OSLOMET

OneCall

OSLO METROPOLITAN UNIVERSITY
STORBYUNIVERSITETET

Bachelor Thesis

OC METRICS - BY GROUP 33

Institutt for Informasjonsteknologi
Postadresse: Postboks 4 St. Olavs plass, 0130 Oslo
Besøksadresse: Holbergs plass, Oslo

Telefon: 22 45 32 00

PROSJEKT NR.
33

TILGJENGELIGHET
Offentlig

BACHELORPROSJEKT

HOVEDPROSJEKTETS TITTEL	DATO
OneCall Metrics	26.05.2023
	ANTALL SIDER / BILAG
	84
PROSJEKTDELTAKERE	INTERN VEILEDER
Jenny Liang Nergård (s312980) Jonas Nicolaysen (s354501) Aksel Holm Jensen (s314807) Aleksandar Perendic (s351906) Elzat Arkin (s354390)	Boning Feng, Førsteamanuensis

OPPDRAGSGIVER	KONTAKTPERSON
Telia Company – avd. OneCall	Tonje Moe Smith-Hansen, Application Manager

SAMMENDRAG

OneCall har et ønske om å vite mer om kundene sine, spesielt med hensyn til kundefrafall. En web-basert applikasjon som kan vise statistikk og risikofaktorer for dette var ønskelig.

Gjennom en smidig utviklingsprosess har vi bygget en fullstack-applikasjon med et dashboard, søk og kundeanalyse. Applikasjonen er er en MVP skrevet i Java med Spring Boot som rammeverk i backend. Vi har brukt React som frontend-rammeverk.

Vi fikk to tabeller fra oppdragsgiver; den ene med CDR data, den andre med kundeinformasjon. Denne dataen har blitt bearbeidet og trent med Random Forest Regression, en 'supervised learning' algoritme, til en maskinlæringsmodell som har som mål å forutsi kundefrafall-risiko. Vi har evaluert modellen vår og kan si at den fikk en god prediksjons score. Denne modellen har blitt tilgjengeliggjort gjennom Python-rammeverket Flask.

3 STIKKORD
Churn risiko
Webapplikasjon
Maskinlæring

Forord

Denne sluttrapporten er en del av vårt bachelorprosjekt ved Fakultet for Teknologi, Kunst og Design (TKD) på OsloMet - Storbyuniversitetet, våren 2023. Bachelorprosjektet er utført for OneCall[1], heretter også kalt oppdragsgiver. Rapporten beskriver utviklingen av en webapplikasjon som har til hensikt å gi oppdragsgiver innsikt, statistikk og churn-prediksjoner[2] om sine kunder.

Rapporten er hovedsakelig skrevet for en leser med grunnleggende forståelse for datateknologi, programmering og maskinlæring, men vi har prøvd å legge til rette for at en leser med lite eller ingen kunnskaper på områdene også kan finne denne rapporten forståelig og interessant. Vi anbefaler å lese dette dokumentet digitalt, da det inneholder interaktive lenker til kilder og andre vedlegg som hører med rapporten. Lenkene forekommer med blå skrift og understrek. Innholdsfortegnelsen er linket opp mot overskrifter i teksten, slik at man kan trykke direkte til dem. Forkortelser, fremmedord og faguttrykk er beskrevet nærmere i fotnoter. Alle vedleggene og kilder oppføres i parentes: (Appendiks X), og ligger vedlagt under avsnittet **Referanser og vedlegg**. For å starte applikasjonen følger man brukermanualen i *README.md* filen i prosjektmappen til selve applikasjonen (zip-filen), eller ved å lese instruksjonene i filen *howto.txt*. Link til vår landingsside for prosjektet: https://telia-bachelor.github.io/BachelorOppgave. Her finnes blant annet dokumentasjon av Statusrapport, Prosjektskisse og Forprosjekt, samt presentasjon av oss.

Dette prosjektet har vært en spennende utfordring, og har gitt oss muligheten til å utforske og lære mer om programvareutvikling og maskinlæring. Vi vil takke Tonje Smith-Hansen (Application Manager) og Cato Berglie (Delivery Manager) som fulgte oss opp og la til rette for at vi kunne gjøre bacheloroppgaven hos dem. Vi takker også Boyd Hermansen (Senior Consultant) som har bidratt med verdifull teknisk kunnskap som senior utvikler, samt gitt oss informasjon om oppdragsgiver sin kundebase. Generelt vil vi takke alle involverte fra OneCall for gode innspill og ressurser vi fikk til disposisjon. Vi håper denne rapporten vil være til inspirasjon og nytte for oppdragsgiveren som ønsker å utforske mulighetene som finnes innenfor datateknologi og maskinlæring i telekombransjen.

God lesing!

[1] Telekomselskap heleid av Telia Company.
[2] Churn er engelsk for "kundefrafall".

Sammendrag

OneCall har et ønske om å vite mer om kundene sine, spesielt med hensyn til kundefrafall. En web-basert applikasjon som kan vise statistikk og risikofaktorer for dette var ønskelig. Gjennom en smidig utviklingsprosess har vi bygget en fullstack-applikasjon med et dashboard, søk og kundeanalyse. Applikasjonen er er en MVP[3] skrevet i Java med Spring Boot som rammeverk i backend. Vi har brukt React som frontend-rammeverk. Vi fikk to tabeller fra oppdragsgiver; den ene med CDR[4] data, den andre med kundeinformasjon. Denne dataen har blitt bearbeidet og trent med Random Forest Regression, en 'supervised learning' algoritme, til en maskinlæringsmodell som har som mål å forutsi kundefrafall-risiko. Vi har evaluert modellen vår og kan si at den fikk en god prediksjons score. Denne modellen har blitt tilgjengeliggjort gjennom Python-rammeverket Flask.

[3] "Minimum Viable Product"
[4] CDR = "Call Detail Records", som viser til mobildata-forbruk og mobilaktivitet.

Innholdsfortegnelse

Introduksjon

OneCall er et telekomselskap eid av Telia Company, men opererer som en separat enhet. OneCall ønsker å forstå kundene sine bedre slik at de kan skreddersy tjenester og identifisere risikofaktorer for kundefrafall. Årsakene til churn generelt er mangefasettert og uklar, og oppdragsgiver er spesielt opptatt av å motvirke dette som utgjør et av deres større økonomiske tap. Derfor ønsket de at vi skulle lage en løsning som hjalp dem til å få bedre oversikt over deres kunder, kundenes forbruk og forhåpentligvis peke mot årsaker til churn. Vår oppgave ble å lage en applikasjon som skulle visualisere og presentere data om forbruket, da dagens analytiske team mener at en av nøkkelfaktorene ligger der. Etter flere møter og samtaler med oppdragsgiver utarbeidet vi en problemstilling rettet mot våre målsettinger;

> **"Hvordan konstruere en løsning som gir OneCall mer kundeinnsikt, spesielt med hensyn til churn risiko?"**

1. Styringsdokumentasjon

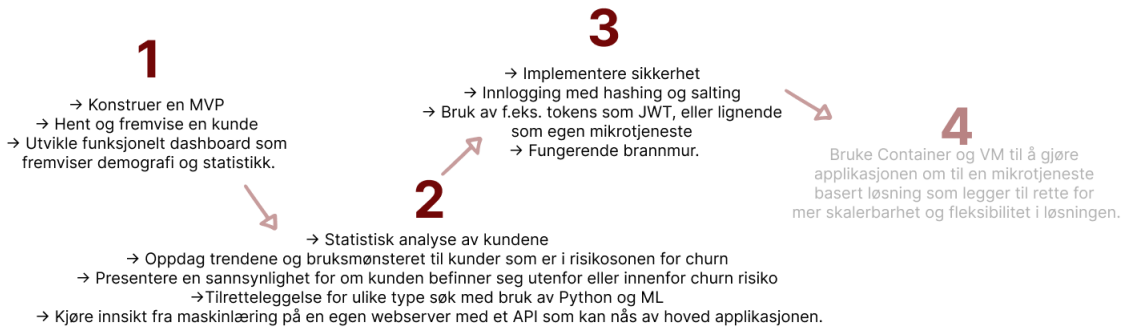
1.1. Planlegging og metode

Bachelorprosjektet startet offisielt i januar 2023. Vi begynte med å planlegge gjennomføringen og lagde en overordnet fremdriftsplan. Gjennom de første møtene med oppdragsgiver samarbeidet vi om å identifisere de grunnleggende kravene til applikasjonen. I løpet av den første fasen definerte vi mål og milepæler, og fastsatte kravspesifikasjoner. Disse vil vi presentere nærmere i kapittel 1.2. Kravspesifikasjon. Deretter utviklet vi en prosjektplan som inkluderte milepælene, tidsrammer og ressursbehov. Planleggingen i starten av prosjektet la grunnlag for en god gjennomføring. Første gang vi ble introdusert for prosjektet var i august 2022, og allerede da ble det opprettet en prosjektdagbok (Appendiks 1). Den brukte vi for å loggføre all aktivitet fra start, møter med oppdragsgiver, gruppemøter, innleveringsfrister og annen informasjon vi tenkte kunne være relevant for senere. Da vi startet med utviklingen av selve applikasjonen i januar, gikk vi over til loggføring i Atlassian sine styringsverktøy som Jira og Confluence. Jira gav oss god oversikt på hva som måtte gjøres, når ting måtte være ferdig, og hvilke ressurser som trengs hvor. Med Jira kunne vi holde orden på fremdriften i Sprinter[5] og ha oversikt over hvem som gjorde hva til enhver tid,

[5] Et tidsrom på ca 4 uker der gruppen skal fullføre en avtalt mengde av arbeidet som er på "tavlen".

og samtidig se hvor langt vi var kommet på hver oppgave. Dette gjorde det enkelt å reflektere over hva som fungerte godt og hva som kunne forbedres til neste Sprint. Ved å definere mål og milepæler tidlig i prosessen, og følge opp med jevnlige møter, ble det lettere å passe på at vi var på rett spor. Vi hadde jevnlige møter med oppdragsgiver for å sjekke om de var fornøyd med fremdriften og det vi hadde gjort. Erfaringer vi har tatt med oss fra dette er at, i tillegg til tekniske ferdigheter kreves det også samarbeid, organisering og planlegging for å gjennomføre et vellykket prosjekt.

Vi samlet oppgavene inn i 4 milepæler. Hver av milepælene representerte en fremgang mot ferdig resultat; å utvikle en fungerende webapplikasjon som kan presentere data om kunder og churn-risiko. Å bruke milepæler gjør det lettere å holde oversikt over fremdriften i et prosjekt. Ved å dele opp et større mål inn i flere mindre delmål vil det bli mer oversiktlig og overkommelig å nå det endelige målet (Rolstadås, 2019). Vi satte målene i rekkefølge etter hva vi prioriterte og hva vi mente var viktigst å få gjennomført for å kunne ha en fungerende applikasjon. Som man kan se på *figur 1.1*, er det fjerde målet "grået ut". Dette fordi vi innså når vi nærmet oss siste halvdel av prosjektet at det ville vært for ressurskrevende og tatt mer tid enn vi hadde til disposisjon. Vi ville heller prioritere de andre milepælene slik at de ble ordentlig gjennomført.

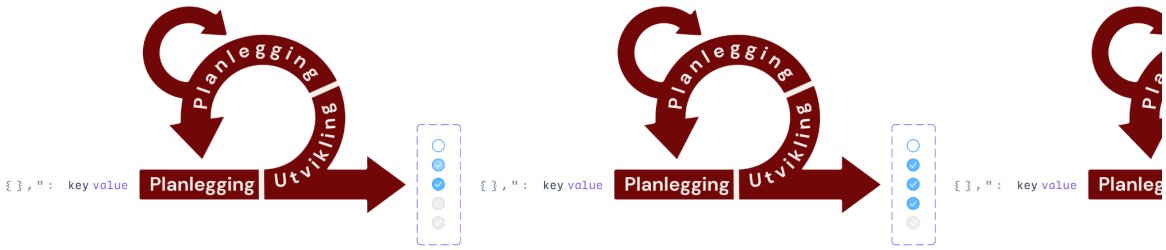


Figur 1.1 - En illustrasjon på Milepæler.

1.1.1. Scrum

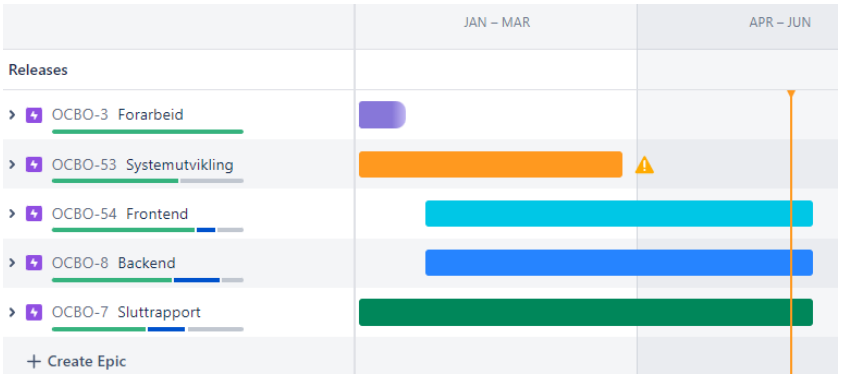
Vi valgte Scrum som arbeidsmetodikk for prosjektet fordi det er en anerkjent og effektiv metode for smidig utvikling av prosjekter som legger vekt på samarbeid, tilpasning, fleksibilitet og iterativ metode (Schwaber, 1997, s. 117-134). Scrum-metoden anerkjenner at endringer vil oppstå i løpet av prosjektet og legger til rette for justeringer og tilpasninger i

prosessen. Vi delte prosjektet inn i 7 Sprinter på ca. 3 uker per Sprint, som er en typisk inndeling for et Scrum-prosjekt (Lindsjørn, 2021).



Figur 1.2 - Sprinter i utviklingsprosess

Vi delte prosjektet inn i ulike Epics[6] etter hvilke oppgaver som skulle utføres (Beerbaum, 2023, s.9 - *figur 1.3*). Vi delte Epics inn i tema etter hvilken del av prosjektet det hørte til. Dette gav oss en bedre oversikt med tanke på å visualisere prosessen på en måte som gav mest mening for vår egen del (*figur 1.3*).



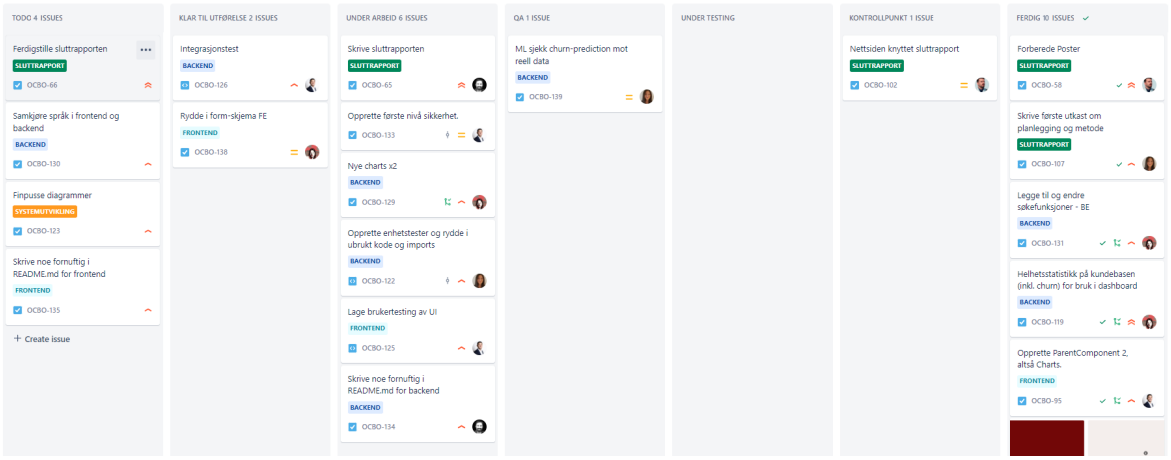
Figur 1.3 - Epics i Jira

Vi delte gruppen inn i et typisk Scrum-team. Dette mente vi var den beste løsningen for at arbeidet skulle bli jevnt fordelt mellom gruppemedlemmene, men samtidig få en god arbeidsflyt uten at arbeidsoppgavene ble for oppstykket. Vi valgte ut en Scrum Master[7] som også ble Product Owner[8]. Denne studenten skulle sørge for at Scrum-prosessen ble fulgt og holde styr på oppgavene i Sprint og Backlog[9]. Samtidig skulle denne personen ha ansvar for kommunikasjon med oppdragsgiver og holde dem oppdatert i prosessen. I utgangspunktet er

[6] En samling relaterte oppgaver knyttet til en brukerhistorie eller tema.
[7] Ansvaret for prosjektstyringen.
[8] Prosjektleder / Ansvarlig for sluttresultat.
[9] Liste med oppgaver som skal gjennomføres, men som ikke er oppført på tavlen / i Sprinten.

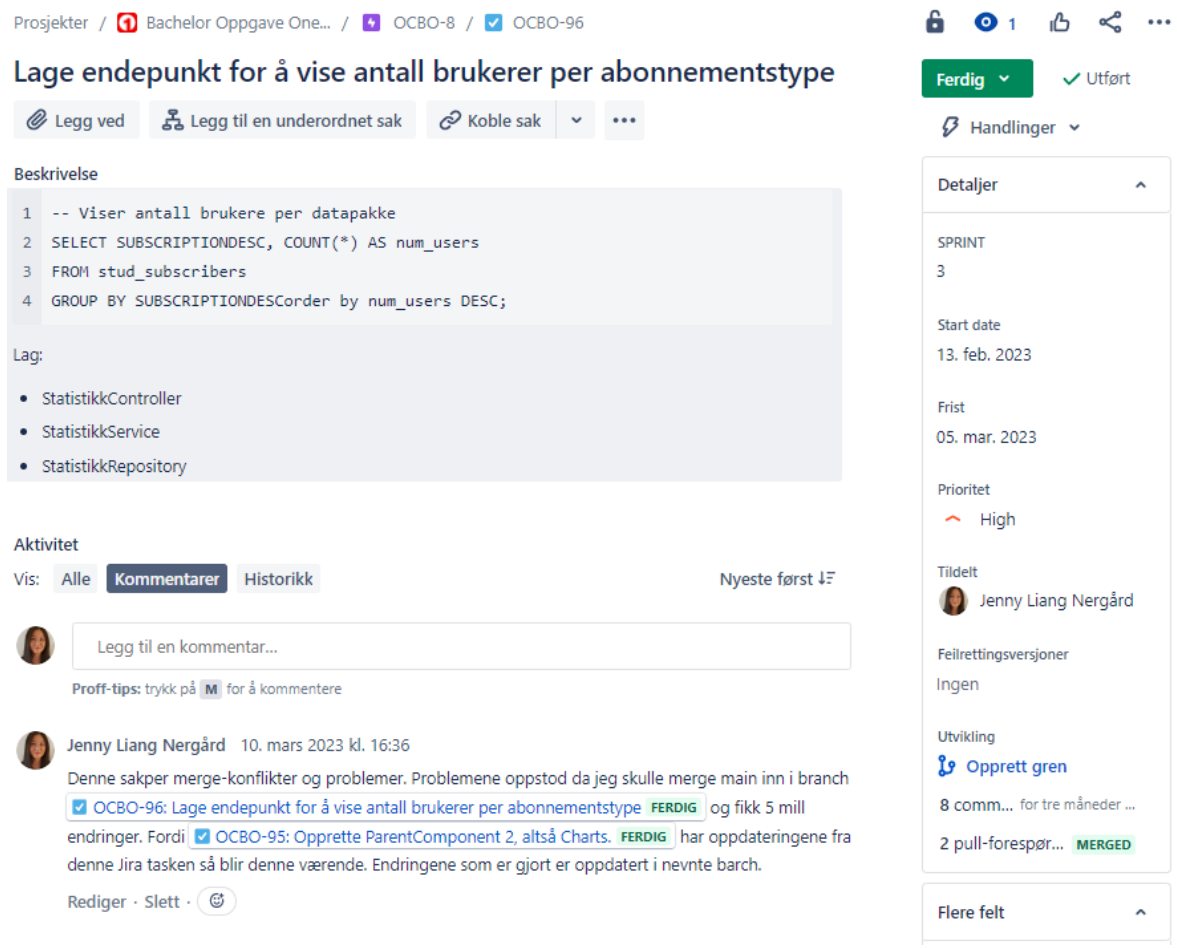
Scrum Master og Product Owner to egne roller som besettes av forskjellige personer, men fordi gruppen var liten fikk en person begge rollene. Et annet gruppemedlem fikk ansvaret for frontend og sørge for at brukergrensesnittet var brukervennlig og i tråd med OneCall sitt designsystem. En tredje fikk ansvar for backend- og API[10]-laget og hvordan logikken bak applikasjonen skulle være. En fjerde fikk ansvar for testing og sikkerhet og sørge for kvalitet og sikker bruk av applikasjonen. Sistemann hadde ansvar for maskinlærings-delen og for å implementere en brukbar modell som skulle gi presise prediksjoner. Vi fordelte oppgavene etter preferanser, men også etter behov og individuelle styrker. Selv om vi fordelte ansvarsområder var vi opptatt av å samarbeide om oppgaver, gjerne i grupper på 2-3 teammedlemmer. Alle i gruppen hadde deltidsjobber og andre fag i tillegg til bacheloroppgaven. Så vi fikk ikke mulighet til å jobbe med prosjektet like mye som et Scrum-team vanligvis gjør. Vi gjorde vårt beste for å møtes minst to ganger i uken for å jobbe sammen og ha stand-ups[11]. Dette med hensikt å følge opp hverandres arbeid og være oppdatert på arbeidsstatus. Etter hver Sprint hadde vi et lengre møte med Retrospektiv (Appendiks 2) for å få en mer helhetlig oversikt over progresjon, evaluere arbeidet og reflektere over hvordan vi synes prosessen hadde gått så langt. En av fordelene med Jira er at man kan koble den til ulike delingsplattformer som GitHub[12] og Slack[13]. Med disse delingsplattformene kunne vi knytte Jira-oppgavene opp mot branches[14]. Slack brukte vi for å kommunisere og GitHub for å administrere og samarbeide på ulike deler av prosjektet. Ved hjelp av funksjoner som “branching”, “merging”[15] og "pull requests”[16] kunne vi samarbeide i en felles kodebase på separate områder på en organisert måte uten å forstyrre hverandres arbeid eller endringer.

[10] Application Programming Interface, utveksling av data mellom ulike teknologier.
[11] Kort dagsmøte med oppdatering på hva som er gjort, gjøres og må gjøres til neste dagsmøte.
[12] Forklares i avsnittet om teknologi.
[13] Se avsnitt “1.3. Valgt teknologi” for nærmere beskrivelse.
[14] En del av “hoved-koden” i en egen gren for å jobbe med den individuelt.
[15] Å slå sammen den individuelle grenen inn i “hoved-koden” igjen.
[16] Be om tillatelse før man kobler sammen den individuelle grenen inn i “hoved-koden” igjen.

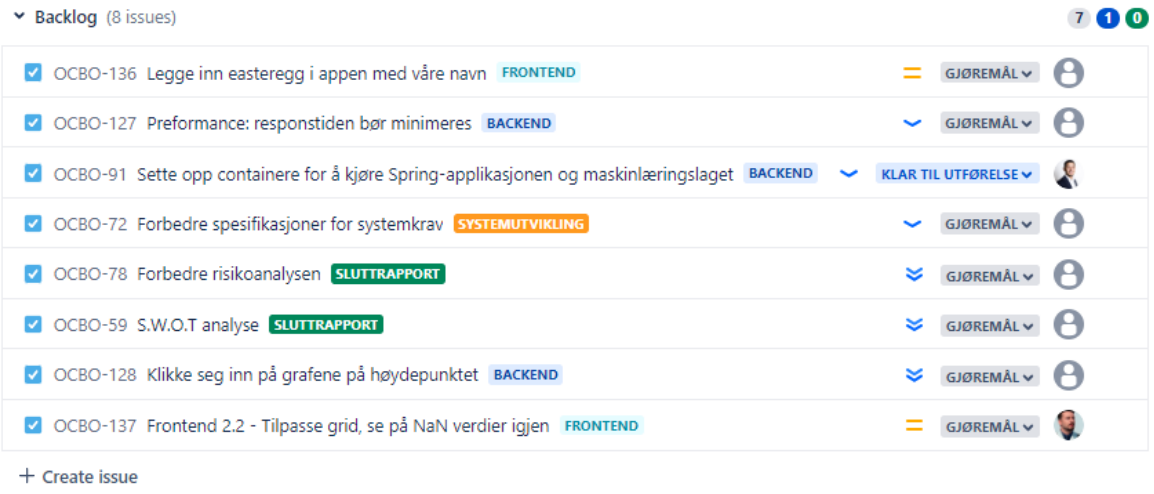


Figur 1.4 - Et klipp fra Scrum Board i Jira

Jira har innebygde funksjoner som er tilpasset Scrum-metodikken og har en digital visuell Scrum Board, Backlog og Roadmap. I Scrum har man et board (tavle) der alle oppgaver som er i den aktuelle Sprinten ligger. Som man kan se av *figur 1.4* gir det en veldig oversiktlig fremstilling. Man ser hvilke oppgaver som er i sprinten, hvilken Epic de har blitt tildelt og hvor langt i prosessen man er kommet. I tillegg kan man se hvem som har fått tildelt oppgaven og hvilken prioritet den har. Man kan også gå inn på hver enkelt oppgave og legge til beskrivende tekst, vedlegg / screenshots og kommentere, slik at man kan følge opp saker underveis (*figur 1.5*). Oppgavene er også nummererte og inneholder en tittel, slik at de skal bli lettere å finne igjen.



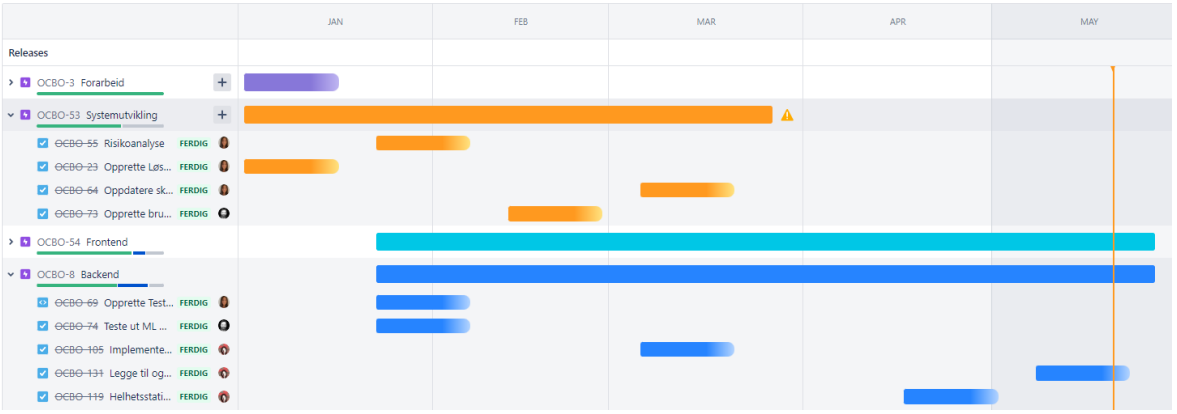
Figur 1.5 - En typisk oppgave (Task) i Jira



Figur 1.6 - Backlog i Jira

Resten av oppgavene ligger i Backlog og er ikke synlig på tavlen (*figur 1.6*). Vi la inn alle oppgavene som måtte fullføres i Backlog med startdato og sluttdato for estimert tidsbruk. I Backlog finnes alle oppgavene som er nedprioritert eller vi ikke er klare til å begynne med.

