escali ettersom de brukte verktøyet selv. Webapplikasjonen som skulle utvikles var en prototype som Escali ønsket å videreutvikle med Blazor. Det var derfor en stor fordel om webapplikasjonen allerede var

utviklet med Blazor. Escali kunne i tillegg bidra med hjelp om gruppen valgte Blazor-løsningen. For gruppen var ikke Blazor kjent fra før, og det krevde at gruppen lærte seg teknologien.

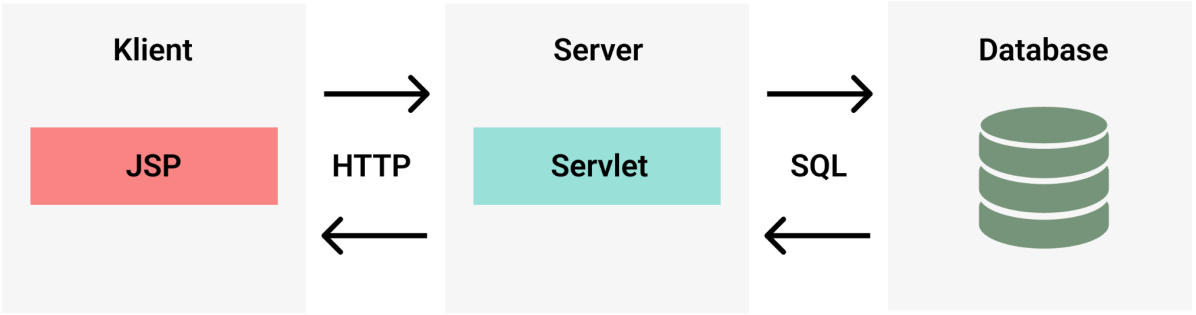
Blazor-apper kan kjøre på både klient-side og server-side (Microsoft, 2022a). Med Blazor var det naturlig, etter innspill fra Escali, å lage en applikasjon som kjører på server-siden (figur 3.2). Fordelen med dette var at database-tilkoblingen kunne skje direkte i webapplikasjonen og det ville ikke være behov for å utvikle et Web API i tillegg. Blazor server-applikasjoner kommuniser med klienten gjennom SignalR, et bibliotek fra Microsoft som opprettholder tilkobling mellom server og klient ved hjelp av WebSockets (Microsoft, 2022d).



Figur 3.2: Systemarkitektur for løsning med Blazor Server-Side

3.1.3 JSP

Et tredje alternativ for å utvikle webapplikasjonen, var Jakarta Server Pages (JSP) (Eclipse Foundation, u.å.).



Figur 3.3: Systemarkitektur for løsning med JSP

Teknologien tillater oppretting av både statiske og dynamiske nettsider, og kan brukes for å utvikle .jsp-sider (figur 3.3). JSP-sider er et tekstbaserte dokumenter som beskriver hvordan en forespørsel skal bli prosessert, og konstruerer deretter en response (Oracle, 2010). Gruppen har tilegnet seg kunnskap til denne teknologien gjennom studieløpet, som var

fordelen ved å velge JSP som utviklingsverktøy. Flere av lærerne på fakultetet har god kunnskap om JSP, som var gunstig dersom gruppen skulle støte på problemer underveis.

3.1.4 Diskusjon av alternativene

Alle de tre løsningsforslagene hadde sine fordeler og ulemper. Løsningen med React var veldig relevant ettersom gruppen hadde kunnskap om dette fra før, men den krevde et Web API i tillegg. En Blazor server-side-løsning krevde ikke Web API, noe gruppen så på som fordelaktig. Samtidig var Blazor nytt for gruppen og ville krevd en del opplæring. Blazor var mest hensiktsmessig for Escali ettersom de hadde erfaring med dette verktøyet og ønsket å bruke det for å videreutvikle webapplikasjonen.

Ulempen med JSP sammenlignet med f.eks. React, er at man får lite hjelp til eksempelvis validering av input, og henting og oppretting av data. Dette gjorde at gruppen anslo at programmeringsdelen for denne løsningen ville ta for lang tid, og ville være en begrensing for kvaliteten på webapplikasjonen.

3.2 Valgt løsning

Den valgte løsningen ble Blazor Server-Side. Her ble den avgjørende faktoren at gruppen så fordelene ved å benytte seg av teknologier som Escali allerede bruker. Denne avgjørelsen ble forsterket ved at det ville være enklere for Escali å fortsette på implementasjonen ved prosjektets slutt, om Blazor ble valgt fra start av. I tillegg hadde gruppen tilgang til kontorer sammen med andre utviklere fra bedriften, som åpnet muligheten til å få hjelp. At Blazor var nytt for gruppen, ble sett på som en mulighet for å heve teknologikunnskaper.

3.3 Valg av verktøy

Som utviklingsverktøy (*engelsk*: Integrated Development Environment, IDE) vurderte gruppen tre ulike løsninger: Microsoft Visual Studio, JetBrains Rider og Visual Studio Code. Microsoft Visual Studio er den offisielle IDEen fra Microsoft for å skrive kode i .NET-miljøet (Microsoft, 2022c). JetBrains Rider er en funksjonsrik IDE som fungerer like bra på alle plattformer (JetBrains, 2022). For Visual Studio Code krevdes det nedlastning av utvidelser for å kunne benytte seg av .NET-miljøet (Visual Studio Code, u.å).

Gruppen valgte å bruke både Microsoft Visual Studio og JetBrains Rider. Det ble ikke ansett å være tidkrevende for gruppen å bytte fra sine tidligere IDEer ettersom mye av funksjonaliteten var lik.

3.4 Prosjektmetodikk

3.4.1 Utviklingsmetodikk

Gruppen ønsket å jobbe iterativ med prosjektet, og valgte derfor å bruke deler av Scrum (Nes, 2019) som utviklingsmetodikk. Scrum er et rammeverk for agil utvikling, og legger til rette for å jobbe gjennom iterasjoner, kalt sprinter. I hver sprint fullfører man utvalgte oppgaver fra "backlogen", en liste over krav for funksjonalitet i produktet. Når man bruker Scrum er det vanlig med et møte på starten av dagen hvor teamet oppdaterer hverandre på hvilke arbeidsoppgaver som er fullført, og hva som må gjøres videre i den pågående sprinten.

Scrum legger til rette for at kunden, i prosjektets tilfelle Escali, kontinuerlig kan komme med tilbakemeldinger på produktet i ukentlige sprintmøter. Dette var en viktig grunn for å velge Scrum, spesielt da gruppen allerede hadde fast møte med Escali hver fredag. Dermed kunne Escali komme med tilbakemeldinger og forslag til forbedringer under disse møtene.

På grunn av ukentlige møter med Escali, var det naturlig for gruppen å velge sprinter som varte i en uke, fra mandag til fredag. Det ble lagt en plan om hva som skulle gjøres i hver sprint, noe som ga en god oversikt over implementasjons-prosessen.

3.4.2 Prosjektplan

Mens sprintene ga en oversikt over implementasjons-delen av prosjektet, ga gantt-diagrammet (vedlegg 9.1) den overordnede planen for helheten av prosjektet. Diagrammet viser de ulike deloppgavene i prosjektet, og antatt varighet for disse. I tillegg får man en oversikt over når de ulike sprintene skal jobbes med. Støttedokumentene, deriblant visjonsdokument, prosjekthåndbok og kravdokument, ble utarbeidet i flere iterasjoner.

3.4.3 Risikovurdering

Det ble gjort en tidlig risikoanalyse av prosjektet. Malen for denne analysen ble gitt av faglærere til gruppens bruk. Her ble det evaluert risiko ut i fra to kriterier, sannsynlighet for at en uønsket situasjon inntraff, og konsekvenser dette ville medføre. Alvorlighetsgraden til en risiko ble bestemt ut ifra disse kriteriene. Begge kriteriene blir delt inn i fem nivåer, 1-5, hvor 1 er lavest. Produktet av de to kriteriene bestemmer alvorlighetsgraden, og kan sees på figur 3.4.

Sannsynlighet						
	Svært Høy (5)	5	10	15	20	25
	Høy (4)	4	8	12	16	20
	Middels (3)	5	6	9	12	15
	Lav (2)	6	4	6	8	10
	Svært Lav (1)	1	2	3	4	5
		Svært Lav (1)	Lav (2)	Middels (3)	Høy (4)	Svært Høy (5)
	Konsekvens					

Figur 3.4: Utregning av alvorlighetsgrad i risikoanalyse

Risikovurdering er en prosess for kartlegging og vurdering av farer og hinder som kan oppstå i et prosjekt. Denne prosessen sørger for å holde risikonivået så lavt som mulig (Arbeidstilsynet, u.å.). Gruppen har gjennom flere iterasjoner kommet frem til en risikoanalyse (vedlegg 9.2). Her kom det frem at det var flere risikoer som hadde høy alvorlighetsgrad for prosjektet, og som var nødvendig å ha i bakhodet for videre prosjektplanlegging og utførelse.

Vedlegg 9.2 viser også tiltak for å unngå uønskede situasjoner, samt en foreslått løsning til hva som skulle gjøres dersom gruppen befant seg i en slik situasjon. Samtlige risikoer ble vurdert til å være håndterbare om de skulle inntreffe.

3.5 Evalueringsplan

Ettersom Scrum ble benyttet som utviklingsmetodikk, ville de ukentlige sprintmøtene bli en naturlig arena for evaluering av løsningen. Planen for møtene var å vise ukentlig fremgang, og få tilbakemeldinger på dette. Dette ville gjøre det mulig for prosjekteier å være med på utviklingen fra start til slutt, og dermed evaluere fremdrift kontinuerlig.

Gruppen ønsket i tillegg en mer omfattende sluttevaluering av produktet gjennom å bruke evalueringssamtaler. Disse samtalene skulle bestå av en gjennomgang av webapplikasjonen med interaksjon fra brukeren, og en samtale hvor brukeren ga tilbakemeldinger. Sluttevalueringen ville være en viktig del av den totale evalueringen av prosjektet.

