Høgskulen på Vestlandet

BACHELOROPPGAVE

Kundeadministrasjonsportal for SAAS

Customer Management Portal for SAAS

Iver Sande, Mikal Bø

Bachelor i ingeniørfag, data
Fakultet for teknologi, miljø- og samfunnsvitenskap
Violet Ka I Pun
12.05.2024

Jeg bekrefter at arbeidet er selvstendig utarbeidet, og at referanser/kildehenvisninger til alle kilder som er brukt i arbeidet er oppgitt, *jf. Forskrift om studium og eksamen ved Høgskulen på Vestlandet, § 10.*

Høgskulen på Vestlandet

Fakultet for teknologi, miljø- og samfunnsvitenskap
Institutt for datateknologi, elektroteknologi og realfag

TITTELSIDE FOR HOVEDPROSJEKT

Rapportens tittel: Kundeadministrasjonsportal for SAAS	*Dato :* *12.05.2024*
Forfatter(e): Iver Sande og Mikal Bø	*Antall sider u/vedlegg : 51* *Antall sider vedlegg : 96*
Studieretning: Dataingeniør	*Antall disketter/CD-er :* 0
Kontaktperson ved studieretning: Violet Ka I Pun	*Gradering:* Ingen
Merknader: Ingen	

Oppdragsgiver: Easly AS	*Oppdragsgivers referanse:* Ingen
Oppdragsgivers kontaktperson: Felix Santhi	*Telefon:* 411 73 113

Sammendrag:

Målet til denne bacheloroppgaven er gjennom utvikling av en full stack applikasjon, å effektivisere of forenkligjøre kundeadministrasjon for Easly AS. Gjennom prosjektet vil utviklingsmetodikk og evaluering være i fokus for å vurdere hvordan den valgte metodikken påvirker resultatene.

Summary:

The goal of the bachelor thesis is to, through the development of a full stack application to streamline and speed up customer administration for Easly AS. The project focuses on development techniques and evaluation to work out how they affect the end result.

Stikkord:

Fullstack	Brukertesting	Ekstrem programmering

Høgskulen på Vestlandet, Fakultet for teknologi, miljø- og samfunnsvitenskap
Postadresse: Postboks 7030, 5020 BERGEN Besøksadresse: Inndalsveien 28, Bergen
Tlf. 55 58 75 00 Fax 55 58 77 90 E-post: post@hvl.no Hjemmeside: http://www.hvl.no

Forord

Denne rapporten dokumenterer arbeidet bak bachelorprosjektet *Kundeadministrasjonsportal for SAAS*. Dette prosjektet gjennomføres i sammenheng med avslutningen av vår treårige bachelorgrad i dataingeniør med spesialisering innen programvareutvikling- og arkitektur ved Høgskulen på Vestlandet (HVL).

Vi vil takke alle bidragende parter på veien mot produktet og den fullførte rapporten. Vi vil takke veileder Violet Ka I Pun for masse gode tilbakemeldinger og god støtte underveis i skriveprosessen. Videre ønsker vi å takke Easly AS for god hjelp med introduksjon til selskapet og evaluering. Sist vil vi takke kjente og kjære for gjentakende gjennomlesing som gjorde rapporten lettleselig og forståelig.

Til slutt ønsker vi innad i prosjektgruppen å takke hverandre for alle timene lagt inn i denne prosessen. Prosjektet har gitt oss en forståelse av hvordan vi jobber og tenker. Dette har gjort at samarbeidet vårt har blitt bedre og vi er takknemlige for hele prosessen.

Innhold

Ordforklaringer

Application Programming Interface Et grensesnitt brukt for å utveksle data mellom systemer eller internt i et system. .

Branch En peker til en versjon av kode i git.

Deployment Prosessen der software blir gjort tilgjengelig til å bli brukt av bruker i systemet og andre program.

Hypertext Transfer Protocol En protokoll for å utføre nettverkskall, standard innenfor web.

Integrated Development Environment Programvare brukt til utvikling.

Java Runtime Environment Bindeleddet mellom Java kode og operativsystemet.

JavaScript Object Notation Dataform basert på nøkkel-verdi par.

Software Programvare er en samling av instruksjoner, data eller programmer som muliggjør at en datamaskin kan utføre spesifikke oppgaver eller operasjoner..

Software as a service Softvare som har en abonnements modell heller enn at du kjøper den en gang.

Uniform Resource Identifier En streng som representerer en adresse eller et navn.

Uniform Resource Locator En streng som representerer en adresse, brukes om f.eks nett-adresser.

Forkortelser

API Application Programming Interface.

HTTP Hypertext Transfer Protocol.

IDE Integrated Development Environment.

JRE Java Runtime Environment.

JSON JavaScript Object Notation.

LTS Long Time Support.

PR Pull request.

SAAS Software As A Service.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

Figurer

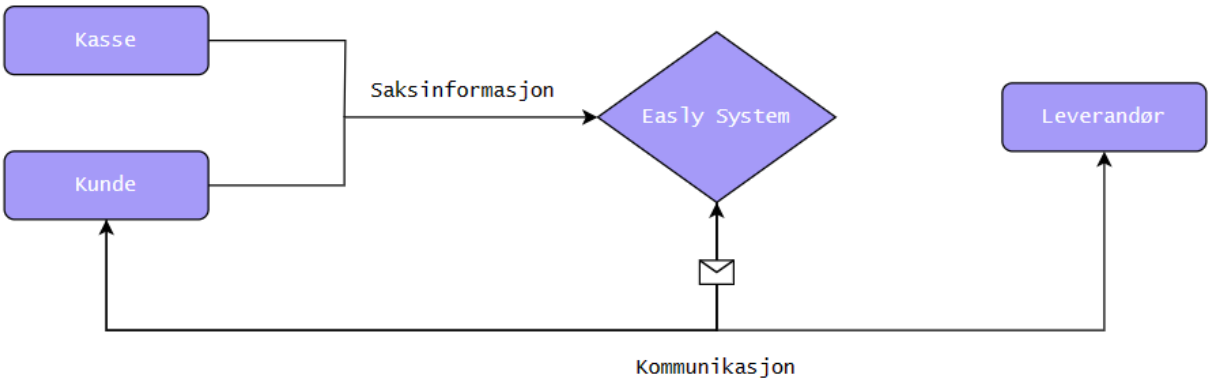
Tabeller

1 INNLEDNING

1.1 Prosjekteier

Easly AS er et SAAS selskap i en kraftig skaleringsfase. Selskapet har sin forretningsvirksomhet i Norge og produktutvikling på Sri Lanka. Easly AS leverer en plattform for kommunikasjon og prosesstyring i detaljhandel bransjen. Problemstillingen selskapet står overfor i dag med tanke på veksten og økt kundemasse er behovet for et system som tilgjengeligjør administrering av brukerene til kunder. Selskapet har i dag noen av Norges største retail kjeder innen møbel, mote og barneklær- og utstyr til å primært håndtere ettersalgsprosesser. Ettersalgsprosesser, som reklamasjoner, retur, reparasjon og erstatning, er ofte komplekse og involverer kommunikasjon med flere aktører – kunde, leverandør, reparatør, regnskap og transportør. Disse prosessene er i dag preget av manuelle prosedyrer hvor hovedverktøyene er excel, post-it og mapper.

Plattformen, som heter Easly, bygger på en konfigurerbar motor som gjør det mulig å tilpasse ulike prosesser og knytte kommunikasjon med interne og eksterne aktører i disse prosessene. Den sørger for sømløs kommunikasjon og saksbehandling med disse på en og samme plattform. Den overordnede saksgangen kan beskrives og kategoriser som følge: 1) forbruker registrer en henvendelse digitalt eller fysisk i butikk med nødvendig informasjon om produktet og dets mangler eller feil, 2) hvis reklamasjon, sendes den til leverandør med nødvendig informasjon for videre saksbehandling og vurdering, 3) leverandør følger opp butikken med utfall (f.eks. reparasjon, kreditnota, ny del, nytt produkt, osv.), 4) butikken følger opp kunde med utfall. Figur 1.1 eksemplifiserer prosessen for en standard sak som inkluderer behov for kommunikasjon med kunde og leverandør.



Figur 1.1: Flytdiagram for Easly

1.2 Kontekst

En del av daglig drift i selskapet er håndtering av brukerene til systemet. Disse brukerene er det som gir kundene mulighet til å opprette saker, samle data og kunne bruke systemet i det hele tatt. Brukeropprettelse behøves ofte, da selskapet jevnlig får nye kunder, tilbakevendende kunder som ønsker flere brukere eller selskaper som bruker systemet i en prøveperiode. Utover opprettelse er det behov for endring av brukere. Endringene kan eksempelvis innebære navnet som vises for kunden og deres kunder eller annen data som er lagret i sammenheng med butikken. I systemet eksisterer unike felter koblet til brukerene. Dersom det lagres duplikater vil data kunne bli koblet til feil bruker.

1.3 Motivasjon

Dagens løsning for opprettelse av brukere er gjennom en tredjepartsapplikasjon som er koblet til systemet. Løsningen er komplisert og lite brukervennlig og det har blitt gjentatt viktigheten av at felter er riktig fylt ut. Det eksisterer ikke en praktisk metode å endre data på når brukerne er laget. Løsningen her er enkeltpersonavhengig og ligger i tekstlig overlevering av ønskede endringer der mottaker må koble seg direkte til databasen for å endre felter. Dette tar lang tid, krever god kommunikasjon og kan resultere i flere feil.

Denne situasjonen gjelder da også for sletting og oversikt over eksisterende brukere.

1.4 Problembeskrivelse og mål

Kundebehandlere, brukerstøtte og ledelsen har en travel hverdag med å holde oversikt over nåværende kunder, svare på henvendelser og gjennomføre opplæring og opprettelse av brukerkontoer for nye kunder. Dette resulterer i et økende behov for en effektiv måte å administrere brukere på, som ikke er avhengig av enkeltpersoner. Dette gjør at en unngår flaskehalser i bruk av systemet. Selskapet mottar stadig flere henvendelser knyttet til brukeropprettelse og administrasjon, og målet med prosjektet er å tilrettelegge for effektiv brukeradministrasjon for de ansatte. Dette vil bidra til raskere og mer nøyaktige gjennomføringer av forespørsler og endringer, og kan potensielt øke kundetilfredsheten samtidig som det åpner opp for flere muligheter for selskapet.

1.5 Oppbygging av rapporten

Rapporten har hittil beskrevet selskapet og problematikken som selskapet står ovenfor. I kapittel to vil det forklares mer spesifikt hva prosjektet går ut på. Det vil også utdypes hva som var startideen og litt om hvordan den ble utformet til sluttideen. Utover det vil det sees på hva som kan begrense og hva som er der for å hjelpe underveis. Videre i kapittel tre blir det lagt fram hvordan løsningen skal utvikles, både arkitekturmessig og rent teknisk. Dette vil gi innsikt i tanker om løsningen og hva som ble det endelige valget. Utover dette vil risikoer, utviklingsplan og utviklingsmetodikk beskrives nærmere. Kapittel fire er et dypdykk i hvordan løsningen fungerer, samt hvordan utviklingen ble utført. Videre i kapittel fem blir resultatet gjennomgått, med hovedfokus på prosesser og evaluering. Kapittel seks og syv ser på deler av prosjektet som ikke ble som planlagt og hvorfor, samt en evaluering av prosjektet som en helhet.