Roadmap brukte vi som en fremdriftsplan. Fordelen med å ha fremdriftsplanen i Jira er at oppgavene kan sorteres etter tidsperioder og type oppgave. Som man kan se på *figur 1.7* har Jira fordelt Epics i ulike farger og sortert dem i kategorier på en tidslinje med oppgavene og tidsperioden de ble satt i. Samtidig viser "Releases" med grønt (ferdig), blå (påbegynt) og grå (gjenstående) hvor mye av oppgavene som er fullførte. I Appendiks 3 finnes en mer detaljert visning av fremdriftsplanen.



Figur 1.7 - Fremdriftsplan / Roadmap (utklippet viser oppgaver som Ferdig, og dermed strøket over)

Vi fastsette egne rammer for Sprinter, Epics og milepæler. Disse utarbeidet vi sammen i starten for at alle kunne bli samkjørt med bruken av Jira. Oversikt med forklaringer, definisjoner og periodisering finner du i Appendiks 4.

1.2. Kravspesifikasjon

Kravspesifikasjon er en formell avtale mellom leverandør, gruppen i dette tilfellet, og oppdragsgiver. Den bør være klart definert og sørge for at begge parter har en enighet om felles mål (Altexsoft, 2021). Kravspesifikasjonen legger grunnlag for arbeidet med applikasjonen og kan være avgjørende for om produktet møter oppdragsgiverens behov og forretningsmål. I vårt prosjekt delte vi disse målene inn i Funksjonelle og Ikke-funksjonelle krav, noe som er vanlig å gjøre i software-utviklingsprosjekter. Funksjonelle krav beskriver hva applikasjonen skal gjøres, f.eks. ulike brukerfunksjoner og andre ting som brukeren

gjerne kan se på sluttproduktet (knapper, menyer, input-bokser, etc). Ikke-funksjonelle krav beskriver gjerne hvordan applikasjonen skal fungere, f.eks. i form av ytelsesevne, sikkerhet og kvalitet. Vi fikk ikke utdelt formelle krav fra oppdragsgiver av den grunn at de ikke hadde kjennskap til våre evner og hva de kunne forvente av oss. Oppdragsgiver økte oppgavens omfang med ytterligere ønsker underveis i prosjektets gang. Kravene som ble opprettet av gruppen i startfasen ble utformet på grunnlag av notater fra initielle møter med oppdragsgiver.

Funksjonelle krav	***Resultat***
De ansatte skal kunne kategorisere data de vil se basert på selvvalgt filter.	*Vi implementerte et søkefelt der den ansatte skal kunne filtrere søk basert på ulike verdier som eks. kjønn, alder, datapakke, etc.*
Løsningen skal kunne gjøre kall til databasen og tilgjengeliggjøre dataene gjennom API'er/endepunkter.	*Med Java lagde vi controller-klasser i backend som administrerer API'ene mot brukergrensesnittet.*
De ansatte skal kunne kategorisere kundene i ulike kategorier basert på churn risiko.	*I samme søkefelt som i første punkt kan kunden også søke etter brukere basert på deres churn-risiko prosent.*
Løsningen skal bruke maskinlæring for å finne frem til en kundes churn-risiko.	*Gjennom Flask henter applikasjonen maskinlæringsmodellen vår, skrevet i Python og med Jupyter Notebook, og kobler den opp mot applikasjonen.*
De ansatte skal kunne logge seg inn på løsningen gjennom credentials, og opprettholde tilkoblingen med sessions. (ikke avklart om det skal være bruk av SAML/intern Azure kobling)	*Vi endte med en løsning med Basic Authentication som krever brukernavn og passord. Den oppretter LocalStorage eller SessionStorage*[17] *som gir videre tilgang til andre sider i applikasjonen. Verken*

[17] LocalStorage og SessionStorage blir nærmere forklart i kapittelet om sikkerhet.

	SAML/intern Azure kobling ble brukt som løsning hittil.
De ansatte skal kunne hente oppdatert informasjon om hver enkelt kunde, og denne oppdateringen skal skje en gang i døgnet.	*Vi har lagt inn funksjoner og endepunkt for at man skal kunne oppdatere datasettet, men dette er ressurskrevende og det vil ta mye tid for oppdragsgiver å hente oppdatert data fra deres interne database og sende det til vår database. Det er ikke mulighet for dette i applikasjonen i dag.*
De ansatte skal kunne laste ned en presentasjon av resultatene i PDF-, csv- eller Excel-format.	*Dette kravet ble ikke prioritert, da det verken var kritisk for applikasjonen at denne funksjonen ikke var på plass, og heller ikke var utslagsgivende for å oppnå andre mål eller krav.*

Ikke-funksjonelle krav	***Resultat***
Løsningen er godt dokumentert med planer, modeller, test-dokumentasjon og loggføring for eventuelt videre drift og utvikling.	*Med denne rapporten og tilhørende vedlegg (appendiks), samt README.md filen, mener vi at applikasjonen er godt dokumentert for videreutvikling etter endt prosjekt.*
Løsningen skal være bygget slik at den er skalerbar og kunne tilpasse seg kontinuerlige oppdateringer av databasen.	*Vi bygget prosjektet slik at den kan skaleres om det er ønskelig på et senere tidspunkt. Mer om dette i kapittelet om Systemarkitektur.*
Løsningen skal ha god responstid. God responstid vil si at det skal ta mindre enn 10 sekunder for datasettet vi har fått.	*På grunn av ferdig lastet inn data og en modell som kjører bra, så tar det rundt 5 sekunder med dagens løsning å hente resultatene fra maskinlæringsmodellen og databasen. Ideelt skulle vi ønske at*

	modellen brukte under 5 sekunder, da det hadde økt kvaliteten på applikasjonen.
Løsningen skal være brukervennlig med tanke på design og utforming. Det er ønskelig at løsningen skal ha et universelt utformet design og brukergrensesnitt.	*Vi har brukt et design-komponenter fra et ferdig bibliotek som er tilpasset universell utforming og brukervennlig design. I tillegg har vi tatt hensyn til oppdragsgiverens eget design på sine produkter.*

1.2.1. Brukerhistorier

Vi utformet noen brukerhistorier (Use Cases) som bygger på noen av kravspesifikasjonene som skulle gjøre det lettere for oss å kunne visualisere utformingen og formålet med noen av kravene.

1.	*Som* *skal jeg kunne* *slik at*	ansatt se statistikk over kundebasen gruppert etter alder, forbruk, abonnement eller churn. jeg kan få overordnet innsikt i forbruksmønsteret til kundebasen.
2.	*Som* *skal jeg kunne* *slik at*	ansatt se statistikk på en valgt bruker jeg kan få innsikt i den enkelte kundes abonnent-bruk.
3.	*Som* *skal jeg kunne* *slik at*	business analytiker se oversikt over hvilke kundegruppe som er i risikosonen for å churne jeg kan vite hvilken kundegruppe som vi bør gjøre proaktive tiltak for å beholde.
4.	*Som* *skal jeg kunne* *slik at*	business analytiker se sannsynlighet for hvorvidt en kunde er i risiko for å avslutte kundeforholdet jeg kan planlegge videre handling for å motvirke churn på den spesifikke abonnenten.

5.	*Som* *skal jeg kunne* *slik at*	utvikler bytte ut datakilden/rådataene og fortsatt ha en fungerende webapplikasjon som er skalerbar.
6.	*Som* *skal jeg kunne* *slik at*	business analytiker spesifisere søk etter ulikt forbruk jeg kan utvikle business-strategier for den valgte gruppen.
7.	*Som* *skal jeg kunne* *slik at*	business analytiker spesifisere søk etter ulik kundeinformasjon og få opp visuelle fremstillinger i form av grafer jeg kan utvikle strategier for den valgte gruppen.

Brukerhistoriene hjalp oss å se hvordan vår applikasjon eventuelt ville bli brukt i praksis. Gjennom akseptansetestingen (kapittel 4.7.4. Brukerstesting (akseptansetest)) fikk vi prøvd ut noen av brukerhistoriene og testet om applikasjonen tilpasset brukernes faktiske behov.

1.3. Valg av teknologi

Valget av teknologier var hovedsakelig basert på erfaring gruppen hadde fra før, men noen av teknologiene ble valgt på bakgrunn av ønsker fra oppdragsgiver. Gruppen la likevel vekt på å ha en attraktiv løsning som følger dagens marked og arbeidslivet generelt. Vi endte med å lære en del nytt underveis og fikk utfordret oss selv og bruke moderne verktøy som vi ikke tidligere hadde brukt.

Java

Java er det programmeringsspråket gruppen har hatt mest erfaring med. Det er et objektorientert språk basert på konseptet om objekter som inneholder data og kode, der metoder eller prosedyrer er knyttet opp mot objektene (IBM, 2023). Det er et robust språk som brukes mye i Enterprise applikasjoner og næringslivet (Stanford, 2008).

Spring Boot

Spring Boot er et rammeverk med åpen kildekode som brukes for å lage Java web-applikasjoner. Det tilbyr et godt system for konfigurasjon av programvare-avhengigheter og gjør det enkelt å lage en frittstående applikasjon. 'Dependency injection' er sentralt her; det lar objekter definere

sine avhengigheter som senere blir tilføyd av rammeverket (IBM, 2023). Det tilbyr også mange innebygde metoder for å behandle data og objekter, samt metoder for å kommunisere med en database.

MySQL

MySQL er en relasjonsdatabase med åpen kildekode som brukes til å lagre og administrere data i tabeller som er knyttet opp mot hverandre. Den støtter SQL (Structured Query Language), som er et standardisert språk for å hente og manipulere data fra en database. SQL er et deklarativt språk, som betyr at man beskriver hva man vil ha fra databasen, og ikke hvordan man vil ha det. Det kan brukes til å hente, legge til nye, oppdatere eksisterende, eller slette data. MySQL er en av de mest populære relasjonsdatabasene og brukes av mange selskaper og organisasjoner. Det er kjent for å være raskt, pålitelig og robust (talend, 2023).

Python

Python er et programmeringsspråk med mange bruksområder. Det er et dynamisk skrevet språk, som (ulikt f.eks. Java) gjør at datatypene/objektene er implisitt basert på verdiene de blir satt til. Det støtter flere programmeringsparadigmer som strukturelt, objekt orientert og funksjonelt. Det er det mest brukte språket for maskinlæring og data science (Raschka et al, 2020, s. 1).

Jupyter Notebook

Jupyter Notebook er et verktøy som brukes til å redigere og kjøre Notebook-dokumenter. Her kan man skrive kode (oftest i Python), men også legge til vanlig tekst. Istedenfor vanlig IDE[18] hvor man skriver all kode før det kjøres, har Notebook muligheten til å skrive en linje kode som kan kjøres. Verktøyet er mye brukt innen data analyse og AI-implementering (ibid.). Vi har brukt bibliotekene Pandas og Numpy for databehandling.

Flask

Flask er et lettvekts rammeverk for Python. Det kan kalles et micro-rammeverk siden det ikke har mange avhengigheter til eksterne biblioteker[19]. Det er mulig å laste opp en Pickle-fil som inneholder en

[18] Integrated development environment (Integrert utviklingsmiljø)
[19] Samling av ferdigbygde kode-elementer, skript eller rutiner som kan brukes for å forenkle utviklingsprosessen.

maskinlærings-modell, og å tilgjengeliggjøre et endepunkt for å analysere data slik som vi har gjort i dette prosjektet (pythonbasics, 2021).

JavaScript

JavaScript er et programmeringsspråk som brukes til å lage interaktive nettsider og applikasjoner. På klientsiden brukes JavaScript for å lage dynamiske og responsive nettsider som kan endre innhold og utseende uten å måtte laste siden på nytt. JavaScript er et viktig språk for utvikling av moderne nettsider og applikasjoner (Nätt, 2020).

React

React er et rammeverk for JavaScript utviklet av Meta, men brukt av utviklere i hele verden. Et av grunnprinsippene for React er muligheten til å gjenbruke koden fordi den er modulær og komponentbasert, samt veldig fokusert på høy ytelse. Reacts virtuelle DOM *(Document Object Model)* gir en raskere og mer effektiv oppdatering av grensesnittet når data endres(Duldulao & Cabagnot, 2021, s.3). Dette gir bedre ytelse og mer responsive applikasjoner.

Ubuntu

Ubuntu er et Linux-basert operativsystem (OS) med åpen kildekode. Ubuntu OS er optimalisert for servere og skyplattformer. Det er utviklet for å gi en mer effektiv og sikker plattform for webtjenere, applikasjonsservere og databaseservere (Loshin & Bigelov, 2021).

GitHub

GitHub er den største leverandøren av hosting av Git Repositories. Git er det mest brukte systemet for versjonskontroll. GitHub tilbyr et godt grensesnitt for å holde oversikt over endringer i koden, spore bugs og gjøre det lettere for flere utviklere å jobbe sammen om et prosjekt.

Postman

Postman er et verktøy for å teste API'er. I vårt prosjekt har vi brukt Postman for å sørge for at endepunktene fungerte som de skulle. Postman er populært blant små bedrifter eller for enkle prosjekter fordi det er lett å anvende og sette opp.

Alternativ teknologi

I dette avsnittet nevner vi et utvalg av de teknologiene vi valgte bort, men som kunne vært aktuelle å bruke om vi hadde hatt andre forutsetninger. Alle disse teknologiene er også anerkjente og godt dokumenterte.

.NET

.NET er en Microsoft-teknologi som fungerer på Windows-plattformen, og bruker hovedsakelig C# som sitt primære programmeringsspråk. .NET bruker en monolittisk arkitektur, som betyr at applikasjonen er bygd som en stor enhet (Microsoft, 2023). Vi valgte rett og slett Spring Boot fremfor .NET da vi hadde mer kjennskap til Spring.

PostgreSQL

PostgreSQL tilbyr avanserte funksjoner og mer fleksibilitet enn MySQL. PostgreSQL har f.eks. innebygd støtte for geografisk informasjonssystem og komplekse datatyper. For gruppens prosjekt ble dette noe mer avansert og unødvendig da MySQL dekket våre behov. (Postgresql, 2023).

Angular er et alternativ til React som bruker TypeScript istedenfor JavaScript. Angular har annerledes arkitektur og tilbyr toveis data-binding[20]. React tilbyr derimot enveis data-binding, der data flyter fra øverste nivå ned til child components[21] (Simplilearn, 2023). Vi valgte React fordi det er mest brukt i bransjen og det gir bedre ytelse og forenkler feilsøking.

Azure Cloud er et alternativ til Ubuntu-server. En mulighet hadde vært å ha applikasjonen i cloud. En av fordelene er blant annet skalerbarhet, pålitelighet og automatisering (Microsoft, 2023). Hovedårsaken til at gruppen har anvendt en Ubuntu-server er at den ble tilgjengeliggjort for gruppen av oppdragsgiver for å utføre prosjektet.

GitLab

GitLab er ofte brukt av mange selskap i Norge, mest på grunn av muligheten til å ha det på egen server (self-hosted GitLab). GitLab er ikke noe gruppen var kjent med fra tidligere, og fordi gruppen hadde studentlisens på GitHub, var det en bra nok løsning for oss.

Swagger

Swagger er et anerkjent verktøy for å implementere, dokumentere og teste API'er. Swagger brukes gjerne i større bedrifter og har flere funksjoner enn Postman. Det er et kraftig verktøy, men også hakket mer avansert å bruke (geeksforgeeks, 2022). Vi valgte Postman av den grunn at vi ikke hadde behov

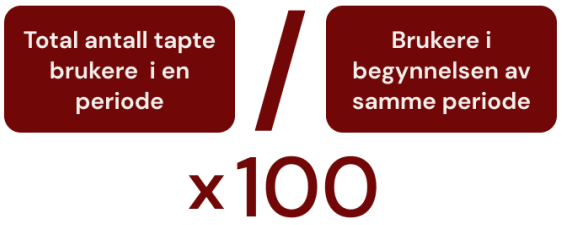
[20] En mekanisme som tillater automatisk synkronisering av data mellom komponenter og brukergrensesnittet.
[21] Blir nærmere beskrevet i kapittelet om Frontend (React.js).

for et så sterkt verktøy, også med tanke på at det hadde tatt oss mer tid å sette oss inn i dette verktøyet.

2. Teori om churn og maskinlæring

Churn

"Churn" er enkelt forklart "kundetap", dvs. når en kunde avslutter sitt kundeforhold (Rautio, 2019, s.3). Det kan måles og omtales da som "churn rate" som er en prosentvis formel som sier noe om hvor mange kunder et selskap mistet innenfor et spesifikt tidsrom



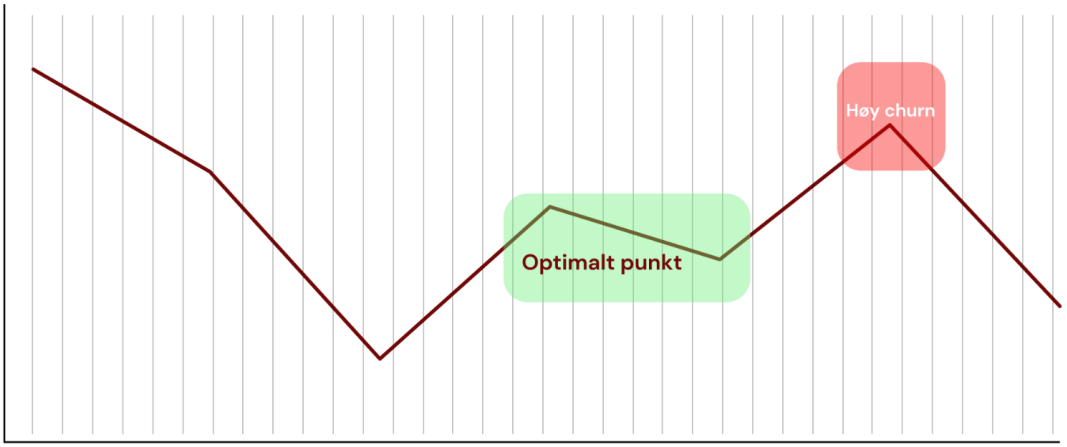
Figur 2.1 - "Formel for utregning av Churn" (Fullview (u.å.))

Innenfor telekombransjen og lignende SaaS-selskaper[22] er churn et utstrakt problem (gjennomsnittlig 13%), og hovedårsaken forblir uklar (ibid.). Man spekulerer i priskonkurranse, generell misnøye med tilbudet/tjenesten, svak tilpasset abonnement for kundens bruk, svak kundeservice, dårlig/ustabil dekning eller lignende (Rautio, 2019, s.7). Det er et stort økonomisk fortrinn å kunne forutse og unngå churn. Det finnes utregninger som sier hvor mye det koster å få inn en kunde, og vice versa hvor mye selskapet taper på at en kunde forlater kundeforholdet. Forskning indikerer at firmaer burde fokusere på å holde igjen eksisterende kunder (Ahn et al., 2020, s.3), og at *"increasing customer retention rates by 5% increases profits by 25% to 95%"* (Reichheld & Sasser, 1990). Oppdragsgiver ønsket at vi kunne bruke maskinlæring for å komme frem til trender og faktorer for å forutsi om kunder ville churne basert på deres dataforbruk.

De hadde noen hypoteser og teorier om årsakssammenheng mellom churn-risiko og enkelte variabler. Disse ønsket vi å se nærmere på og finne ut om det kunne være noe sannhet i. Om

[22] SaaS (Software-as-a-service) er en distribusjonsmodell hvor programvare leveres over internett som en tjeneste, slik at brukere kan få tilgang til og bruke applikasjonene uten å måtte installere eller vedlikeholde programvaren lokalt.

etablerte teorier fra oppdragsgivers churn-analyse-team stemte overens med de resultatene vi fant ville det også indikere at vi var på rett spor. En større hypotese var at det kan være en sammenheng mellom churn og kundenes databruk i forhold til inkludert data[23]. Oppdragsgiveren fant at det var en terskelverdi (*figur 2.2*). Disse kundene som ligger på "det gylne snittet" blir som regel lenger hos teleoperatøren enn de andre gruppene.



Figur 2.2 - Rekonstruksjon av graf vi fikk fra oppdragsgiver. Reell data er konfidensiell. Grafen illustrerer områder og punkter som er avgjørende for churn rate. (T. Felix, personlig kommunikasjon, 04. april 2023)

En annen hypotese de hadde var at "value loads" kan være attraktivt for kunden. "Value loads" er å tilby ekstra funksjoner eller fordeler til et produkt eller tjeneste (B. Hermansen, personlig kommunikasjon, 30. januar 2023), noe som bidrar til økt kundetilfredshet og lojalitet. I forbindelse med dette kunne man sett på oppsalg[24] eller nedsalg[25] som brukes som strategier av oppdragsgiver for å øke inntektene og maksimere verdien av hver kunderelasjon. Oppsalg og nedsalg er interessant å se på i forhold til hvilke tendenser kunden har rundt eget forbruk. Disse faktorene tror oppdragsgiver kan være av stor betydning for om kunden er i risiko for å churne. Men denne hypotesen er noe vi ikke fikk utforsket, da vårt datasett ikke inneholdt informasjon om kostnader og betalinger. Vi hadde ikke tilstrekkelig informasjon for å undersøke dette nærmere. Med tanke på målet hadde det nok vært lurt å inkludere disse variablene, men på grunn av tiden til rådighet og arbeidet med datasettet hadde blitt enda mer omfattende falt dette bort. Likevel ser vi ikke bort ifra at denne vil være meget aktuelt å inkludere i videreutvikling av maskinlæringsmodellen vår.

[23] Mobildata / datapakke som inngår i abonnementet.
[24] Refererer til prosessen med å selge et dyrere eller høyere kvalitetsprodukt eller tjeneste til kunden.
[25] Refererer til motsatte, hvor en kunde tilbys et billigere eller lavere kvalitetsprodukt eller tjeneste.

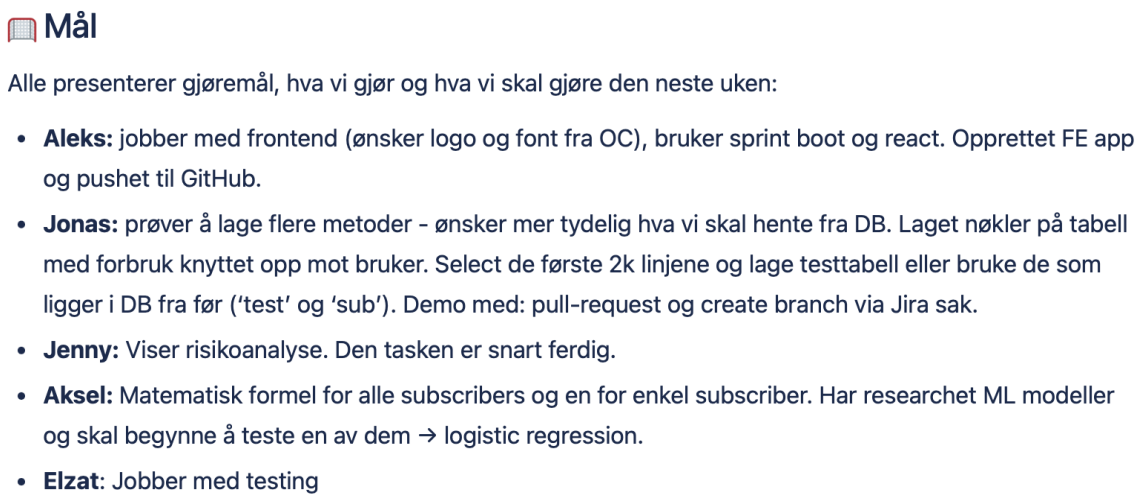
Maskinlæring

Maskinlæring er prosessen med å lære opp en maskin til å utføre en kompleks oppgave som mennesker kan ha vanskeligheter med å gjøre alene. Man mater en maskin med data og får maskinen ved hjelp av algoritmer til å forsøke å gjøre en prediksjon og foreta en beslutning eller beregning ut fra dette. Churn-prediksjon er en utfordrende oppgave, og et utbredt felt innen forskningen. Det brukes mange ulike typer algoritmer avhengig av hvilket formål eller svar man vil ha. Men i arbeidet med churn-prediksjon, blir det vanligvis ansett som et binært klassifiseringsproblem. I forskningsartikkelen til Lalwani, P. et al. har de brukt blant annet klassifiserings-algoritmene[26] Logistic Regression, Naive Bayes, Support Vector Machine, Random Forest, Decision Trees, blant flere, som fremgangsmåter for churn-prediksjon (Lalwani et al., 2022, s. 272-273). De kom frem til at AdaBoost og XGBoost Classifier gav høyest treffsikkerhet på 80% (ibid. s. 288-290). Ut fra artikkelen kan vi ikke se hvilke features de tok med, så det kan spille en faktor på resultatene. En annen forskningsartikkel skrevet av Ahmad, A.K. et al, anvender de også klassifiserings-algoritmer som Decision Tree, Random Forest, Gradient Boosted Machine Tree "GBM" og Extreme Gradient Boosting "XGBoost" for å finne frem til churn-risiko. Med deres metoder kom de fram til at XGBoost gav best resultat på 93% treffsikkerhet (Ahmad et al., 2019, s. 18-20). De hadde en litt annen tilnærming ved at de hadde 70% treningsdata og 30% testdata. De har heller ikke inkludert detaljer om features i artikkelen. Ullah, I. et al prøvde de på flere algoritmer innenfor klassifisering, og fant at Random Forest og J48 gav best resultat; hele 88% (Ullah et al., 2019, s. 60141-60142). I deres forskningsartikkel har de brukt reell CDR som utgangspunkt og laget features som totalt antall samtaler, totale minutter, totalt innkomne og utgående minutter, samtaler på nett, samtaler på nett i minutter osv (ibid., s.60146). Det er den siste artikkelen som vi har lagt mest vekt på når vi lagde vår modell. Deres features har likhetstrekk med våre egne og de har hatt fokus på Random Forest algoritmen som vi også har anvendt i vår modell. Forskjellen er at vi har brukt Regression[27] istedenfor klassifisering. Vi vil snakke mer om vår modell i kapittel 4.2. Maskinlæringsmodellen.

[26] Klassifisering forklares nærmere i kapittelet om Maskinlæringsmodellen.
[27] Forklares nærmere i kapittelet om Maskinlæringsmodellen.

3. Utviklingsprosessen

Utviklingsprosessen var organisert gjennom bruk av Jira, som nevnt i kapittel 1.1. Planlegging og metode. For å oppdatere hverandre på fremdriften, delegere oppgaver og identifisere utfordringer, holdt gruppen møter. Disse møtene ga oss muligheten til å diskutere hva hver enkelt av oss jobbet med og hvor langt vi var kommet med oppgaven, om det var noen utfordringer og hva som burde fokuseres på fremover. I starten prøvde vi å få til å holde disse møtene ukentlig med fast agenda, men etterhvert ble disse møtene mer flytende da timeplanene våre ble forskjøvet. Dessuten merket vi at disse møtene kunne bli unødvendig lange og tok opp tid vi heller kunne bruke på å jobbe med selve prosjektet. Det var viktig at vi da fortsatte å kommunisere godt gjennom Slack og sosiale medier for å holde hverandre oppdaterte. Under kan du se et eksempel på notater fra et av møtene vi hadde i Sprint 2 (*figur 3.1*).