4 DETALJERT LØSNING

4.1 Systemarkitektur

Systemets arkitektur består av tre lag som kommuniserer seg imellom: Database, server og klient (se figur 3.2 på side 20). Databasen er av typen Azure SQL[7] (Microsoft, u.å. a), og har ansvar for lagring av data i systemet. Data kan bli hentet, opprettet, oppdatert og slettet fra databasen ved bruk av SQL-spørringer.

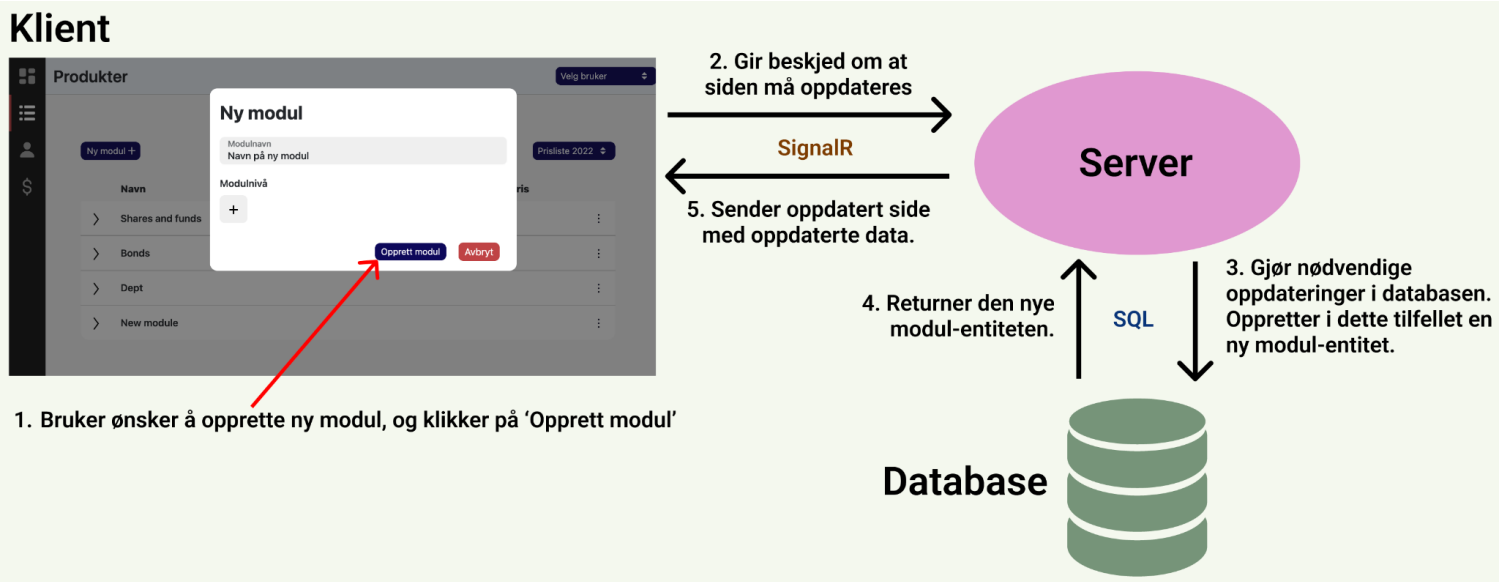
På serveren kjører det en Blazor-app som er ansvarlig for kommunikasjon med databasen, samt generering av webapplikasjonen. Serveren kobles til databasen ved hjelp av en "Connection string". Dette er en tekststreng som blant annet inneholder databasens adresse, brukernavn og passord. Figur 4.1 viser et eksempel på hvordan deler av en slik tekststreng kan se ut (ConnectionStrings.com, u.å.).

```
Server=tcp:myserver.database.windows.net,1433;Database=myDataBase;User
ID=mylogin@myserver;Password=myPassword;
```

Figur 4.1: Eksempel på connection string

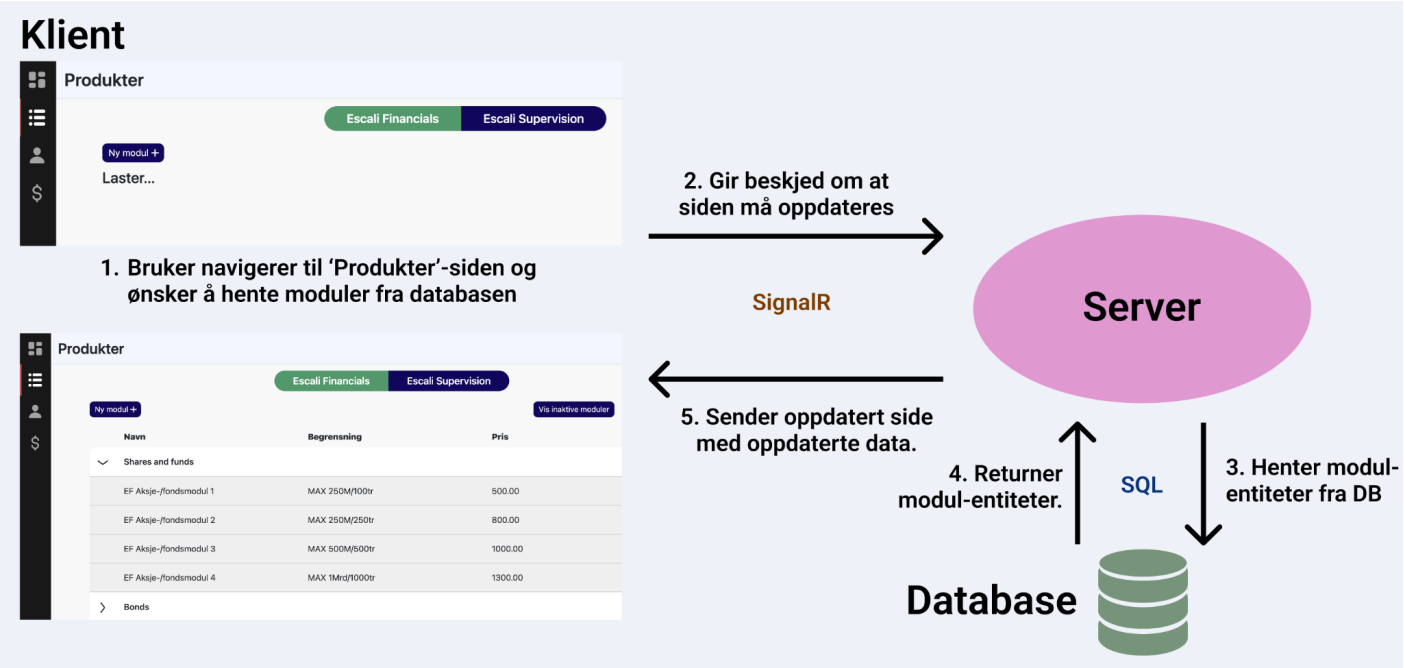
Når en klient ønsker å bruke webapplikasjonen, opprettes det en SignalR-forbindelse mellom server og klientens nettleser, og en ny brukersesjon er i gang. Serveren har ansvar for å generere webapplikasjonen når forbindelsen med klient er opprettet, for så å sende den videre til klienten. Dette innebærer å lage de grafiske komponentene, men også hente relevant data fra databasen. Klientens eneste ansvar er å sende forespørsler til serveren når den ønsker å laste inn nytt innhold. Om klienten f.eks. navigerer til en ny side i webapplikasjonen, eller oppdaterer data, håndterer serveren dette.

[7] Tjeneste fra Microsoft for SQL-relasjonsdatabaser i "skyen"



Figur 4.2: Eksempel på kommunikasjon mellom klient, server og database.

Figur 4.2 viser et eksempel på hvordan klient, server og database kommuniserer sammen når bruker ønsker å utføre en handling på klient-siden. I dette tilfellet ønsker bruker å opprette en ny modul-entitet. Når nødvendige informasjonsfelter er fylt ut, og knappen for å opprette ny modul klikkes, gir klienten beskjed til serveren om at en oppdatering av applikasjonen er nødvendig. Serveren utfører handlingene som er nødvendig for at modul-entiteten skal opprettes, og deretter oppdateres webapplikasjonens grafiske grensesnitt. Under denne prosessen oppretter serveren en ny modul-entitet i databasen med den informasjonen gitt av bruker på klient-siden. Modul-entiteten returneres til serveren, og kan dermed vises i det oppdaterte grensesnittet i webapplikasjonen.

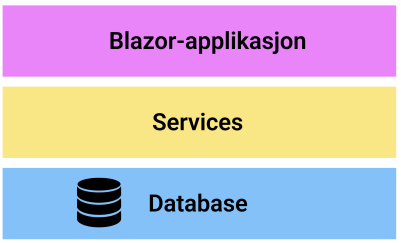


Figur 4.3: Eksempel på kommunikasjon under innlasting av Modul-entiteter på "Produkter"-siden.

Figur 4.3 er et eksempel på henting av data fra databasen. Når "Produkter"-siden lastes inn, ønsker klienten å vise valgt produkt og dets moduler. Serveren får beskjed om dette og henter alle modul-entitetene fra databasen. Deretter oppdateres brukergrensesnittet med ny data, og sender dette videre til klienten.

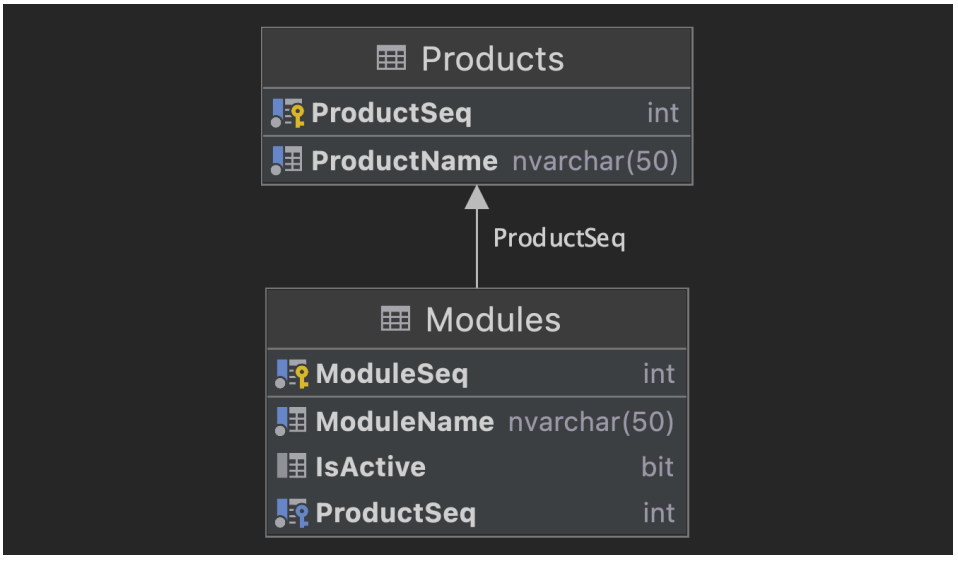
4.2 Databasestruktur

Kodebasen er delt opp i en tre-lags struktur bestående av databasemodeller, service-klasser med ansvar for å hente og oppdatere data i databasen og en Blazor-applikasjon. Blazor-applikasjonen bruker service-klassene til å vise data til brukeren (figur 4.4).