2 PROSJEKTBESKRIVELSE

2.1 Praktisk bakgrunn

I dette kapittelet skal prosjektets grunnlag gjennomgås. Hva det baserer seg på, hva som initielt var tenkt om prosjektet, hvilke forutsetninger som eksisterer og litt mer om problemstillingen.

2.1.1 Tidligere arbeid

Det er ikke gjort noe tidligere arbeid i sammenheng med løsningen som ønskes utviklet. Det som eksisterer er opprettelse av brukere, denne løsningen skal brukes via prosjektet. Dette vil si at applikasjonen skal kobles på og sende data til brukeroppretteren, noe som samler alt ansvaret for brukeradministrasjon. For resten av funksjonaliteten blir alt bygget fra bunnen av. Dette gjør at den kan skreddersys til dagens problematikk.

2.1.2 Initielle krav

Easly er i stor grad avhengig av at det kan konfigureres og håndteres av alle de ansatte. Dette fører til det overordnede kravet om brukervennlighet. De funksjonelle kravene til systemet ligger i administrasjon av kundebrukere. Derunder opprettelse, endring, sletting, visning og historikk over brukeropprettelser. Utover dette ble det satt krav til hvilke teknologier som skulle brukes i systemet. Det ble også besluttet at tjenesten skulle bygges utvidbar og dokumentert slik at videre utvikling kunne finne sted. Dette gjør at det senere kan utvikles ytterlige funksjonalitet dersom det skulle være behov.

2.1.3 Initiell løsnings-ide

Den initielle ideen var en omfattende portal hvor det eksisterte opprettelser, endringer,

sletting av brukere og faktura til kundene. I sammenheng med fakturaløsningen skulle tilgangen til brukeren automatisk fjernes ved ubetalt faktura. Kundene skulle også selv kunne logge inn i portalen å gjøre endringer på egen data.

2.1.4 Avgrensninger

Det ble tidlig avklart at det måtte kuttes i funksjonalitet for at det skulle være mulig å gjennomføre prosjektet innenfor tidsrammen. Fakturahåndteringen ble tidlig forstått å være av for stort omfang. Denne funksjonaliteten ville hatt behov for opp til flere integrasjoner, enten med annen faktureringssoftware, med banken, eller at noen manuelt registrerte mottatt innbetaling. Siste punkt ville jobbet mot sin hensikt da det ville lagt til mer komplikasjoner enn hjelp, dette med tanke på effektivitet og brukervennlighet i systemet.

Muligheten for kunder å logge inn var en interessant tanke som dessverre også ble vurdert til for stort omfang. Dette ville ikke bare ha krevd mye mer fokus på sikkerhet med brukere, det ville også vært behov for begrensninger og differensiering i brukerrettigheter. Dette ville skapt mye arbeid og det ville også vært mindre feilmargin for både forståligheten av systemet og ikke minst sikkerheten i form av mulighet for reell endring av data for enkeltbrukere.

2.2 Ressurser

Selskapet stiller ikke med noen fysiske ressurser da det ikke er noen faste kontorer i Bergen. Studentene vil bruke egne PC'er og gratis programvare til selve utviklingen. Det selskapet stiller opp med er git server og hosting av prosjektet.

Ressurser eksisterer også i form av mennesker og hjelp. Selskapet har godkjent brukertester med flere av de ansatte. I tillegg til dette får vi et design laget, dette vil spare mye tid på utforming og designvalg. En systemarkitekt vil også være åpen for spørsmål i løpet av prosjektet og det blir gjennomgang av kode og funksjonalitet mot slutten av prosjektløpet.

3 DESIGN AV PROSJEKTET

3.1 Forslag til løsning

I dette kapittelet ser vi på tre konkrete løsninger. Disse blir veid opp mot hverandre og det reflekteres på fordeler og ulemper med begge to. Utover dette ser vi på en løsning som krever minimal utvikling og ser på fordelene og ulempene dette medfører.

3.1.1 Alternativ løsning 1

Systemet skal lese, endre og slette data fra en postgres database. Disse operasjonene kan legges i en Spring applikasjon. Denne applikasjonen har metoder for å svare på HTTP forespørsler. Dette gjør at dataen kan hentes ut uten å kjøre logikken i nettleseren. For visningen av dataen kan det utvikles en Angular applikasjon, denne vil kun ha informasjon om hvordan den kan kalle Spring applikasjonen sine endepunkter. Dette fører til en avkobling som gjør at systemet er mer modulært og at ikke de forskjellige delene er direkte avhengig av hverandre.

3.1.2 Alternativ løsning 2

Operasjonene med datahåndtering og uthenting kan legges i en .Net applikasjon. Denne kan utføre de samme operasjonene som Spring applikasjonen. Deretter kan man lage en Blazor applikasjon som tar hånd om visningen av dataen. Dette gjør at all logikk i systemet kan bli kodet i samme språk. Dette kan minke tiden det tar å gå mellom programmerings tankesett, og dermed potensielt øke effektiviteten.

3.1.3 Alternativ løsning 3

Databasehåndtering er en problemstilling løst av mange aktører. PGAdmin er den mest

populære open source administrasjon og utviklings platformen for postgres (pgadmin.org, 2024), vi har derfor valgt å utforske denne. PGAdmin gir direkte tilgang til databasen, men med et grafisk brukergrensesnitt. Dette brukergrensesnittet er brukervennlig for en erfaren utvikler. Det kan bli overveldende for en uten innsikt i hvordan relasjonsdatabaser er satt opp, blir interagert med eller har interagert med en database tidligere. Dette vil kreve en del opplæring, men vil kunne bli en god løsning for endring av felter når denne fasen er over. Det som fortsatt må brukes fra det eksisterende systemet er brukeropprettelsen. Denne prosessen involverer mer enn kun nye "entries" i databasen. Dette gjør at denne løsningen deler opp interagering med kunde brukerene til to forskjellige applikasjoner og kan øke kompleksiteten og opplæringstid.

3.1.4 Diskusjon av alternativene

Begge utviklingsløsningene er fornuftige alternativer, da begge baserer seg på en utviklingsmetodikk der man skiller datahåndteringslaget fra visningslaget (Oluwatosin, 2014). Dette gjør at man kan utvikle dem separat, og dermed ikke skaper for mange avhengigheter. Det som kan være fordelen med .Net er at all logikken kan skrives i C#, dette kan som nevnt hjelpe på flyten i programmeringen. Det som derimot er fordelen med Spring og Angular er at hele gruppen har kjennskap til, og har brukt disse teknologiene tidligere i store og små prosjekter. Selskapets eksisterende løsninger er bygget i Spring og Angular noe som gjør at det kan hentes inspirasjon fra eksisterende kodebase. Med dette er også selskapet kjent i disse rammeverkene og tilhørende språk, så det er lavere terskel for innspill og veiledning om noe skulle mangle, enten i funksjonalitet eller i deployment delen av prosjektet.

Den utviklingsfrie løsningen vil gjøre utvikling overflødig, men vil skape mer kompleksitet i systemet. Kompleksiteten vil øke da ansvaret for kundebrukerene blir fordelt over flere applikasjoner. Databaseadministrasjon er en komplisert oppgave med flere risikoer, og da spesielt datatap. Nye brukere kan fort slette eller endre data de ikke skulle røre. PGAdmin løser dette ved brukerprivilegier, dette gjør at det ikke er mulig for brukerne å utføre

handlinger på databasen som de ikke har fått tilgang til. PGAdmin kan brukes gjennom en server eller via et lokalt nedlastet program. Dette gjør at den utviklingsfrie løsningen vil kreve enten at selskapet deployer en PGAdmin instans, eller at alle som trenger tilgang må laste ned programvaren.

3.2 Valgt løsning

Den valgte løsningen er Spring backend og Angular frontend. Dette både etter ønske av selskapet, og basert på kunnskapen som gruppen innehar. Dette gjør at det blir en slakere læringskurve da ikke alt må læres helt fra bunnen av. Dette gir mer tid til læringen av de mer kompliserte aspektene i prosjektet. Selskapet har også videre tanker om hva denne applikasjonen kan bli, noe som gjør at de ønsker et spesifikt verktøy utviklet heller enn å legge til flere eksterne løsninger som ikke gjør nøyaktig det som trengs.

3.2.1 Teknikker og teknologier

Delkapittelet tar for seg valgene innenfor arkitekturen i prosjektet i form av etablerte teknologier og standarder innefor utviklingsfaget. Med det hovedsaklig valg av metodikker eller tankeganger for å holde prosjektet skalerbart, lesbart og effektivt.

REST API

REST API eller RESTful API er en måte å designe API'er som utviklet seg tidlig på 2000 tallet. Denne tankegangen rundt API design er basert på noen grunnregler. De mest relevante av disse er ingen tilstand, ingen forskjell på kall, og klient server avkobling. Disse tre tingene baserer seg på at uansett hvor kallene kommer fra og hvem som kaller API'en så skal den alltid få samme informasjon og returnere samme informasjon. Ingen tilstand betyr at serveren ikke skal gi forskjellig svar avhengig av hva som har skjedd tidligere. Ingen forskjell på kall betyr at uansett hvem som kaller API'et så skal det gjøres på samme måte, i vårt tilfelle med HTTP og JSON. Klient server avkobling vil si at serveren på ingen

måte skal være avhengig av klienten, det eneste klienten skal vite om serveren er URI'en til den etterspurte dataen og det eneste serveren skal vite om klienten er hva den har spurt etter og sende dataen. (IBM, 2024)

Testing av kode

For backenden i kodebasen så vil testing av kode være av høy prioritet. Utviklingen vil basere seg mye på enhetstester og være såkalt test drevet, definert i kap 3.4.1. Dette gjør at man fort kan vite om koden man har skrevet kjører og deretter vurdere revisjon om noe mangler.

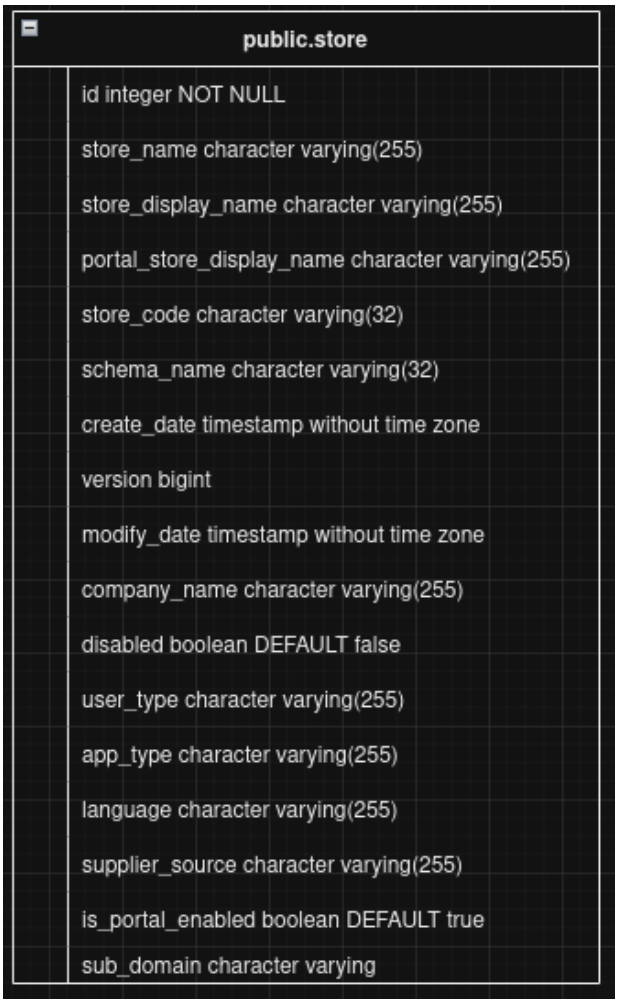
CRUD

CRUD er en forkortelse som står for create, read, update og delete. Dette er grunnmuren i hva applikasjonen skal gjøre. CRUD operasjonene vil i applikasjonen bli gjennomført ved at brukeren gjør en handling i frontenden, denne sender et API'kall til den RESTfulle API'en i Spring som gjør selve CRUD operasjonen på databasen. Det som skiller applikasjonen fra en ren CRUD applikasjon er koblingen til byggeverktøyet Jenkins, dette gjør applikasjonen mer kompleks og mer skreddersydd til selskapet sine behov.