📖 Mål

Alle presenterer gjøremål, hva vi gjør og hva vi skal gjøre den neste uken:

- **Aleks:** jobber med frontend (ønsker logo og font fra OC), bruker sprint boot og react. Opprettet FE app og pushet til GitHub.
- **Jonas:** prøver å lage flere metoder - ønsker mer tydelig hva vi skal hente fra DB. Laget nøkler på tabell med forbruk knyttet opp mot bruker. Select de første 2k linjene og lage testtabell eller bruke de som ligger i DB fra før ('test' og 'sub'). Demo med: pull-request og create branch via Jira sak.
- **Jenny:** Viser risikoanalyse. Den tasken er snart ferdig.
- **Aksel:** Matematisk formel for alle subscribers og en for enkel subscriber. Har researchet ML modeller og skal begynne å teste en av dem → logistic regression.
- **Elzat**: Jobber med testing

Figur 3.1 - Mål for uke 4, Sprint 2

De mer strukturerte møtene, som f.eks. møtene med oppdragsgiver eller Retrospektiv, ble loggført og lagt inn i Confluence (Appendiks 5). På Retrospektiv-møtene hadde vi fokus på å presentere status på prosjektet og hvor vi lå an, diskutere videre plan og mål for neste Sprint, samt organisere og (om)prioritere de nye oppgavene. Det var også et viktig tema for alle Retrospektiv å evaluere hva gruppen var god på og hva vi burde bli bedre på i tiden fremover. Det gav oss læring og mulighet til forbedring. Gruppen var bevisst på at læring og forbedring var viktige elementer i prosessen, så vi prøvde å være åpne på kunne diskutere og komme med konstruktive tilbakemeldinger til hverandre. Vi evaluerte hva som fungerte bra og hva som kunne forbedres, og var villige til å prøve nye tilnærminger for å forbedre prosessen.

Tilbakeblikk på retrospektiv sprint 1

- Handlingselementene ble fullført.
- Stand-up møter hver uke kan vi fremdeles bli flinkere til å etterstrebe.
- Vi er flinke til å bruke slack som kommunikasjonsverktøy.
- Vi er blitt flinkere til å kommentere og skrive i Jira sakene.
- Vi er flinkere til å create ny branch og jobbe med de for hver Jira sak, men vi kan bli flinkere til å sende pull request før vi committer til main.
- Vi må fremdeles bli flinkere til å bruke samme språk i koden.

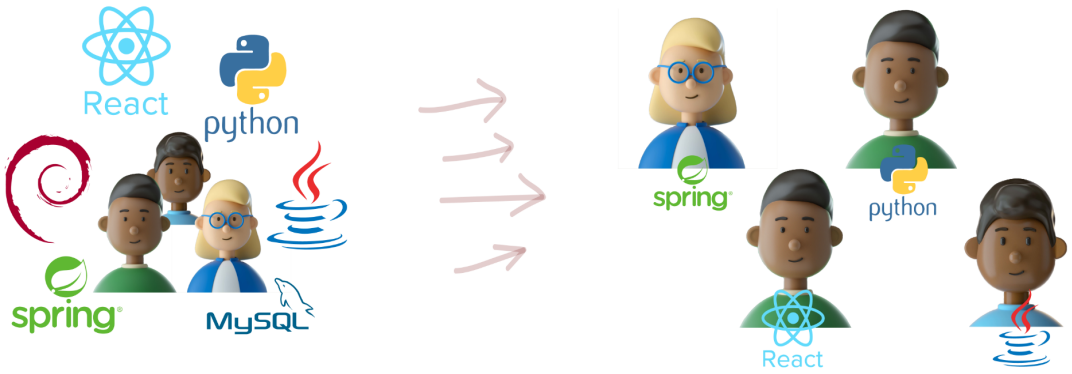
Figur 3.2 - Retrospektiv Sprint 2 med Tilbakeblikk på Sprint 1 (læring og forbedring fra forrige Sprint)

Begynn med	Slutt med	Fortsett med
• Stand-up møter hver uke, helst 2-3 ganger i uken. • Alle presenterer det de holder på med og hva de skal gjøre. • Å bruke Slack som kommunikasjonsverktøy.	• Gjøre oppgaver uten at det er en oppgave i Jira.	• Ukentlige samtaler. • Samarbeide om oppgaver underveis. • Skrive i rapporten underveis når vi har noe å tilføye til den.
• Kommentere på sine saker de har fått tildelt.	• Gjøre endringer uten at vi har blitt enige.	• Følge saker i Jira man har fått tildelt.
• Lav Dev-branch fra repo på GitHub. • Sende Pull Request før man kan pushe til Main-branch. • Minst et annet teammedlem må godkjenne før merge til Main-branch.	• Å commite rett til Main-branch. • Bruke forskjellig språk i koden og ikke bruke forståelige metodenavn.	• Å jobbe med saker via Jira tasks.

Figur 3.3 - Retrospektiv Sprint 2 (læring og forbedring inneværende Sprint)

Selv om vi opprinnelig planla at alle i gruppen skulle få jobbe med alt slik at alle fikk så likt læringsutbytte som mulig, utviklet det seg etter hvert mer tydelige ansvarsområder i prosjektet. Dette var en naturlig utvikling som var basert på «spesialkompetanse» og behovet for å fokusere på spesifikke oppgaver. Vi innså også etterhvert at det ville blitt brukt mye tid på at alle skulle lære seg alle teknologier like godt og være like flinke til alt. Det ville gått utover produktiviteten da vi antakelig hadde brukt mer tid på opplæring og å sette alle inn i

hver enkelt oppgave. Dette igjen hadde ført til at det hadde tatt lengre tid å fullføre prosjektet. Selv om det påvirket den opprinnelige planen, ble kommunikasjonen og samarbeidet mellom gruppemedlemmene opprettholdt, og gruppen fortsatte å jobbe produktivt. Det tillot oss å fokusere på spesifikke oppgaver og følge dem opp på en mer effektiv måte.



Figur 3.4 - Endring i fordeling av oppgaver gjennom tid - ingen kilde, laget av oss selv i figma

Ettersom vi kommuniserte godt var det mulig å jobbe individuelt, men også at flere jobbet sammen om en oppgave, eller at noen byttet mellom en oppgave eller to. Med denne arbeidsstrukturen var det lett å holde oversikt over arbeidet og følge med på hva som fungerer og ikke fungerer i prosessen. Dette gav utbytte i form av at vi kunne lære av egne feil og gjøre forbedringer underveis. Gruppen var også nødt til å være åpen for endringer i ansvarsområder for å sikre at prosjektet ble gjennomført på en mest effektiv måte. Eksempelvis hvis en i gruppen var ferdig med en oppgave og ikke hadde fått tildelt en ny, kunne den personen hjelpe en annen med å bli ferdig med sin oppgave i Sprinten. Alternativt kunne den som var ferdig ta over en av de andres oppgaver som ikke var påbegynt.

Vi brukte GitHub Actions (GitHub, 2023) for å automatisk bygge Spring- og React-applikasjonene hver gang vi godkjente en kodebit til main branchen. Denne arbeidsflyten til GitHub Actions kjørte hver gang noen pushet noe til main branchen og fungerer som en sikring på at det ikke legges til noen endringer som gjør at hoved-koden ikke kompileres. Det er en form for Continuous Integration, og er en viktig del av DevOps prinsippene. Vi har gjennom utviklingsprosessen kjørt applikasjonene lokalt, men hvis vi skulle deploye dem hos en skyleverandør for eksempel, kunne vi gjort dette med en GitHub Actions workflow.

Figur 3.5 - Godkjent byggeprosesser

3.1. Risikoanalyse

En risikoanalyse er en viktig del av software-utviklingen, og den bør gjøres i en tidlig fase. Den har til hensikt å identifisere og vurdere potensielle risikofaktorer som kan ha påvirkning på utviklingsprosessen. Risikoanalysen vil i tillegg hjelpe gruppen til å tidlig innføre tiltak og endringer i planleggingen slik at eventuelle negative påvirkninger reduseres og uheldige scenarioer kan unngås (Hayes, 2023). Derfor er risikoanalyse svært viktig. Følgelig gjorde vi også risikoanalyse tidlig i prosessen. Gruppen gjorde en helhetlig vurdering av både interne og eksterne faktorer som kan påvirke prosjektets fremdrift og suksess. Sammen identifiserte vi potensielle risikoer som kan føre til forsinkelser, uenigheter og avsporing under utviklingsprosessen. I tillegg kategoriserte vi disse risikoene basert på hvor stor sannsynlighet det var for at de kunne oppstå underveis. Et eksempel på en risiko som skjedde som hadde veldig høy sannsynlighet, men som vi vurderte av liten konsekvens var R8.

ID	Risiko	Konsekvens	Mulige tiltak	Kategori
R8	Feilestimering av tid på utviklings-prosessen.	> Forsinkelser i arbeidet som gjør at oppgaver ikke blir fullført i tide eller etter planen.	> Ha en detaljert arbeidsplan og godt dokumentert fremdriftsplan. > Fordele arbeidet mellom studentene på en fornuftig måte.	*Estimering*

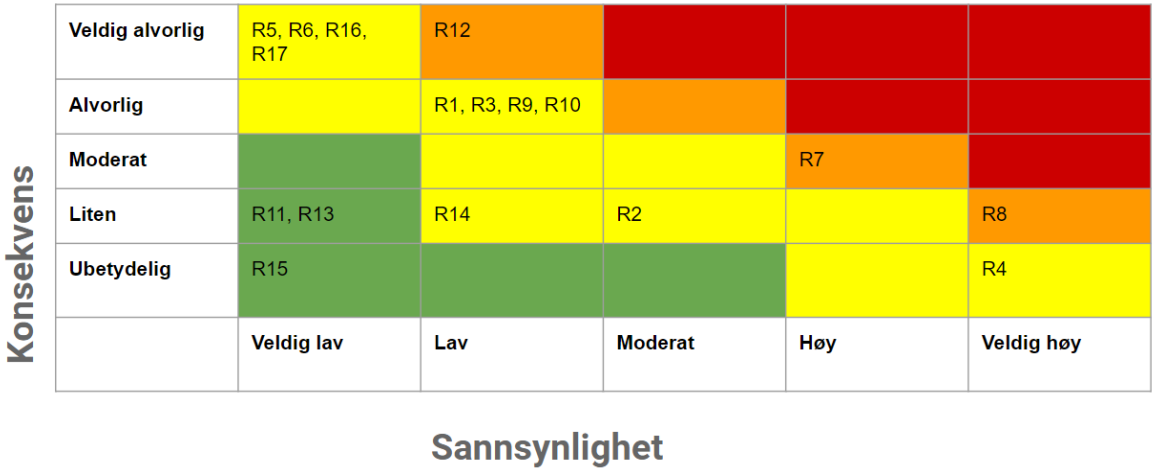
Vi implementerte mulige tiltak, og da vi forsto at arbeidsmengden ville overstige tidsrammen, ble det behov for å endre fokus på oppgavene. Heldigvis hadde vi planlagt milepæler som gjorde det enklere å prioritere bort det som kunne gjøres på et senere stadium. Et annet eksempel på en risiko vi vurderte og som vi var nødt til å håndtere var R7.

ID	Risiko	Konsekvens	Mulige tiltak	Kategori
R7	Endringer i krav-spesifikasjoner underveis.	> Endringer i kravene medfører omfattende "redesign" av løsningen, som igjen kan føre til alvorlige forsinkelser i prosessen.	> Sørge for at kravspesifikasjonene er godt utarbeidet fra start, så det ikke blir nødvendig å gjøre større endringer. > Sørge for å ha god dialog med oppdragsgiver fra start og få kartlagt mål og ønsker så presist som mulig.	*Krav*

Det var ingen store krav som medførte store endringer på løsningen, men krav fra oppdragsgiver var i startfasen løst definert. Vi fikk en oppgave med frie rammer som vi måtte utarbeide. Dette viste vi til oppdragsgiver som var positivt innstilt til våre planer. Ettersom vi startet på utviklingen av løsningen ble det etterhvert tydeligere for oppdragsgiver hva de ønsket ut av vårt arbeid. Forslag ble til ønsker som senere ble forventninger. Slik som forrige beskrevne risiko, ble det behov for å justere målene og oppgavene underveis slik at vi kunne tilpasse oss ønskene til oppdragsgiveren. Når det gjelder R12 har denne risikoen ennå ikke oppstått. Vi anser sikkerheten som er implementert for løsningen som svak og dermed utgjør en reell fare for prosjektet. Dette gjør at vi valgte å ta forhåndsregler for å beskytte oss selv, løsningen og oppdragsgiver og sterkt begrense applikasjonens tilgjengelighet. Dette minsker sannsynligheten for at R12 vil inntreffe.

ID	Risiko	Konsekvens	Mulige tiltak	Kategori
R12	Hacking, angrep på brannmur eller andre sikkerhets-mekanismer.	> Sikkerheten til løsningen blir truet og det kan potensielt gå utover personvern og sensitive opplysninger blir lekket eller uvedkommende får tilgang.	> Sørge for gode rutiner for å lagre passord og tilgang på en sikker måte. > Være bevisst på mulige trusler og følge standarder for god sikkerhetshåndtering.	*Produkt*

Vi mener den grundige risikoanalysen har bidratt til at gruppen oppnådde en bedre forståelse av de potensielle utfordringene som kan være kritiske for utviklingen underveis. Dette har gjort det mulig for oss å være proaktive og løsningsorienterte allerede fra begynnelsen av. Dette tenker vi er en nøkkelfaktor som fører til at prosjektet blir vellykket og samarbeidet i gruppen har fungert godt.

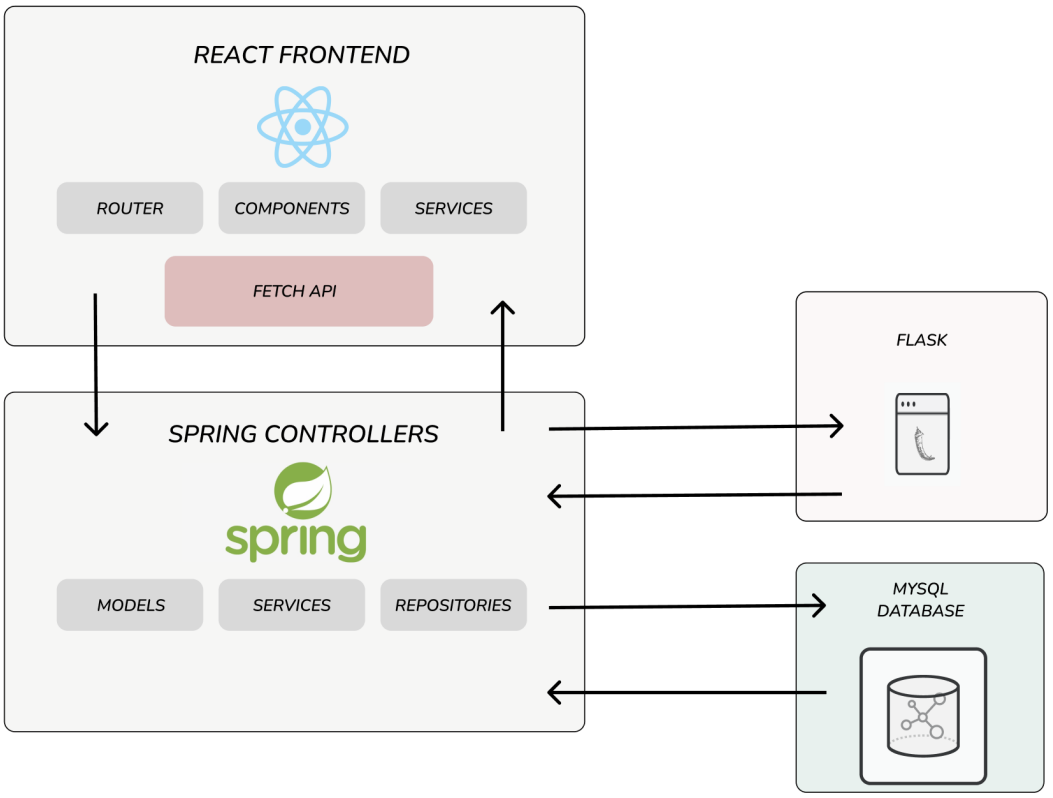


Konsekvens	Sannsynlighet				
Veldig alvorlig	R5, R6, R16, R17	R12			
Alvorlig		R1, R3, R9, R10			
Moderat				R7	
Liten	R11, R13	R14	R2		R8
Ubetydelig	R15				R4
	Veldig lav	**Lav**	**Moderat**	**Høy**	**Veldig høy**

Figur 3.6 - Vår risikoanalyse. Beskrivelse av risikofaktorene finnes i Appendiks 6

3.2. Systemarkitektur

Backend er kjernen i systemet som binder de forskjellige delene sammen. Spring Boot tilbyr et rammeverk for å utvikle Java-baserte webtjenester og RESTful API-er. REST (Representational State Transfer) API er en tilnærming til å designe og bygge webtjenester som tillater kommunikasjon mellom forskjellige datamaskiner og programmer over internett. REST API bruker den eksisterende HTTP-protokollens metoder, som `GET`, `POST`, `PUT` og `DELETE`, for å utføre operasjoner på data (IBM, 2023). Backend-applikasjonen inneholder kontroller-klasser som definerer API-endepunkter og behandler forespørsler fra klienter. Den kommuniserer med de andre komponentene i systemet for å hente, lagre og behandle data. *Figur 3.7* under viser den endelige systemarkitekturen for vårt prosjekt. Dette er den nyeste versjonen som skiller seg noe fra våre første utkast (Appendiks 7).

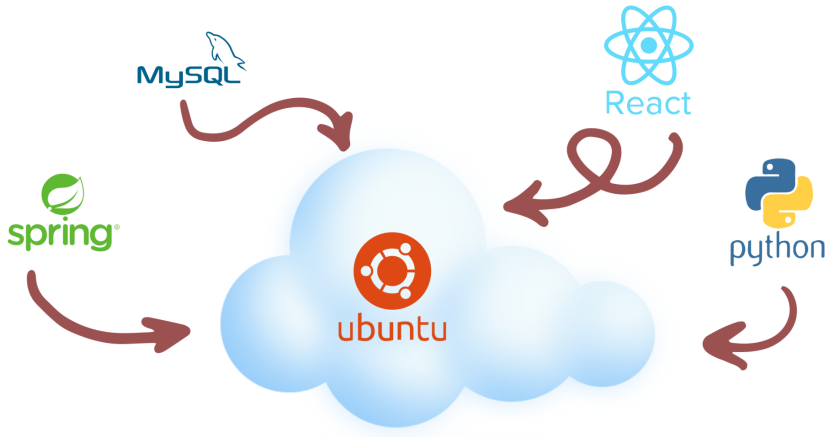


Figur 3.7 - Systemarkitektur

MySQL-databasen er integrert med JPA (Java Persistence API) og tilbyr enkel integrasjon med Spring Boot. Ved hjelp av JPA-repositorier og Hibernate-ORM, (Hibernate, 2023) kan Spring Boot forenkle repository og håndtere spørringer og transaksjoner. Vi har anvendt ORM (Object Relational Mapping) som er en teknikk som brukes i programvareutvikling for å knytte sammen OOP (Objektorientert Programmering) og relasjonsdatabaser. Frontend er brukergrensesnittet til systemet, utviklet ved hjelp av React-biblioteket. Den kommuniserer med backend via HTTP-forespørsler til API-endepunkter definert i Spring sine kontroller-klasser. Frontend henter og viser data til brukeren, og sender også brukerhandlinger og data tilbake til backend for behandling. Flask-serveren tilgjengeliggjør de trente maskinlæringsmodellene vi har utviklet. Den tar imot spørringer fra backend, sender data til modellen for prediksjon, og returnerer resultatene. Dette resulterer i en helhetlig løsning der hver komponent utfører hver sin funksjon og samhandler gjennom Spring Boot.

3.2.1. Cloud Hosting vs egen hosting av server

Kundedataene ble tilgjengeliggjort av oppdragsgiver gjennom en MySQL-server som er hostet i en Ubuntu virtuell maskin. Gruppen så på Cloud-alternativer som Amazon Web Services og Microsoft Azure. Vi gjorde egne undersøkelser og sammenligninger for å finne ut pris, effektivitet og tilpasning, men på grunn av høye kostnader og applikasjonens formål ble det sett på som mest hensiktsmessig av begge parter å bruke egen hosting av servere. Oppdragsgiver hadde et ønske om å ha minimale kostnader for utførelsen av prosjektet. I tillegg hadde oppdragsgiver tilbudt å opprette en Ubuntu-server fra leverandører de selv benytter.



Figur 3.8 - Mikro-tjeneste arkitektur med Ubuntu som server

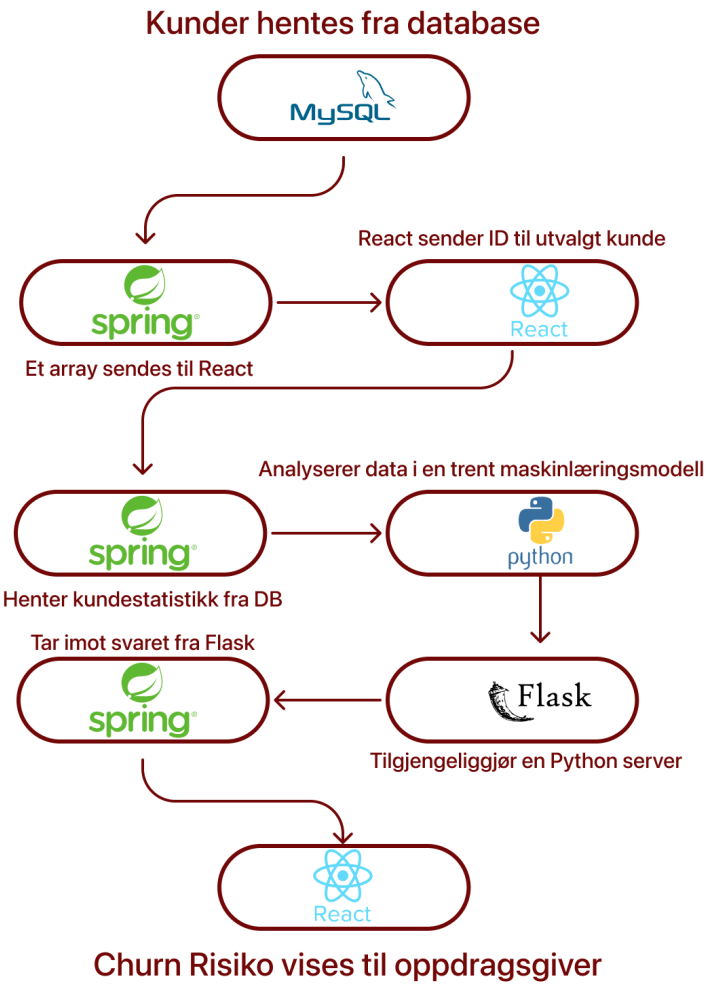
3.2.2. Monolittisk arkitektur vs mikrotjeneste-arkitektur

Monolittisk arkitektur er en måte å bygge et system som en enkelt enhet, der alle komponentene er tett koblet og avhengige av hverandre. Fordeler med monolittisk arkitektur er at den er enkel å utvikle, teste og distribuere. Ulemper er at slik arkitektur kan være vanskeligere å vedlikeholde, skalere og oppdatere over tid (Gos et al., 2020, s. 153). Mikrotjeneste-arkitektur derimot er en måte å bygge et system som består av mange små og autonome tjenester som kommuniserer med hverandre. Noen fordeler med mikrotjeneste-arkitektur er at den gir bedre modularitet, fleksibilitet og skalerbarhet. Ulempene er at den krever mer komplekst design, koordinering og overvåking (Bårdgård, 2021). Blant kravene for å ha mikrotjeneste-arkitektur er at alle tjenester (services) har sin egen database, og kobling mellom tjenester skjer gjennom API-kall. Det er tre hovedkomponenter som primært utgjør hele løsningen. Spring inneholder både backend og

frontend, Flask kjører maskinlæringsmodellen i python og MySQL-databasen kjører separat i Ubuntu-miljøet. Konklusjonen er at arkitekturen i dette prosjektet er en hybrid av både monolittisk og mikrotjeneste. Den følger prinsipper for mikro-tjeneste arkitektur, selv om ikke alle lagene inneholder sin egen database. Vi valgte å gjøre det på denne måten fordi det var lettere å jobbe på et og samme prosjekt i samme database. Vi delte opp prosjektet inn i tjenester som skal være enkle å dele opp. Dette er noe som eventuelt vil bli aktuelt dersom oppdragsgiver ønsker å videreutvikle løsningen vår.

4. Produktdokumentasjon

4.1. Introduksjon av løsningen



Figur 4.1 - Flytdiagrammet viser hvordan data overføres mellom de ulike komponentene

Oversikt over systemflyten er synlig i *figur 4.1* vist over. Som nevnt er vår løsning satt sammen av tre separate applikasjoner og en database. Vi har gjennomgående brukt

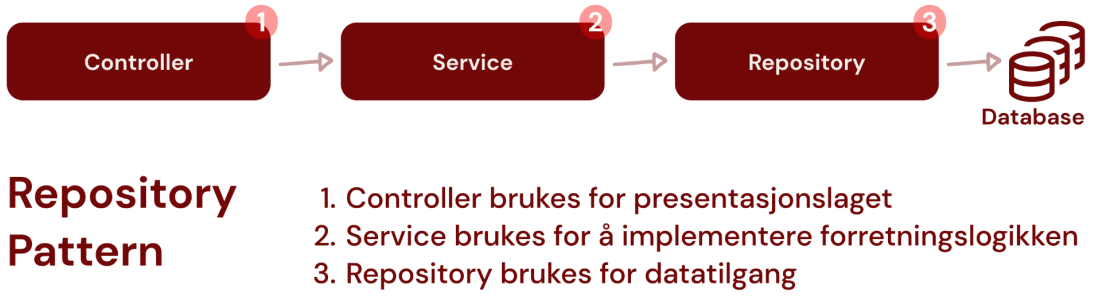
støttebiblioteker for alle tre. I backend er de definert i en "pom.xml"-fil, i frontend er disse definert i en “package.json”-fil og i maskinlæringsserveren er de en del av Python VirtualENV. Kort oppsummert: Backend er utviklet med Spring Boot og Java som håndterer logikken og databehandlingen på servernivå. Frontend er bygget med React og JavaScript som er ansvarlig for å presentere brukergrensesnittet til brukerne. Det er også rammeverket oppdragsgiver bruker. Maskinlæringsanalyse av kunder utføres med Flask API. MySQL-databasen brukes til lagring og henting av data mot Backend. Ubuntu-serveren som ble tilgjengeliggjort for gruppen av oppdragsgiver krever brukernavn og passord for å få innvilget tilgang (*figur 4.2*). Samlet sett ga dette et brukervennlig grensesnitt som kommuniserer med en pålitelig og skalerbar backend.

```
spring.datasource.url=jdbc:mysql://URL
spring.datasource.username=root
spring.datasource.password=<password>
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

Figur 4.2 - Tilkoblingsinformasjon til databasen (sensurert)

4.2. Java og Spring Boot

I strukturen til backend har vi ønsket å ha en logisk oppdeling av ansvarsområder og tjenester. Vi har gått for arkitekturen Controller-Service-Repository (Tom Collings, 2021). Dette mønsteret følger prinsippet om “Separation of Concerns” ved å dele opp ansvarsområdene til ulike komponenter. Termen “Separation of Concerns” ble først definert av Edsger W. Dijkstra i 1974 (ibid.). Man deler opp ansvarsområdene så hvert lag av applikasjonen har sin rene funksjon. Dette gjør det lettere å holde kontroll på hva som ligger hvor og kan være enklere å debugge senere. Det kan også gjøre det lettere å skalere opp senere ved at man legger til flere kontrollere, services eller repositories uten å påvirke resten av applikasjonen. En ulempe med et slikt type mønster er at det kan medføre større kompleksitet i koden. For mindre applikasjoner som vårt prosjekt var det strengt tatt ikke nødvendig, men vi ønsket å følge en slik struktur for å lære oss konseptet og bruke et etablert designmønster.



Figur 4.3 - Illustrasjon på design mønster.(Kouomeu, 2022)

En annen mulighet vi så på var å bruke arkitekturen Model-View-Controller (MVC). MVC fungerer ved at den deler ansvar mellom modell, visning og kontroller. Modellen representerer data og logikk, visningen håndterer brukergrensesnittet, og kontroller styrer kommunikasjonen mellom modellen og visningen (Tutorialspoint, u.å.). I motsetning til vår løsning skjer logikken i modell-laget som gjør at all logikken og databehandlingen samles og blir mer kompleks. I vårt prosjekt er det mye data og logikk som må bearbeides. Service-laget i Controller-Service-Repository gjør akkurat denne delen enklere og mer fleksibel.

Spring Boot tar seg av innkommende forespørsler og delegerer videre til de riktige tjenestene. Tabellene i vår MySQL-database er koblet til Java-klasser, eller `@Entities`, som gjør det mulig å behandle tabellene i databasen som objekter. De klassene som direkte korresponderer med databasetabellene ligger i prosjektet vårt under mappen `/models/entities`. Vi har også andre modell-klasser som f.eks. `SubscriberFilter.java`, og en Data Transfer Object (DTO) klasse kalt `SubcriberDTO.java`. DTO brukes f.eks. for å kombinere data fra flere tabeller inn i en klasse. Vi bruker da en felles primærnøkkel kalt subscriberID for å trekke ut de relevante dataene vi trenger. Eksempelvis ville vi ha alle attributtene til en Subscriber fra kundetabellen, men også Churn-prediksjonen som ble lagret i en annen tabell. En fordel med bruk av DTO kan være redusert nettverkstrafikk. Ved å ha færre spørringer til databasen kan det bidra til bedre ytelse. Det var også en fordel å forholde seg til kun et objekt-type som inneholdt alt vi ville sende til klienten. Da mottas det enkelt som en liste over objekter i den typen som kunne itereres over og håndteres av frontend. En ulempe med å gjøre det på denne måten kan være at det påvirker DTO-objektene om strukturen til en av tabellene i databasen blir endret. F.eks. hvis det blir lagt til eller fjernet features. Hvis prosjektet skulle utvides med en større database eller andre tabeller senere må DTO

oppdateres for å gjenspeile de nye endringene, ellers kan det oppstå avvik mellom databasen og DTO-objektene og/eller konflikt mellom klient og server.