Figur 4.4: Kodebasens tre-lags struktur

Databasen er bygget opp som en relasjonsdatabase (se seksjon 2.4) med relasjonelle entiteter/tabeller. Figur 4.5 viser to entiteter, Produkter og Moduler. En produkt-entitet inneholder en nøkkel for å identifiseres, samt et navn. En modul har blant annet en nøkkel og et navn, i tillegg til en referanse til et produkt (ProductSeq). Dette gir grunnlag for at en modul har en relasjon til et produkt, mens et produkt kan ha relasjon til flere moduler.



Figur 4.5: Databasemodell - Produkter og Moduler.

Hver enkelt entitet i databasen er bygget ved hjelp av Microsofts database-rammeverk, Entity Framework Core (Microsoft, 2021). Entity Framework Core genererer database-entiteter og relasjoner ut ifra C#-objektklasser. Dette skjer ved å utføre en migrasjon (*eng: migration)*. En migrasjon kan kjøres flere ganger, selv etter at man har gjort

endringer i C#-klassene. Dette gjør det enkelt å gjøre endringer i database-entitetene, både under utvikling, og når databasen kjører i produksjon med reell data.

Figur 4.6 viser C#-objektklassen for modul-entiteten. Klassen brukes av Entity Framework Core for å lage modul-entiteten under migrasjon. Attributtene i Module-klassen gjenspeiler entitetstabellen for Module i figur 4.5.

```
public class Module
{

    [Key]
    public int ModuleSeq { get; set; }

    [Required]
    [MaxLength(50)]
    public string ModuleName { get; set; }

    public bool? IsActive { get; set; }

    [Required]
    [ForeignKey("Product")]
    public int ProductSeq { get; set; }
    public virtual Product Product { get; set; }

}
```

Figur 4.6: C#-klasse som representerer databasens modul-entitet

4.3 Service-klasser

For håndtering av database-operasjoner er det utviklet et eget bibliotek med serviceklasser. Biblioteket inneholder én serviceklasse per entitet, hvorav hver klasse inneholder metoder for å hente, opprette, endre og slette data for den aktuelle entiteten. Hver metode bruker Entity Framework Core for å koble til og oppdatere databasen. Figur 4.7 viser et utsnitt av modul-serviceklassen som inneholder to metoder: "*GetAllModules*" er en metode for å hente alle modul-instanser fra databasen, mens "*AddModule*" legger til en ny modul i databasen.

```
public async Task<List<Module>> GetAllModules()
{
    var res = await _db.Modules.ToListAsync();
    return res;
}

public async Task<Module> AddModule(Module module)
{
    // ...
    var res = _db.Modules.Add(module);
    await _db.SaveChangesAsync();
    return res.Entity;
}
```

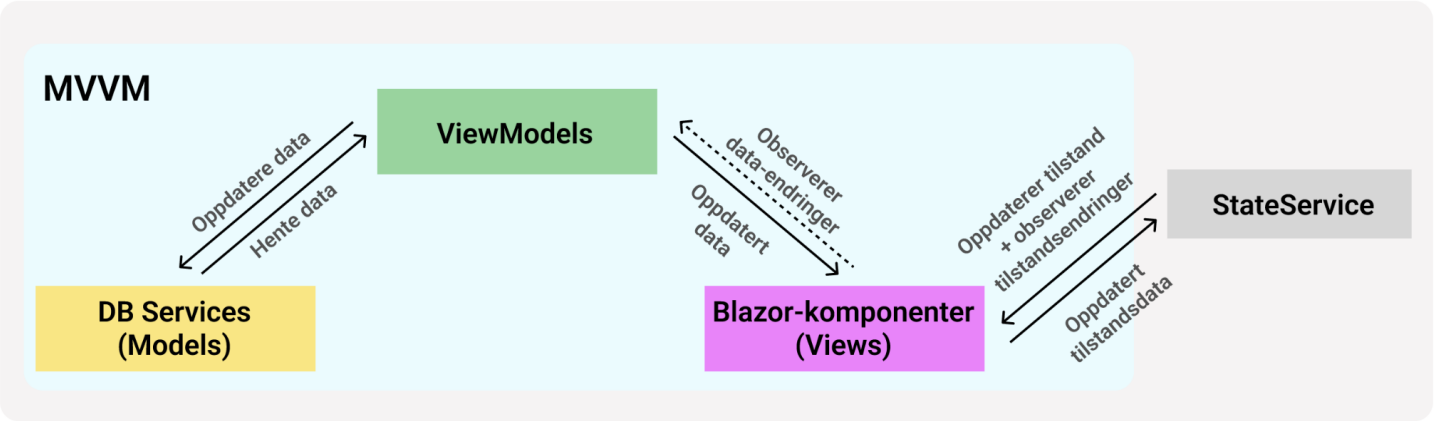
Figur 4.7: Utsnitt av "ModuleService"-klassen

Serveren bruker disse klassene når den ønsker å kommunisere med databasen. Med andre ord vil aldri serveren gjøre operasjoner i databasen uten at det skjer gjennom disse klassene. Dette medfører at klassene blir løsere koblet, som igjen gjør det lettere å gjøre endringer i serviceklassene på et senere tidspunkt. I tillegg blir database-tilgangen lettere å gjenbruke i andre deler av applikasjonen.

Service-klassene har ansvar for feilhåndtering under database-operasjoner, og at data som legges til i databasen er korrekt i henhold til reglene som er satt. Om f.eks. en ny modul skal legges til vha. "*AddModule",* og navnet på ny modul allerede finnes i databasen, vil service-klassen gi en feilmelding.

4.4 Blazor-applikasjon

Blazor-applikasjonen er den biten av systemarkitekturen som brukeren interagerer med, og er til enhver tid ansvarlig for å hente og vise data som brukeren ønsker. Som nevnt i seksjon 4.1, kjører Blazor-applikasjonen på serveren og sender den genererte webapplikasjonen tilbake til klienten ved hjelp av SignalR. Blazor-applikasjonen det øverste laget i arkitekturen, og bruker service-klassene til å hente data fra databasen (se figur 4.4, side 28).



Figur 4.8: Designmønster i utviklet Blazor-applikasjon

4.4.1 MVVM

Blazor-applikasjonens arkitektur baserer seg på MVVM-designmønsteret, og består i hovedsak av Models, Views og Viewmodels[8], som vist i figur 4.8. Modellene er i dette tilfellet database-services, som er ansvarlig for datalagring i databasen. Tilstandsdata ("StateService") blir forklart senere i kapittelet.

Videre har visningsmodellene ansvar for å bruke modellene til å hente data som skal vises i webapplikasjonen. Det er en én-til-én korrespondanse mellom visningsmodellene og database-services. Med andre ord er det én visningsmodell per database-entitet.
I det Blazor-applikasjonen lastes inn for første gang, henter visningsmodellene en liste av sine respektive entiteter fra databasen. All nødvendig data for videre bruk av webapplikasjonen lastes dermed inn. For eksempel vil modul-entitetens visningsmodell hente alle modul-instanser fra databasen. Disse dataene lagres midlertidig i visningsmodellene for hver brukersesjon, noe som reduserer antall database-forespørsler.

[8] Refereres til som henholdsvis modell, visning og visningsmodell.

Visninger har kun ansvar for å generere brukergrensesnittet som sluttbrukeren interagerer med, og henter nødvendige data fra visningsmodellene. Når visningsmodellene henter oppdaterte data, vil visningene bli varslet om dette, og endres i henhold til ny data. Dermed er visninger kun avhengig av visningsmodellene, og visningsmodellene kun avhengig av modellene (database-services), noe som viser et tydelig MVVM-mønster.

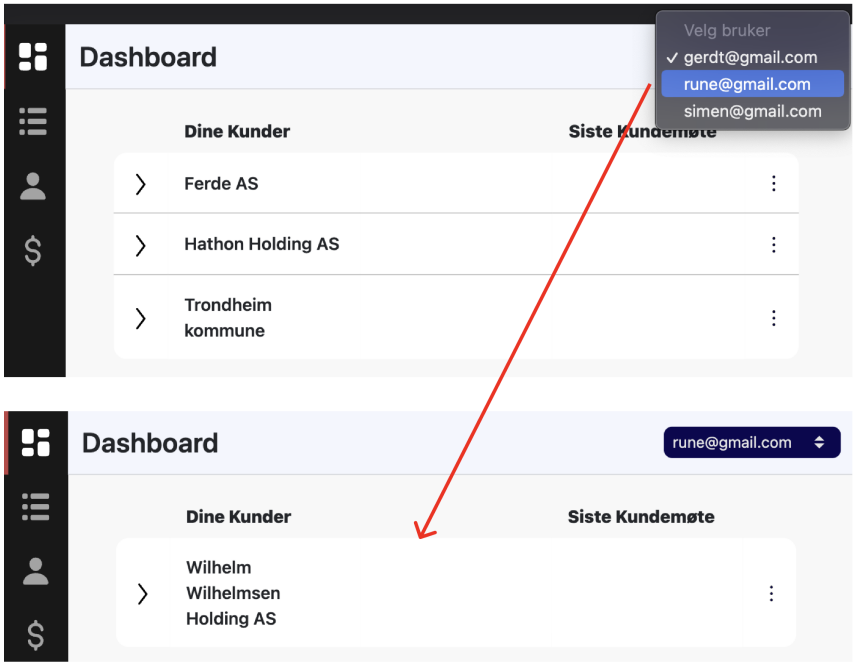
Blazor-komponenter legger grunnlag for alle visninger i applikasjonen. En komponent er en blanding av C#, HTML og CSS, hvor HTML og CSS bidrar til det statiske brukergrensesnittet, mens C# brukes for å gjøre siden dynamisk. Et eksempel på en del av en Blazor-komponent er vist på figur 4.9. Mesteparten av koden er HTML og CSS, med unntak av "*() => Modal.OpenModal()*". Dette er en C#-funksjon, og når HTML-knappen klikkes vil det åpnes et modal-popup-vindu.

```
<button class="btn btn-primary" @onclick="() ⇒ Modal.OpenModal()">
    Ny modul
    <img class="button-icon" src="Icons/add-icon.svg" alt="" />
</button>
```