Relasjonsdatabaser

En relasjonsdatabase er en måte å lagre data på basert på tabeller. Disse tabellene har noen krav. Hver rad i tabellen må være unik identifiserbar, dette vil si ingen like rader. Dette kan oppnås med kombinasjoner av felt som sammen blir unike eller et unikt felt f.eks et løpenummer. Rekkefølgen på rader og attributter er ikke signifikant, altså trenger det ikke være noen satt rekkefølge på hverken attributtene eller radene. Det siste ved relasjonsdatabaser er at attributtene skal være atomære. Det vil si at felter i tabellen skal ikke inneholde lister med data eller flere verdier for samme rad (Eidsmo, 2001). Et eksempel på en tabell ses i figur 3.1, hvilket er databasetabellen som blir brukt i prosjektet. Her kan

man se attributter som er definert med ulike datatyper og spesifikke egenskaper som NOT NULL og DEFAULT. Disse egenskapene betyr henholdsvis at et felt må ha en verdi og hva verdien til et felt skal være om den ikke blir tilordnet noe spesifikt. For selskapet er dette en god form for lagring av data da en kun behøver en rad per bruker. Det er også en billig og utbredt form for lagring som gjør at sikkerhet og implementering er godt dokumentert.



Figur 3.1: Databasetabell

3.3 Valg av verktøy

IDE

Til frontend utviklingen er Visual Studio Code valgt, dette er en tekstbehandler med flere utvidelser spesifikt mot det vi skal utvikle. Dette gjør at vi vil ha full støtte og mye dokumentasjon for språk- og rammeverk spesifikke problemstillinger. For backenden har vi valgt Eclipse. Dette fordi Eclipse er en komplett IDE med de verktøyene som trengs for å kunne utvikle en god løsning, hvilket inkluderer enkel testing av kode samt en god brukeropplevelse. I tillegg til disse punktene er begge studentene komfortable og kjent med verktøyene som gjør at oppstarten ikke blir hindret av ukjente miljøer.

Bitbucket og Git

Bitbucket er en tjeneste som tilgjengeliggjør git sin funksjonalitet. Git er et revisjonskontroll system som gjør at flere utviklere kan jobbe på samme prosjekt. Dette gjøres ved at den skaper versjoner av koden og tilbyr muligheter for sammenslåing av kode fra forskjellige versjoner. Det er normalt å lage egen versjon for fiksing av feil i systemet eller for implementering av ny funksjonalitet. Disse versjonene kalles for grener og ses i figur 3.2, her forgrener koden seg og hver node representerer en versjon. Sammenslåinger av grener, som sees på de to nederste forgreningene i figuren, foregår som regel via PR'er. Disse er en metode for å oversiktlig se hva noen ønsker å legge til i kodebasen. De viser alle forskjellene fra den nåværende koden og gir muligheter for tilbakemeldinger og kommentarer før en sammenslåing blir godkjent. (Spinellis, 2012)

Figur 3.2: Visualisering av versjoner i git

3.4 Prosjektmetodikk

Gruppen har valgt å gjennomføre prosjektet gjennom å benytte oss av de smidige verdiene (Martin, 2012). De smidige verdiene inkluderer:

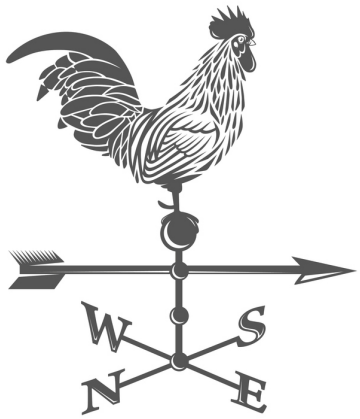
- Prioritere godt samarbeid og interaksjoner over prosesser og verktøy
- Fungerende software over omfattende dokumentasjon
- Fokusere på samarbeid med kunden istedenfor å henge seg opp i formaliteter
- Respondere på endringer fremfor å følge planen.

Dette prosjektet gjennomføres ved bruk av smidig utvikling for å øke sannsynligheten for å bygge riktig produkt. Dette gjennom hyppige tilbakemeldinger på implementert funksjonalitet og samtidig respondere på ønskede endringer fra selskapet. (Martin, 2012)

Figur 3.3 illustrerer denne tilnærmingen ved å vise en værhane på et kirkespir, som selv om den er laget av jern, raskt vil bli ødelagt av stormvinden hvis den ikke forstår den edle kunsten å snu seg mot enhver vind. Dette sitatet av Heinrich Heine understreker viktigheten av å være fleksibel og tilpasningsdyktig. Ved å justere oss i henhold til skiftende krav og forhold kan vi bedre møte kundens behov og skape et produkt av høy kvalitet (Martin, 2012).

3.4.1 Utviklingsmetodikk

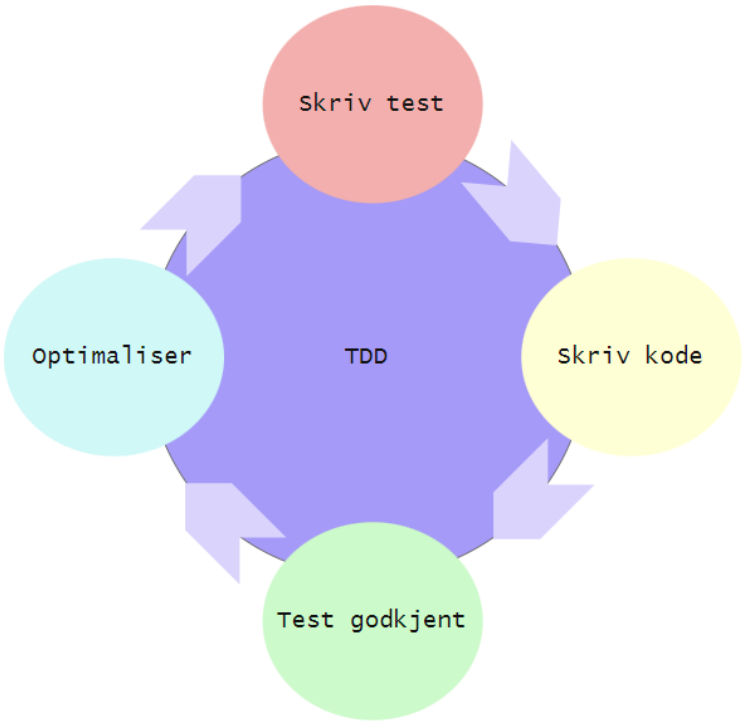
Utviklingsmetodikken som har blitt tatt utgangspunkt i heter ekstrem programmering. Ekstrem programmering er en av de mest utbredte smidige utviklingsmetodikkene. Metodikken angriper utviklingen gjennom effektivitet og enkelhet med korte utviklingsiterasjoner. Metodikken bygger på fem veiledende verdier, fem regler og tolv praksiser for koding. (Raeburn, 2024)



Figur 3.3:
"The weather-cock on the church spire, though made of iron, would soon be broken by the storm-wind if it did not understand the noble art of turning to every wind."
-Heinrich Heine

En av de tolv praksisene vi har fokusert på er test drevet utvikling. Test drevet utvikling er en prosess der en først skriver tester til ønsket funksjonalitet, for å videre skrive kode for å bestå testen. Etter at testen er godkjent kan en optimalisere koden. Dette blir gjentatt frem til all ønsket funksjonalitet er implementert. Prosessen kan du se visualisert i figur 3.4. Figuren beskriver at når en test er godkjent skal koden optimaliseres. Men viktig å huske på at optimaliseringen er et ønske men ikke et krav og skal ikke prioriteres over introduksjon av ny funksjonalitet (Martin, 2012).

For å planlegge prosjektet var det nødvendig å få klarhet i selskapets krav til applikasjonen. Disse kravene ble formidlet gjennom brukerhistorier, som gir en god oversikt over funksjonaliteten uten å gå for dypt inn i detaljer. Selv om disse brukerhistoriene inneholder de overordnede kravene, forventes det mindre justeringer i løpet av prosjektgjennomføringen etter hvert som selskapet får mulighet til å interagere med løsningen. (Martin, 2012)



Figur 3.4: Test drevet utvikling

Når en gjennomfører ekstrem programmering er det ekstra viktig å høre godt etter i samtale med kunden. Dette vil føre til at en forstår behovene til kunden. Med en helhetlig forståelse av behovene kan man velge arkitektur og implementering som ikke krever store endringer i løpet av utviklingsprosessen. Dette vil videre være en pådriver mot å oppnå ryddighet i kodebasen. Dette hjelper oss mot målet om at riktig produkt blir bygget ikke bare at produktet er bygget riktig.

Begge gruppemedlemmene jobber deltid i selskapet og innehar en god forståelse av systemet. Dette gjør at oppgaven kunne i stor grad blitt utviklet uten kontakt med kunden. Dette gjør at det er ekstra viktig å vurdere tilbakemeldingene fra selskapet for å være sikre på at produktet utfyller kundens behov, og ikke vår tolkning av problemet.

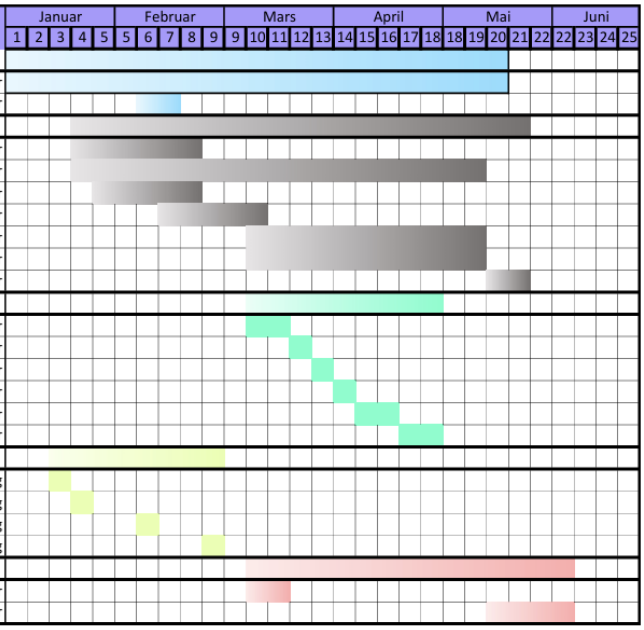
I ekstrem programmering er det ofte benyttet planlegging og prioritering i møtene (Martin,

2012). I denne konteksten vil nok dette være mindre relevant enn i større prosjekter da løsningen ikke vil bli tatt i bruk før endelig produkt er ferdig. Funksjonene i systemet, beskrevet i kravdokumentet (vedlegg 3), vil ha høyest prioritet da de fleste funksjonene ikke eksisterer i dagens system. Ved manglende tid mot slutten av utviklingsfasen vil de ikke-funksjonelle kravene bli nedprioritert.