```
// Kombiner Subscriber og MLSubscriber data til en liste med SubscriberDTO objekter
            List<SubscriberDTO> subscriberDTOs = new ArrayList<>();
            for (Subscriber subscriber : subscribers) {
                MLSubscriber mlSubscriber = mlSubscriberMap.get(subscriber.getId());
                SubscriberDTO subscriberDTO = new SubscriberDTO();
                subscriberDTO.setId(subscriber.getId());
                subscriberDTO.setSubscriptionName(subscriber.getSubscriptionName());
                subscriberDTO.setStatus(subscriber.getStatus());
                subscriberDTO.setStatusDesc(subscriber.getStatusDesc());
                subscriberDTO.setBucketSize(subscriber.getBucketSize());
                subscriberDTO.setAge(subscriber.getAge());
                subscriberDTO.setStartDate(subscriber.getStartDate());
                subscriberDTO.setEndDate(subscriber.getEndDate());

                if (mlSubscriber != null) {
                    subscriberDTO.setChurnRisk(mlSubscriber.getChurn());
                }
                subscriberDTOs.add(subscriberDTO);
            }
            return subscriberDTOs;
        } catch (Exception e) {
            logger.error("Kunne ikke finne subscriber, fra MainService " + e.getMessage());
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, e.getMessage());
        }
```

Figur 4.4 - Utklipp fra kode som slår sammen data fra 2 tabeller i et objekt

"`/filter`"-endepunktet har til hensikt å ta imot et SubscriberFilter-objekt fra frontend. I Repository tar `@Validated` seg av input-validering for å sjekke at det er gyldige verdier før forespørselen blir sendt videre til neste lag. `@RequestBody` er en REST-annotasjon som forteller Spring at forespørselen, her et JSON-objekt[28] fra frontend, skal bli konvertert til et Java-objekt som vi har definert (Spring Framework, u.å.).

[28] JavaScript Object Notation, universal format for å dele objekter mellom teknologier.

```
@PostMapping(path = "/filter")
    public ResponseEntity<?> getFilteredSubscribers(@Validated @RequestBody SubscriberFilter filter,
BindingResult bindingResult) {
        logger.info("Received filter request with filter: {}", filter);
        if (bindingResult.hasErrors()) {
            List<String> errors = bindingResult.getAllErrors().stream()
                    .map(ObjectError::getDefaultMessage)
                    .toList();

            logger.info("Error fra server:" + errors);
            return ResponseEntity.badRequest().body("Feil med input validering:" + errors);
        }
        List<SubscriberDTO> filteredSubs = mainService.getSubscribers(filter);
        if (filteredSubs.isEmpty()) {
            return ResponseEntity.status(HttpStatus.NO_CONTENT).body("Ingen subscribers funnet for
dette søket.");
        }
        return new ResponseEntity<>(filteredSubs, HttpStatus.OK);
    }
```

Figur 4.5 - Illustrasjon av /onecall/filter endepunktet i MainController

```
models
 ├── LoginRequest.java
 ├── SubscriberDTO.java
 ├── SubscriberFilter.java
 └── entities
     ├── MLSubscriber.java
     ├── MachineLearningPred.java
     ├── Subscriber.java
     └── SubscriberUsage.java
```

Figur 4.6 - Utklipp av strukturen i models mappen

Controller/API-laget har ansvaret for å eksponere tjenestene og endepunktene så de kan brukes av andre tjenester. I vårt system er det React og Flask som vil bruke dette API'et. Det er definert endepunkter som korresponderer med ulike metoder. Det er vår `MainController.java`, som blant annet tar seg av forespørsel til `MainService.java`. Alle endepunktene under MainController starter med adressen `/onecall` (*figur 4.7*).

```
controllers
 ├── AuthController.java
 ├── ChurnPredictionController.java
 ├── MainController.java
 ├── MetricsController.java
 ├── SubscriberController.java
 └── SubscriberUsageController.java
```

Figur 4.7- Utklipp av controllers mappen

Som nevnt tar servicelaget seg av databehandlingen og logikken. I vår applikasjon har vi lagt mesteparten av logikken i `MainService.java`. MainService innsetter tilgang til tre forskjellige repositories gjennom konstruktøren. `@Service` gjør at Spring Boot automatisk skanner og registrerer denne klassen som en Service-entitet. Dette er også kjent som "dependency injection" hvor avhengighetene blir introdusert eksternt gjennom konstruktører, metoder eller egenskaper fremfor at et objekt oppretter og håndterer sine egne avhengigheter. Dette betyr at et objekt ikke trenger å være klar over hvordan avhengighetene opprettes, men kan fokusere på sitt eget ansvarsområde. Vår Service-klasse har tilgang til Repository/database-laget, og siden Service blir injisert i kontrolleren er dette nå også tilgjengelig implisitt. Service fungerer altså som et mellomledd mellom Controller og Repository. Det er for å unngå at det kommer en request direkte til et API som kaller rett på datalagrings-laget.

```
@Service
public class MainService {

    private final SubscriberRepository subscriberRepository;
    private final MLSubscriberRepository mlSubscriberRepository;
    private final MachineLearningRepository machineLearningRepository;

    final Logger logger = LoggerFactory.getLogger(MainService.class);

    public MainService(SubscriberRepository subscriberRepository, MLSubscriberRepository
mlSubscriberRepository,
                       MachineLearningRepository machineLearningRepository) {
        this.subscriberRepository = subscriberRepository;
        this.mlSubscriberRepository = mlSubscriberRepository;
        this.machineLearningRepository = machineLearningRepository;
    }
```

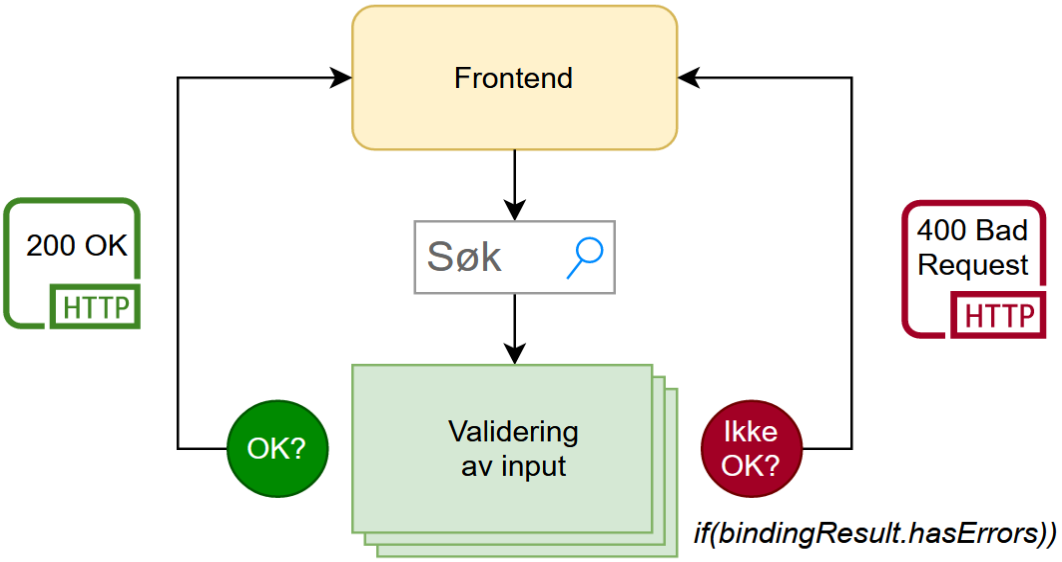
Figur 4.8 - Koblingen mellom Service-laget og Repository

```
services
├── MainService.java
├── MetricsService.java
├── PredictionService.java
└── SubscriberUsageService.java
```

Figur 4.9 - Strukturen i services mappen og oversikt over serviceklasser

Hvis forespørselen som kommer inn fra et kundesøk i applikasjonen ikke samsvarer med grenseverdiene, blir inputfeilene samlet ved hjelp av et BindingResult-objekt og sendt tilbake til klienten med en HTTP statuskode "`Bad Request`" (400). Det tar den innebygde klassen

ResponseEntity seg av. Valideres forespørselen returneres statuskode "`OK`" (200) og søkeresultatet i form av en liste med SubscriberDTO objekter.



Figur 4.10 - Validering av input og HTTP status retur.

Våre Repositories er bygget på høynivå-abstraksjon som gjør det lettere å implementere tilgang til en database ved hjelp av Java Persistence API (JPA). (Oracle, 2015). I eksempelet under (*figur 4.11*) definerer vi et MachineLearningRepository-grensesnitt som utvider det innebygde JpaRepository. Den angir at Repository skal håndtere entiteter av typen MachineLearningPred-objekt med en unik ID. Når vi oppretter en instans av typen MachineLearningRepository, får vi tilgang til en rekke ferdig implementerte metoder for å utføre operasjoner på MachineLearningPred-entiteter. Eksempler på dette er `save()`, `findAll()`, `findById()`og `delete()`, og mange flere. JPA lager automatisk SQL-spørringer basert på metode-navnet som reduserer vår arbeidsmengde med data-aksesslaget(DAL).

```
@Repository
public interface MachineLearningRepository extends JpaRepository<MachineLearningPred, Integer> {
    MachineLearningPred findMachineLearningPredById(Integer id);
}
```

Figur 4.11 - Utklipp av MachineLearningRepository

Siden vi hadde tjenester fra forskjellige kilder, var det nødvendig for oss å sette opp tillatelse-håndtering. Vi hadde kontroll over serveren, så vi kunne benytte oss av Cross-Origin Resource Sharing (CORS) i konfigurasjonsinnstillingene. CORS er en standard som tillater eller nekter forespørsler som kommer fra et annet opphav (Mozilla, u.å.).

Vi implementerte logging av hendelser som hjelper med konkrete og verdifulle beskrivelser av det som blir kjørt. Dette er svært behjelpelig for feilsøking og feilhåndtering. Når det kommer til unntakshåndtering (Exception Handling), blir alt håndtert gjennom standardmetoder fremfor egendefinerte. Exceptions fanger opp feil eller uforutsette hendelser som kan oppstå ved kjøringen av applikasjonen (Grønning & Vihovde, 2015). Tanken bak de egendefinerte Exceptions var å kunne gi konsise og gode tilbakemeldinger til både serveren og klienten hvis noe gikk galt. Fordelen med å ha egendefinert feilhåndtering er at det blir lettere å løse feilen ettersom man raskere finner hva som har gått galt og på hvilket stadie i utføringen feilen hadde skjedd. Den opprinnelige kjøreplanen hadde ikke dette under målet om en MVP og ble derfor en nedprioritert arbeidsoppgave. Ettersom standardmetodene er implementert, er ikke egendefinerte Exceptions en nødvendighet. Derimot når applikasjonen ekspanderer og øker i kompleksitet vil egendefinerte exceptions bli svært fordelaktig.

4.3. Frontend (React.js)

Frontend-utvikling omfatter tradisjonelt sett bruk av JavaScript, HTML og CSS. I dagens landskap er fokuset i stor grad rettet mot ulike rammeverk som forenkler utviklingsprosessen, automatiserer oppgaver og muliggjør gjenbruk av kode (Hutagikar et al., 2020, s.3317). Disse rammeverkene er hovedsakelig basert på JavaScript eller TypeScript. I et utviklingsprosjekt velger man det rammeverket som egner seg best for det man utvikler. Det er små, men viktige forskjeller mellom de ulike rammeverkene. For eksempel kan Angular være det riktige valget hvis man har behov for toveis data-binding. React er et JavaScript/Typescript-basert bibliotek for å bygge brukergrensesnitt for webapplikasjoner. En av de største fordelene med React er at det bruker en deklarativ tilnærming til å bygge brukergrensesnitt, som gjør det enklere å lage interaktive og dynamiske applikasjoner.

4.3.1. Struktur

Applikasjonen vår utviklet i React er delt inn i komponenter. En komponent er en gjenbrukbar byggeblokk som består av HTML, CSS og JavaScript-kode. Komponentene kan

enten være basert på klasser eller funksjoner. Bruk av Klassekomponenter var den opprinnelige måten å definere React-komponenter på. De kan bruke livssyklusmetoder, og de har også mulighet til å lagre tilstand i komponenten selv og har derfor mer funksjonalitet enn funksjonskomponenter. Klassekomponenter er også kompatible med tredjeparts-biblioteker og React-komponenter som er basert på klasser. Funksjonskomponenter er enklere og lettere å forstå enn klassekomponenter (Wieruch, 2019). De er vanligvis kortere og krever mindre kode, og gir derfor også bedre ytelse enn klassekomponenter. De er definert som en funksjon som tar inn props-objektet[29] som parameter og returnerer en React-komponent. Etter hvert som React-økosystemet har utviklet seg, har funksjonskomponenter økt i popularitet. I det siste har de blitt mye mer komplekse som har ført til at det er funksjonskomponenter som er anbefalt for å lage enkle og rene komponenter (ibid.). Funksjonskomponentene er brukt gjennom hele utviklingen. Vi ville gjennom søkeskjemaet vårt gjøre et POST-kall til backend med søkekriteriene som returnerte et resultat. Vi ville vise dette resultatet i kundetabellen, men det var ikke mulig å sende data mellom søket og tabellen. For å få til dette ble løsningen å lage en felles forelderkomponent som inneholdt søkeskjemaet og tabellen. Når brukeren trykker på søkeknappen ble det brukt en callback-funksjon for å sende dataen fra skjemaet opp et nivå.

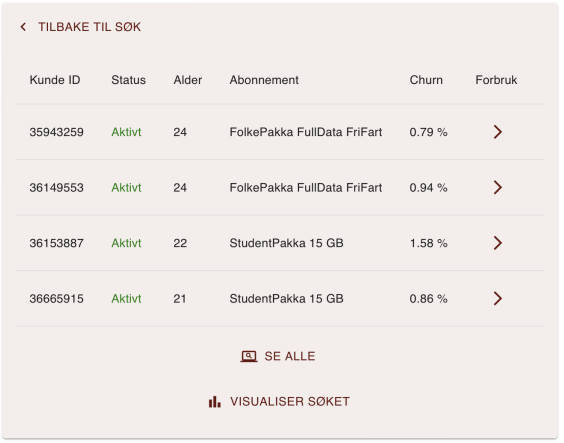
```
const handleSubmit = (formData) => {
constcredentialsEncoded=btoa(`${credentials.username}:${credentials.password}`);


   fetch('http://localhost:8080/onecall/filter', {
       method: 'POST',
       headers: {'Content-Type': 'application/json',
               Authorization: `Basic ${credentialsEncoded}`},
       body: JSON.stringify(formData)
   })
       .then(response => {
           if (response.ok) {
               return response.json();
           } else {
               console.log("Fra response i handleSubmit");
               console.log(response);
               return response.text().then(error => {
                   throw new Error("Kunne ikke finne subscriber" + error);});
           }
       })
       .then(data => {
           setTableData(data);
       })
       .catch(error => {
           alert(error.message);
           console.error(error);
       });
};
```

Figur 4.12 - Utdrag fra koden der kall til backend blir gjort.

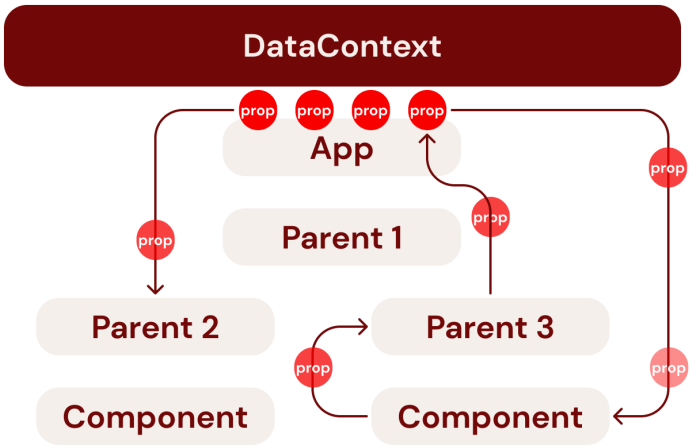
[29] Data som utveksles mellom ulike komponenter.

Figur 4.13 - Søkefelt



Figur 4.14 - Resultat

`formData` tas imot av ParentCard, som sender en POST-request til backend. Kontekst er en mekanisme som gjør det mulig å dele data på tvers av komponenter uten å måtte sende props gjennom alle mellomliggende komponenter (React, 2023).



Figur 4.15 - Diagram over dataflyt i frontend

Når dataen blir satt gjennom `setAllMLSubscribers(data)` gjør vi den tilgjengelig til alle komponenter lenger ned i applikasjonsstrukturen. Dette gjøres gjennom DataContext som også blir brukt i `/Visualizations` som er en egen side som visualiserer søkeresultatet i grafer. Derfor var det også viktig å gjøre denne tilgjengelig gjennom DataContext.

Ved siden av den viktigste komponenten (App.js) som bygger selve nettsiden, så har vi følgende struktur:

```
// Eksempel
function Dashboard() {
     return (
          <Box style={{marginTop: '1.8em'}} sx={{
flexGrow: 0 }}>
               <ParentCard/>
               <ChartBox/>
          </Box>
) }
  export default Dashboard;
```

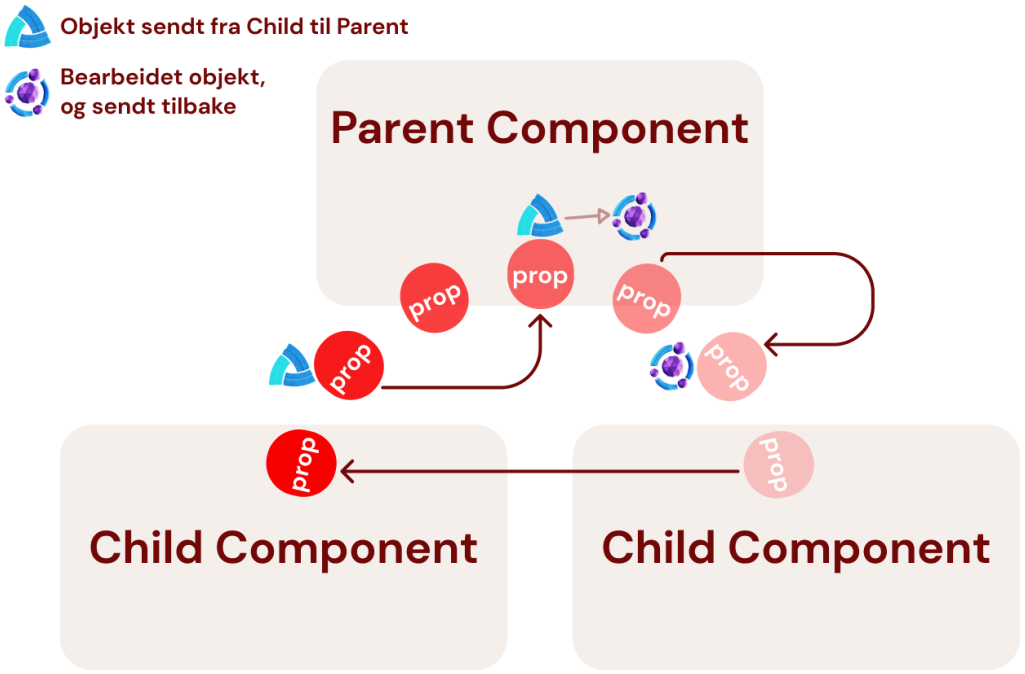
Pages *- det er sider i webapplikasjonen, og de er bare en samling av ulike komponenter. Disse komponentene formaterer data vist til kunde (oppdragsgiver)*

```
// Eksempel
function ParentCard() {
    return (
         <Grid container spacing={2}>
             <Grid item xs={12}>
                 <SearchForm onSubmit=
{handleSubmit}/>
             </Grid>
             {tableData && tableData.length > 0 ? (
                 <Grid item xs={12}>
                     <UserInfo data={tableData}/>
                 </Grid>
             ) : (
                 <Grid item xs={12} md={6} >
                     <ShowStatic/>
                 </Grid>
             )}
         </Grid>
     );
 }
  export default ParentCard;
```

Hovedkomponenter *(også kjent som Parent Components) - er komponenter som inneholder flere mindre komponenter. Det kan f.eks. være blanding av en tabell, et bilde og en graf, men i sin helhet er det innhold som er synlig for brukeren*