Figur 4.9: Venstre: Kode for en knapp i en Blazor-komponent. Høyre: Visuell fremstilling av hvordan knappen ser ut i nettleser

4.4.2 Tilstandsdata

Tilstandsdata i Blazor-applikasjonen er lagret i "StateService", og er tilgjengelig for alle visnings-komponenter (figur 4.8) . Hver bruker-sesjon har sin unike "StateService" som genereres i det applikasjonen starter opp. Et eksempel på tilstandsdata er innlogget bruker. Hvilken bruker som er innlogget avgjør, i flere deler av applikasjonen, hvilken data som skal vises. Ettersom flere komponenter er avhengig av å vite hvilken bruker som er logget inn, er det hensiktsmessig å lagre denne dataen i et felles tilstandslager. Komponenter kan abonnere på oppdateringer av data i tilstandslageret, og får varsel når data oppdateres.



Figur 4.10: Endring av innlogget bruker vil oppdatere tilstandslager. Varsling blir sendt til tabellen som viser "Dine kunder"

Figur 4.10 viser et eksempel på endring av bruker[9]. Når valgt bruker endres, oppdateres også tilstandslageret i henhold til dette. Tilstandslageret varsler så alle komponentene som har abonnert på oppdateringer. I dette tilfellet blir tabellen "Dine kunder" oppdatert til å vise innlogget brukers kunder.

4.4.3 Mobilvennlighet

Under implementasjon av Blazor-applikasjonen var hovedfokuset at den skal brukes på desktop-klienter. I tillegg til desktop, ønsket også Escali at applikasjonen skulle være mobilvennlig. Ettersom over 40% av alle nettsider i Norge i 2021 ble lastet inn fra mobil-klienter (Statcounter, 2021), er det tydelig at webapplikasjoner må tilpasses mobil.

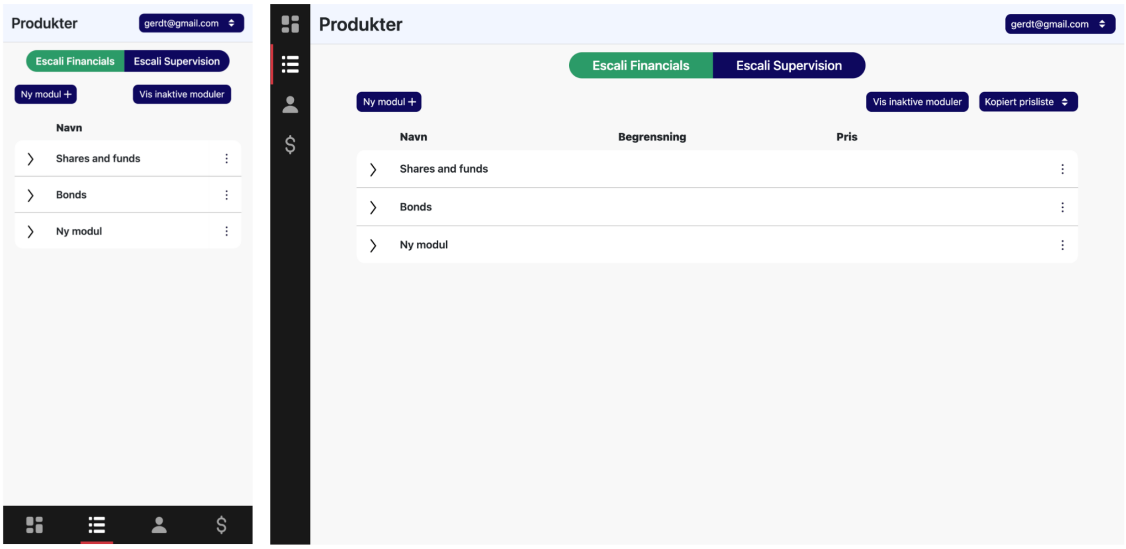
[9] Menyen for å endre bruker er prosjektets løsning. Ved fremtidig implementasjon vil denne byttes ut med en realistisk login-funksjonalitet. Tilstandslagring av innlogget bruker vil uansett være relevant i begge tilfeller.

```css
@media screen and (min-width: 576px) {
  .nav-bottom {
    display: none;
  }
}

@media screen and (max-width: 575px) {
  .sidebar {
    display: none;
  }
}
```

Figur 4.11: Media Queries for navigasjonsmeny

For å implementere en mobilvennlig tilpasning ble CSS Media Queries tatt i bruk (MDN Web Docs, 2022c). Media Queries legger til rette for å endre utseende på en webapplikasjon basert på skjermstørrelse. Figur 4.11 viser Media Queries for å bytte mellom navigasjonsmeny på bunnen, og på venstre side av skjermen. Max-width og min-width indikerer hvor stor skjermbredden må være (målt i piksler) for å bytte posisjon på navigasjonsmeny. Resultat av Media Queries illustreres i figur 4.12.



Figur 4.12: Venstre: Meny for mobil, vist på bunnen. Høyre: Meny for større skjerm.

5 RESULTATER

Evaluering er et kraftig verktøy for å utvikle og gjennomføre et prosjekt. Det kan være til hjelp i det noe går galt, eller være en veiledning når noe går bra. Uansett vil evaluering alltid være nødvendig for de som strever etter en bedre løsning. Ærlige samtaler, kontinuerlig forbedring og klarhet innad i gruppen er noen av flere grunner til at gruppen har valgt en kontinuerlig evalueringsprosess (Grosvenor, u.å.). Prosessen har gått ut på å ha jevnlig kontakt med brukere, veiledere og utviklere som har bidratt med evaluering av webapplikasjonen.

5.1 Evalueringsmetode

5.1.1 Sprintmøter

Gjennom prosjektet har gruppen brukt de ukentlige sprintmøtene som kontinuerlig evaluering. Hver sprint ble avsluttet med et møte med prosjekteier for å vise progresjon i arbeidet, rette opp feil eller misforståelser og stille spørsmål angående prosjektet.

Sprintmøtene kan oppsummeres i tre deler, hvor første del gikk ut på å vise hva gruppen hadde jobbet med i løpet av ukens sprint. Deretter var det tilbakemeldinger fra Escali, etterfulgt av diverse spørsmål som gruppen eller Escali hadde.

5.1.2 Evalueringssamtaler med webapplikasjon-interaksjon

En annen type evaluering gruppen benyttet seg av var evalueringssamtaler med ansatte i Escali. Her var intensjonen at en ansatt skulle bruke webapplikasjonen og teste funksjonalitet, navigasjon og flyt. Med tanke på at det er ansatte i Escali som skal bruke webapplikasjonen, er det viktig med tilbakemelding fra denne gruppen.

I seksjon 2.1.2 kan man finne oversikt over funksjonalitet som er et “minimumskrav”. Disse går i korte trekk på at det skal være mulig, gjennom webapplikasjonen, å legge til og vedlikeholde data i databasen. Med bakgrunn i disse kravene har gruppen diskutert, sammen med ansatte, hva som kan forbedres og hvilke minimumskrav som har blitt oppfylt.

5.2 Evalueringsresultat

5.2.1 Sprintmøter

Til tross for en god planleggingsfase med Escali, har det vært nødvendig med små justeringer underveis i prosessen. Sprintmøtene har, gjennom hele prosjektet, bidratt til gode tilbakemeldinger til gruppen, som igjen har forbedret sluttproduktet.

5.2.2 Evalueringssamtaler med webapplikasjon-interaksjon

Som en grundig sluttevaluering av produktet, gjennomførte gruppen evalueringssamtaler med to ansatte i Escali: Gruppens kontaktperson i Escali og salgs- og markedssjef. Selv om gruppen kun har hatt to evalueringssamtaler, mener gruppen samtalene er veldig relevante,og av et tilstrekkelig antall. Prosjektet har hatt få involverte fra Escali, og i første omgang er det et fåtall fra Escali som skal bruke webapplikasjonen.
Det er i hovedsak Gerdt Vedeler, hovedkontaktperson i Escali, som har vært med å definere systemkrav, og har dermed godt grunnlag for å gi tilbakemeldinger om applikasjonen. I tillegg vil salgs- og markedssjef i Escali være en fremtredende bruker av sluttproduktet, og det er derfor også viktig med en evaluering herfra.

Evalueringssamtalene ga grundige sluttevalueringer av produktet, og er viktig for videre implementasjon. Det første som ble vurdert, var til hvilken grad de ulike minimumskravene for applikasjonen var oppfylt. Evalueringen viste at samtlige minimumskrav, bortsett fra prisliste-funksjonalitet, i stor grad var oppfylt. Innlegging og vedlikehold av prislister er ikke ferdig implementert, så dette evalueringsresultatet var som forventet.

Videre ble det stilt spørsmål vedrørende brukergrensesnittet: Er knapper, design og navigasjon intuitivt å forstå? Konklusjonen ble at systemet virker veldig intuitivt, og ingen hadde problemer med å sette seg inn i løsningen uten opplæring.
Evalueringen viste også at produktet stemte godt overens med de ansatte i Escali sine forventninger ved prosjektstart. Samtidig kom det kommentarer om at “det fortsatt er litt knirk her og der”, men at det er forventet, og kan løses med enkel testing og fiksing. Ellers viste testingen ingen feil i den konseptuelle løsningen.