En ofte brukt metode innen ekstrem programmering er parprogrammering. Denne metoden vil bli benyttet i mindre grad i dette prosjektet da utviklingsteamet er bestående av to personer. Parprogrammering er en effektiv metode med fordeler deriblant "forbedret programvarekvalitet, kortere utviklingstid, forbedret arbeidsmoral og motivasjon, økt tillit og forbedret samspill, effektiv kunnskapsdeling og forbedret læring" (Dahle, 2007). Med ressursene vi har tilgjengelig har vi lagt mer vekt på code review. Code review er en prosess der en annen en forfatteren går igjennom koden som er produsert og gir en vurdering av koden. Dette gir flere av de samme fordelene som parprogrammering men kan gjennomføres mer kostnadseffektivt med tanke på tidsbruk. Dette vil hjelpe med å få ferdigstilt løsningen før deadlinen. Mer informasjon vil komme om code review i kapittel 5.1.2. (Bacchelli og Bird, 2013)

3.4.2 Prosjektplan

ID	Fullført	Oppgave navn	Start	Varighet
		Forarbeid		
1	✓	Kompetanseheving	01.01.2024	135 dager
2	✓	Analyse av tidligere bachelorarbeid	05.02.2024	10 dager
		Dokumentasjon		
3	✓	Visjonsdokument	20.01.2024	35 dager
4	✓	Prosjekthåndbok	20.01.2024	114 dager
5	✓	Kravspesifikasjon	27.01.2024	28 dager
6	✓	Midtveisrapport	10.02.2024	29 dager
7	✓	Sluttrapport	03.03.2024	71 dager
8	✓	Evaluering	03.03.2024	71 dager
9		Expo plakat	13.05.2024	18 dager
		Utvikling		
10	✓	Iterasjon 1	04.03.2024	13 Dager
11	✓	Iterasjon 2	18.03.2024	8 Dager
12	✓	Iterasjon 3	25.03.2024	8 dager
13	✓	Iterasjon 4	01.04.2024	8 dager
14	✓	Iterasjon 5	08.04.2024	14 dager
15	✓	Iterasjon 6	22.04.2024	14 dager
		Møter		
16	✓	Oppstartsmøte	17.01.2024	1 dag
17	✓	Statusmøte 1	22.01.2024	1 dag
18	✓	Statusmøte 2	08.02.2024	1 dag
19	✓	Statusmøte 3	26.02.2024	1 dag
		Presentasjon		
20	✓	Midtveispresentasjon	06.03.2024	8 dager
21		Sluttpresentasjon	13.05.2024	22 dager



Figur 3.5: Gantt-diagram

Prosjektplanen er presentert av figur 3.5 i form av et Gantt-diagram. Gantt-diagrammet gir en god oversikt over planlagt tidsbruk på ulike deler av prosjektet. Diagrammet er delt inn i 5 forskjellige underkategorier og blir målt på varighet i uker og dager og når oppgaven ble startet. De ulike delene er forarbeid, dokumentasjon, utvikling, møter og presentasjon. Prosjektplanen blir fulgt så godt det lar seg gjøre og oppdateres med jevne mellomrom. Totalt i prosjektet er det medberegnet 690 timer for fullføring av alle oppgaver frem til innlevering av sluttrapporten. Dette timebudsjettet er basert på at en gjennomsnittsuke vil være mellom 20 og 30 timer.

3.4.3 Risikovurdering

	Hendelse / Risiko	Årsak	Sannsynlighet	Konsekvens	Risiko-produkt	Tiltak
1	Sykdom		3	3	9	Godt kosthold, jevnlig fysisk aktivitet
2	Problemer med samarbeid		2	4	8	Snakke om ting heller enn å sitte inne med dem
3	For treg læring	Mye nytt stoff	3	5	15	Begynne tidlig
4	Bygger feil produkt	Dårlig kommunikasjon med selskapet	3	4	12	Flere brukertester og smidig utviklingsmetodikk
5	Dårlig tidsstyring	Dårlig planlegging	3	4	12	Planlegge ofte og nøye
6	Uatorisert tilgang	Dårlig sikkerhet	2	4	8	Fokus på sikkerhet
7	Tredjeparts programvare		2	4	8	Fokus på tidlig testing av api

Tabell 3.1: Risikotabell

"Risiko er sannsynligheten for at en spesifikk usikker hendelse inntreffer og medfølgende konsekvenser. Enhver faktor som påvirker prestasjonen til ett prosjekt kan være en kilde til risiko" (Abreu, 2022). Ved å være bevisste på risikoene som er involvert kan en lettere etablere tiltak. Tiltakene vil bidra med å minimere sannsynligheten for uønskede hendelser og vil hjelpe mot å overlevere ønsket resultat. (Abreu, 2022)

Med bakgrunn i dette valgte vi å gjennomføre en risikoanalyse. Risikoanalysen vår baserer seg på en mal vi ble tildelt i emnet "DAT191 Bacheloroppgave". Tabellen 3.1 viser vår fullstendige risikoliste basert på malen der sannsynlighet og konsekvens blir tildelt et tall mellom 1 og 5 og det beregnes et risikoprodukt med maksimumsverdi 25.

Samarbeid

Manglende informasjonsflyt og mangel på interne møter kan skape dårlig samarbeid innad i bachelorgruppen. Dårlig samarbeid kan videre gjøre at oppgaver ikke fullføres innen gjeldende frister, og prosjektet blir i værste fall ikke gjennomført innen gjeldende tidsramme. For å avverge dette vil arbeid fordeles jevnt mellom partene, og det legges til grunn en lav terskel for å ta opp problemstillinger og gi tilbakemeldinger.

Tidstyring

Risikoer relatert til tidstyring vil kunne kontrolleres til en viss grad gjennom fokus og å sette forventninger til hverandre og selskapet. Forventningene til hverandre ble ofte gitt i form av interne frister. Da det alltid kan dukke opp uforutsette hendelser vil vi aldri kunne få sannsynligheten for dårlig tidstyring helt vekk.

Planlegging

Dårlig planlegging er forsøkt avverget gjennom kontinuerlige møter både med intern og ekstern veileder. Disse kontinuerlige møtene er også med på å senke sannsynligheten for at vi bygger feil produkt.

Gjennom teknologivalgene våre, som begrunnet i kapittel 3.1 sikres det at selskapet har god kunnskap innenfor valgte teknologier og det er teknologier vi har erfaring med fra tidligere arbeid både via skole og jobb.

Sikkerhet

Dårlig sikkerhet er teamet sin største bekymring da dette er gruppen sitt første prosjektet som skal utleveres direkte til sluttkunde. Gjennom utdanningen på HVL har vi gjennomført flere prosjekter som har gitt oss praktisk erfaring. Selskapet har stort fokus datainnsamling, og de blir derfor viktig at håndteringen av disse kundebrukerne er sikker og feiltolerant slik at data ikke blir tapt.

Bygger feil produkt

Risikoen for å bygge feil produkt vil i tillegg kunne være grunnet at prosjektteamet overvurderer deres kunnskap til dagens system og nedprioriterer tid brukt i møter da en allerede innehar mye info om systemet. Her er det viktig å huske på at selskapet har visjoner/endringer som planlegges innenfor dagens system som vi også må ta høyde for.

3.5 Evalueringsplan

Formålet med evalueringen er å vurdere applikasjonen sin brukervennlighet og kodekvalitet. Derunder opprettelse, endring, sletting og visning av brukere.

Evaluering av prosjektet er viktig for gjennomføringen og personene som gjennomfører. Fordelene av evalueringen for gruppen er effektivisering av arbeidsprosessen og justering av korrekt utfall. Fordelene med evaluering for den enkelte i prosjektgruppen er gode erfaringer som en kan ta med seg videre og vil være verdifull i andre prosjekter. Evalueringsprosessen skal gi rom for læring og refleksjon samtidig som en holder kostnadene nede. Hovedmålet vil alltid være å lage best mulig løsning til gitt problemstilling. (Clark, 2021)

Evalueringsplanen til prosjektet deles inn i 2 faser, underveisevaluering og sluttevaluering. I underveisevalueringen vil fokuset være for resten av prosjektgjennomføringen og sluttevalueringen vil gi mer rom for drøfting og refleksjon med høy kompetanseheving for de involverte.

Underveisevalueringen skal gjennomføres ukentlig og er todelt. Første del vil fokusere på det tekniske ved koden, og i andre del vil fokusere på gjennomføring av prosjektet.

Den tekniske evalueringen skal gjennomføres av prosjektgruppen og blir gjennomført individuelt gjennom Unit testing og gjennom metoden code review, som nevnt i kapittel 3.4.1. Code review gjennomføres når en har fullført kodingen av ny funksjonalitet. Reviewen gir en bedre mulighet til å oppdage og gjennomføre konkrete forbedringer, og identifisere mu-

lige “bugs” og tilfeller som systemet ikke svarer på. Planen er å gjennomføre code review ukentlig, og vil være mest relevant i sammenheng med å legge ny kode til hovedgrenen. Vi benytter oss av verktøyet bitbucket for å se over endringer hverandre har gjort dette i form av PR’er.

Andre del av underveisevaluering skal være vurdering av produktet med selskapet og prosessene internt i prosjektgruppen. Som en del av underveisevalueringen vil det i uke 16 bli gjennomført en brukertest der brukerne tester og gir tilbakemelding på første versjon av løsningen. Mer informasjon om brukertesten vil bli gitt i kapittel 5.1.1

Sluttevalueringen av prosjektet vil være en helhetlig vurdering av prosessen og produktet. I den teknologiske vurderingen vil det fokuseres på funksjonalitet og kodekvaliteten til løsningen, denne vil bli gjennomført med en systemarkitekt fra selskapet. Produktets forbedring med hensyn til brukermiljøet blir testet ved sammenligning av hvor mye raskere endringene blir gjennomført. Videre om kundeopprettelse ble effektivisert og om tiden for overlevering av bruker til ny kunde ble drastisk forkortet. Vurdering av prosessen vil bli gjennomført av prosjektgruppen der vi vil vurdere om budsjettet ble holdt og om prosjektet ble gjennomført i henhold til planen presentert i prosjekthåndboken (vedlegg 2). Det vil i uke 18 bli planlagt produkttesting der brukerne tester og gir tilbakemelding på siste versjon av løsningen og om de initielle krav til løsningen er møtt. For å foreta en slik evaluering vil vi knytte løsningen vår til en testdatabase. Med ønske om å vurdere løsningens brukervennlighet vil det bli gitt begrenset med opplæring til brukerne før testing.