```
// Eksempel
function UserInfo(props) {
return (
         <Grid container spacing={2}>
             <Grid item xs={4}>
                 <Card >
                     <CardContent >
                         <h2 style={{textAlign:
'center'}}>
         Bruker #{userData.id}</h2>
                         <h3>Status: <span>
{userData.statusDesc}</span></h3>
                         <h3>Alder: <span>
{userData.age}</span> </h3>
                         <h3>Abonnement: <span>
{userData.subscriptionName}</span>
                        </h3>
                         <h3>Kunde siden: <span>
{userData.startDate}</span></h3>
                     </CardContent>
                 </Card>
             </Grid>
  )
}
export default UserInfo;
```

Komponenter *(også kjent som Child Components) - de aller minste komponenter som ofte gjør kun en ting, f.eks. henter data fra en database til en tabell, eller sender data ved ulike JSON-kall.*



Figur 4.16 - Visualisering av flyten mellom komponentene

Som forklart ovenfor og vist med deler av koden, så er alt organisert hierarkisk, og komponentene er satt sammen til et helhetlig brukergrensesnitt. Dashboard (under Pages) er det øverste nivået i hierarkiet og fungerer som en side som vises til kunden. ParentCard (under Hovedkomponenter) er en underordnet komponent til Dashboard, og samler små komponenter som vises på siden. Denne komponenten tillater også dataflyt mellom mindre komponenter. I vårt eksempel sender søke-komponenten data til backend, og resultatet som kommer tilbake fra backend blir vist i en komponent som viser utskrift av det som er søkt etter. UserInfo (under Komponenter) er igjen en underordnet ParentCard og ansvarlig for å hente informasjon om en kunde fra databasen. Dette betyr at hele frontend er bygget opp av flere komponenter slik at hver enkelt komponent har sin spesifikke funksjonalitet.

Flere komponenter

Sorting & Selecting Tables er en serie av tabeller som gir en enkel og intuitiv måte å lage sorterings- og filtreringsfunksjonalitet for tabeller. For oppdragsgiver var det viktig å ha enkel navigering i tabellen med veldig mange kunder. De ville sortere kunder etter churn risiko, alder eller status.

Grid tilpasser seg skjermstørrelse og orientering, og sikrer konsistens på tvers av ulike oppsett. Grid skaper visuell konsistens mellom innhold samtidig som den tillater fleksibilitet for ulike design. Material Design sin responsive brukergrensesnitt er basert på et 12-kolonners grid-layout.

Card gir en enkel og fleksibel måte å presentere innhold på. På grunn av universell utforming har vi valgt å implementere Cards på nesten alle deler av applikasjonen. I tillegg til Grid tilpasser den nettsiden til ulike skjermer.

Appbar er en komponent som ikke brukes aktivt på nettsiden ettersom fokuset ligger på oversikt gjennom et dashboard, og filtrering basert på innhold fra database, samt AI-algoritmen vår.

Date and Time Pickers brukes mest for å velge dato og tid når man vil søke på kundene basert på start og sluttdato.

Tabs organiserer og tillater navigering mellom grupper av innhold som er relatert og på samme nivå av hierarki. Oppdragsgiver ønsker f.eks. å se oversikt over databruk for en kunde fordelt på flere måneder.

4.4. Design

Vårt designsystem består av verktøy, komponenter, farger, font og prosesser som er i harmoni med OneCall sitt designsystem. Det overhengende mål var at det skal være lett å gjenkjenne at produktet kommer fra OneCall. I tillegg var det et ønske fra begge parter om å bruke React UI-komponent fordi det er åpen kildekode, lett tilgjengelig og kan tilpasses. Ettersom fokuset i prosjektet ikke var på selve designet, valgte vi Material UI slik at komponentene kan endres av oppdragsgiver etter eget ønske og behov. Fordelen med å bruke slike verktøy og komponenter er at de er bygget for å følge designprinsipper og støtte for universell utforming.

Verktøy

Material UI

Material UI (MUI) er et gratis og åpent bibliotek med React UI-komponenter som implementerer Googles Material Design. Noen fordeler med Material UI er at det følger et konsistent og moderne designsystem med støtte for flere temaer og tilpasninger (Mui, 2023).

Recharts for React

Recharts for React er et bibliotek med React-komponenter som tilbyr et enkelt og kraftig API for å opprette interaktive diagrammer og grafer, som kan brukes til å visualisere komplekse datasett. Disse ulike grafene gjorde det enkelt for oppdragsgiver å få en oversikt over forbruket av deres kunder, samt churn risiko på en intuitiv måte (Recharts, 2023).

Figma

Figma er en skybasert designplattform som brukes av designere, utviklere og samarbeidsgrupper for å opprette, dele og samarbeide om digitale designprosjekter. Plattformen tilbyr et bredt utvalg av funksjoner og verktøy for å hjelpe brukerne med å opprette alt fra wireframes til komplekse interaktive prototyper. Vi ønsket å ha alle opplysninger, standarder og retningslinjer knyttet til design på ett sted. Figma var et produkt som ble brukt for å utvikle wireframe og poster (Figma, 2023).

Farger og kontrast

Farger er delt i de som brukes for forgrunn og bakgrunn. Her var det mest fokus på kontrast og tilgjengelighet.

Hovedfarge	Alternative farger		
#720707	#8A0808	#A20A0A	#BA0B0B

Hovedfarge	Alternative farger		
#EE0000	#FF0808	#FF2222	#FF3C3C

Alternative farger			**Hovedfarge**
#DBC5BA	#E3D3CB	#ECE1DB	#F5EFEC

Farge (bakgrunn og forgrunn)	**Kontrast**
Eksempel på tekst	**10.54:1**
Eksempel på tekst	**10.54:1**

Fargene vi har brukt i vår applikasjon er hovedsakelig #720707 og #F5EFEC, som oppnår kravet til kontrast. I arbeidet med undersøkelse av fargevalg oppdager gruppen at oppdragsgiver IKKE har godkjent kontrast i noen av de elementene på nettsiden (eks. Nettbutikk, Prøvekjøp).

OneCall farger (bakgrunn og forgrunn)	**Kontrast**
Prøvekjøp	**3.85:1**
Totalpris	**4.2:1**
Chat med Uno	**4.18:1**

> "*Kravet til kontrast gjelder for både tekst og bilde av tekst. Liten tekst krever høyere kontrast for å ha god lesbarhet, enn stor eller fet tekst. WCAG 2.0 stiller derfor ulike krav til kontrastverdi for stor og liten tekst. Kravet til kontrast mellom tekst og bakgrunn er:* ***4.5:1*** *for liten tekst* ***3.0:1*** *for stor eller fet tekst Kontrastforholdet i kravet på 4.5:1 er satt fordi dette vil kompensere for synstapet som er vanlig når man blir eldre. Denne kontrastverdien vil gjøre teksten lettere å lese for alle brukere, f.eks. når en er ute i skarpt sollys.*" (Universell Utforming Tilsynet, 2023)

Oppdragsgiveren er informert om at deres nettside inneholder fargevalg som ikke innfrir krav til kontrast.

Font

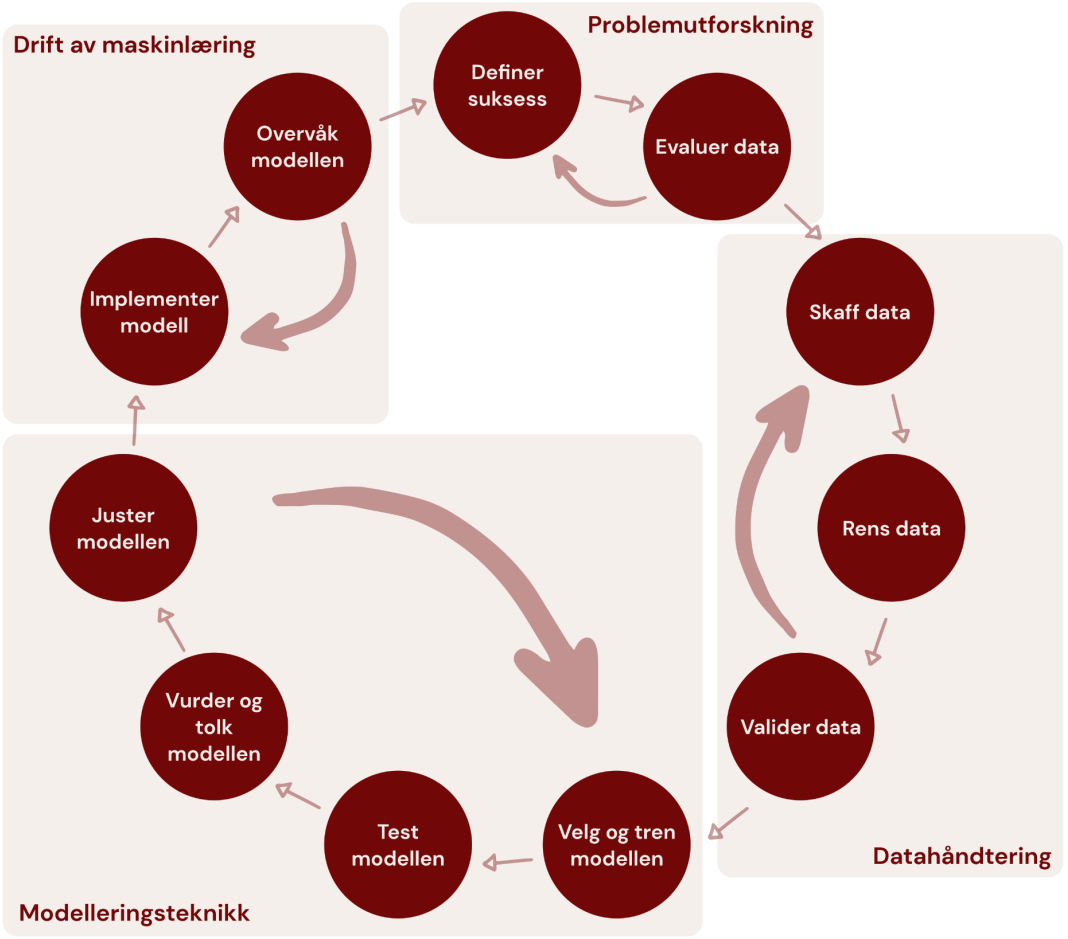
På sine sider bruker OneCall en font som heter *Stabil Grotesk*. Denne fonten var ikke åpent tilgjengelig og ville kostet oss 70€, så vi ble nødt til å finne en annen. DM Sans har en rekke egenskaper som gjør den til en god skrifttype å bruke. Den er tilgjengelig i fire forskjellige stiler: regular, italic, medium og medium italic. I tillegg er den veldig lesbar på skjermen, noe som gjør den godt egnet for digitale plattformer som nettsider, apper og e-bøker. Skrifttypen passer også godt for trykte materialer som aviser, magasiner og bøker.

Font	**DM Sans**				
HTML tag	<h1>	<h2>	<h3>	<h4>	<p>
Normal	H1	H2	H3	H4	p
Bold	**H1**	**H2**	**H3**	**H4**	**p**
Italic	*H1*	*H2*	*H3*	*H4*	*p*

4.5. Maskinlæringsmodellen

4.5.1. Maskinlæringsprosessen

Maskinlæringsprosessen er en omfattende prosess som krever mye arbeid. Den består stort sett nøyaktig av den samme syklusen med iterative segmenter som illustrert på *figur 4.17* under. Selv om dette er en typisk prosess og består av mange av de samme handlingene, så blir hver modell unikt satt sammen gjennom ulike faktorer som datatilgjengelighet, rensing, analysering, valg av algoritme og målsetting (Saltz, 2022). Vi har forsøkt ulike tilnærminger som blir beskrevet i dette kapittelet.



Figur 4.17 - Maskinlæringsprosess (Saltz, 2022)

Churn hos OneCall

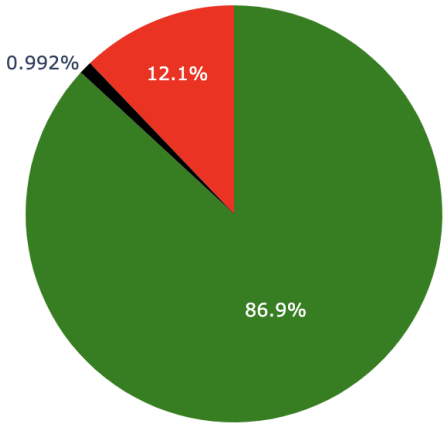
OneCall har ca. 400.000 kunder i Norge (T. Felix, personlig kommunikasjon, 19.05.2023). Ettersom vårt datasett inneholder data om ca. 25000 av kundene betyr det at vi har tilgang til kun 7% av kundebasen. Resultatene under er dermed ikke representative for faktisk churn hos OneCall. Vi vet ikke hva som er OneCall sin reelle churn rate. Dette for å opprettholde konfidensialitet ovenfor oppdragsgiver og skjerme forretningshemmeligheter. Utfallet vi har fått gjennom prosjektet gjenspeiler derfor først og fremst modellens kvalitet basert på et utvalg av OneCall sin kundebase.

For datasettene vi har tilgang til finner vi at totalt 23.380 kunder starter medlemskap i løpet av perioden mars - november 2022, men 17.865 kunder velger å forlate i samme periode. Det betyr ikke at OneCall har en høy churn, men at OneCall har netto økning i antall kunder.

Kundebasen øker med i snitt 1985 kunder pr. mnd tilsvarende 23%. I figur *4.18* kan man se et utsnitt av dataene. Det er verdt å nevne at bare fordi noen kunder regnes som churnet i en spesifikk måned er det ikke nødvendigvis tilfelle at de ble aktive i samme måned eller i det hele tatt noen av månedene vi har tilgang til. Likevel, om vi ser på vårt ferdige datasett etter rensing kommer vi frem til at vi har ca. 12% som er inaktive mot ca. 87% aktive kunder (ca. 1% er sperret).

Month	Active_Subscribers	Churned_Subscribers	Ratio	Gained
03, 2022	1878	721	0.3839	1157
04, 2022	2098	956	0.4557	1142
05, 2022	2405	876	0.3642	1529
06, 2022	2883	923	0.3202	1960
07, 2022	2800	770	0.2750	2030
08, 2022	3764	716	0.1902	3048
09, 2022	3833	401	0.1046	3432
10, 2022	3266	141	0.0432	3125
11, 2022	453	11	0.0243	442

Figur 4.18 - Oversikt over startede og avsluttede medlemskap fordelt på måned



Figur 4.19 - Kundefordeling på status. Grønn er aktive, rød er inaktive, sort er sperret

OneCall hadde allerede laget en ml-modell basert på Random Forest som skulle predikere churn, men som var lite brukt innad i selskapet. Grunnen til det er at den var mangelfull og slo kun ut på feil overforbruk (B. Hermansen, personlig kommunikasjon, 06.03.2023). Litt nærmere forklart: En kunde i Estland nær grensen til Russland fikk en utenfor EU-tilkobling som økte regningen betydelig, og det var fare for at kunden ville slutte å bruke tjenesten. Vi fikk ikke mer innsikt i hvordan denne prediksjonen var utformet eller anvendt, annet enn at

den ga gjennomsnittlig 12-14% churn-prosent på kunder. Som vi nevnte kort i kapittel 2. Teori om churn og maskinlæring, hadde oppdragsgiver en rekke spørsmål rundt databruk, pris og "value loads" som de gjerne ønsket mer klarhet i. De ønsket også innsikt i hvordan de ulike pakkene blir brukt, hvor mye data noen bruker av tilgjengelig data og om andelen av forbruk har innvirkning på churn. Oppdragsgiver var især interessert i å finne kandidater og det optimale punktet som hjalp dem å bestemme om det skulle gjennomføres oppsalg eller nedsalg basert på kundens forbruk og behov. Men vi ble enige om at det som skulle være vårt fokus var å se først på churn-risiko, og deretter databruk opp mot churn.

Beskrivelse av datasettene

Oppdragsgiver bearbeidet to datasett til oss som ble lastet opp til Ubuntu-serveren der vi kunne aksessere dataen via MySQL. Den ene inneholdt kunde-attributter (*stud_subscribers*), den andre inneholdt forbrukstransaksjoner (*stud_cdr*). Datasettet som beskrev kunder og deres abonnementstyper hadde 13 features og 28.895 rader.

Beskrivelse av stud_subscribers

SUBSCRIBERID	er en åttesifret ID som var unik for hver kunde (Primærnøkkel).
PRIS	viser hvor mye hver kunde betalte for abonnementet.
SUBSCRIPTIONTYPE	var en numerisk verdi for abonnementstype.
SUBSCRIPTIONDESC	var tekstforklaring på abonnementstypen.
STATUS	var en numerisk verdi for kundens abonnementstatus.
STATUSDESC	var tekstforklaring på kundens status.
OPERATORPRICEPLAN	viste hvordan betalingen var gjort.
BUCKET_MB	viste hvor mye data pakkeløsningen inneholdt.
USAGEMB	var en kategori laget av OneCall intendert for vår bruk.
FAMILYSERVICE	viser om kunden har et abonnement omfattet av Familieabonnement.
AGE	viste kundens alder.
STARTDATE	viste når kunden startet sitt abonnement.
ENDDATE	viste når kunden valgte å avslutte abonnementet sitt.

Datasettet som viste kunders CDR hadde 7 features og inneholdt 14.648.566 rader.

Beskrivelse av stud_cdr

SUBSCRIBERID	er en åttesifret ID som var unik for hver kunde (ikke Primærnøkkel).
CHARGE	viste enten antall (f.eks. SMS) eller sekunder(f.eks. ringeminutter).
VOLUME	viste størrelsen på bruken i byte.
RATEZONE	viste det geografiske området hvor bruken foregikk.
RATEPRODUCT	viste hvilken type handling som var gjort, f.eks. samtale ut, samtale inn, sende SMS, motta SMS eller nettsurfing.
INCOMEPERIOD	forteller hvilken måned cdr-handlingen skjedde.
CLEARINGDATE	er dato og tidspunkt for oppstarten av bruksdata-transaksjonen.

Vi vil nevne at dataene vi fikk utdelt i CDR gjaldt for perioden 1.juni-13.november 2022. Det var noe skjev fordeling hvor transaksjoner for august, september og oktober viste 84% av datasettet, mens juni og juli hadde under 0,5% av datasettet. November var ikke komplett da forbruket sluttet midt i, men teller likevel 15% av total data.

month	transactions	percentage_ra...
June	21317	0.14552
July	34169	0.23326
August	3477533	23.73975
September	4126699	28.17135
October	4777642	32.61508
November	2211206	15.09503

Figur 4.20 - CDR fordelt på måned, antall og hvor stor prosentandel av datasettet transaksjonene utgjør.

Maskinlæringsalgoritme Logistic Regression

Vi fikk ekte data med innsikt om kunder som hadde churnet og kunder som var aktive. Dette skulle vi bruke for å trene en modell som skulle forutsi hvor stor sannsynlighet det var for at en gitt aktiv kunde kom til å churne i fremtiden. Dette er egentlig et regresjonsproblem[30]. Men gjennomgående i forskningsartiklene vi skrev om i kapittel 2. Teori om churn og

[30] Omtales mer i gjennomgang av Random Forest Regression i den endelig versjonen.

maskinlæring anvendes hovedsakelig klassifiseringalgoritmer. Vi valgte derfor å starte med en typisk algoritme for en slik problemstilling som kunne gi oss en basis å jobbe videre med. Basert på dette valgte vi å anvende maskinlæringsalgoritmen Logistic Regression, som vi hadde kjennskap til fra tidligere.

Logistic Regression er en populær maskinlæringsmodell som ligger innenfor den delen av maskinlæring som kalles Supervised (javatpoint, u.å.). Supervised maskinlæring krever treningsdata som inneholder både input og de forventede output-verdiene. I supervised læring er målet å lære modellen å gjøre prediksjoner på nye, ukjente data basert på mønstrene den har lært fra treningsdataene. Innenfor Supervised har man mulighet for både Klassifisering og Regression. Ved klassifisering er målet å gruppere output i ulike klasser, i regression er målet å estimere eller forutsi kontinuerlige numeriske verdier som output. Tross bruken av "regression" som brukes ved kontinuerlige verdi-prediksjoner er logistic regression-algoritmen som oftest brukt til å forutsi binære utfall, eksempelvis churn (Aktiv v.s. Kansellert). Det er en modell som kan gi en baseline for et godt resultat veldig raskt i tilfeller hvor man har tilgang på mye data. Blant enkelte som forsker på churn-analyse, ble logistic regression nevnt som noe man hadde brukt, men ikke som den som gav det beste resultat (Berkeley School of Information, 2017).

Versjon 1

En utfordring var at vi hadde begrenset kjennskap til maskinlæring. Vi spurte oppdragsgiver om hjelp og innsikt i deres implementerte maskinlæring, men de oppfordret oss til å eksperimentere på egen hånd. Vår basiskunnskap i maskinlæring var fra et introduksjonsfag til kunstig intelligens som lærte oss det overordnede (DAVE3625 Introduksjon til Kunstig Intelligens). For tips om fremgangsmåte ble vi henvist denne artikkelen skrevet av Felix Frohböse (2020) kalt *Machine Learning Case Study: Telco Customer Churn Prediction | by Felix Frohböse | Towards Data Science*. Videre forsøkte vi å finne andre løsningsmetoder, men samtlige var lite anvendelige ettersom de viste til features vi ikke hadde til disposisjon.

Tanken bak første versjon var å samle så mye data som mulig fra begge tabeller. Vi gjorde grunnleggende bearbeiding av dataene. Dette gjorde vi ved å gjennomføre diverse endringer som forenklinger, legge til features og endre formater på datatypene.

```
Int64Index: 875803 entries, 0 to 899999
Data columns (total 14 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   SUBSCRIBERID      875803 non-null  int64
 1   PRIS              875803 non-null  int64
 2   SUBSCRIPTIONTYPE  875803 non-null  int64
 3   STATUSDESC        875803 non-null  int64
 4   BUCKET_MB         875803 non-null  int64
 5   AGE               875803 non-null  int64
 6   STARTDATE         875803 non-null  int64
 7   ENDDATE           875803 non-null  int64
 8   CHARGE            875803 non-null  int64
 9   VOLUME            875803 non-null  float64
 10  RATEZONE          875803 non-null  int64
 11  RATEPRODUCT       875803 non-null  int64
 12  CLEARINGDATE      875803 non-null  int64
 13  DAYS_MEMBER       875803 non-null  int64
dtypes: float64(1), int64(13)
memory usage: 100.2 MB
```

Figur 4.21. - Versjon 1 (renset)

Versjon 1 gav oss innsikt i tabellene og hvordan vi skulle utnytte de bedre for de neste iterasjonene. Her ble det oppdaget og utformet taktikk som ble videreført til vår endelige bearbeiding. Eksempelvis vil maskinlæringsmodeller helst operere med tall og datatyper som `int`, `float` og `double`. Vi skjønte at vi måtte bearbeide tekstfeatures av typen `object` og `string` om til tallfeatures. Mye tid ble satt av til å jobbe med denne omformateringen. Vi lærte også at datasettet om CDR kunne kraftig reduseres ved å samle all databruken fra enkelttransaksjoner i rader, og flytte de til features som samlet bruken i volum per måned.

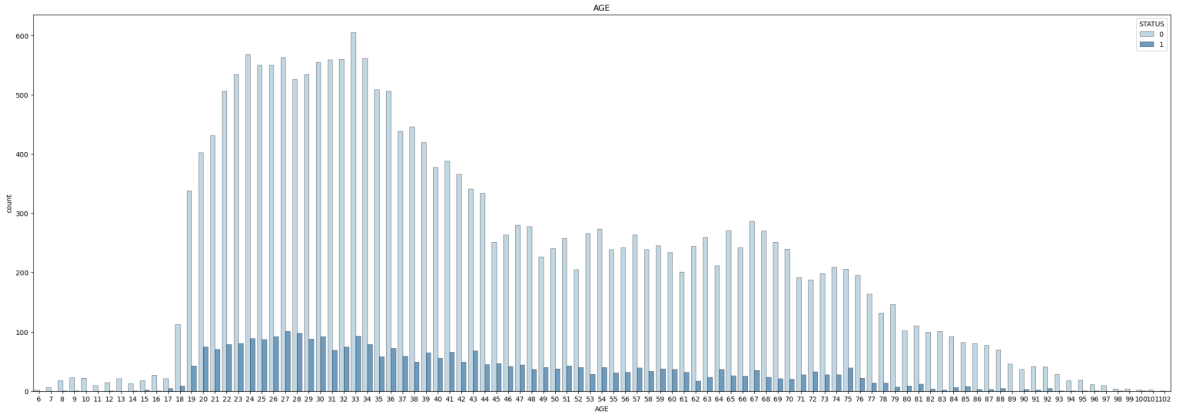
SUBSCRIBERID	CHARGE	VOLUME	RATEZONE	RATEPRODUCT	INCOMEPERIOD	CLEARINGDATE
36503973	0	2504	NULL	GPRS	202212	2022-11-09 22:11:36
36503973	0	4239	NULL	GPRS	202212	2022-11-09 22:11:36
36503973	0	4239	NULL	GPRS	202212	2022-11-09 22:11:36
36563912	0	40566	NULL	GPRS	202212	2022-11-09 22:11:44
36563967	440	46123…	NULL	GPRS	202212	2022-11-09 22:11:26
36564110	290	NULL	NMOB	MO	202212	2022-11-09 22:11:31
36564110	6323	NULL	NMOB	MO	202212	2022-11-09 22:11:48
36564110	270	NULL	NMOB	MO	202212	2022-11-09 22:11:30
36564110	290	NULL	NMOB	MO	202212	2022-11-09 22:11:32

Figur 4.22 - Utsnitt av CDR tabell

SUBSCRIBERID	VOL_JUNE	VOL_JULY	VOL_AUGUST	VOL_SEPTEMBER	VOL_OCTOBER
36282983	0	0	11455	65836	88024.9
36283049	0	0	1540.63	16317.4	21212.3
36283632	0	0	1794.26	4908.73	2883.02
36284347	0	0	0.0324	33260.5	10944.9
36284369	0	0	61.6566	92.9812	86.4738

Figur 4.23 - Utsnitt av tabell for Versjon 2

Datarensing er en av de viktigste oppgavene man kan gjøre for å få et godt resultat. Mye tid ble her brukt på null-verdier da det ble ansett som det mest presserende. Et element som skapte usikkerhet var spørsmålet vedrørende null-verdier og svært lave aldre. Ved undersøkelse av age-features ble det oppdaget at to av abonnentene var under 6 år (dvs. 0 og 1 år gamle), og at det også fantes null-verdier på alder. Ved plotting av churn basert på alder virket det som om alderen kunne bidra til å vippe beslutningen for modellen. Eksempelvis viste det seg at det ble mest kunder i aldersspennet 18-44 (se figur 4.24). På figuren virker det som at trenden for churn følger trenden for antall kunder i samme aldersgruppe. Ser vi nærmere på figuren ser vi at det er små, men potensielt utslagsgivende forskjeller. De som har churnet når en topp ved 27 år og får et fall ved 31 år. De aktive når sin topp ved 33 og har sitt fall ved 28. Det ble også ansett at alder ikke burde settes til snitt eller median fordi det kunne være en utslagsgivende faktor for churn. Vi droppet derfor radene hvor kundene hadde "null" i alder og var under 4 år. Det var høyst usannsynlig at det forbruket vi så hos de som var 0 og 1 år gamle var reelt.



Figur 4.24. - Diagram viser aldersfordelingen blant aktive (lyseblå) og kansellerte (mørkeblå) kunder.

Det ble også valgt å fjerne alle kundene med status ‘Sperret’. Det var usikkert om disse kundene ønsket å være kunder, men manglet betalingsevne eller om de ikke ønsket å være kunder og unnlot å betale til de ble sperret. Det var heller ikke interesse for at modellen skulle predikere om noen ville bli ‘Sperret’, kun om sannsynligheten for at aktive abonnenter ville churne. For at modellen skulle bli best mulig trent på å predikere churn-risiko ble de som hadde status ‘Sperret’ ansett som potensielt forstyrrende og droppet.

En større utfordring gjaldt ‘null’-verdier i ENDDATE. ENDDATE kunne ikke være ‘null’, men var satt til ‘null’ når kundene ikke hadde avsluttet kundeforholdet og var aktive. Kunder som hadde avsluttet kundeforholdet hadde derimot et tidsstempel i ENDDATE. Det ble forsøkt med å endre ‘null’-verdiene slik at de sto til dagens dato, og altså ville bli regnet som mer langvarige kunder, men det ville gitt upresise beregninger. For oppdragsgiver som har oppdatert data kunne man få en feature med korrekt varighet på kundeforhold, men for oss som ikke hadde tilgjengelighet på kunders status ville dette gi inkorrekt varighet på kundeforhold. Vi kan ikke si noe sikkert om lengden på kundeforholdene, men vi antar at dette kunne vært en betydelig faktor for å gjøre modellen mer presis.

Datasettet (som vist på figur 4.21) ble brukt som utgangspunkt til å trene datasettet på maskinlæringsalgoritmene Logistic Regression og Keras Neural Network. Begge ble testet for Accuracy og begge oppnådde en score på 82% (se figur 4.25 og 4.26). Selv om dette ikke var et dårlig resultat, forsto vi at datasettet inneholdt masse feil som måtte endres mer for å kunne gi noe meningsfull output.

```
Accuracy: 0.8238877375671525
Accuracy: 82 %
```

```
5474/5474 [==============================]
Test Accuracy: 0.8198971152305603
Test Accuracy: 82 %
```

Figur - 4.25 LogisticRegression (t.v.), Keras Neural Network (t.h.)

Versjon 2

For denne versjonen ble dataen utvalgt med tanke på å se nærmere på forbruk og generelt bruksmønster, som man ser i figur 4.23. Det klokeste var å se på hver kundes totale forbruk fordelt på hver måned og linke det opp mot SUBSCRIBERID. På denne måten ville vi samle alle CDR-dataene slik at hver rad inneholdt databruken til kun en kunde. Dette reduserte

datasettet fra 14 millioner rader til 25.368 rader og gjorde det mer håndterbart for modellen å tolke. Vi la også til noen features fra kunde-tabellen. Ferdigopprettet tabell før rensing ser man på figur 4.26. Å utføre bearbeiding av features ble fra dette punkt og for senere versjoner utført direkte i MySQL fremfor i Jupyter Notebook. Det var den letteste måten å synkronisere resultater fra maskinlæring med vår applikasjon på det tidspunktet, ettersom de samme input-verdiene måtte sendes inn. En annen løsning hadde vært å bearbeide mer i Python slik at man kunne lettere filtrere rådata og benytte flere datakilder med samme modell. Eksempelvis hvis dette skulle brukes på live og kontinuerlig data hadde det vært mer fordelaktig for oppdragsgiver med så mye databehandling i Python/Pandas som mulig.

Det mest optimale oppsettet hadde vært om databehandlingen i Notebook hadde vært lik i form som databehandlingen på Flask serveren. Da hadde vi kunnet gjort samme prosessering med Pandas uten å måtte endre databasestrukturen. I neste fase av en videre prosess ville dette vært et fokusområde.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25368 entries, 0 to 25367
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   SUBSCRIBERID      25368 non-null  int64
 1   VOL_JUNE          25359 non-null  float64
 2   VOL_JULY          25366 non-null  float64
 3   VOL_AUGUST        25327 non-null  float64
 4   VOL_SEPTEMBER     25344 non-null  float64
 5   VOL_OCTOBER       25332 non-null  float64
 6   VOL_NOVEMBER      25293 non-null  float64
 7   AGE               25305 non-null  float64
 8   BUCKET_MB         25368 non-null  int64
 9   SUBSCRIPTIONTYPE  25368 non-null  int64
 10  SUBSCRIPTIONDESC  25368 non-null  object
 11  STATUS            25368 non-null  int64
 12  STATUSDESC        25368 non-null  object
dtypes: float64(7), int64(4), object(2)
memory usage: 2.5+ MB
```

Figur 4.26 - Versjon 2 av maskinlæring-datasett (ubearbeidet)

Ettersom datasettet baserte seg på tabellen som inneholdt cdr, oppdaget vi at den ikke inkluderte alle kundene vi hadde i kunde-tabellen. 3527 kunder som ikke hadde dataforbruk var ikke interessante å se på siden vi skulle trene en modell som så på forbruk og de ble droppet. Versjon 2 ble også testet med Logistic Regression, som ga en Accuracy-Score på 90%, men evalueringen av modellen gjorde betydelig dårligere (figur 4.28).

```
Accuracy: 0.8922370784274596 AKA 89 %
Precision: 0.7627118644067796 AKA 76 %
Recall: 0.2132701421800948 AKA 21 %
F1: 0.33333333333333337 AKA 33 %
```

Figur 4.28 - Score for Accuracy, Precision, Recall og F1(logistic regression) for versjon 2 av maskinlæring-datasett

Vi har brukt Accuracy, Precision[31], Recall[32] og F1[33] for å evaluere vår maskinlæringsmodell (Nighania, 2019). Disse evalueringsresultatene brukes på binære klassifiseringsproblem for å sjekke maskinlæringmodellers kvalitet. Resultatene viser at modellen gjorde det bedre enn vår forrige, men fortsatt ikke var god nok. Om et firma f.eks. har en churn på 20% gjør firmaet det veldig dårlig. En prediksjon om at ingen kunder churner vil da tilsi at man har en accuracy på 80%. Den er ikke dårlig, men oppdager ikke falske negativer. En mer ideell modell for maskinlæring i forbindelse med churn er derfor høy precision og en god recall (Berkeley School of Information, 2017). Dette er ikke tilfelle for evalueringen av daværende modell.