Evalueringen ga også forslag til funksjonalitet som i utgangspunktet ikke er planlagt. Eksempelvis ønsket salgs- og markedssjef funksjonalitet for å avslutte kundeforhold, og hvor alle ekstierende avtaler hos den aktuelle kunden blir avsluttet i samme prosess. Det kom også forslag om statistikk som i fremtiden vil være aktuelt å ha på "Dashboard"-siden. Noe av dette krever å legge til attributter til entiteter i databasen, som igjen, om ønskelig, er mulig å implementere i fremtiden.

Totalvurderingen av produktet var stort sett positiv, med noen kommentarer på forbedringer. Gruppen var klar over at noe funksjonalitet manglet, så mange av de konstruktive tilbakemeldingene var forventet.

5.3 Prosjektresultat

Det overordnede resultatet av prosjektet er et system med en webapplikasjon som kan utføre de fleste forespurte operasjoner, altså minimumskravene for prosjektet. Systemet består av en klient-del og en server-del som gruppen har konstruert fra bunn. Webapplikasjonen er en prototype på en løsning som, etter videre utvikling, kan gjøre arbeidet lettere for selgerne i Escali. Operasjoner som tidligere har gått via salgs- og markedssjef (se figur 1.1, side 8), vil i fremtiden kunne håndteres via webapplikasjonen.

Alt i alt tror gruppen at det har blitt levert en god løsning, som fungerer, og har fokus på gjenbrukbar kode. Dette har lagt et godt grunnlag for videre implementasjon og utvikling. Diverse funksjonalitet mangler fra minimumskravene, og ville blitt implementert dersom gruppen hadde hatt mer tid disponibelt.

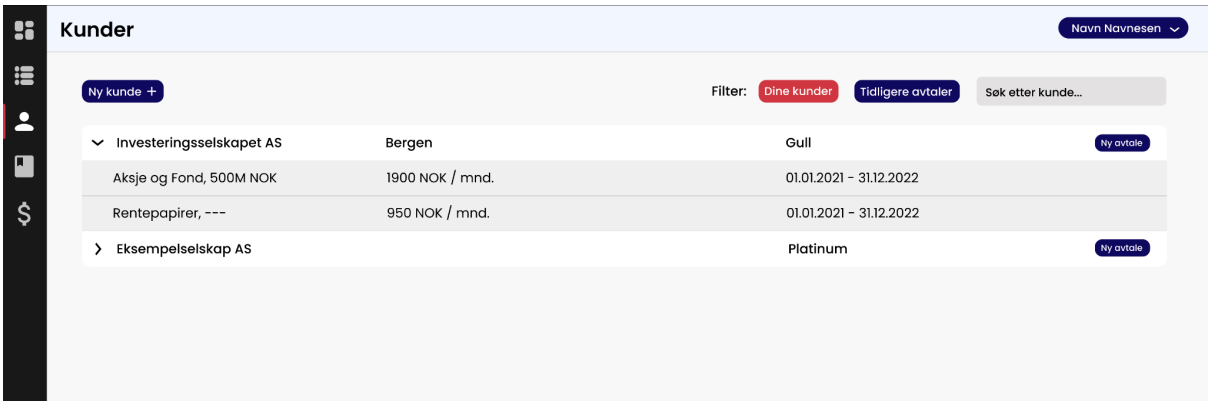
5.4 Prosjektgjennomføring

I starten av prosjektet ble det satt av mye tid til definering og diskusjon av oppgaven. Gruppen hadde hyppige møter med Escali for å få et godt bilde av oppgavebeskrivelsen, samt oppklaring av eventuelle feiltolkninger. I tillegg var det lav terskel for å stille spørsmål utenom møtene. Utfallet av dette ble at gruppen og Escali, fra start av, fikk en god oversikt over et felles mål.

Et godt utgangspunkt for databasestrukturen, definert av Escali, la til rette for en god start på planleggingen av prosjektet. Gruppen brukte, sammen med Escali, mye tid på å utarbeide

og forbedre databasestrukturen, og fikk gjennom dette en felles forståelse av hvordan den skulle se ut.

Etter at databasemodellen var godt definert, startet gruppen å lage en prototype av webapplikasjonen ved å bruke Figma (Figma, 2022). Prototypen ble detaljert og ga et godt utgangspunkt for designet, i tillegg til at den viste hvordan interaksjon med applikasjonen skulle fungere i praksis (eksempel av skisse vises i figur 5.1). Godt definerte systemkrav, sammen med en gjennomarbeidet prototype og databasemodell, førte til at gruppen hadde et godt grunnlag for å starte med implementasjon av database og webapplikasjon.



Figur 5.1: Figma-skisse av Kunde-side

Under implementasjonen var planen å jobbe agilt. Her benyttet gruppen seg av sprinter på en uke, hadde ukentlige sprintmøter med Escali og daglige scrummøter innad i gruppen. Tidlig i fasen ble sprintene fullført, men det viste seg raskt vanskeligheter med dette da gruppen noen ganger ikke ble ferdig med sprint-oppgavene for en uke, og måtte utsette disse til den påfølgende uken.

Gruppen hadde daglige scrummøter hvor medlemmene ga en forklaring på hva som var gjort og hvilke oppgaver som skulle jobbes videre med i løpet av dagen.
Hver sprint ble avsluttet med et møte med oppdragsgiver. Sprintmøtene har vært et godt verktøy for tilbakemeldinger på løsningen, og viktig for å bekrefte at gruppen og oppdragsgiver har vært samstemte.

Gruppen har blitt eksponert for utfordringer i form av tidsklemmer og tekniske problemer, og uforutsette oppgaver kom som et resultat av dette. Som regel har løsningen vært å tilpasse seg dette, og innhente “mistet” tid ved å benytte seg av hjelp fra utviklere i Escali.

Ved tekniske problemer som var mer avanserte, ble det benyttet parprogrammering. Dette fungerte meget bra, og gruppen kom frem til gode løsninger gjennom slikt samarbeid.

Gruppen mener prosessen har vært gjennomgående god, og at oppsettet i stor grad har fungert som tiltenkt. Det som ikke har fungert like bra er estimering av tid per sprint-oppgave, og oppdatering av disse oppgavene. Dette gjenspeiles i timelisten (vedlegg 9.5), hvor det har vært varierende timetall per uke.

6 DISKUSJON

6.1 Utviklingsprosess

Sprinter er i utgangspunktet veldig nyttige, men dersom oppsettet av sprintene blir uoversiktlig, kan det resultere i utydelige arbeidsoppgaver. I starten av implementasjonen hadde gruppen god oversikt, fulgte sprintene og hadde en oppdatert fremdriftsplan. Ettersom tiden gikk, ble sprintene til mindre grad fulgt opp og ikke oppdatert når uforutsette tekniske problemer oppstod. Dette førte til at de påfølgende sprintene ikke ble planlagt på forhånd. Da ikke oppgavene ble planlagt på forhånd, endte gruppen opp med å fordele oppgavene seg imellom som større, overordnede oppgaver. En mer effektiv løsning kunne vært å vedlikeholde sprintene, og definere mindre, mer spesifikke oppgaver. For eksempel ble oppgaver gitt som: “Implementer hele Produkt-siden”. Her kunne man delt opp i mindre oppgaver som: “Implementer filtreringsknapper på siden” og “Implementere tabell-visning på siden”.

Ukentlige møter har vært en arena hvor gruppen har presentert ukentlig fremgang, og fått tilbakemelding som har ført til småjusteringer. Ingen nye krav til systemet har blitt fremlagt på disse møtene ettersom de initielle kravene var så godt formulert av oppdragsgiver. Dette, i tillegg til god planlegging, førte til at implementasjonen ble jobbet med som én iterasjon med mindre justeringer underveis, i stedet for flere iterasjoner, som er vanlig i agil utvikling. Ved å ha hatt godt definerte krav fra starten, har det ikke vært behov for mye fokus på planlegging underveis. Dette førte til at det ble mer tid til utvikling, noe som var en fordel.

Samarbeidet mellom gruppen og Escali har fungert veldig bra. Sett i retrospekt ville gruppen ønsket seg et tettere samarbeid med Escali når det kommer til kodebasen. Det hadde vært en fordel om en eller flere utviklere i Escali hadde vært delaktig med å sette opp programmeringsprosjektene og jobbet tett på gruppen i startfasen. Da kunne strukturen for applikasjonen blitt definert på en god måte allerede fra starten av, og gruppen ville brukt mindre tid på omstruktureringer. Samarbeid på kodebasen ville også resultert i at utviklerne lettere kunne satt seg inn i problemer gruppen hadde. Med tanke på at ressursene var tilgjengelig, burde gruppen tatt initiativ for å få til et slikt kodesamarbeid.

Planleggingsfasen er noe gruppen er veldig fornøyd med. Det at det ble satt av god tid for å diskutere design og krav, har vært med på å holde gruppen på en stødig kurs hele veien. Ved å bruke god tid på diskusjon av design og krav, ble det ikke behov for store justeringer i løsningen etter tilbakemeldinger fra sprintmøter. Dersom prosjektet skulle blitt gjennomført på nytt, er planleggingen noe som ville fått like mye fokus.

6.2 Teknisk løsning

Da gruppen implementerte databasen, var det utelukkende fokus på dette. Gruppen skrev tester på database-servicene, og alt så tilsynelatende ut til å fungere. Da arbeidet med databasen var over, startet gruppen på webapplikasjonen. Det ble i denne overgangen oppdaget flere feil i databasen. Et eksempel var oppdatering av entiteter gjennom Entity Framework Core i Blazor, noe gruppen brukte lang tid på å løse.
I etterkant ser gruppen at det kunne vært mer tidseffektivt å starte implementasjonen av webapplikasjonen på et tidligere tidspunkt i prosjektet. Ettersom flere av de tekniske problemene til gruppen var knyttet til kommunikasjon mellom database og webapplikasjon, kunne det vært en fordel å jobbe mer parallelt med disse. For eksempel kunne man implementert noe funksjonalitet på database-siden, deretter testet fortløpende i webapplikasjonen, og gjennom dette oppdaget relevante problemer på et tidligere tidspunkt.