4 DETALJERT LØSNING

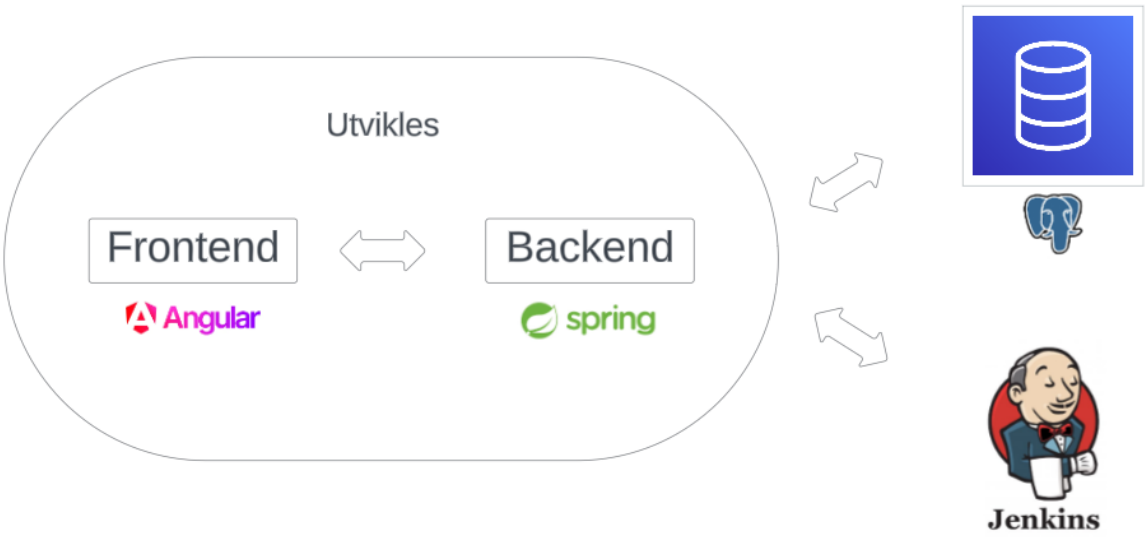
Kapittel fire inneholder en total oversikt over hvilke teknologier, designmønstre og teknikker som har blitt brukt for utviklingen av applikasjonen.

4.1 Systemdesign

Designet til et system påvirker hvorvidt det er skalerbart, brukervennlig, effektivt og mulig å utvikle videre.

4.1.1 Overordnet arkitektur

Arkitekturen som ble utarbeidet i planleggingsfasen ble brukt gjennom prosjektet og forble uendret. Arkitekturen er basert på en klient-server modell. Dette vil si at brukeren sitter på klient programmet, dette programmet vil sende en forespørsel over internett til serveren og serveren vil utføre forespørselen. I dette tilfellet vil Angular frontenden være klienten og Spring backenden være serveren. Veiene for kommunikasjon er illustrert i figur 4.1. Kommunikasjonsleddene går i begge retninger da det sendes forespørsler og data begge veier i alle ledd. Fra backenden etableres det kommunikasjon til de "eksterne", det vil si de modulene som ikke skal utvikles. Denne kommunikasjonen går gjennom API kall til Jenkins og direkte kobling til postgres serveren. Dette vil være separate moduler innad i backenden slik at det enkelt kan kobles av eller på flere moduler senere uten store endringer i kodebasen.



Figur 4.1: Overordnet systemflyt

4.1.2 Teknologier og Systemer

Dette kapittelet vil gi dypere innsikt i teknologiene og systemene som er brukt i applikasjonen.

Angular

Angular er et komponent-basert Typescript rammeverk til å bygge skalerbare applikasjoner. Det at Angular er komponentbasert vil si at istedenfor å ha en HTML fil som er koblet til en URL, så lager man komponenter som kan injiseres flere steder. Dette gjør at komponentene man lager kan gjenbrukes og gjør skaleringsprosessen mer effektiv. I tillegg til dette inneholder Angular mange biblioteker med kode som gjør operasjoner som HTTP forespørsler enklere å implementere, men også sikrere og raskere å jobbe med (angular.io, 2024).

Spring

Spring er det mest populære rammeverket for Java (spring.io, 2024). Rammeverket inneholder en mengde med under-moduler som tar seg av forskjellige områder av utviklingen. En av hovedmodulene til rammeverket heter Spring Boot. Spring Boot gjør at spring applikasjoner "bare kjører". Dette gjør den med innebygde servere og autokonfigurasjon av av avhengigheter. Dette gjør at fokuset vårt ved utvikling av applikasjonen kan ligge i å utvikle og lage kode heller enn å bruke timesvis på feilsøking grunnet manglende konfigurasjoner, kjøring av web servere og oppsett av endepunkter. (spring.io, 2024)

Jenkins

Jenkins er et selvstendig, open source, integrasjon og automasjonsplatform (jenkins.io, 2024). Selvstendig betyr her at det kan kjøre uten noen andre verktøy eller applikasjoner. Jenkins kan kjøre på enhver maskin som har JRE installert. Bruksområdet til selskapet for Jenkins er automatisering av deployment og scripts som lager nye database entries.

Postgres

Postgres er en open source relasjonsdatabase. Den har vært i utvikling siden 1986 så den har mye funksjonalitet og er meget stabil (Postgres, 2024). Selskapet bruker og har brukt Postgres siden start i 2018. Dette er en velfungerende database for bruksområdet fordi dataen er simpel og passer godt inn i tabellformatet. Dette gjør at man kan utnytte seg av postgres sin optimalisering og innebygde søking noe som gjør systemet raskt og effektivt.

4.2 Flyt internt i systemet

Dette kapittelet handler om hva som skjer internt i hver del av prosjektet, hvordan ansvaret er fordelt og hvordan de forskjellige applikasjonene kommuniserer med hverandre.

Frontend

Angular var frontend rammeverket som ble brukt i dette prosjektet, altså klienten. Dette er bindeleddet til brukeren av systemet. Dette vil si at den skal vise fram dataen på en fornuftig måte samtidig som brukeren skal kunne sende data gjennom den og til backenden. Som nevnt i forrige kapittel er det fordeler med Angular over klassisk HTML og Javascript. Når man lager restfulle API'er som det er gjort i applikasjonen så er det standard kommunikasjonsformatet JSON. JSON objektene som frontenden mottar, for eksempel ved kall til "api/get-by-storename" blir castet til typene konstruert i prosjektet. I dette tilfellet parserer frontenden JSON objektet, og caster det til en intern type. I figur 4.2 vises det hvordan man kan sende en forsespørsel og tilegne svaret man får direkte til typer. I Angular prosjektet er det to hovedtyper som omhandler data, typene er navngitt "Build" og "Tenant". Build typen er den som sendes til backenden for å bygge en ny bruker, men den blir aldri hentet da den ikke er en komplett bruker (se Kap. 4.2.1 Jenkins). Tenant er typen som sendes til backenden ved endringer, men også den typen som hentes for visning av brukere i systemet.

```
getAllTenants(companyName: string): Observable<string[]> {
  return this.http.get<string[]>
  (this.requestUrl + '/api/users/get-all-tenantsInCompany?com=' + companyName)
};

getTenant(store_name: string): Observable<Tenant[]> {
  return this.http.get<Tenant[]>
  (this.requestUrl + '/api/users/get-by-storename?storename=' + store_name)
};
```

Figur 4.2: HTTP forespørsel i Angular

Backend

Spring er der databehandlingen, valideringen og persisteringen blir gjennomført. Hele spring backenden er skrevet i Java. Backenden består av to hoveddeler. Disse to er "UserApi" og "JenkinsApi". UserApi er delen av applikasjonen som behandler all data omhandlende brukerene. Dette er da all uthenting av data fra databasen og endring i databasen. Fra endepunktet "/api/users" så kan man f. eks hente ut alle brukerene i systemet, dette som fullstendige objekter eller som en liste med string. Den andre delen, JenkinsApi, er der kommunikasjon med Jenkins foregår. Denne utfører bygging av nye brukere ved forespørseler til Jenkins (se Kap. 4.2.1 Jenkins). I kapittelet om Frontenden ble det etablert at Backenden sender data i JSON format, dette gjelder for alle endepunktene noe som gjør at man enkelt kan kommunisere med alle endepunktene og forvente konsistent respons. For kommunikasjon til Backenden så er det også JSON som er formatet. De endepunktene som skal motta data utover det som forespørres trenger å få en body, dette er for eksempel ved opprettelse av en ny bruker. Denne bodyen kommer som en Streng, men er egentlig et JSON objekt. Ett eksempel på bodyen til et kall fra frontenden ses i Figur 4.3. Denne konversjonen fra JSON string blir løst med en generisk funksjon og google sitt JSON bibliotek GSON. Denne konverterer strengen til objektet som brukes videre i validering og deretter sendes enten til Jenkins eller til databasen. Denne formen for Object-relational mapping gjør at objektene som blir sendt enkelt kan behandles og brukes videre i programmet. I backenden eksister typer som tilsvarer "Tenant" og "Build" i frontenden. Disse typene heter "User" og "JenkinsBuildUser".

```
{
    "id": 0,
    "store_name": "demoStore",
    "store_display_name": "Demo Store",
    "portal_store_display_name": "Demo Store",
    "store_code": "demo",
    "schema_name": "demo",
    "create_date": "2023-02-07T11:49:48.16449",
    "version": 0,
    "modify_date": "2023-02-07T11:49:48.16449",
    "company_name": "demos",
    "disabled": false,
    "user_type": null,
    "app_type": null,
    "language": "furniture",
    "supplier_source": "external",
    "is_portal_enabled": true,
    "sub_domain": "@demostore.no"
}
```


Figur 4.3: JSON med brukerdata

4.2.1 Jenkins

Som nevnt tidligere (se 4.1.2 Jenkins) så er en av selskapet sine bruksområder for Jenkins automatisering av scripts for opprettelser i databaser. Dette er den delen av Jenkins som er blitt integrert med i prosjektet. I Jenkins heter automatiseringer jobber. Av jobbene som selskapet har er det en vi bruker. Denne jobben heter "new tenant". For å gjøre denne jobben kreves en del parametere som sier hva den nye brukeren skal inneholde. Denne dataen blir sendt gjennom typene nevnt tidligere, Build og JenkinsBuildUser. For at denne