Likevel valgte vi å ha med denne modellen i vår applikasjon for å kunne sammenligne forskjellene med flere modeller. Vi anvendte metoden "predict_proba" for å forutsi sannsynligheten for churn. Dette er tallet som er synlig i vår applikasjon. "Predict_proba" forutsier egentlig ikke sannsynlighet, men faktisk modellens sikkerhet på at den har predikert riktig. Vi lot den stå ettersom "predict_proba" fortalte hvilket binære utfall som er gitt avhengig av om tallet var under eller over 50%. Vi tenkte at i samsvar med en bedre prediksjonsmodell kan dette være en ekstra beslutningsfaktor; høy churn på begge modellene øker sannsynligheten for at en aktiv kunde er i risikosonen for å churne.

Maskinlæringsalgoritme: Random Forest Regression

Gjennom testing av dette datasett med flere forskjellige maskinlæringsalgoritmer kom vi kom frem til at den algoritmen som ga best evaluering for vårt formål var Random Forest Regression. Denne algoritmen er innenfor det emnet av maskinlæring som kalles Ensemble methods. Dette er algoritmer hvor flere individuelle modeller kombineres for å oppnå bedre

[31] Presisjon kan uttrykkes som antall sanne positive (True Positives, TP) delt på summen av antall sanne positive og antall falske positive (False Positives, FP).
[32] Recall kan uttrykkes som antall sanne positive (True Positives, TP) delt på summen av antall sanne positive og antall falske negative (False Negatives, FN).
[33] F1-score beregnes som harmonisk gjennomsnitt av presisjon og recall, og uttrykkes som følger: F1-score = 2 * (precision * recall) / (precision + recall)

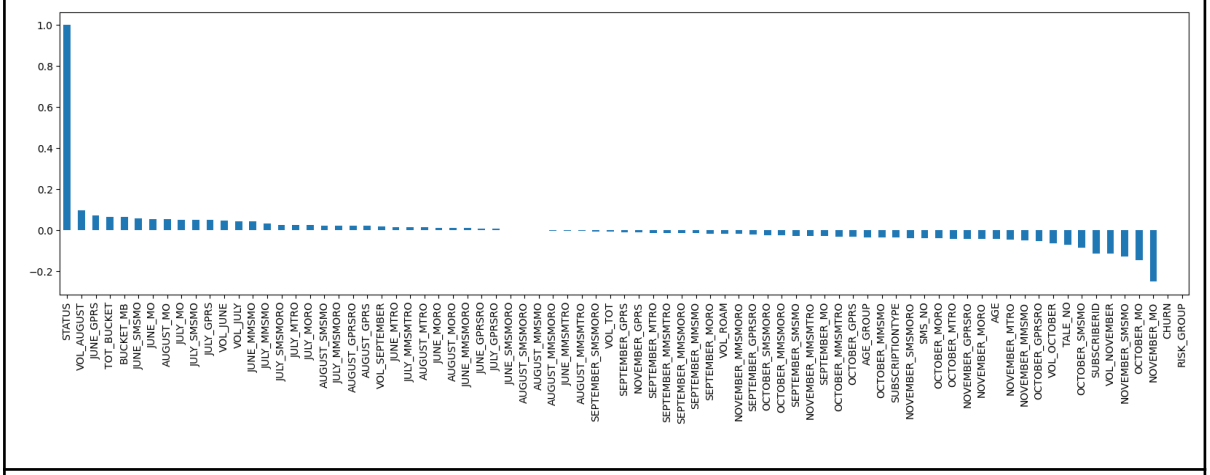
prediksjoner eller beslutninger enn hva hver enkelt modell kan oppnå alene (Brownlee, 2021). Bagging[34]-metoder bygger flere base-modeller på forskjellige tilfeldig utvalgte undergrupper av treningsdataene. En kjent bagging-metode er Random Forest, som kombinerer en mengde avgjørelsestrær for å gjøre prediksjoner (ibid). Den blir også regnet for å være god på churn-prediction. Ensemblemetoder kan gi flere fordeler, inkludert økt prediksjonsnøyaktighet, bedre generell ytelse, reduksjon av overtilpasning (overfitting) og økt robusthet mot støy i dataene. Dette gjør den spesielt egnet til vårt datasett fordi økt. Dette er en noe utradisjonell tilnærming, da det som nevnt er vanligere å se på churn som et klassifiseringsproblem. For vår del derimot ønsket vi å se på risikofaktorer som hadde påvirkning på churn. Vi vil heller dele opp kundene i risikosoner og oppdage nyanser i forbruket. Denne problemstillingen mener vi er nærmere knyttet opp mot regresjon. Vi vil ikke finne ut om en kunde forlater selskapet eller ikke, men heller se en prosent over hvor utsatt en kunde er. Dette gjenspeiler også virkeligheten ved at årsaker til at en kunde forlater et selskap er komplekse og uklare.

Andrew Ng underbygger en av våre teorier, han mener at selv om datarensing er sentralt, er det likevel undervurdert i arbeidet for å oppnå de beste resultatene (DeepLearningAI, 2021). Han mener at hvis man står fast og ikke får et bedre resultat, så ligger nøkkelen i datarensingen. Denne tankegangen er interessant fordi han mener modellen alltid kan bli bedre, selv om det virker som modellen har nådd sin topp. Med tanke på vår modell er det interessant å se på hvilke muligheter dette hadde gitt oss og gav oss ettersom modellen ble bedre og bedre for hver iterasjon. Vi kunne eksempelvis sett nærmere på skjulte outliers som ikke burde være representative for den alminnelige bruker. Av alle kundene tegnet 6381 aktive kunder sitt abonnement før 01.06.2022 (når CDR-datasettet begynner). Per nå er disse kundene inkludert i treningen av maskinlæringsmodellen selv om deres fulle aktive periode ikke stemmer overens. Det motsatte tilfellet inntreffer også, hvor kunder begynner sitt abonnement sent i perioden vi har CDR for og ble dermed registrert med inaktiv databruk selv om de da ikke var kunde hele perioden. Denne skjeve fordelingen har vi ikke funnet en god løsning på. Likevel mener vi at det kan være mulig å komme frem til en bedre løsning.

[34] Bootstrap Aggregation

4.5.2. Den endelige versjonen

For vår endelige versjon av modellen var planen å inkludere så mange features som mulig i håp om at modellen ville ha flere tall å se på og dermed kunne gjøre mer korrekte prediksjoner. Videreutviklet fra versjon 2 tok denne versjonen med antall av samtlige ulike typer transaksjoner en kunde kunne ha fordelt på hver måned. For eksempel SMS, samtaler, bruk i og utenfor Norge, m.m. I tillegg ble det også opprettet features for Churn og aldersgrupper. Totalt ga dette oss hele 79 features. Det vi kom frem til var at enkelte typer features som for eksempel samtale og SMS tilsynelatende ikke hadde stor betydning for churn-risikoen, men de var likevel viktig å inkludere i datasettet fordi det gjorde modellen mer presis. En teori på dette kan være at det oppstår en form for informasjonssynergi. Det kan bidra til at modellen kan bli bedre på å fange opp komplekse mønstre når tilsynelatende uviktig data blir tatt med i beregningen. De fungerer som vektorer på en likevekt og kan gjøre utslag den ene veien eller den andre.



Figur 4.29 - Plot som viser betydningen av features (bilde med bedre kvalitet i Appendiks)

Vi kunne rangere features i datasettet som hadde innvirkning på å bestemme kundens status. Det var stor spredning, men en ting var åpenbart. Forbruk som fant sted i de første månedene (Juni, Juli, August) ville øke sannsynligheten for churn, mens forbruk som fant sted i de siste månedene (September, Oktober, November) minket sannsynligheten for churn. Features som inneholdt data om forbruk i november viste seg også å ha en betydelig innvirkning på modellen. Dette er likevel en måned hvor vi har ufullstendig data; kun frem til 13. november. Vi vurderte å droppe denne måneden nettopp fordi den ikke viser en hel måneds-forbruk og derfor kanskje ikke er representativ, men den er beholdt.

```
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   SUBSCRIBERID        25303 non-null  int64
 1   VOL_JUNE            25303 non-null  float64
 2   VOL_JULY            25303 non-null  float64
 3   VOL_AUGUST          25303 non-null  float64
 4   VOL_SEPTEMBER       25303 non-null  float64
 5   VOL_OCTOBER         25303 non-null  float64
 6   VOL_NOVEMBER        25303 non-null  float64
 7   JUNE_MO             25303 non-null  int64
 8   JULY_MO             25303 non-null  int64
 9   AUGUST_MO           25303 non-null  int64
 10  SEPTEMBER_MO        25303 non-null  int64
 11  OCTOBER_MO          25303 non-null  int64
 12  NOVEMBER_MO         25303 non-null  int64
 13  JUNE_SMSMO          25303 non-null  int64
 14  JULY_SMSMO          25303 non-null  int64
 15  AUGUST_SMSMO        25303 non-null  int64
 16  SEPTEMBER_SMSMO     25303 non-null  int64
 17  OCTOBER_SMSMO       25303 non-null  int64
 18  NOVEMBER_SMSMO      25303 non-null  int64
 19  JUNE_GPRS           25303 non-null  int64
 20  JULY_GPRS           25303 non-null  int64
 21  AUGUST_GPRS         25303 non-null  int64
 22  SEPTEMBER_GPRS      25303 non-null  int64
 23  OCTOBER_GPRS        25303 non-null  int64
 24  NOVEMBER_GPRS       25303 non-null  int64
 25  JUNE_GPRSRO         25303 non-null  int64
 26  JULY_GPRSRO         25303 non-null  int64
 27  AUGUST_GPRSRO       25303 non-null  int64
 28  SEPTEMBER_GPRSRO    25303 non-null  int64
 29  OCTOBER_GPRSRO      25303 non-null  int64
 30  NOVEMBER_GPRSRO     25303 non-null  int64
 31  JUNE_MMSMO          25303 non-null  int64
 32  JULY_MMSMO          25303 non-null  int64
 33  AUGUST_MMSMO        25303 non-null  int64
 34  SEPTEMBER_MMSMO     25303 non-null  int64
 35  OCTOBER_MMSMO       25303 non-null  int64
 36  NOVEMBER_MMSMO      25303 non-null  int64
 37  JUNE_SMSMORO        25303 non-null  int64
 38  JULY_SMSMORO        25303 non-null  int64
 39  AUGUST_SMSMORO      25303 non-null  int64
 40  SEPTEMBER_SMSMORO   25303 non-null  int64
 40  SEPTEMBER_SMSMORO   25303 non-null  int64
 41  OCTOBER_SMSMORO     25303 non-null  int64
 42  NOVEMBER_SMSMORO    25303 non-null  int64
 43  JUNE_MMSMORO        25303 non-null  int64
 44  JULY_MMSMORO        25303 non-null  int64
 45  AUGUST_MMSMORO      25303 non-null  int64
 46  SEPTEMBER_MMSMORO   25303 non-null  int64
 47  OCTOBER_MMSMORO     25303 non-null  int64
 48  NOVEMBER_MMSMORO    25303 non-null  int64
 49  JUNE_MMSMTRO        25303 non-null  int64
 50  JULY_MMSMTRO        25303 non-null  int64
 51  AUGUST_MMSMTRO      25303 non-null  int64
 52  SEPTEMBER_MMSMTRO   25303 non-null  int64
 53  OCTOBER_MMSMTRO     25303 non-null  int64
 54  NOVEMBER_MMSMTRO    25303 non-null  int64
 55  JUNE_MORO           25303 non-null  int64
 56  JULY_MORO           25303 non-null  int64
 57  AUGUST_MORO         25303 non-null  int64
 58  SEPTEMBER_MORO      25303 non-null  int64
 59  OCTOBER_MORO        25303 non-null  int64
 60  NOVEMBER_MORO       25303 non-null  int64
 61  JUNE_MTRO           25303 non-null  int64
 62  JULY_MTRO           25303 non-null  int64
 63  AUGUST_MTRO         25303 non-null  int64
 64  SEPTEMBER_MTRO      25303 non-null  int64
 65  OCTOBER_MTRO        25303 non-null  int64
 66  NOVEMBER_MTRO       25303 non-null  int64
 67  VOL_TOT             25303 non-null  float64
 68  TALE_NO             25303 non-null  float64
 69  SMS_NO              25303 non-null  int64
 70  VOL_ROAM            25303 non-null  float64
 71  AGE                 25303 non-null  int64
 72  BUCKET_MB           25303 non-null  int64
 73  SUBSCRIPTIONTYPE    25303 non-null  int64
 74  TOT_BUCKET          25303 non-null  int64
 75  AGE_GROUP           25303 non-null  int64
 76  STATUS              25303 non-null  int64
 77  CHURN               0 non-null      float64
 78  RISK_GROUP          0 non-null      float64
dtypes: float64(11), int64(68)
memory usage: 15.3 MB
```

Figur 4.29 - Versjon 3 av maskinlæring-datasett

Evalueringen av Random Forest Regression-modellen bruker ikke Accuracy, Precision, Recall og F1, ettersom den ikke predikerer faktiske klassifiserte utfall. For regresjon må man derfor anvende f.eks. Mean Absolute Error[35], Mean Squared Error[36], Root Mean Squared Error[37] og R-Squared[38]. Våre verdier ble henholdsvis 0.06, 0.02, 0.15 og 0.78(se figur 4.30).

[35] MAE er en evalueringsmetrikk som måler gjennomsnittlig absolutt differanse mellom prediksjonene og de faktiske verdiene. MAE gir et mål på gjennomsnittlig avvik i størrelse mellom prediksjonene og de faktiske verdiene.

[36] MSE er en evalueringsmetrikk som beregner gjennomsnittet av kvadratet av differansen mellom prediksjonene og de faktiske verdiene. MSE gir et mål på gjennomsnittlig kvadratisk avvik mellom prediksjonene og de faktiske verdiene.

[37] RMSE er en evalueringsmetrikk som er kvadratroten av MSE. Det gir et mål på gjennomsnittlig kvadratrot avvik mellom prediksjonene og de faktiske verdiene.

[38] R-Squared er en evalueringsmetrikk som måler hvor godt en regresjonsmodell passer til dataene. Det representerer andelen av variasjonen i målvariabelen som kan forklares av modellen

De tre første skal være så lave som mulig, mens den siste skal i utgangspunktet være så høy som mulig.

```
Mean Absolute Error: 0.06
Mean Squared Error: 0.02
Root Mean Squared Error: 0.15
R-squared: 0.78
```

Figur 4.30 - Evaluering av modellen

En annen evaluering er det som kalles for Cross-Validated R-squared Scores. Da får man f.eks. fem forskjellige R-squared scores som ideelt sett skal være høye og jevne. Hvis man får høyt sprik i scores betyr det gjerne at modellen er ujevn.

```
Cross-validated R-squared scores: [0.81956764 0.8286122  0.80373456 0.706345   0.3738141 ]
```

Figur 4.31 - Resultat av Cross-Validated R-squared

Dette ga oss gode nok resultater og spredning, men det var fortsatt sterk overvekt mot at de aller fleste kunder fikk en churn risiko på under 5%. Det gledet dog at det var en håndfull kunder som spredde seg jevnt over helt opp mot 90-99.99%. Vi undersøkte og sammenlignet de samme kundene og så på fordelingen opp mot oppdaterte kundedata fra oppdragsgiver (se figur 4.32).

```
Antall aktive med churn over 90: 4 av 1567
Antall churna med churn over 90: 1561 av 1567

Antall aktive med churn mellom 80-89.99: 8 av 529
Antall churna med churn mellom 80-89.99: 518 av 529

Antall aktive med churn mellom 70-79.99: 17 av 367
Antall churna med churn mellom 70-79.99: 347 av 367

Antall aktive med churn mellom 60-69.99: 35 av 200
Antall churna med churn mellom 60-69.99: 161 av 200

Antall aktive med churn mellom 50-59.99: 31 av 173
Antall churna med churn mellom 50-59.99: 140 av 173

Antall aktive med churn mellom 40-49.99: 64 av 189
Antall churna med churn mellom 40-49.99: 121 av 189

Antall aktive med churn mellom 30-39.99: 89 av 222
Antall churna med churn mellom 30-39.99: 122 av 222

Antall aktive med churn mellom 20-29.99: 185 av 307
Antall churna med churn mellom 20-29.99: 109 av 307

Antall aktive med churn mellom 10-19.99: 293 av 448
Antall churna med churn mellom 10-19.99: 148 av 448

Antall aktive med churn mellom 0-9.99: 16671 av 21050
Antall churna med churn mellom 0-9.99: 4189 av 21050
```

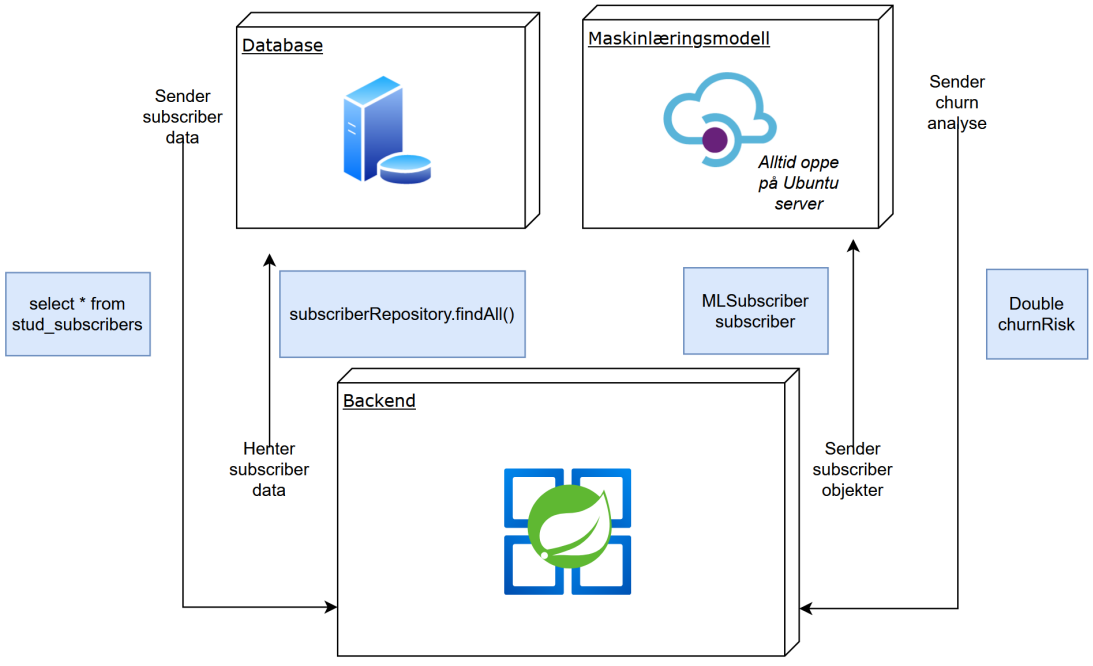
Figur 4.32 - Sammenligning med med dagens kundeinfo fra oppdragsgiver

Resultatene vises som forventet. Kunder som allerede har avsluttet kundeforholdet har konsekvent en markant høyere risiko for kundefrafall. Dette avtar mer jo mer churn-risikoen

minker, men andelen kunder som har churnet representerer en overvekt blant alle som får churn-risiko over 30%. Problematisk er dog at der det burde være en tydelig overvekt blant aktive medlemmer (0% - 10%), finner vi at omkring 25% faktisk har churnet. Modellen har altså utfordringer med falske negativer. Det er med andre ord fortsatt rom for forbedring av modellen, men vi er positivt innstilt til resultatet. En imperfekt modell kan forklares med at vårt datasett ikke fullt dekker seks måneder og hovedsakelig kun tar utgangspunkt i forbruksmønster.

Flask

Flask-applikasjonen brukes for å koble maskinlæringsmodellen til resten av applikasjonen. Flask ble først utviklet i PyCharm som en separat tjeneste. Etter en stund ville vi ha alle delene av prosjektet under samme mappe og i samme repository, så det var enkelt å starte det samtidig. Dette fungerte, men det innebar en del konfigurering med Python Interpreters[39] for IntelliJ og det fungerte ikke for alle gruppemedlemmene. Det ble bestemt å benytte seg av Ubuntu-serveren for å hoste Flask-applikasjonen så den kunne kjøre konstant i bakgrunnen. Dette også med tanke på at det skulle være enkelt å starte opp når vi skulle avlevere prosjektet til oppdragsgiver og sensor.



Figur 4.33 - Illustrasjon av applikasjons-flyt med Flask

[39] Python Interpreter tolker og utfører Python-kode linje for linje.

```
onecall 1377052 0.0 0.5 1568220 SSL Apr12 /usr/bin/python /home/onecall/flask-ml/app/app.py
```

Figur 4.34 - Bakgrunnskjøring av Flask-applikasjon på Ubuntu server

Random Forest modellen ble trent på 75 features med data. Endepunktet vist under velger ut de aktuelle, omformer dataen til en flat struktur, mater modellen med dataen, regner ut en prosent og returnerer resultatet til Spring Boot.

```python
# Endpoint for å forutsi sannsynlighet for churn for en enkelt subscriber
# Bruker Random Forest modellen
@app.route("/predict_rf_subscriber", methods=["POST"])
def predict_rf_subscriber():
    data = request.get_json()
    # Kolonnene modellen trenger
    attributes_to_keep = [
        'volumeJune', 'volumeJuly', 'volumeAugust', 'volumeSeptember', 'volumeOctober',
        'volumeNovember',
        'juneMo', 'julyMo', 'augustMo', 'septemberMo', 'octoberMo', 'novemberMo',
        'juneSmsMo', 'julySmsMo', 'augustSmsMo', 'septemberSmsMo', 'octoberSmsMo', 'novemberSmsMo',
        'juneGprs', 'julyGprs', 'augustGprs', 'septemberGprs', 'octoberGprs', 'novemberGprs',
        'juneGprsRo', 'julyGprsRo', 'augustGprsRo', 'septemberGprsRo', 'octoberGprsRo',
        'novemberGprsRo',
        'juneMmsMo', 'julyMmsMo', 'augustMmsMo', 'septemberMmsMo', 'octoberMmsMo', 'novemberMmsMo',
        'juneSmsMoRo', 'julySmsMoRo', 'augustSmsMoRo', 'septemberSmsMoRo', 'octoberSmsMoRo',
        'novemberSmsMoRo',
        'juneMmsMoRo', 'julyMmsMoRo', 'augustMmsMoRo', 'septemberMmsMoRo', 'octoberMmsMoRo',
        'novemberMmsMoRo',
        'juneMmsMtRo', 'julyMmsMtRo', 'augustMmsMtRo', 'septemberMmsMtRo', 'octoberMmsMtRo',
        'novemberMmsMtRo',
        'juneMoRo', 'julyMoRo', 'augustMoRo', 'septemberMoRo', 'octoberMoRo', 'novemberMoRo',
        'juneMtRo', 'julyMtRo', 'augustMtRo', 'septemberMtRo', 'octoberMtRo', 'novemberMtRo',
        'totalVolume', 'taleNo', 'smsNo', 'volumeRoam',
        'age', 'bucketMb', 'subscriptionType', 'totalBucket', 'ageGroup'
    ]
    # Dictionary med kun de valgte kolonnene
    filtered_data = {attr: data[attr] for attr in attributes_to_keep}
    data_array = np.array(list(filtered_data.values()))
    data_array = data_array.reshape(1, -1)
    prediction = rf_model.predict(data_array)
    probability_percent = round(prediction[0] * 100, 2)

    return jsonify(probability_percent.tolist())
```

Figur 4.35 - Endpoint for å gi en prediksjon for en subscriber

4.6. Sikkerhet

Sikkerhet utgjorde en mindre del av løsningen, da det ikke var vektlagt en omfattende sikkerhet fra oppdragsgiverens side. Dette fordi oppdragsgivers systemer allerede var beskyttet av komplekse og solide sikkerhetsmekanismer. Av den grunn brukte vi ikke mye ressurser på å implementere omfattende sikkerhet rundt applikasjonen. Men for å gjøre vår

oppgave mer komplett og for å vise at vi tok høyde for sikkerhet, implementerte vi en grunnleggende logg-inn funksjonalitet. Vi brukte Spring sin innebygde Basic Authentication. Denne mekanismen sikrer brukergrensesnittet og endepunktene til backend med krav om brukernavn og passord. Vi definerte en 'in memory' sikkerhetskonfigurering som inneholdt brukernavn og passord. Dette krypteres før det sendes over nettverket fra frontend til backend. Det ble lagret en brukerlegitimasjon i LocalStorage[40] eller SessionStorage[41] som ble brukt for å få tilgang til andre endepunkter i systemet. Ved utlogging blir disse legitimeringene fjernet.

Den store fordelen med Basic Authentication er at det er enkelt å sette opp. Dessverre har den også den ulempen at brukernavn og passord ligger lagret i selve kildekoden. En sikrere tilnærming er f.eks. å lagre hash-verdien av passordet separat i en database, men for vår demonstrasjon var dette akseptabelt.

Alternativt kunne vi brukt JSON Web Token (JWT), som er en mer komplett, sikker, og omfattende sikkerhetsmekanisme. JWT fungerer ved at det blir generert en token som autentiseres og signeres med en privat hemmelig nøkkel eller et offentlig/privat nøkkelpar, og godkjent token ville gitt brukeren videre tilgang.

4.7. Testing

Testing utgjør en stor del av utviklingen. Dermed er det også viktig at man planlegger, utfører og rapporterer testingen på en strukturert, gjennomtenkt og systematisk måte. Dette er på grunn av at eventuelle feil bør oppdages både tidlig og fortløpende, slik at de kan rettes så tidlig som mulig. I tillegg er det viktig at man rapporterer riktig, som gjør det lettere å spore tilbake til der feilen ligger og se om den har blitt rettet. I vårt prosjekt har vi i utgangspunkt vurdert å basere oss på Test Driven Development (TDD) som er en utviklingsmetode som har stor fokus på testing og utviklingen starter med bygging av tester. Men på grunn av lite kjennskap til denne tilnærmingen, udefinerte kravspesifikasjoner i startfasen, og i tillegg til et komplekst datasett har det vært utfordrende for oss å bruke TDD (IBM, u.å.). Derfor har vi kjørt testingen parallelt med utviklingen på slutten av hver sprint, og i tillegg ved slutten av hele utviklingen. De ulike test-metodene vi har brukt er enhetstesting, integrasjonstesting og

[40] Funksjon i nettleseren som lar nettsider lagre og hente data på brukerens enhet.
[41] Funksjon i nettleseren som lar nettsider midlertidig lagre data under brukersesjoner.

til slutt brukertesting som også har fungert som akseptansetesting for oppdragsgiver. Disse testmetodene er hierarkisk bygget opp med ulike nivåer (*figur 4.36* - Gupta, u.å.) for å teste så mye som mulig og mot så mange aspekter ved systemet som mulig over flere faser (Gupta, u.å.). Testene kan avdekke ulike type feil og øke dekningsgraden for testing av hele applikasjonen.