I seksjon tre diskuteres ulike utviklingsverktøy som er relevante, og som kunne brukes til å løse problemstillingen. Escalis anbefaling av Blazor kan ha vært med på å tåkelegge det faktum at gruppen allerede hadde kunnskap om de to andre forslagene. I et tidsbegrenset prosjekt slik som en bachelor er, kunne gruppen med fordel valgt en annen løsning. Samtidig er det er vanskelig å si om løsningen ville blitt bedre med et annet valg, men gruppen tror derimot at implementasjon av funksjonalitet kunne gått raskere og mer knirkefritt. Samlet sett er det rimelig å anta at gruppen kunne implementert mer funksjonalitet dersom valget hadde på for eksempel React, tatt tidsbegrensningen og kunnskap om verktøyet i betraktning. Læringsutbyttet ved å bruke Blazor er derimot større, og gruppen tar med seg mye god erfaring videre.

Blazor var, som nevnt, en ny teknologi for gruppen, og dette prosjektet var første gang det ble tatt i bruk av gruppemedlemmene. Som følge av dette har gruppen naturligvis lært mye

nytt på veien. Strukturen til webapplikasjonen har endret seg drastisk fra start til slutt, da gruppen gjennom prosjektet hele tiden har lært nye måter å løse utfordringer på. Applikasjonen har blitt omstrukturert flere ganger. Disse omstruktureringene har tatt tid, men også resultert i en mer oversiktlig kodebase, som er lettere å jobbe videre med.

6.3 Problemstilling

Sluttproduktet ble, som nevnt i seksjon 5.3, en webapplikasjon med de fleste minimumskrav oppfylt. Minimumskravene som ikke er oppfylt, er vedlikehold av prislister og vedlikehold av avtaler. På disse to punktene mangler det noe funksjonalitet for å kunne si at de er komplette. Gruppen vet hvordan dette skulle blitt løst, men tidsrammen for prosjektet var ikke tilstrekkelig.

Evalueringsresultatene var veldig positive. Brukertestingen viste at løsningen svarte til oppdragsgivers forventninger og at alle minimumskrav, bortsett fra krav nevnt ovenfor, var oppfylt.

Gruppen har utviklet en løsning som oppfyller de fleste minimumskravene gitt av Escali, i tillegg til å ha fått positive tilbakemeldinger fra Escali under evaluering. Som følge av dette mener gruppen at løsningen er et godt svar på problemstillingen beskrevet i seksjon 1.4.

7 KONKLUSJON OG VIDERE ARBEID

Målet for prosjektet var å utvikle et system som gir Escalis selgere en oversikt over produkter, kundedata og avtaleinformasjon, samt mulighet for å oppdatere og vedlikeholde dette. For å beregne hvorvidt målet ble nådd, tok gruppen utgangspunkt i minimumskravene for prosjektet. Gruppen mente at samtlige minimumskrav måtte være tilfredsstilt for å ha nådd målet.

Ettersom ikke alle minimumskrav ble ferdig implementert, er ikke målet for prosjektet nådd. Som diskutert tidligere, er det flere årsaker til hvorfor gruppen ikke kom helt i mål. Det var en del teknisk trøbbel som tok tid under implementering av løsningen, noe som er vanskelig å forutse. Samtidig var Blazor nytt for gruppen, og det ble derfor naturligvis brukt mye tid på læring underveis. Om gruppen hadde hatt mer kunnskap om Blazor før prosjektstart, er sannsynligheten for at målet ville blitt nådd, betydelig høyere.
Det er rimelig å anslå at to-tre uker ekstra, med fullt fokus på implementasjon, ville gitt en løsning som hadde nådd minimumskravene, med forbehold om at det ikke ville oppstå noen teknologiske utfordringer.

Det er viktig å nevne at selv om målet ikke ble nådd i full grad, var det ikke langt unna. Resultatet av prosjektet er en webapplikasjon med nesten alle minimumskrav oppfylt. Gruppen er alt i alt fornøyd med resultatet, i tillegg til prosessen frem mot dette. Det faktum at oppdragsgiver også er fornøyd, er noe gruppen verdsetter.

Webapplikasjonen som gruppen har utviklet er en prototype på et nytt system, og en fremtidig løsning hos Escali. Planen er at applikasjonen skal videreutvikles og senere bli tatt i bruk. Det er ønske om enda mer funksjonalitet i applikasjonen, noe Escali vil fortsette å implementere. Under implementasjon i dette prosjektet ble det f.eks. ikke implementert funksjonalitet for kommunikasjon med Escalis systemer, noe som er nødvendig for at systemets skal fungere som ønsket.
Det er ønskelig for Escali å kunne overføre data mellom eksisterende og nytt system. Dette er helt nødvendig da det nye systemet skal lagre alle lisenser, som igjen skal brukes til autorisering for kunder i eksisterende Escali-applikasjoner.
Login-funksjonalitet er heller ikke på plass i webapplikasjonen, og må implementeres før

systemet kan bli tatt i bruk. Escali har allerede kode for autentisering, og vil kunne implementere mye av den samme funksjonaliteten i webapplikasjonen.

Gruppen mener at webapplikasjonen gir Escali et godt utgangspunkt for videre utvikling, av flere grunner: Blazor er et foretrukket frontend-rammeverk hos Escali, og de slipper dermed å sette seg inn i nye teknologier. Gruppen mener at applikasjonen er godt strukturert vha. designmønsteret MVVM, og at det på grunn av dette er relativt enkelt å legge til ny funksjonalitet. I tillegg er mye av koden gjenbrukbar. Som følge av dette kan eksisterende kode brukes på nytt under videre implementasjon av systemet.

Under prosjektet har gruppen tilegnet seg mye ny kunnskap. Blant annet har gruppen lært nye teknologier, deriblant C#, Entity Framework Core og Blazor. Gruppen har også lært mye når det kommer til prosjektarbeid, hvor planlegging, problemløsing og samarbeid har stått i fokus. Å planlegge godt er viktig, men også vanskelig, spesielt for programvareutvikling, ettersom det alltid er en risiko for at uforutsette hendelser inntreffer. Prosjektet har gitt mye erfaring i problemløsing, både i planleggingsfasen og under utvikling, i tillegg til samarbeid og kommunikasjon for å oppnå en felles visjon og et felles mål.

8 REFERANSER

Arbeidstilsynet (u.å.) *Risikovurdering*. Tilgjengelig fra: https://www.arbeidstilsynet.no/hms/risikovurdering/ (Hentet: 24.02.22)

Bronson, Nathan et al (2013) *TAO: Facebook's Distributed Data Store for the Social Graph.* Tilgjengelig fra: https://www.usenix.org/system/files/conference/atc13/atc13-bronson.pdf (Hentet: 08.03.22)

ConnectionString.com (u.å.) *Azure SQL Database connection strings.* Tilgjengelig fra: https://www.connectionstrings.com/azure-sql-database/ (Hentet 19.04.22)

Eclipse Foundation (u.å.) *Jakarta Server Pages*. Tilgjengelig fra: https://projects.eclipse.org/projects/ee4j.jsp (Hentet 04.03.22)

Elmasri, R., Navathe, S.B. (2016) *Fundamentals of Database Systems*. 2. utg. USA: Pearson

Figma (2022) *Design features*. Tilgjengelig fra: https://www.figma.com/design/ (Hentet: 18.05.22)

Gamma, E. mfl. (1994) *Design Patterns: Elements of Reusable Object-Oriented Software* Indianapolis: Addison-Wesley Tilgjengelig fra: http://www.javier8a.com/itc/bd1/articulo.pdf (Hentet: 29.04.22)

Grosvenor (u.å.) *6 reasons why evaluation is a great opportunity for program managers.* Tilgjengelig fra: https://www.grosvenor.com.au/insights-resources/public-sector-advisory/6-reasons-why-evaluation-is-a-great-opportunity-for-program-managers/ (Hentet: 05.05.22)

IBM (2021a) *What is SQL?* Tilgjengelig fra: https://www.ibm.com/docs/en/developer-for-zos/14.0.0?topic=support-what-is-sql (Hentet: 09.03.22)

Internet Engineering Task Force (2011) *The WebSocket Protocol.* Tilgjengelig fra: https://datatracker.ietf.org/doc/html/rfc6455 (Hentet: 29.04.22)

JetBrains (2022) *Rider: The Cross-Platform .NET IDE from JetBrains.* Tilgjengelig fra: https://www.jetbrains.com/rider/ (Hentet 08.03.22)

MDN Web Docs (2022a) *An overview of HTTP.* Tilgjengelig fra: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview (Hentet: 29.04.22)

MDN Web Docs (2022b) *MVC.* Tilgjengelig fra: https://developer.mozilla.org/en-US/docs/Glossary/MVC (Hentet: 29.04.22)

MDN Web Docs (2022c) *Using media queries.* Tilgjengelig fra: https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries (Hentet: 03.05.2022)

Meta Platforms (u.å.) *Tutorial: Intro to React*. Tilgjengelig fra: https://reactjs.org/docs/getting-started.html (Hentet 04.03.22)

Microsoft (2012) *The Model-View-ViewModel Pattern.* Tilgjengelig fra: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10)?redirectedfrom=MSDN (Hentet: 29.04.22)

Microsoft (u.å. a) *Azure SQL Database.* Tilgjengelig fra: https://azure.microsoft.com/nb-no/products/azure-sql/database/ (Hentet 21.05.22)

Microsoft (u.å. b) *Microsoft Certifications.* Tilgjengelig fra: https://docs.microsoft.com/en-us/learn/certifications/ (Hentet: 04.05.22)

Microsoft (2021) *Overview of Entity Framework Core*. Tilgjengelig fra: https://docs.microsoft.com/en-us/ef/core/ (Hentet 19.04.22)

Microsoft (2022a) *ASP.NET Core Blazor hosting models*. Tilgjengelig fra: https://docs.microsoft.com/en-us/aspnet/core/blazor/hosting-models?view=aspnetcore-6.0 (Hentet 23.02.22)

Microsoft (2022b) *What is .NET? Introduction and overview*. Tilgjengelig fra: https://docs.microsoft.com/en-us/dotnet/core/introduction (Hentet 04.03.22)

Microsoft (2022c) *Visual Studio 2022 IDE.* Tilgjengelig fra: https://visualstudio.microsoft.com/vs/ (Hentet 08.03.22)