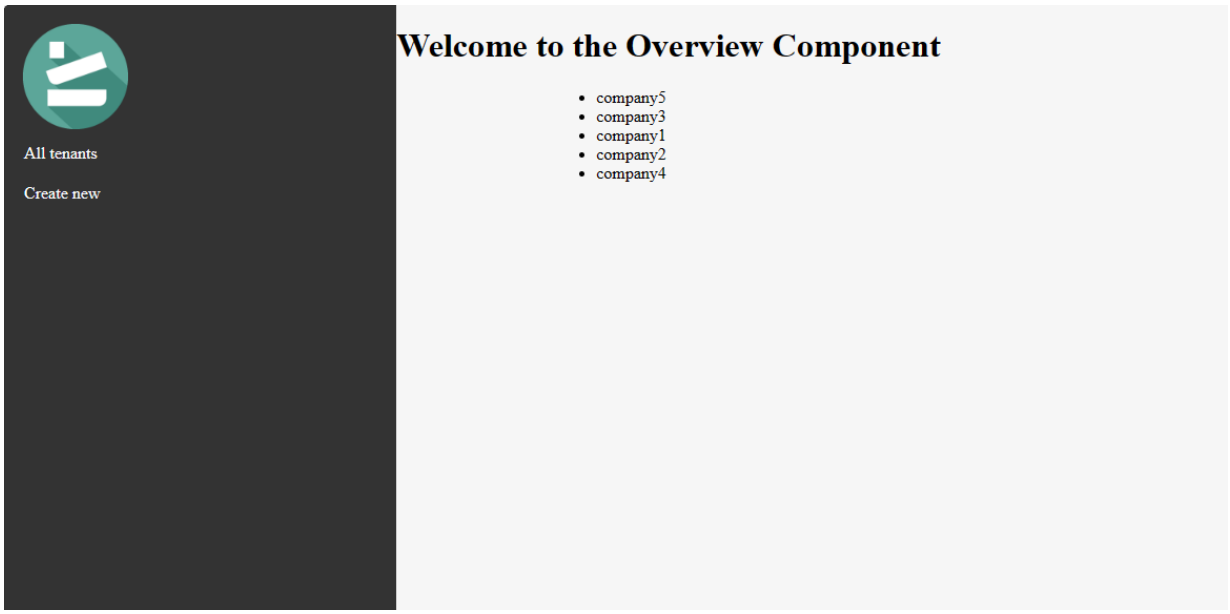
dataen skal bli oppfattet av Jenkins må den sendes i bodyen til forespørselen. I tillegg til dette må forespørselen inneholde autentisering samt en "Jenkins-Crumb". Denne smulen er ett ekstra lag med sikkerhet for Jenkins, smulen oppdateres regelmessig og må sendes med hver forespørsel. For å få tak i smulen må det sendes en forespørsel med brukernavn og passord. For å starte en jobb må det sendes en smule, brukernavn og en API-nøkkel. Denne nøkkelen er noe som må genereres inne på brukeren, og ikke kan hentes på noen ekstern måte.

4.3 Utviklingsiterasjonene

Vi gjennomførte en planleggingsprosess der vi delte implementeringsfasen inn i iterasjoner. Planen var å gjennomføre 6 iterasjoner med varighet på mellom 1-2 uker. Målet for hver iterasjon var å implementere en brukerhistorie. Brukerhistoriene ble gjennomført i rekkefølgen basert på det som kunne gjennomføres på kortest tid og det som var mest kritisk for selskapet. Denne prioriteringen er gjort i henhold til ekstreme programmering sine retningslinjer (Martin, 2012). Se kapittel 3.4 i visjonsdokumentet (vedlegg 1) for prioritering av funksjoner. Frontenden ble laget med kun fokus på funksjonalitet i iterasjon 1 og 2 siden vi mottok design da vi startet på iterasjon 3.

Iterasjon 1, Full oversikt over eksisterende brukere

Første del av iterasjon 1 ble benyttet til oppsett av utviklingsmiljøet. Gjenstående tid i første iterasjon besto av implementering av første brukerhistorie. Brukerhistorien med høyest prioritet var i utgangspunktet å kunne endre eksisterende bruker. Men her ble oversikt over eksisterende brukere valgt fremfor endring av eksisterende bruker. Dette begrunnes av mindre utviklingstid da vi har fokus på hyppige leveranser. Arbeidet med backenden startet med å utvikle endepunktet for å hente ut alle selskapene og brukerne til hvert selskap. Frontenden koblet seg på endepunktet og visuelt representerte dette i frontenden som en enkel liste, presentert i figur 4.4.



Figur 4.4: Bilde av oversikten over brukere før mottak av design

Iterasjon 2, Endre eksisterende bruker

Utviklingstiden for endring av eksisterende brukere er nå den med høyest prioritet. Endepunktet for endring av eksisterende bruker ble opprettet. Frontenden koblet seg på endepunktet og utviklet funksjonalitet for å motta endring fra bruker og videre sende dette til backenden. Funksjonaliteten ble implementert ved brukt av et enkelt skjema som figur 4.5 viser er mottakelig for inndata fra bruker.

Figur 4.5: Bilde av muligheten for å endre eksisterende brukerdetaljer dette gjennom et enkelt skjema

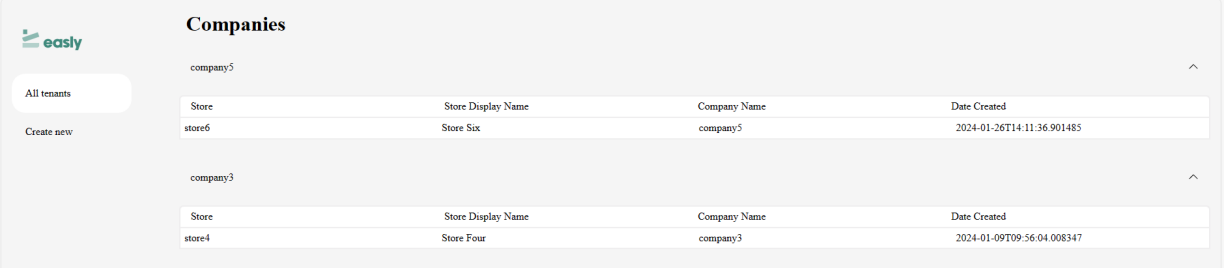
Iterasjon 3, Lage nye brukere

I denne iterasjonen var fokuset å implementere brukerhistorien for å lage nye brukere. Endepunktet for å lage nye brukere ble opprettet i backenden og arbeid med integrasjon mot Jenkins ble gjennomført. Frontenden koblet seg på endepunktet og implementerte funksjonaliteten, dette visualisert av figur 4.6. I tillegg til implementering av ny funksjonalitet ble også planleggingen mot å utbedre det visuelle i frontenden startet. Planleggingen ble gjennomført i sammenheng med mottagelse av design til løsningen i løpet av denne iterasjonen.

Figur 4.6: Bilde av muligheten for å lage en ny bruker ved å fylle ut brukerdetaljer gjennom ett enkelt skjema

Iterasjon 4

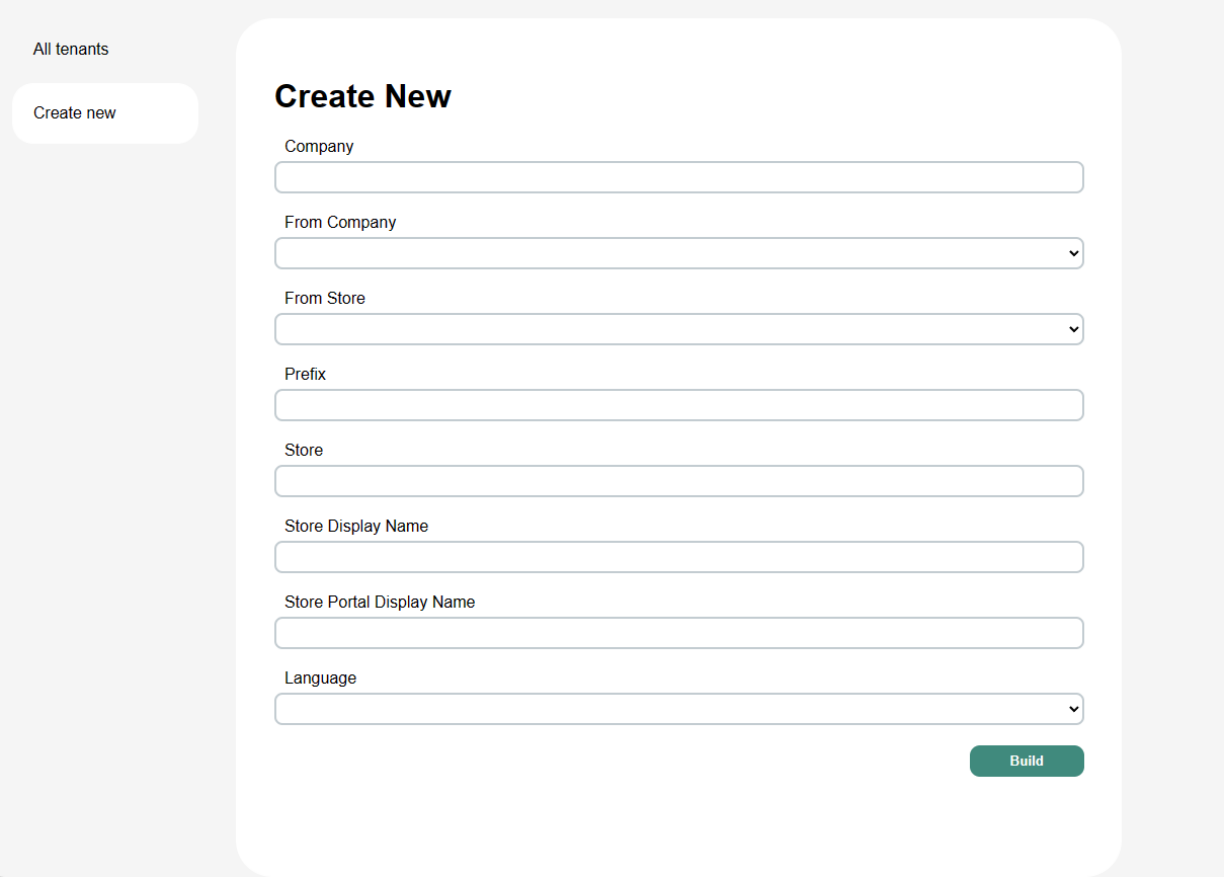
Iterasjon 4 besto av feilhåndtering og retting av feil i backenden. I frontenden ble forbedring av brukergrensesnittet for listen over alle selskapene ferdigstilt. Forbedringen er presentert av figur 4.7. Videre ble arbeidet med designimplementering for brukerhistoriene fra iterasjon 2 og 3 planlagt og påbegynt.



Figur 4.7: Oversikten over brukerene forbedret i henhold til design

iterasjon 5 Utbedring og brukertest

Iterasjon 5 som vist i prosjektplanen (Figur 3.5) er en lengre iterasjon enn de tidligere. Dette fordi iterasjonen består av utbedring av løsningen og gjennomføring av brukertest. Uke 1 av Iterasjonen ble innledet med å injisere testdata og grundig testing av applikasjonen som forberedelse til brukertesten. Utbedringen av løsningen besto av refaktorering av kode i backenden og forbedring av brukergrensesnittet i frontend. Forbedringen vises av figur 4.8 og fokuserte på opprettelse av ny bruker og endring av eksisterende bruker. Figuren viser forbedringer for opprettelse av ny bruker men forbedringen for endring av eksiterende bruker er gjennomført i tilsvarende stil. Uke 2 av iterasjonen besto av gjennomføring av brukertesten og planlegging av ønskede endringer fra brukertesten.



Figur 4.8: Brukergrensesnittet for opprettelse av bruker forbedret

Iterasjon 6

I iterasjon 6, som er siste iterasjon, ble de prioriterte endringene etter gjennomført brukertest fullført. I tillegg gjennomførte vi klargjøring av kodebasen for overlevering til selskapet. Klargjøring besto av gjennomgang av dokumentasjonen utbedring av kommentarer og mindre feilrettelser. I denne iterasjonen ble også den tekniske verifiseringen i samarbeid med selskapets systemarkitekt gjennomført, mer om denne i kapittel 5.1.2.