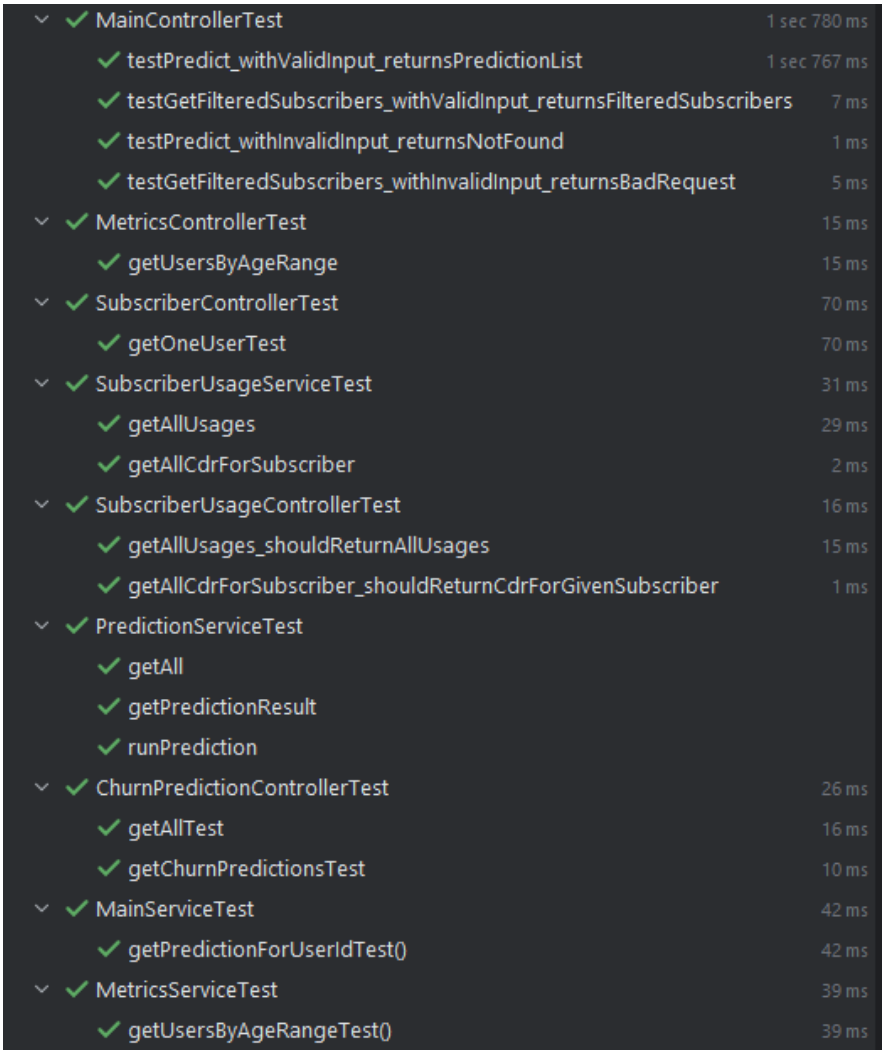
Figur 4.36 - Test hierarkiet

Vi har som nevnt tidligere utført testingen parallelt med utviklingen, og har dermed ikke planlagt eller strukturert det på en detaljert måte, ved for eksempel bruk av et test-management verktøy. Dette er noe vi kunne tenkt oss å implementere hvis vi hadde hatt mer tid til rådighet. Testverktøyene hadde økt effektiviteten i utviklingen, og med bedre strukturert testing kunne vi forbedret kvaliteten til produktet vi leverte.

4.7.1. Enhetstest

Det enhetstestingen gjør er å teste at individuelle enheter eller komponenter fungerer som forventet. Den bidrar til at vi som skriver koden kan få bedre forståelse av hvordan kodebiten fungerer (javapoint, u.å.). Denne typen test kan avdekke feil tidlig i utviklingen fordi det gjerne er det første som testes i utviklingsfasen. Det er ofte programmerere som lager enhetstestene fordi de er nærmest selve koden. I større firmaer er det egne testfolk som tar seg av de andre testene i hierarkiet. Enhetstester er gjerne knyttet opp mot funksjonelle krav, da de tester disse funksjonene som f.eks. handler om validering, input og output-metodekall som sendes til frontend og brukergrensesnittet. Spring Boot har innebygd støtte for enhetstesting som vi benyttet oss av. Til å utføre enhetstester brukte vi de innebygde

bibliotekene som JUnit og Mockito. Disse bibliotekene gir støtte for å mocke[42] de ulike klassene og metodene slik at man lage egne testklasser som simulerer de ekte klassene. Disse klassene og metodene er isolerte og skal ikke påvirke resten av applikasjonen, men kun brukes til å teste at de enkelte metodene fungerer etter sin hensikt.



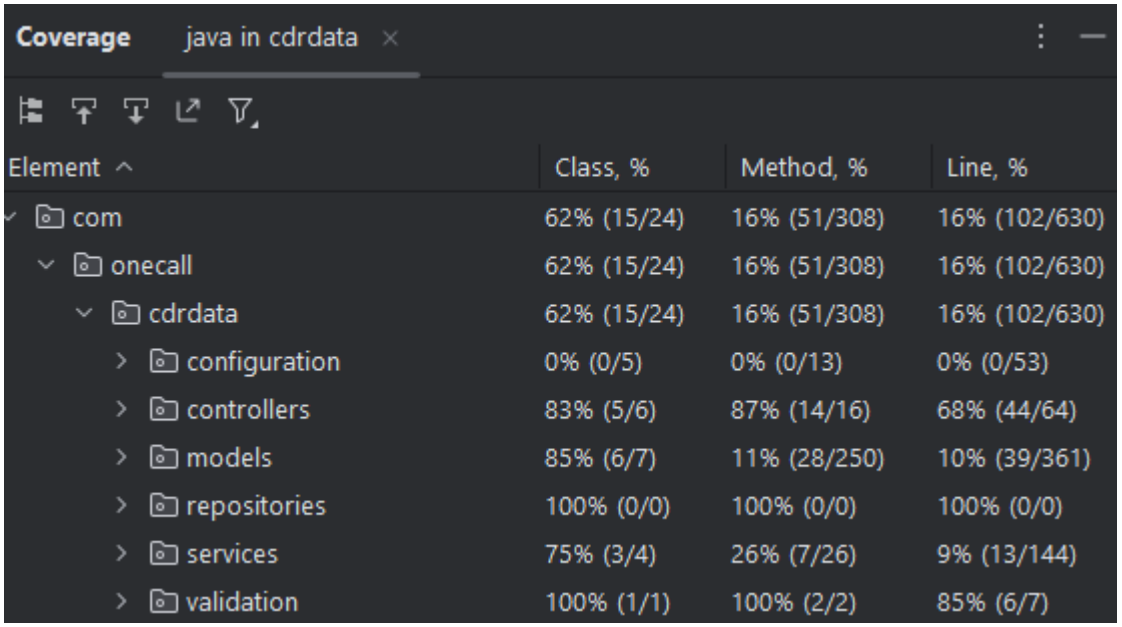
Figur 4.37 - Enhetstesting av xx-klasse og dens metoder

I figur 4.37 kan du se en oversikt på resultatene fra en typisk enhetstest. Etter at testene har blitt kjørt passerer alle metodene i klassen testen og fått grønn sjekk-markør. Det viser at forventet utfall ble det samme som det faktiske utfallet. Mot siste halvdel av prosjektet ble det en del flere metoder og komponenter å teste og vi rakk dessverre ikke å få godkjent på alle før innlevering. På figur ser man at selv om testene under PredictionServiceTest har fått grønn sjekkmarkør er ikke disse metodene dekket 100%.

[42] Kopi eller klone av den originale klassen for å simulere den til bruk i testing.

Vi lagde også en testmanuskrift for enhetstestingen (Appendiks 8), noe som er lurt å ha for å sjekke at man dekker så mange metoder og komponenter som mulig. Det var en del av planleggingsarbeidet. Et slikt dokument kan gi en god oversikt over hva som blir testet. Dessverre rakk vi aldri å implementere resultatene, ettersom enhetstester ble opprettet parallelt med kodingen. Hadde vi hatt bedre tid, ville vi prøvd å få alle testene grønne og med full dekningsgrad. Noe av grunnen til at vi ikke fikk testet alle metodene var fordi applikasjonen var i stadig utvikling og det var nye metoder som måtte testes og java-klasser som ble endret, noe som gjorde at tidligere fungerende tester feilet etter endringene.

Noe vi også ønsket for enhetstestingen om vi hadde hatt bedre tid var å bruke et bibliotek som heter PIT Mutation Testing (pitest, u.å.). Slike mutant-tester finner flere testtilfeller. Selv om alle testene vi har er dekket, betyr ikke det at vi automatisk har avdekket alle feil som kan oppstå. Fordeler med å bruke dette verktøyet er at den setter inn mutasjoner i test-koden og hvis testen ikke klarer å "drepe" mutanten, så har man en test som ikke dekker alle utfallene.



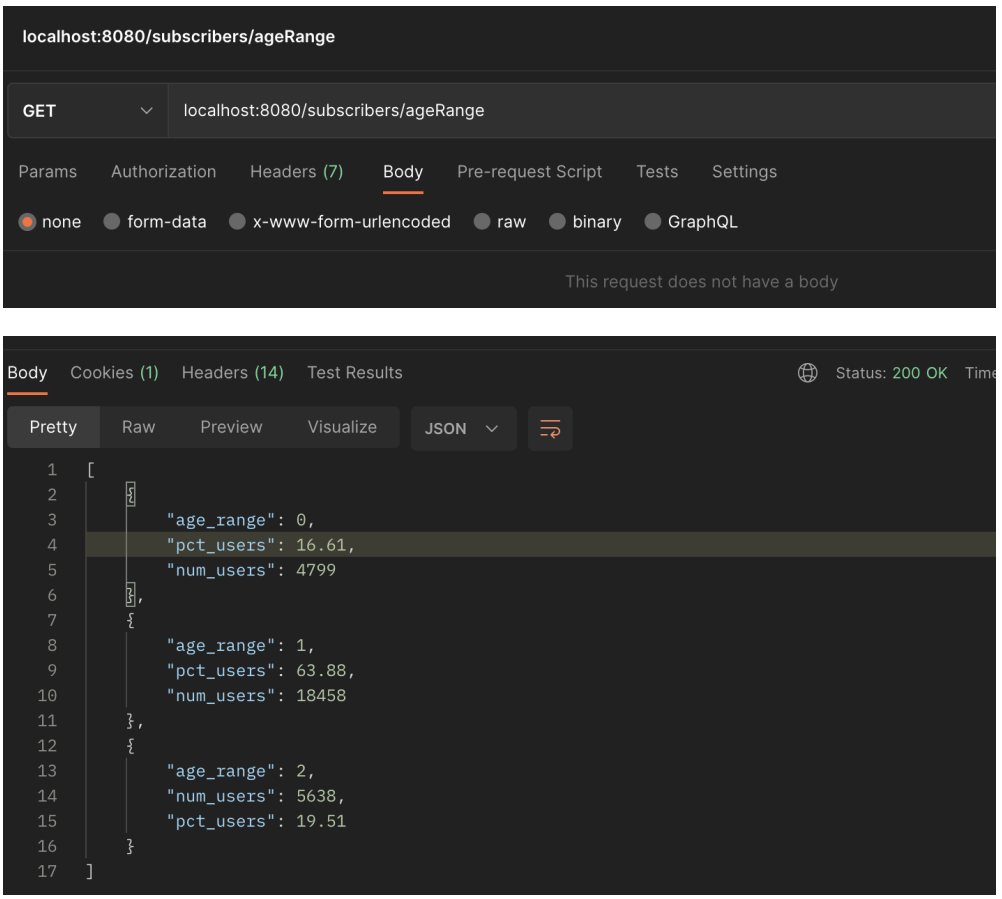
Figur 4.38 - Utklipp fra Code Coverage[43]

4.7.2. Integrasjonstest

Integrasjonstest er en essensiell og viktig del av utviklingsprosessen, særlig i testfasen. I integrasjonstest tester vi endepunkter enkeltvis og i sammenheng dersom de samspiller med

[43] Dekningsgrad, hvor mye av komponentene og metodene er blitt testet.

hverandre. Hensikten med integrasjonstest er å avdekke feil som kan befinne seg i endepunktene. Samtidig er denne fasen nødvendig for å sikre at henting og sending av data foregår på riktig måte. Dessuten kan man her avdekke om dataen som blir utvekslet er i riktig format og dekker systemets behov og krav. Vi har på grunn av mangel på tid og ressurser ikke klart å dekke alle endepunktene med integrasjonstest, men har testet de mest kritiske og sårbare endepunktene. Ut ifra resultatene vi har fått fra integrasjonstestingen, kan vi si at datautvekslingen mellom databasen og backend for det som er implementert foregår på en korrekt måte. For utførelse av integrasjonstestingen har vi brukt verktøyet Postman, som er mye anvendt i bransjen.



Figur 4.39 - Et eksempel fra en endepunkt-test i Postman

Se Appendiks 9 for hele tabellen over alle endepunktene som testes.

4.7.3. Systemtest

Under testfasen av utviklingen har vi vurdert å gjennomføre en systemtest i tillegg til integrasjonstest. Systemtest er en viktig del av testingen som skal dekke løsningen i sin helhet. I denne prosessen kan man avdekke feil i større grad som angår både

brukergrensesnitt og koblingen mellom frontend og backend. For typisk gjennomføring av systemtest ville vært å bruke verktøy som for eksempel Selenium og Jira Zephyr. Hvor Selenium blir brukt for selve testprosessen og Jira Zephyr for planlegging/strukturering/administrering av test-utførelsen. På grunn av at vi jevnlig testet applikasjonen underveis og avdekket oppståtte feil og mangler fortløpende har vi gått bort fra å utføre en systemtest, men heller rette fokus på akseptansetest.

4.7.4. Brukertesting (akseptansetest)

For å gjøre utviklingsprosessen så komplett som mulig, har vi bestemt oss for å kjøre en avsluttende brukertest, som også fungerer som en akseptansetest. For gjennomføringen av brukertesting har vi avtalt med oppdragsgiver å benytte 3 brukere som kommer til å anvende applikasjonen for å utføre deler av arbeidsoppgaven sin. For selve brukertestingen har vi satt av 45 minutter totalt, hvor hver bruker kommer til å få 15 minutter (10 minutter for å teste applikasjonen, 5 minutter for tilbakemelding og kommentar). Fremgangsmåten er som følger:

Steg 1	Før vi setter i gang brukertesting med en bruker, vil brukeren få en generell introduksjon over løsningen, og hva brukertestingen går ut på. I tillegg vil de bli informert hvor mange minutter vedkommende kommer til å få, og hvilke punkter vi vil mest ha tilbakemelding på.
Steg 2	Brukeren begynner å anvende applikasjonen uten veiledning, og prøver å finne informasjon ved å navigere i brukergrensesnittet.
Steg 3	I løpet av testingen vil brukeren bli bedt om å utføre spesifikke oppgaver eller scenarioer som er relevante for applikasjonens formål. Disse oppgavene kan f.eks. være å finne informasjon og en spesifikk type data om en kunde.
Steg 4	Observatører(gruppen) noterer brukerens handlinger, problemer og utfordringer de møter underveis, samtidig som vi noterer kommentarene som kommer fra brukeren.
Steg 5	Etter at brukertestingen med hver av brukerne er avsluttet, vil de få et skjema hvor det skal gis poeng fra 1 - 6 (6 er best) på noen konkrete områder. Disse punktene er: fargebruk, brukervennlighet, tilgjengelighet og organisering.

Steg 6	Vi kommer også til å ha en dialog på 5 minutter med brukerne for å høre deres kommentarer og tilbakemeldinger som vil bli notert.

Hensikten med denne brukertestingen er å høre oppdragsgiver og brukernes endelige kommentarer om løsningen, spesielt det som angår brukergrensesnittet. Et annet mål er å finne eventuelle feil, mangler, eller forbedringspunkter som kan brukes til fremtidig videreutvikling av applikasjonen. I tillegg er det også viktig for gruppen å vite i hvilken grad vi har klart å dekke kravspesifikasjonene fra oppdragsgiver, og samtidig få en pekepinn på hva som kunne forbedres eller gjøres annerledes.

Det er flere grunner til at det ikke har blitt utført noen brukertester tidligere i prosessen. Den første er at vi har hatt jevnlige møter med oppdragsgiver hvor applikasjonen ble demonstrert og tilbakemeldinger notert. Slik har vi prøvd å rette opp eventuelle feil som har blitt gjort. Den andre grunnen er at maskinlæring-delen, som er den mest essensielle delen i hele løsningen, ikke har vært ferdig utviklet som gjorde det utfordrende å gjennomføre en brukertesting som gir nytte. Likevel så tenker vi at vi burde ha testet brukergrensesnittet tidligere for å unngå å innføre store endringer i etterkant.

Resultater fra brukertestingen finner du i Appendiks 10.

Refleksjon etter brukertesting:
Vi gjennomførte en brukertesting helt på slutten av prosessen som en avsluttende fase. Likevel har denne delen gitt oss enorm utbytte i form av konstruktive tilbakemeldinger til selve løsningen, men også et solid grunnlag for mer refleksjon rundt gjennomføring av et slikt prosjekt. Med brukertestingen ønsket vi å få en pekepinn på hvor tilfredsstilt oppdragsgiveren ble av applikasjonen, og i tillegg fange opp forbedringspunkter som vi reflekterer over og ser på eventuelle fremtidige muligheter.

Herunder vil vi kort oppsummere tilbakemeldingene vi har fått under og etter brukertestingen og kategorisere de i tilsvarende fokusområder:

Fargebruk: I alle tre testrundene vi har utført, har vi kun fått positive kommentarer om fargebruken. Dette mener vi er på grunn av at vi har nøye valgt ut fargekombinasjoner som

også brukes på nettsidene til oppdragsgiveren som gjør at brukeren ikke føler løsningen er helt ukjent. Dermed vil vi si at dette er et område vi har lyktes med.

Brukervennlighet: Brukervennlighet handler om hvor enkelt og intuitivt et produkt eller en tjeneste er å bruke for brukerne. Dette har vært et område vi har forsøkt å vektlegge mer. Applikasjonen skal være enkel å bruke, og innholdet og navigeringen skal være intuitiv og skje på en naturlig måte. Basert på tilbakemeldingene vi har fått, så har vi dessverre ikke helt kommet i mål. Applikasjonen mangler for eksempel tilbakeknapp et par steder der det kunne ha vært nødvendig.

Tilgjengelighet: Tilgjengelighet går ut på at alle skal kunne anvende applikasjonen, uten hindringer eller utfordringer. Dette gjelder også brukere med funksjonshemminger, som har ulike utfordringer med bruk av digitale verktøy. Da vi hadde som mål å bygge en MVP, har vi dessverre ikke hatt stort fokus på tilgjengelighet under utviklingen. Til tross for at det har blitt lagt mindre vekt på tilgjengelighet, så har vi likevel sørget for at applikasjonen er enkel å anvende. Det er for eksempel mulig å bruke TAB eller ENTER i siden for å navigere og bekrefte valg.

Organisering: Under hele utvikling av frontend har vi vært opptatt av å ha en oversiktlig struktur i applikasjonen. Dette gjorde vi ved å plassere komponentene på en ryddig måte, som gjør det lettere å hente informasjon effektivt. Hver av våre brukere har uttrykt at de er fornøyde med plassering av komponenter og hvor ting er plassert.

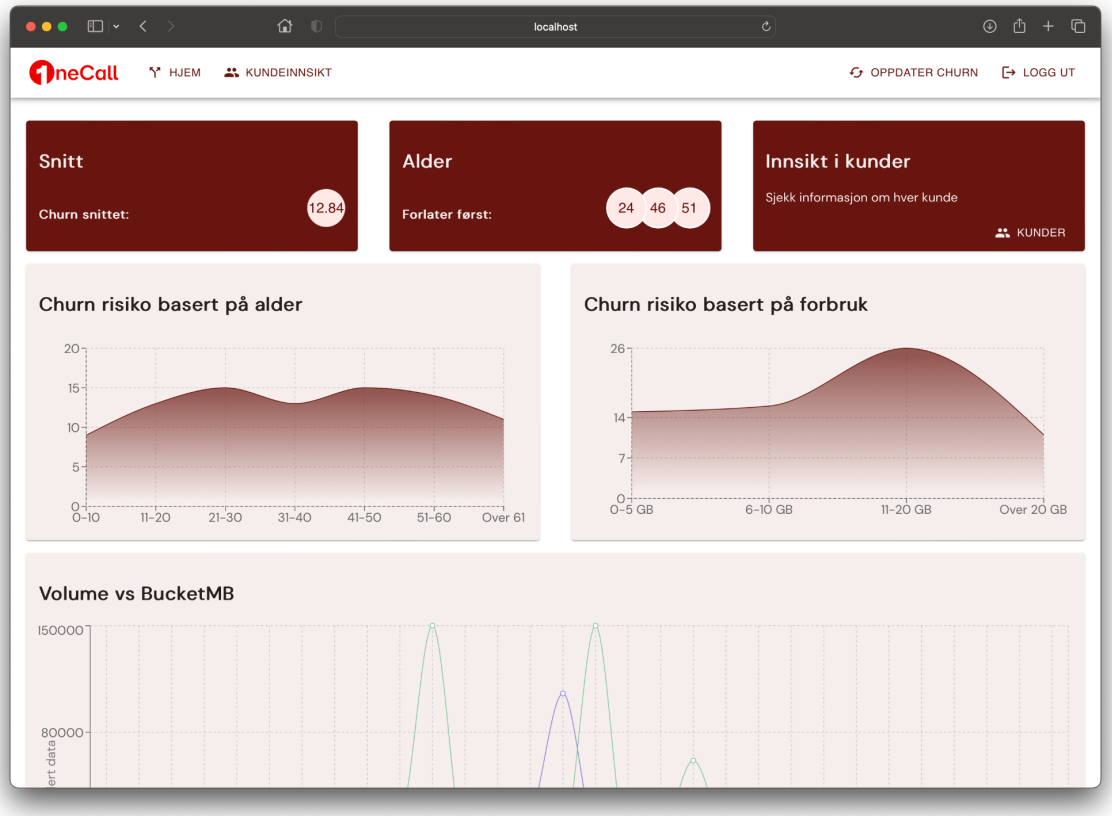
Ideelt sett burde brukertestingen inkludere brukere med behov innen et større spekter, men på grunn av at dette er en intern applikasjon har vi nøyd oss med å teste på OneCall ansatte fra ulike fagområder. Tilbakemeldingene vi fikk er positive. Dette skyldes mest at det ikke har vært implementert utstrakte tilgjengelighetsfunksjoner og at brukerne ikke har noen særskilte behov.

4.8. Brukerveiledning

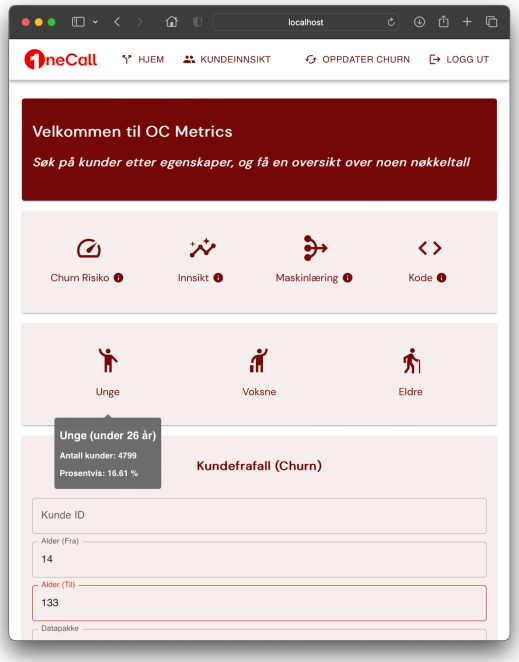
I denne delen viser vi noen skjermbilder av brukergrensesnittet og muligheter en bruker har.

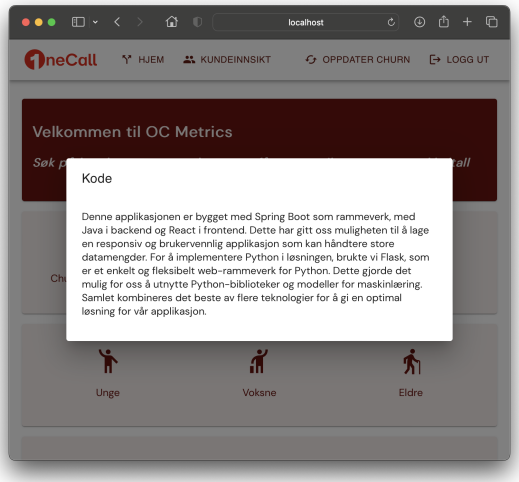
Det er delt i forskjellige sider:

- Dashboard med overordnet statistikk over forbruket, og churn risiko
- Kundeinnsikt som har et søkefelt og flere visualiseringer
- Tabell med brukere fra søket
- Detaljerte opplysninger om churn per kunde
- Visualisering

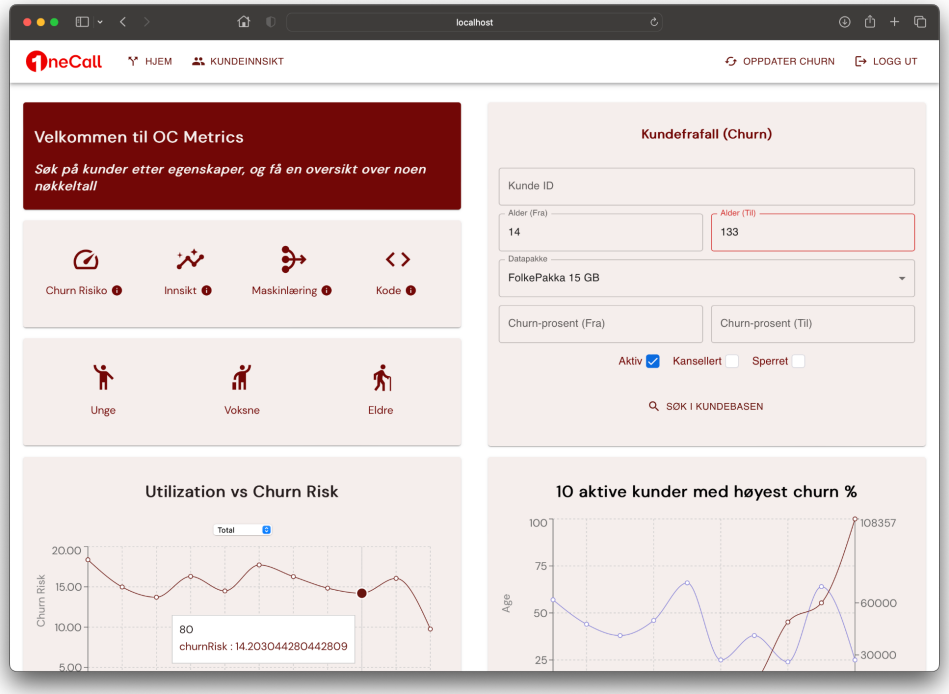


Når brukeren går inn på nettsiden, er det dette de vil møte først. Denne siden har en generell oversikt. Den viser noe statistikk og grafer.