Microsoft (2022d) *Overview of ASP.NET Core SignalR*. Tilgjengelig fra: https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-6.0 (Hentet 19.04.22)

Nes, S. M. (2019) *En kort introduksjon til Scrum*. Tilgjengelig fra: https://www.visma.no/blogg/en-kort-introduksjon-til-scrum/ (Hentet 24.02.22)

Oluwatosin, H.S. (2014) Client-Server Model, i *IOSR Journal of Computer Engineering* (Volume 16). Tilgjengelig fra: https://www.researchgate.net/profile/Shakirat-Sulyman/publication/271295146_Client-Server_Model/links/5864e11308ae8fce490c1b01/Client-Server-Model.pdf, s. 67-68 (Hentet 29.04.22)

Oracle (2010) *The Java EE 5 Tutorial.* Tilgjengelig fra: https://docs.oracle.com/javaee/5/tutorial/doc/bnagx.html (Hentet: 08.03.22)

Oracle (u.å. a) *What Is a Database?* Tilgjengelig fra: https://www.oracle.com/database/what-is-database/ (Hentet: 29.04.22)

Oracle (u.å. b) *What is a Relational Database (RDBMS)?* Tilgjengelig fra: https://www.oracle.com/database/what-is-a-relational-database/ (Hentet: 08.03.22)

Semilof, M., Casey, K., Montgomery, J. (2021) *What is cloud migration? An introduction to moving to the cloud*. Tilgjengelig fra: https://www.techtarget.com/searchcloudcomputing/definition/cloud-migration (Hentet 08.03.22)

Stacc Escali (u.å. a) *Om Oss*. Tilgjengelig fra: https://escali.no/Om-oss (Hentet: 18.02.22)

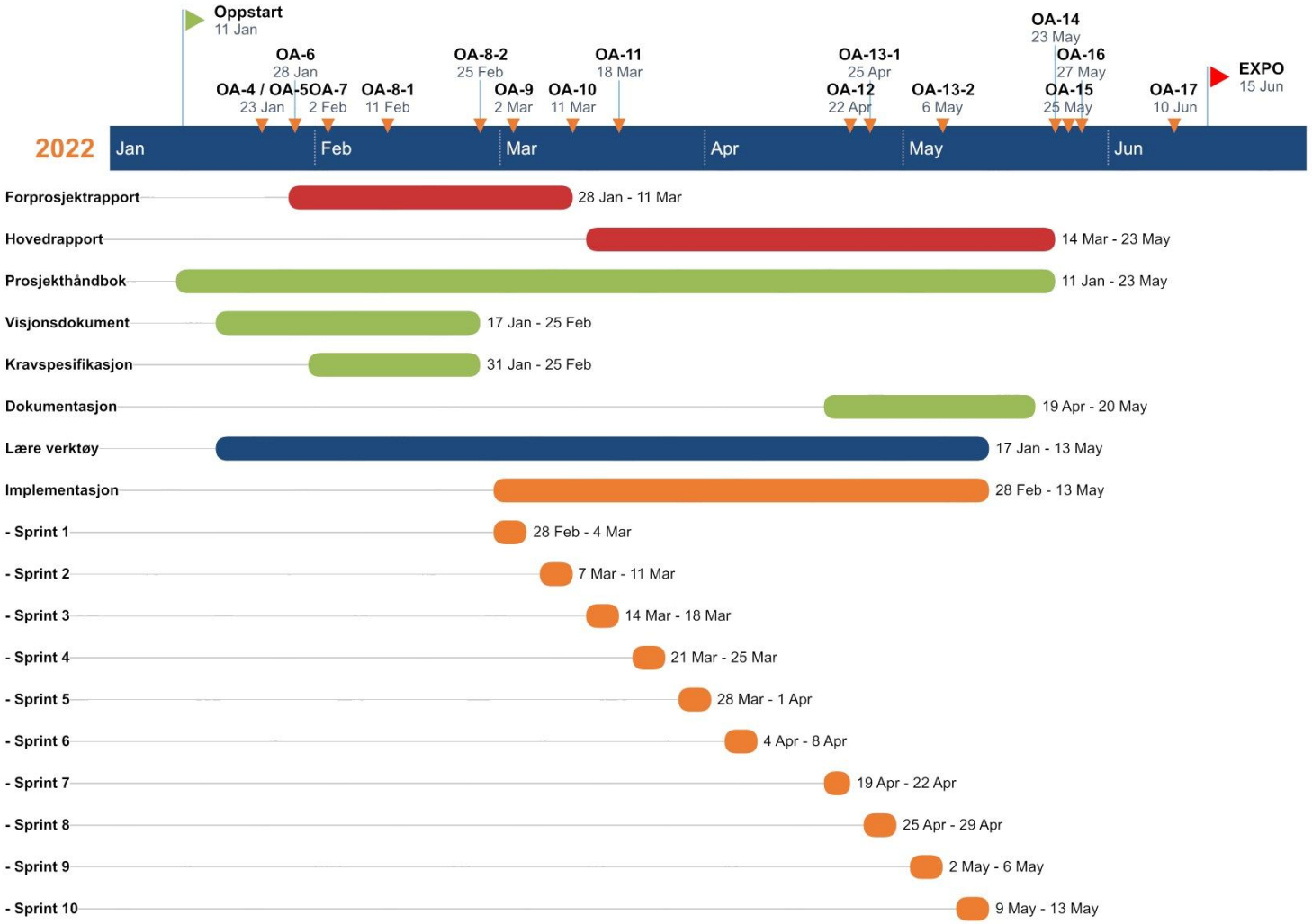
Statcounter (2021) *Desktop vs Mobile vs Tablet Market Share Norway.* Tilgjengelig fra: https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/norway/2021 (Hentet: 03.05.22)

State of JS (2020) *Front-end Frameworks*. Tilgjengelig fra: https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/ (Hentet 23.02.22)

Visual Studio Code (u.å) *Getting Started.* Tilgjengelig fra: https://code.visualstudio.com/docs (Hentet: 08.03.22)

9 VEDLEGG

9.1 Fremdriftsplan med sprinter



Rapporter er markert i rødt, støttedokumenter i grønt, læring av verktøy i blått og implementasjon/sprinter i oransje.

Rapporter og dokumenter vil jobbes med i flere iterasjoner. OA-X er innleveringsfrister i

løpet av semesteret.

OA-leksikon

1. OA-4: Oppstartsmøte med oppdragsgiver og veileder innen uke 3. Frist: 23. januar 2022
2. OA-5: Møte med HVL-veileder i løpet av uken. Frist: 23. januar 2022.
3. OA-6: Prosjekttittel. Prosjekthåndbok itr.1. Visjonsdokument itr.1. Frist 28. januar 2022.

4. OA-7: Statusmøte med HVL-veileder. Oppfølging av OA-6. Frist: 6. februar 2022.
5. OA-8-1: Analyse av tidligere bacheloroppgaver. Frist: 16. februar 2022.
6. OA-8-2: Forprosjektrapport itr 1. Prosjekthåndbok itr 2. Visjonsdokument itr 2. Kravspesifikasjon itr 1. Frist: 25. februar 2022.
7. OA-9: Statusmøte med HVL-veileder. Oppfølging av OA-8-2. Frist: 6. mars 2022.
8. OA-10: Forprosjektrapport. Frist: 13. mars 2022.
9. OA-11: Forprosjekt-presentasjon. Frist: 18. mars 2022.
10. OA-12: Prosjektstatus. Prosjekthåndbok itr 3. Frist: 24. mars 2022.
11. OA-13-1: Utkast-1 til rapport. Frist: 25. mars 2022.
12. OA-13-2: Utkast-2 til rapport. Frist 8. mai 2022.
13. OA-14: Endelig rapport innlevering. Wiseflow. Frist: 23. mai 2022.
14. OA-15: Refleksjonsnotat. Frist: 25. mai 2022.
15. OA-16: EXPO poster. Frist: 29. mai 2022.
16. OA-17: Presentasjon. Frist: 10. juni 2022.

9.2 Risikoanalyse

Hendelse	Årsak	S	K	R	Tiltak for forebygging	Løsning om risiko inntreffer
Gruppen	Prosjektmedlemmer sterkt uenige	SL (1)	M (3)	3	Kommunisere godt og spørre dersom noe er uklart/uforstått. Ikke være enig bare for å være enig.	Finne ut av hvorfor man er uenig og finne en god løsning for begge parter. Diskusjon.
Applikasjon	Tekniske problemer som gjør at implementasjon tar lengre tid enn antatt	H (4)	M (3)	12	Vanskelig å forebygge at en slik hendelse inntreffer.	Feilsøking. Få hjelp fra erfarne utviklere hos oppdragsgiver.
Langtidssykdom	Covid etc.	L (2)	H (4)	8	Vaske hender og bruke munnbind.	Fordele arbeidsoppgaver slik at innleveringsfrister fortsatt vil være mulig.
Tregt læringsutbytte	Ikke nok tid innen prosjektstart	M (3)	H (4)	12	Bruke mer tid på læring.	Sette av mer tid til å lære seg verktøy. Bruke tilgjengelige ressurser på nett og arbeidsplass. Eventuelt finne nye minimumskrav
Applikasjon	Mangel på tid	M (3)	H (4)	12	Høy konsentrasjon og effektiv arbeid.	Sette oss sammen med oppdragsgiver for å finne nye minimumskrav. Eventuelt bytte utviklingsverktøy til noe gruppen har god kjennskap til.
Applikasjon	Feil på IT systemer (f.eks Azure SQL)	L (2)	M (3)	6	Bruke lokale test-databaser under utvikling.	
Applikasjon	Personopplysninger/data på avveie	L(2)	H (4)	8	Være bevisst på at det innehas sensitiv data.	Kontakte personer hvor informasjon er blitt lekket. Finne hullet og tette det.
Applikasjon	Ineffektiv app pga. lite erfaring med teknologier som brukes	M (3)	M (3)	9	Lære teknologiene godt før implementasjon. Bruke tilgjengelige ressurser på arbeidsplass.	
Applikasjon	Appen ble ikke som oppdragsgiver så for seg	M (3)	H (4)	12	God kommunikasjon med oppdragsgiver.	Finne ut hva som har gått galt. Reflekter hvorfor ting har gått slik som det har gått. Skrive ned i hovedrapport.