4.4 Tidstyver i utviklingsprosessen

Dette kapittelet vil inneholde noen av problemene gruppen støtte på under utviklingen. Disse problemene ble løst, men tok flere dager å feilsøke og teste.

4.4.1 Docker

Docker er et verktøy som ønsker å forhindre problemet hvor programvare kjører på et system, men ikke et annet. Dette gjør den ved å pakke inn programmene dine i "containers". Disse konteinerne er i praksis små egenstående operativsystemer med alt som trengs for å kjøre programvaren din. Dette gjør at en kan kjøre flere slike små operativsystemer heller enn å kjøre applikasjonene lokalt (Docker, 2024).

For å teste brukte vi Docker til å hoste databasen og å hoste Jenkins. Når alt kjørte og alle problemene var nøstet opp var dette en fungerende og effektiv løsning. Det som ikke var medberegnet var alle timene med feilsøking som lå før denne fungerende versjonen.

4.4.2 Jenkins

Jenkins er som tidligere etablert en platform for automatisering. I researchen ble det etablert at det fantes en API for å styre jobber eksternt, og det ble funnet et bibliotek som hadde all funksjonaliteten for dette bygget inn.

Jenkins REST

Jenkins REST er et bibliotek i Java som inneholder Jenkins API kallene som ferdiglagde funksjoner. Dette biblioteket eksisterer for å forenkle og muligjøre integrasjon med Jenkins via Java. Problemet med dette biblioteket var at det ikke var kompatibelt med Spring Boot 3.2 som var versjonen vi hadde valgt til prosjektet. Denne versjonen var valgt grunnet at den var den nyeste LTS versjonen. Dette førte til at all implementasjon måtte gjøres ved

fullstendige HTTP forespørsler, noe som tok en del lengre tid. Dette førte også til neste problem.

Jobs API

Jenkins jobs API er navnet på endepunktene man bruker for å eksternt koble til jobber i Jenkins. Automatisering er en av nøkkelordene til Jenkins. Ønsket for prosjektet var å automatisere automatiserings verktøyet. Dette er ikke et veldig vanlig bruksområde, noe som viste seg i researchen til denne modulen i prosjektet. Måten det viste seg var at det eksisterte veldig lite gode ressurser på nett om hvordan bruke denne API'en. Blant annet hvilke autentisering som trengtes i de forskjellige forespørslene, hva slags format body'en til forspørsler skulle komme på og generelt hva de forskjellige endepunktene gjorde. Dette tok lang tid å teste og førte til noen forsinkelser i leveranser for backenden.

5 RESULTAT

I dette kapitlet går vi inn på gjennomføringen av prosjektet. Kapittelet vil forklare hvordan evalueringen ble gjennomført, om den ble gjennomført etter planen og resultatet av den.

5.1 Evalueringsmetode

Evalueringsprosessen delte vi i to. Første prosess er validering, dette er med på å sikre at vi utvikler et produkt som svarer på problemstillingen. Andre prosess omhandler verifisering av løsningen og at vi utvikler produktet riktig.

5.1.1 Validering

Valideringen ble gjennomført gjennom Summativ og Formativ testing. "Summative testing gjennomføres etter produktet er ferdigutviklet, med mål om å validere at produktet møter kravene" (Barnum, 2010). "Formativ testing gjennomføres samtidig som produktet er i utvikling, med mål om å identifisere og fikse problem" (Barnum, 2010). Vi gjennomførte den formative testingen gjennom ukentlig code review gjennomført av prosjektgruppen kombinert med en brukertest i samarbeid med selskapet. Den summative testingen ble gjennomført i samarbeid med en systemarkitekt fra selskapet.

Brukertesting

Brukertesten ble gjennomført som en usability test. Usability test er "Prosessen om å lære om brukerene fra brukerene gjennom å observere dem bruke produktet til å oppnå spesifikke mål som har interesse for dem" (Barnum, 2010). På denne måten forenkler brukertestingen identifiseringen av grunnleggende problemer i løsningen og bidrar til den endelige utformingen. Dette i tillegg til å ikke bruke tid på å implementere funksjonalitet/utformingsforbedringer som er unødvendig.

Gjennomføring av brukertest

Vi sørget for å gjennomføre testingen når all funksjonalitet var på plass slik at når fokuset var på utforming og brukervennlighet kunne vi få ønsker direkte fra bruker.

Brukertestingen ble gjennomført som et nettmøte med ansatte fra selskapet, disse ansatte er fremtidige brukere av kundeadministrasjonsportalen. Målet med testen var å identifisere brukerflyt og til hvilken grad løsningen vår kommuniserte til testpersonen om eventuelle feil som hindret en fra å oppnå målet med oppgavene. Ansatte fra selskapet inntok rolle som testpersoner og vi inntok roller som henholdvis leder og medhjelper. Leder rollen hadde i oppgave å introdusere møte og informere om oppgavene. Medhjelper rollen hadde som hovedoppgave å observere og notere under gjennomføringen.

Møtet ble innledet med en introduksjon som informerte testpersonen om hva vi ønsket fra dem. Introduksjonen informerte testpersonen om å tenke høyt og å gjennomføre oppgavene etter beste evne i tillegg til hva målet med testen var. Under gjennomføringen var det viktig at vi holdt hovedfokus på å være observatører og ikke veiledere, for å få mest mulig informasjon fra testpersonen. Etter testpersonen hadde gjennomført alle oppgavene ble testpersonen bedt om å gi en helhetlig tilbakemelding med fokus på de mest kritiske forbedringene de ønsket.

Brukerne ble tildelt oppgaver å utføre ved å benytte seg av løsningen. Disse oppgavene inkluderte:

1. Du ønsker å finne alle aktive selskaper i systemet i dag. Hva gjør du for å finne utav dette?
2. Du ønsker å finne ut hvilke og hvor mange brukere som er knyttet til selskapet "Easlytest". Hvordan går du frem for å finne ut av dette?
3. Endre "Store name" til easlytest. Hvordan får du til ønsket endring?
4. Lag en ny bruker som skal hete "nyBruker1" som skal tilhøre selskapet admin.

Spørsmålene er viktige for å gi brukeren følelsen av å være i en situasjon der de skal bruke systemet. Spørsmålene er med på å skape et mønster i bruken av systemet og vil være gode for å gjenskape eventuelle problem. Alternativt å gjennomføre testen uten presise spørsmål vil bruken være veldig varierende mellom testpersonene. Dette i tillegg til at spørsmålene forsikrer oss om at all funksjonalitet ble testet. (Barnum, 2010)

Spørsmålene ble forsøkt formulert åpne og ikke ledende. I planleggingen og gjennomføringen av brukertesten benyttet vi oss av tipsene til Universitet i Oslo kombinert med retningslinjene til boken "Usability Testing Essentials : Ready, Set... Test!". (UIO, 2017; Barnum, 2010)

Summative testing

Summativ testing ble også gjennomført som en del av valideringsprosessen. Den ble gjennomført i samarbeid med en systemarkitekt fra selskapet. For gjennomføringen inviterte vi til et teams møte. I møte demonstrerte vi all implementert funksjonalitet og dette ble gjennomført for å validere at produktet møter de satte kravene.

5.1.2 Verifisering

Unit testing

I vår test drevne utvikling har vi skrevet unit tester. Unit tester er en prosess for å forsikre seg om at viktige komponenter av systemet møter kravene. Testene er en måte for å fange opp eventuelle feil i program-logikken. En bør etterstrebe å teste komponenten individuelt og isolert fra resten av systemet. Når en er sikker på komponenten møter kravene og funksjonaliteten blir gjennomført uten å oppleve feil kan en gå over til å teste komponenten i samarbeid med andre komponenter. Unit testing kan brukes til å identifisere feil og rom for forbedring. Ofte kan testene oppfatte at beregninger er korrekte og eventuelle prestasjons relatert problematikk gjennom gjentakende kall av komponent kan bli fanget. Et annet ord

som også er brukt om unit testing er komponent testing. (Watkins, 2001)

Det er flere fordeler med Unit Testing. En fordel er at de oppfattes supplerende til dokumentasjonen av koden. I tillegg til å supplere dokumentasjonen forenkler unit testing prosessen med å refaktorere kode. Refaktorering er en optimaliseringsprosess og skal gjennomføres etter en test er godkjent som nevnt i kapittel 3.4.1. "Refaktoring er prosessen å endre et software system på en slik måte at det ikke endrer ytre oppførsel av koden men forbedrer den indre strukturen" (Martin, 2012). Refaktoreringen gjennomføres gjennom gjentagende små endringer der unit testene vil bli kjørt etter hver endring. Hver lille endring i seg selv utgjør liten forskjell men flere små endringer resulterer i forbedret arkitektur og design av applikasjonen. (Martin, 2012)

```
@Test
public void testExistsByID() {
    insertTestUsers();
    assertTrue(userDB.existsById(2L));
    assertFalse(userDB.existsById(4L));
}
```

Figur 5.1: Unit test

Code review

Code review er en utbredt praksis innefor systemutvikling, det gjennomføres i både open source prosjekter og i industriell kontekst. "Code review gir fordeler som inkluderer kunnskapsoverføring, teambevissthet og opprettelse av alternative løsninger til problemet" (Bacchelli og Bird, 2013). Vi har valgt å gjennomføre manuell code review dette kan gi bedre oversikt over systemets helhetlige kvalitet. Unit testing sikrer som nevnt at funksjonaliteten ikke feiler under kjøring, men garanterer ikke god kodekvalitet. Derfor bør en kombinere bruken av code review og unit testing. (Carullo, 2020)

Resultatet av å ha gjennomført denne metoden vil også være å opprettholde en enhetlig kodestil over hele løsningen. En enhetlig kodestil betyr at når en leser gjennom koden, vil

personen ikke kunne skille mellom ulike bidragsytere. Dette gir inntrykk av at koden ble utviklet av en enkelt person, uavhengig av hvor mange som faktisk har bidratt til den.

Enhetlig kodestil er viktig for enkelt vedlikehold og lesbarhet av koden dette er også en av extreme programmering sine praksiser. "All kode i systemet skal se ut som den er skrevet av ett veldig kompetent individ" (Martin, 2012; Raeburn, 2024). Denne metoden er god å kombinere med Unit Testing, fordi en identifiserer mangler og feil som unit testing ikke gjenkjenner i tillegg til økt kodekvalitet. (GitLab, 2023)

Ulempen med code review er at det er tidkrevende noe som også er en av våre største risikoer. Gjennomføringen av denne metoden knyttes også opp mot ett av punktene i ekstrem programmering som omhandler kollektivt eierskap. Vi prioriterte gjennomføring av denne metoden da vi vurderte fordelene som større enn ulempene.

Teknisk verifisering

Den tekniske gjennomgangen av løsningen besto av ukentlig code review av hverandres kode og en vurdering fra selskapets systemarkitekt. Vurderingen ble gjennomført som et teams møte etter implementeringsfasen var avsluttet. Møtet ble introdusert med gjennomgang av diagrammer og forklaring av arkitekturvalg. Videre gikk det over til en code review der vi gikk igjennom backenden og frontenden til løsningen.

5.2 Evalueringsresultat

I dette delkapittelet presenteres og tolkes resultatene av gjennomført validerings og verifiseringprosess.