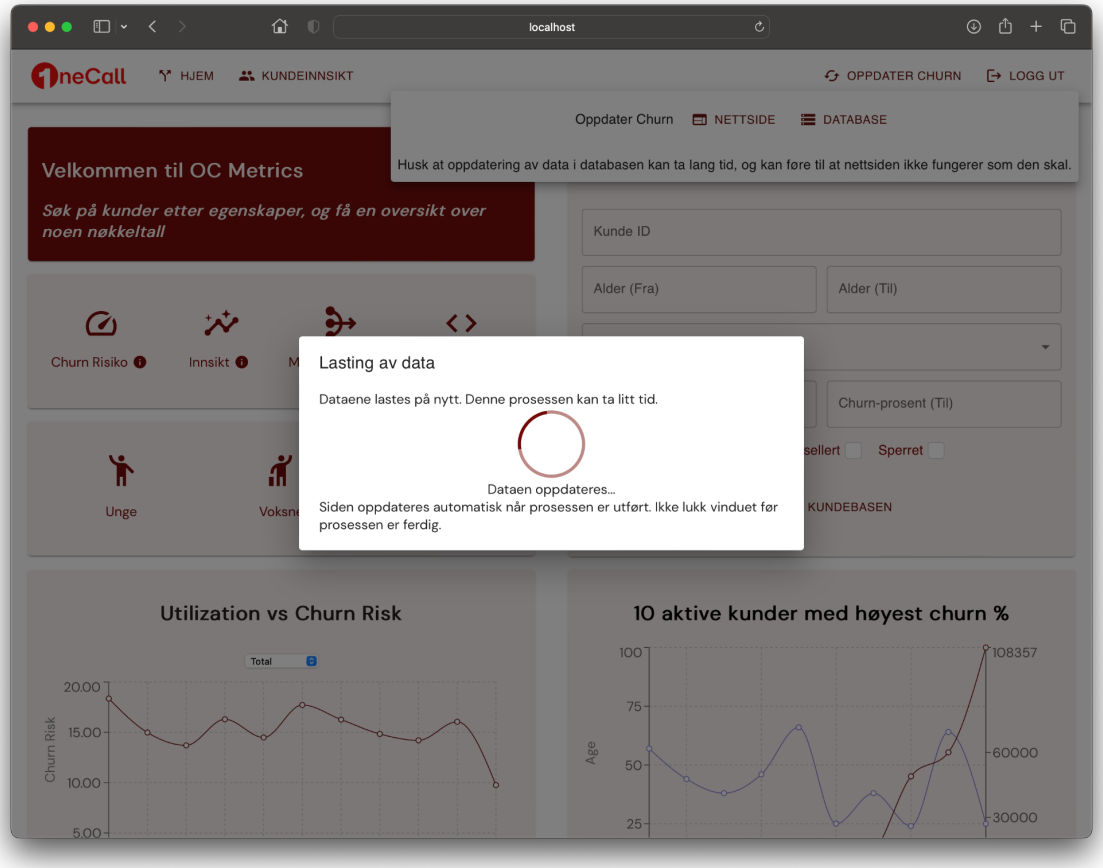


Vi har implementert mobile view for å gjøre applikasjonen lett tilgjengelig på mobile enheter. I tillegg vi også implementert informasjonsbokser på en estetisk måte, som gjør det mulig for brukere å se definisjoner på nøkkelord.

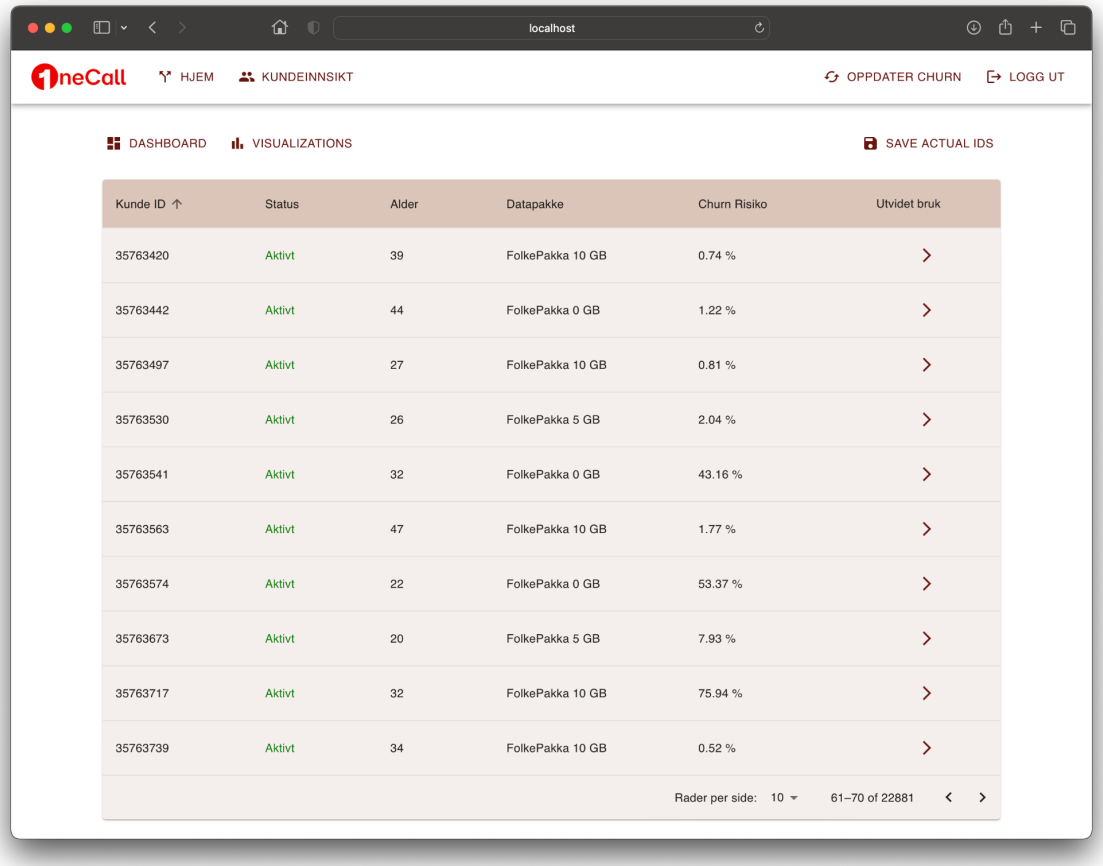


Kundeinnsikt-siden er der brukerne får flere grafer. Hovedfunksjonaliteten på denne siden

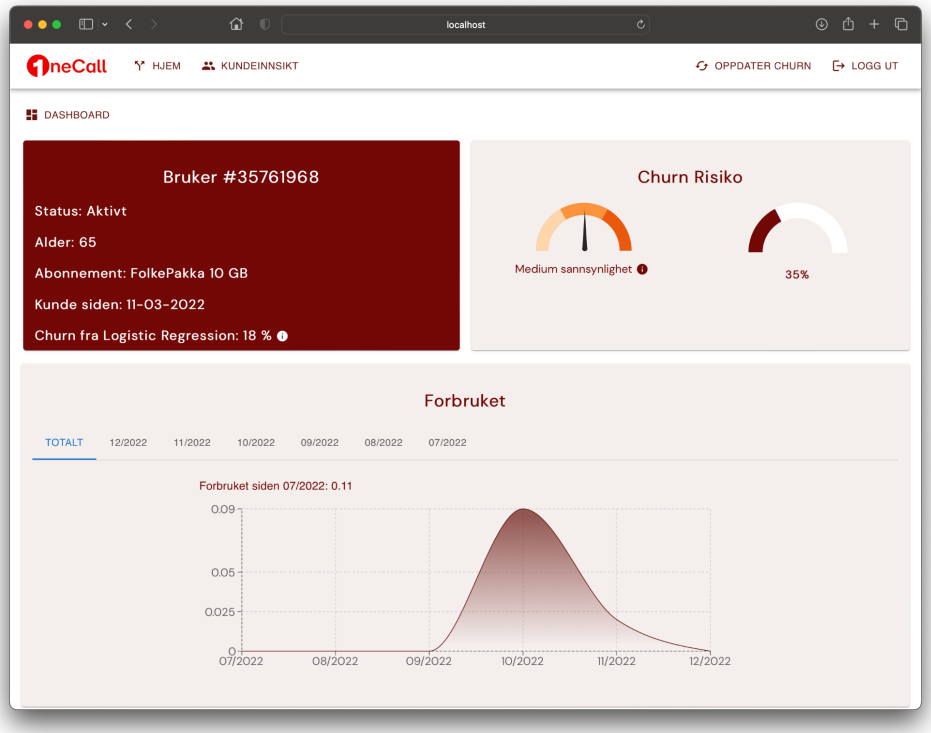
er søkefeltet hvor brukere kan søke gjennom kundebasen etter diverse parametre.



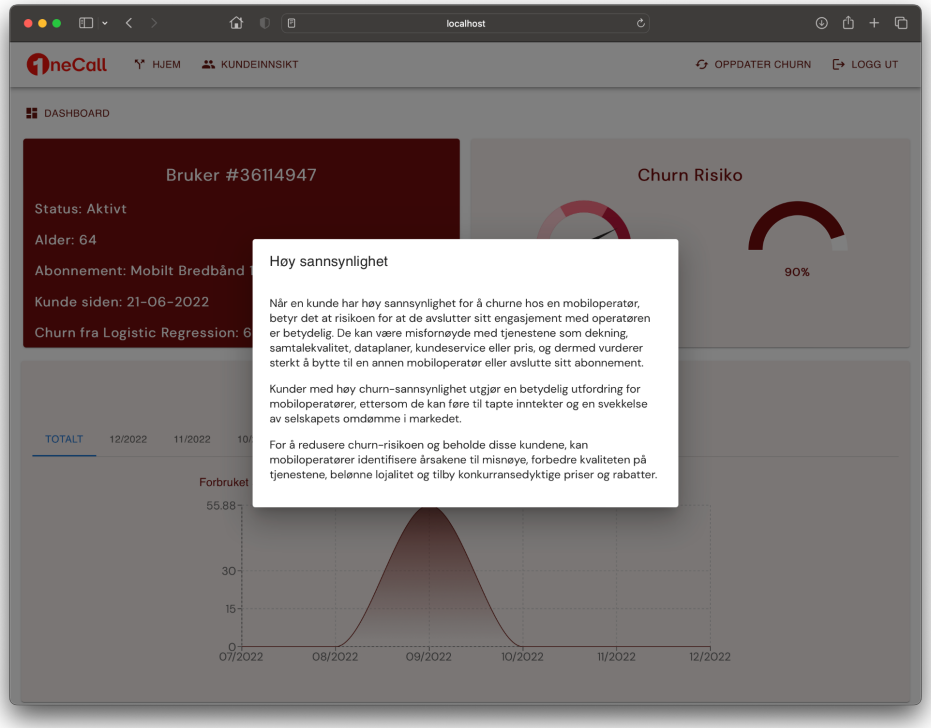
Brukere av applikasjonen vil ha mulighet til å oppdatere churn-analysen dynamisk, samt lagre resultatet i databasen.



Dette er søkeresultatet som vises. Det viser status, alder, datapakke og churn-prosent. Her er det mulig å ha flere kunder per side, bla gjennom og se deres forbruk og detaljert oversikt. Utvidet bruk knappen tar en videre til den detaljerte oversikten.



Denne siden viser oversikt over en bruker fra databasen. Her vises det oversikt over status og noen personlige opplysninger, samt churn risiko representert i form av prosent og churn-indikator. Vi viser også resultatet fra Logistic Regression modellen.



Churn-indikatoren gir en beskrivelse av risiko, og generelle anbefalinger rundt dette.

5. Avsluttende del

5.1. Oppsummering

Vi mener at vi har utført prosjektet på en tilstrekkelig måte, både i forhold til oppdragsgiver og som et bachelorprosjekt. Vi har utviklet en løsning som fungerer etter oppdragsgiverens bestilling og deres kravspesifikasjoner. Samtidig har vi jobbet strukturert og fordelt arbeidsoppgaver rettferdig mellom gruppemedlemmene slik at alle fikk omtrent like mye å gjøre og like mye læringsutbytte av prosjektarbeidet. Gruppen konkluderer med at oppdragsgiver har fått mer innsikt i hvordan de kan implementere maskinlæring for churn-prediction, som gir dem bedre forutsetninger for å utarbeide gode strategier for kundelojalitet. Løsningen vår er et godt utgangspunkt for videreutvikling og kan bidra til å sette et grunnlag for videre arbeid.

5.2. Refleksjon

Vi tror at løsningen vår fungerer såpass bra at den vil kunne ha betydning for oppdragsgiverens forretningsstrategi og at det vil være hensiktsmessig for dem å videreutvikle løsningen. Det var noen opplysninger som oppdragsgiver besitter som kan spille en rolle i sammenheng med churn, men som ikke var inkludert i treningen av modellen. Vi tror et bedre resultat blant annet vil oppnås om oppdragsgiver velger å utvide og videreutvikle modellen vår slik at den blir trent ytterligere med ytterligere informasjon om flere kunder over en lengre periode.

Bachelorprosjektet har fungert som en god introduksjon til arbeidslivet som utviklere. Vi har fått et praktisk innblikk i programvareutvikling og prosessene som er involvert. Vi har måttet sette oss inn i nye problemstillinger og lære oss nye teknologier når det var nødvendig. Arbeidsgiver ville helst ha maskinlæring som en del av prosjektet, og ingen av oss hadde inngående kunnskaper om dette bortsett fra et introduksjonskurs. Dette gjaldt også React, da ingen av oss hadde brukt dette noe særlig ved oppstart. Det å kunne plukke opp ny kunnskap og anvende tidligere erfaring med praktisk kunnskap for å se mønstre og sammenhenger er uvurderlig i en jobb som programvareutvikler. Dette har vært en gjentagende faktor i utviklingen av dette prosjektet. Vi vil ta med oss dette videre ut i arbeidslivet, hvor teknologien er i stadig utvikling, og som gjør at en softwareutvikler er nødt til å stadig sette seg inn i nye problemstillinger.

Det var motiverende å gå løs på en problemstilling som en etablert oppdragsgiver ikke hadde løst selv. Vi så på dette som en unik mulighet til å kunne fullføre et utviklingsprosjekt fra bunnen av hvor vi er med på å planlegge og utforme et helt produkt. Vi hadde veldig mye frihet i hvordan vi gjennomførte prosjektet og hvilke teknologier vi valgte. Det ga oss fleksibilitet, men i ettertid kunne vi gjerne vært tydeligere i å avgrense hva oppdragsgiver ønsket. Dette kjente vi på underveis i prosjektet, og spesielt etter avsluttende brukertesting av brukergrensesnittet.

Vår MVP var en applikasjon for fremvisning av statistikk av kundebasen, og vi ville gjerne få til en nøyaktig Churn-prediksjon av kundene med bruk av maskinlæring. Vi synes vi har nådd dette målet, da testing av våre prediksjoner på den faktiske kundebasen fikk god evaluering (Appendiks 11).

Som gruppe er vi fornøyd med vår egen innsats med tanke på måloppnåelsen innenfor den gitte tidsrammen og prosjektets vanskelighetsgrad. Vi kunne laget en enda mer komplett løsning med et mer utfyllende dashboard hadde rammene vært annerledes, men vi mener at vi har klart å produsere en god løsning for kjerne-problemstillingen som var:

Hvordan konstruere en løsning som gir OneCall mer kundeinnsikt, spesielt med hensyn til churn risiko?

5.3. Prospekt - videre prosess

For videre arbeid ville vi ha sett på muligheten for å integrere vår løsning inn i OneCall sitt interne system. Dette ville dog innebære oppkobling til datakilder som inneholder produksjonsdata, og vi måtte trolig sett nærmere på forbedring av ytelsen til vår løsning. Den fungerer fint med 25.000 kunder, men med hele kundemassen vil ytelsen bli en utfordring. Vi gjør en del av kalkulasjonene til visualiseringer i JavaScript i klienten. Noe av dette kunne ved en videre prosess blitt flyttet til backend. Da måtte vi eventuelt ha satt oss inn i de forskjellige mulighetene og sett på fordeler og ulemper de medførte.

Vi har tilrettelagt funksjoner som gjør det mulig for OneCall å kjøre prediksjoner på systemer som har data som er i endring. Det kan lagres en oppdatert churn-prosent på alle kunder ved

oppstart av applikasjonen. Vi har derimot hatt statiske data som det ikke har vært nødvendig å oppdatere fortløpende. Ved en videre prosess ville vi lagt til en mer avansert databasestruktur med flere muligheter for oppdatering, lagring og sletting.

En nærmere testing av churn-analysens nøyaktighet er en faktor vi gjerne ville jobbet mer med. Med mer tid til rådighet ville vi også sett på å trene maskinlærings-modellen med en større og mer komplett datamengde.

Mer grundig testing av samtlige applikasjoner er noe vi anbefaler. Vi har testet hovedfunksjonalitet, men vil ønske en større testdekning for backend, samt implementere tester for frontend og ML-API ved neste iterasjon.

Et bedre brukergrensesnitt hvor vi tar mer høyde for WCAG-score for universell utforming er også anbefalt.

Tross alt har oppdragsgiver uttrykt tilfredshet med resultatet. Vi håper at prosjektet ikke bare har vært nyttig og lærerikt for oss, men også gitt verdi for oppdragsgiveren. Det hadde morro om noen av våre funn kunne tas med videre i deres arbeid.

Takk for oss!

6. Referanser og vedlegg

6.1. Kildeliste

ABAP. (2019). Separation of concerns - ABAP keyword documentation. https://help.sap.com/doc/abapdocu_753_index_htm/7.53/en-US/abenseperation_concerns_guidl.htm

About. PostgreSQL. (2023). https://www.postgresql.org/about/

Ahmad, A. K., Jafar, A., & Aljoumaa, K. (2019). Customer churn prediction in telecom using machine learning in big data platform. Journal of Big Data, 6 (1), 1-24.

Ahn, J., Hwang, J., Kim, D., Choi, H., & Kang, S. (2020). A survey on churn analysis in various business domains. IEEE Access, 8, 220816–220839. https://doi.org/10.1109/access.2020.3042657

Arefeen, M. S., & Schiller, M. (2019). Continuous Integration Using Gitlab. Undergraduate Research in Natural and Clinical Science and Technology Journal, 3, 1-6.

Authentication. Authentication : Spring Security. (2023). https://docs.spring.io/spring-security/reference/servlet/authentication/index.html

Beerbaum, D. O. (2023). Agile Strategy–Achieving Sustainable Advantage. Available at SSRN.

Brownlee, J. (2019) A Gentle Introduction to Ensemble Learning Algorithms. machinelearningmastery https://machinelearningmastery.com/tour-of-ensemble-learning-algorithms/

Built-in react components. React. (2023). https://react.dev/reference/react/components

Collings, T. (2021, August 10). Controller-service-repository. Medium. https://tom-collings.medium.com/controller-service-repository-16e29a4684e5

Deshpande, C. (2023, February 23). What is Angular?: Architecture, features, and advantages [2022 edition]. Simplilearn.com. https://www.simplilearn.com/tutorials/angular-tutorial/what-is-angular

Duldulao, D. B., & Cabagnot, R. J. L. (2021). Getting Ahead in React. In Practical Enterprise React: Become an Effective React Developer in Your Team (pp. 1-10). Berkeley, CA: Apress.

Editor. (2019, October 18). Functional and nonfunctional requirements: Specification and types. AltexSoft. https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/

Fordeler Ved Skyoverføring: Microsoft Azure (n.d.). https://azure.microsoft.com/nb-no/resources/cloud-computing-dictionary/benefits-of-cloud-migration/#benefits

Fullview (u.å.) What Is A Good Churn Rate For SaaS Companies? (2023) https://www.fullview.io/blog/average-churn-rate-for-saas-companies

Friess, E. (2023). Scrum in Classroom Collaborations: A Quasi-Experimental Study. Journal of Business and Technical Communication, 37(1), 68-94.

Front-end frameworks popularity (React, Vue, Angular and Svelte). (2023). Gist. https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190

GeeksforGeeks. (2022, April 26). Postman vs Swagger. GeeksforGeeks. https://www.geeksforgeeks.org/postman-vs-swagger/

Gos, K., & Zabierowski, W. (2020, April). The comparison of microservice and monolithic architecture. In 2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH) (pp. 150-153). IEEE

Hayes, A. (2023, January 6). *Risk analysis: Definition, types, limitations, and examples*. Investopedia. https://www.investopedia.com/terms/r/risk-analysis.asp

Hutagikar, V., & Hegde, V. (2020). Analysis of Front-end Frameworks for Web Applications. Int. Research J. of Engineering and Tech, 7(4), 3317-3320.

IBM. (u.å.). Test-Driven Development. Hentet 2023 fra https://www.ibm.com/garage/method/practices/code/practice_test_driven_development/

IBM. HTTP basic authentication. (2023). https://www.ibm.com/docs/en/cics-ts/5.4?topic=concepts-http-basic-authentication

Java Basic Refresher - Stanford University. (2008). https://web.stanford.edu/class/archive/cs/cs108/cs108.1082/JavaBasicRefresher.pdf

Kofler, M., & Kofler, M. (2004). What Is MySQL? (pp. 3-19). Apress.

Kontrast. Tilsynet for universell utforming av IKT. (2023). https://www.uutilsynet.no/veiledning/kontrast/48

Kouomeu, K. (2023, May 12). Unit testing the service layer of Spring Boot Application. 1kevinson.https://1kevinson.com/testing-service-spring-boot/

Lalwani, P., Mishra, M. K., Chadha, J. S., & Sethi, P. (2022). Customer churn prediction system: a machine learning approach. Computing, 1-24.

Lindsjørn, Y. (2021). Prosjektledelse. Systemutvikling DAFE2200. OsloMet.

Logistic regression in machine learning - javatpoint. www.javatpoint.com. (n.d.-a). https://www.javatpoint.com/logistic-regression-in-machine-learning

Logo, farger. OneCall. (2023). https://onecall.no/

Loshin, P., & Bigelow, S. J. (2021, October 6). What is the linux operating system?. Data Center. https://www.techtarget.com/searchdatacenter/definition/Linux-operating-system

Myhr, R., & Bårdgård, T. (2022, June 16). Mikrotjenester - Driftsstøtte (IM-ITK VG2) - NDLA. ndla.no. https://ndla.no/nb/subject:26f1cd12-4242-486d-be22-75c3750a52a2/topic:63796e04-10bd-47d7-b3d9-ffc0272e8744/resource:4764c942-f580-49d4-8013-1c2bc258133a

N. Sharma. (2020, April 16). React vs angular vs ember vs Vue: Compare javascript framework. Apptunix Blog. https://www.apptunix.com/blog/react-vs-angular-vs-ember-vs-vue-best-javascript-framework/

Nighania, K. (2019, January 30). Various ways to evaluate a machine learning models performance. Medium.

https://towardsdatascience.com/various-ways-to-evaluate-a-machine-learning-models-performance-230449055f15

Nätt, T. H. (2020). JavaScript. Store norske leksikon. https://snl.no/JavaScript

Package javax.persistence (Java(TM) EE 7 Specification APIs). (2015, June 1). https://docs.oracle.com/javaee/7/api/javax/persistence/package-summary.html

Rahaman, S. (2023, May 15). Average churn rate for SAAS companies (2023 update).

Raschka, S., Patterson, J., & Nolet, C. (2020). Machine learning in Python: Main developments and technology trends in data science, machine learning, and artificial intelligence. Information, 11(4), 193.

Rautio, A. J. O. (2019). Churn prediction in SaaS using Machine Learning (Master's thesis)

Real world mutation testing. PIT Mutation Testing. (2023). https://pitest.org/

Rolstadås, A. (NTNU). (2019). Milepæl – Prosjektledelse. Store norske leksikon. https://snl.no/milep%C3%A6l_-_prosjektledelse

Reichheld F. & Earl Sasser, W. (2014, August 1). Zero defections: Quality Comes to services. Harvard Business Review. https://hbr.org/1990/09/zero-defections-quality-comes-to-services

Sabar, G. (2020, January 28). Deploy a flask application to ubuntu 18.04 server. Medium. https://medium.com/analytics-vidhya/deploy-a-flask-application-to-ubuntu-18-04-server-69b414b10881

Schwaber, K. (1997). Scrum development process. In Business Object Design and Implementation: OOPSLA'95 Workshop Proceedings 16 October 1995, Austin, Texas (pp. 117-134). Springer London.

Software Testing Genius, & Gupta, Y. (2018, August 30). Simple explanation of hierarchy of testing levels. Software Testing Genius. https://www.softwaretestinggenius.com/simple-explanation-of-hierarchy-of-testing-levels/

Spring Framework. Chapter 10. Application layering. (u.å.). https://docs.spring.io/spring-roo/reference/html/base-layers.html

Takeuchi, H., & Nonaka, I. (1986). The new new product development game. Harvard business review, 64(1), 137-146.

Testing matrices. Testing Matrices - BrownDog - Confluence. (2023). https://opensource.ncsa.illinois.edu/confluence/display/BD/Testing+Matrices

The economics of E-loyalty. HBS Working Knowledge. (2000). https://hbswk.hbs.edu/archive/the-economics-of-e-loyalty

The React Component Library You always wanted. MUI. (2023). https://mui.com/

Ullah, I., Raza, B., Malik, A. K., Imran, M., Islam, S. U., & Kim, S. W. (2019). A churn prediction model using random forest: analysis of machine learning techniques for churn prediction and factor identification in the telecom sector. IEEE access, 7, 60134-60149.

Unit testing - javatpoint. www.javatpoint.com. (n.d.). https://www.javatpoint.com/unit-testing

UseContext. React. (2023). https://react.dev/reference/react/useContext

Vihovde & Grønning. (2015). Unntakshåndtering. https://www.cs.hioa.no/~evav/uvstoff/intro/exceptions.html

What is .NET? An open-source developer platform. Microsoft. (2023). https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet

What is Flask Python? - Python Tutorial. (2021). https://pythonbasics.org/what-is-flask-python/

What is Java Spring Boot?. IBM. (2023). https://www.ibm.com/topics/java-spring-boot

What is Java?. IBM. (2023). https://www.ibm.com/topics/java

What is Mysql? Everything you need to know. Talend. (2023). https://www.talend.com/resources/what-is-mysql/

What is Object/Relational Mapping? - Hibernate Orm. Hibernate. (n.d.). https://hibernate.org/orm/what-is-an-orm/

What is test-driven development?. testdriven.io. (2023). https://testdriven.io/test-driven-development/

Wieruch, R. (2019, March 18). React function components. RWieruch. https://www.robinwieruch.de/react-function-component/

YouTube. (2017, November 13). The challenge of predicting churn in the Enterprise World (data dialogs 2017). YouTube. https://www.youtube.com/watch?v=WO_aXPdjbS4

YouTube. (2021, March 24). A chat with Andrew on MLOps: From model-centric to data-centric AI. YouTube. https://www.youtube.com/watch?v=06-AZXmwHjo

6.2. Appendiks

Notasjon: Appendiks - nummer - tittel - format

Appendiks 1: Prosjektdagbok (PDF)

Appendiks 2: Retrospektiv Sprint 1-5 (PDF)

Appendiks 3: Roadmap-Fremdriftsplan (PNG)

Appendiks 4: Planer (PDF)

Appendiks 5: Sprinter og møtenotater (JPG)

Appendiks 6: Risikoanalyse (PDF)

Appendiks 7: Systemarkitektur ulike utkast (PDF)

Appendiks 8: Testmanuskript (PDF)

Appendiks 9: Integrasjonstest (PDF)

Appendiks 10: Brukertesting (PDF)

Appendiks 11: Churnprediction status sensurert (XLSX)

Appendiks 12: Plot med features (PNG)

Appendiks 13: Aktivitetsdiagram (PDF) - *ikke nevnt i rapporten*