9.3 Oppgavebeskrivelse

EB-13 Web App for lisenskontroll (Stacc Escali AS)

Stacc Escali AS er et datterselskap i Stacc-konsernet. Stacc er et fintech-selskap med hovedkontor i Bergen og ca 150 ansatte. Stacc leverer ulike løsninger til bank og finanssektoren.

Stacc Escali er norgesledende leverandør av et porteføljе- og treasurysystem, hvor kunder innen bank, forsikring, pensjonskasser, investeringsselskaper og store virksomheter holder styr på sine finansielle instrumenter som aksjer, fond, obligasjoner, lån og derivater.

Mer informasjon på www.stacc.com

Oppgave

Stacc Escali leverer skybaserte løsninger hvor kundene har adgang til ulike moduler i systemet. Dette betaler de en fast årlig lisens for. Innenfor hver modul er det en prisstruktur som varierer ut ifra kompleksiteten hos kunden, slik at de komplekse kundene betaler mer, og de enkle kundene betaler mindre.

Her er et eksempel fra modulprisen på aksje- og fondsmodulen (hvor det er begrensninger i forhold til forvaltningsvolum (MNOK) og transaksjonsvolum:

PRICELIST ESCALI FINANCIALS 2021 – NOK per MONTH							
Stocks and Funds							
250MNOK/100	500MNOK/250	1MRD/500	2,5MRD/1000	5MRD/2500	5MRD/5000	10MRD/Unlim.	Unlimited
950	1900	2850	3800	4750	5700	6650	7600

I dag styres tilgangen på moduler i systemet ut ifra en installert lisenskode, mens avtaleopplysninger/lisens-avtaler lagres i selskapet CRM-system. Lisenskodene produserer i dag ved hjelp av en enkel egenutviklet applikasjon.

I fremtiden ønsker vi å gjøre det slik at kundene våre selv kan utvide og bestille moduler på nettet i løsningen vår direkte. Det krever at vi flytter ut avtaleinformasjonen fra CRM-systemet vårt, og ut til «skyen».

Vi ser for oss en applikasjon som inneholder følgende funksjonalitet/innhold

- En «produktkatalog» og prisliste
- Alle kundenes lisensavtaler
- Muligheter for å legge inn nye og utvidede avtaler direkte i applikasjonen
- Muligheter for å lage tilbud på lisenser direkte i applikasjon.
- Muligheter for å prisjustere avtalene og prislisten
- Muligheter til å hente ut informasjon til ledelsesrapportering, som lisensendringer (frafall, utvidelser, fordelt på kunder, bransjer, moduler etc)
- Eksportere fakturagrunnlag

Applikasjonen skal kommunisere mot vår løsning via web-services, med følgende informasjon:

- Escali's løsninger skal skulle kunne hente lisensbegrensninger fra web-servicer i web-appen. Dette skal erstatte dagens lisenskoder
- Informasjon om lisensutvidelser som kundene gjør direkte i Escali's løsninger

Vår teknologiplatform er basert på følgende:

- Azure SQL database
- C# / .NET
- Blazor

Som en viktig del av prosjektet vil du bygge opp kompetanse og erfaring på angitte teknologier.

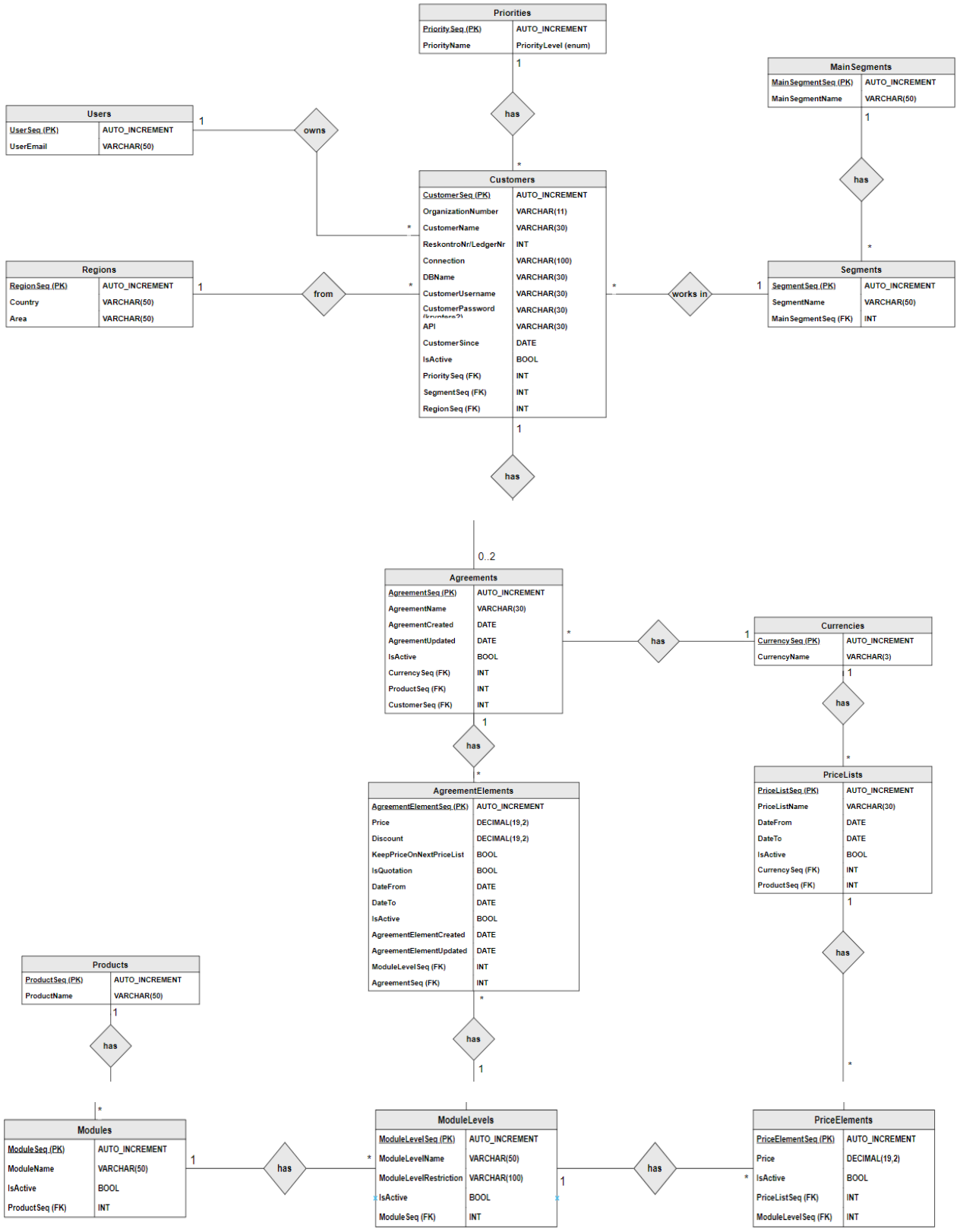
Ved å jobbe med dette prosjektet vil du jobbe i et innovativt og aktivt miljø, hvor flesteparten har stor IT- og systemutvikligskompetanse.

Vi vil tilby arbeidsplasser ved vårt kontor i Thormøhlensgate 53 på Marineholmen, men vi er i stor grad fleksibel i forhold til om man ønsker å jobbe i deler av prosjektet hjemmefra.

Kontaktpersoner

- Gerdt Vedeler
- Forretningsutvikler
- gerdtv@stacc.com
- Mobil 926 22 063

9.4 Databasemodell



9.5 Timeliste

Simen																							
Oppgave / Uke	**2+3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**Sum oppgave**
Forprosjektrapport					3	17	2	9															**31**
Hovedrapport											1		6	11	9	27	8	26	6				**94**
Prosjekthåndbok	1	5	2			1			1			1			1			1					**13**
Visjonsdokument		4	1															1					**6**
Kravspesifikasjon			5	13		1												1					**20**
Systemdokumentasjon														8									**8**
Lære verktøy	24	14	12	3	1	3																	**57**
Implementasjon							16	11	18	25	15	12	4	3	18	2	18	1		6			**149**
Refleksjonsnotat																		2	5				**7**
EXPO & EXPO poster																			8			8	**16**
Presentasjoner (m/forberedelser)					10				3											8	8		**29**
Møter	5	1	2	1		1	2	1			1	1			1	1	1	1	1	1			**21**
Forelesninger	8	2	2	6	4				1														**23**
Sum uke	**38**	**26**	**24**	**23**	**18**	**23**	**20**	**21**	**23**	**25**	**17**	**14**	**10**	**22**	**29**	**30**	**27**	**33**	**20**	**15**	**8**	**8**	**474**

Total oppgave
62
190
26
12
40
16
121
318
12
32
58
43
47
977

Rune																							
Oppgave / Uke	**2+3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**Sum oppgave**
Forprosjektrapport					3	17	2	9															**31**
Hovedrapport											1			7	9	23	8	42	6				**96**
Prosjekthåndbok	1	5	2			1			1			1			1			1					**13**
Visjonsdokument		4	1															1					**6**
Kravspesifikasjon			5	13		1												1					**20**
Systemdokumentasjon														8									**8**
Lære verktøy	28	18	15		3																		**64**
Implementasjon							16	13	17	26	16	22	8	8	20	4	6	1	6	6			**169**
Refleksjonsnotat																			5				**5**
EXPO & EXPO poster																			8			8	**16**
Presentasjoner (m/forberedelser)					10				3											8	8		**29**
Møter	5	1	2	1	1	1	2	1			1	1			1	1	1	1	1	1			**22**
Forelesninger	8	2	2	6	5				1														**24**
Sum uke	**42**	**30**	**27**	**20**	**22**	**20**	**20**	**23**	**22**	**26**	**18**	**24**	**8**	**23**	**31**	**28**	**15**	**47**	**26**	**15**	**8**	**8**	**503**

Totalt uke	**80**	**56**	**51**	**43**	**40**	**43**	**40**	**44**	**45**	**51**	**35**	**38**	**18**	**45**	**60**	**58**	**42**	**80**	**46**	**30**	**16**	**16**	**977**