Resultat etter brukertest

Etter brukertestene var resultatet en kombinasjon av medhjelperens notater av observasjoner kombinert med den helhetlige tilbakemeldingen fra brukerene.

Resultatene gjorde at vi kunne skreddersy løsningen mot systemet og brukernes behov. Tilbakemeldingene fokuserte hovedsakelig på brukergrensesnittet med fokus på navigasjon og tilbakemelding. Testpersonene sine ønsker er listet under.

- Listen over selskapene må være i alfabetisk rekkefølge.
- Muligheten til å kunne se antall brukere tilknyttet ett selskap. Antallet brukere skal vise når ett selskap er valgt fra listen.
- Ønskelig med et søkefelt på toppen av listen for raskere navigering til et selskap
- Ønskelig med tilbakemelding når en oppretter ny bruker eller når en har forsøkt å endre verdiene til en bruker at det ble gjennomført med suksess.

En kan konkludere med at løsningen har behov for forbedring, spesielt knyttet til oversikt og navigasjon. Selv om brukerne var fornøyde med de implementerte funksjonene, ønsket de forbedringer i design og navigasjon.

Resultat av den tekniske verifiseringen

Etter den tekniske verifiseringen med selskapets systemarkitekt ble det utarbeidet noen anbefalinger. Det ble anbefalt å sørge for at modulene i backend og frontend har navn som samsvarer med navnene de har i databasen. Den primære tilbakemeldingen var å fokusere på riktig navngiving av klasser basert på databasen, dette vil gjøre løsningen enklere å vedlikeholde og administrere.

Å benytte JPA repostitory til database spørringer er fordelaktig for disse spørringene er databaseuavhengige. Vår løsning benytter seg av JPA repostitory men også noen "Native query" med postgres dialekt som kan fungere kun for postgresdatabaser. Selskapet har et ønske om høyest mulig grad av modularitet noe dette ikke oppfylte. Et eksempel kan ses i figur 5.2, her vil JPA generere spørringer som passer til den databasen den er koblet på basert på navnet til funksjonen.

```java
@Query(value = "SELECT distinct company_name FROM store", nativeQuery = true)
List<String> findAllCompanies();

List<String> findDistinctCompany_name();
```

Figur 5.2: Native query vs JPA generated query

I frontend var det overkompliserte API-kall og manglende bruk av environment variables som ble fokuset. Mangelen på dette fører til lavere grad av modularitet og fleksibilitet.

Evalueringen resulterte i klare retninger for forbedring, inkludert å justere bruken av rammeverk og optimalisere kode for bedre vedlikehold og ytelse. Det positive resultatet fra evalueringen var at systemet ikke inneholdt store feil og mangler.

5.3 Prosjektresultat

Prosjektet resulterte i en leveranse som oppfylte de fleste kravene som ble satt fra starten. De funksjonelle kravene som ble møtt inkluderte å gi en oversikt over alle brukere i systemet, muligheten til å endre brukere og opprettelse av nye brukere.

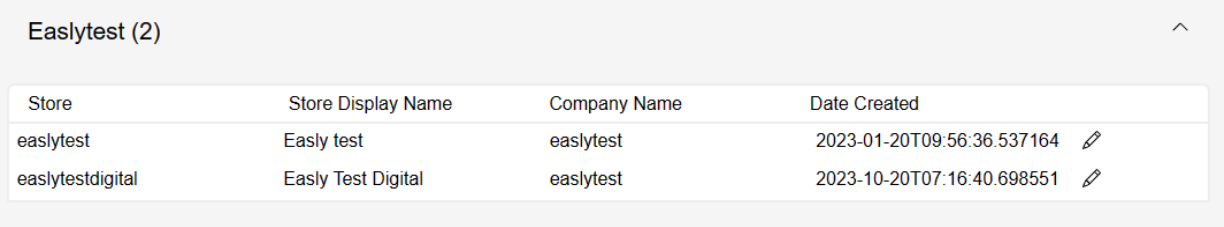
De ikke-funksjonelle kravene som løsningen oppnådde var ryddighet i kodebasen, dette forenkler vedlikehold av koden og fremtidig utvikling. Vi forbedret prosessen for å opprette nye systembrukere gjennom tilpasning av feltene som skal fylles ut. En av verdiene som skal fylles ut for å opprette en Easly bruker er å oppgi en allerede eksisterende bruker. Dette skal brukes for å kopiere filer som hver bruker må inneholde for å være funksjonelle. Tidligere har denne verdien måtte bli skrevet inn korrekt noe som gir rom for menneskelige feil. Løsningen vår her var at dette feltet er en liste med valg over alle eksisterende brukere slik at en ikke kan fylle denne verdien feil. Dette er noe som forbedrer brukeropplevelsen. I tillegg fokuserte vi på et godt og brukervennlig design, for å oppnå et mer intuitivt og tiltalende system for brukerene.

Prosjektet har oppnådd sitt overordnede mål om å tilgjengeliggjøre og effektivisere kunde-

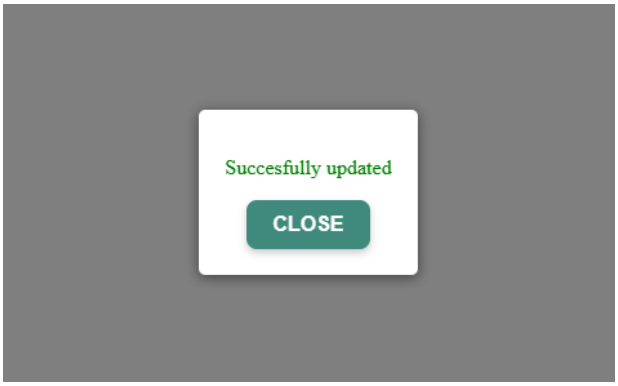
administrasjon. Merk at systemet ennå ikke er satt i produksjon, så selv om målet er nådd, vil det bare bli realisert fullt ut når systemet blir satt i drift. Når systemet er operativt, vil det styrke selskapets effektivitet og evne til å administrere kundedata.

Resultat etter brukertesten

I tillegg til de opprinnelige funksjonelle og ikke-funksjonelle kravene ble de fleste ønskede forbedringene fra brukertesten implementert. Listen over selskaper ble sortert i alfabetisk rekkefølge, noe som gjorde det enklere å finne spesifikke selskaper. Det er nå mulig å se antall brukere knyttet til hvert selskap, som vist i figur 5.3. I tillegg mottar brukerne tilbakemeldinger når de oppretter en ny bruker eller forsøker å endre verdier til en eksisterende bruker, som vist i figur 5.4.



Figur 5.3: Utdrag fra listen over alle brukere i systemet i dag. Tallet ved siden av selskapsnavnet beskriver antall brukere i selskapet



Figur 5.4: Tilbakemelding til kunde basert på API request response code

5.4 Prosjektgjennomføring

Dette prosjektet utføres som en del av utdanningen ved gjennomføringen av ingeniørstudie ved Høgskolen på vestlandet. Oppgaven ble som tidligere nevnt gitt i oppgave fra Easly AS. Gjennomføringen av dette prosjektet har en varighet på hele vårsemestert år 2023/2024 og ble innledet med et møte med intern og ekstern veilder. Der intern veileder representerer skolen og ekstern veileder er representant fra Easly AS.

Arbeidet med oppgaven er dokumentert i prosjekthåndboken (vedlegg 2). I dokumentet er progresjon og timeføring presentert i tabeller. Prosjektet hadde en påregnet total timebruk på 690 og vi endte opp med å bruke 700.

Videre fulgte vi prosjektplanen med iterasjonene beskrevet i Gantt-diagrammet. Vi arbeidet i korte sprintsykluser som tillot oss å regelmessig levere funksjonelle komponenter og få tilbakemeldinger fra kunden. Et avvik fra Gantt-diagrammet var at første iterasjon av utviklingsfasen ble forskjøvet med to uker. Dette er dokumentert i timelistene (vedlegg 2) og utdypet videre i kapittel 6.5 tidsstyring.

6 DISKUSJON

Diskusjonskapittelet tar for seg store deler av oppgaven og ser på hvordan de har påvirket prosjektet og resultatene.

6.1 Iterasjonsbasert jobbing

Å jobbe med iterasjoner påvirket resultatet i positiv grad. Sammenlignet med tidligere prosjekter uten iterasjonsbasert tilnærming, var det enklere å opprettholde motivasjonen. Jobbing mot delmål kontinuerlige leveranser var mer effektivt enn å fokusere på en stor endelig leveranse og langtidsmål.

Dette, i tillegg til at selskapet hadde mulighet til å gjøre endringer i andre brukerhistorier samtidig som vi jobbet med implementeringen av en, bidro til effektiviteten. (Martin, 2012)

6.2 Hvordan påvirket brukertestingen resultatet?

Å observere brukerne og hvordan de samhandlet gjorde at vi var i stand til å gjøre brukergrensesnittet mer brukervennlig og intuitivt. Videre flyttet brukertesten fokuset til kritiske områder som behøvde forbedring. Fokuset ble flyttet mot områder som har størst innvirkning på sluttresultatet. I sum var brukertestingen med på å skreddersy løsningen mot brukeren og ga oss informasjon om åpenbare mangler og tolkningen av systemet sett utenifra. Brukertesten påvirket med dette resultatet kun i positiv grad.

6.3 Teknologier og verktøy

I prosjektløpet ble det ikke møtt på problematikk knyttet til hverken programmeringsspråk, IDE'er eller rammeverk. Dette ble forhindret i stor grad fordi valgene av teknologi var basert på tidligere erfaringer og kunnskap. Dette var en styrke for prosjektet da tiden ble

mer effektivt brukt på utvikling og læring av konkrete problemstillinger for prosjektet, heller enn verktøyene som kun skulle brukes til å utføre utviklingen. Det som tok lengre tid enn ønsket, var det som ikke var utforsket i dybden før oppstart, her hovedsaklig Docker og Jenkins. Gitt at dette var vår største risiko, så ble det satt av mye tid til læring. Dette gjorde at vi kom i mål med disse, selv om det tok lengre tid enn forventet.

6.4 Mangler

Tiden i prosjektet strakk dessverre ikke til for alt som var ønsket fra selskapet. I dette delkapittelet ser vi på to deler av prosjektet som ikke kom helt i mål.

6.4.1 Sikkerhet

Sikkerhet er listet som en av de ikke-funksjonelle kravene og ble nedprioritert da tiden ikke strakk til. Dette er gjort i henhold til plan presentert i (3.4.1) prosjektmetodikk. Systemet bruker en standard for innlogging og sikkerhet, og denne ble ikke implementert i løsningen, hverken i form av brukere for ansatte eller kunder. Den siste delen var noe vi allerede hadde avgrenset i den initielle planen, men ønsket å ha en base for med tanke på videre utvikling.

6.4.2 Deploy

Prosjektet ble laget til det stadiet at det kjørte. Dette vil si at det fortsatt inneholdt kode som tilsa at det kjørte på localhost, og at man måtte kjøre alle prosjektene lokalt for å bruke hele programmet. Det som ble gjort her var å utarbeide Docker filer og metoder for å kjøre programmene lokalt, slik at det enkelt kunne endres til å kjøre på skyen eller en ekstern server.

6.5 Om det skulle blitt gjort igjen

Dersom prosjektet skulle bli gjennomført på nytt, er det situasjoner vi ville ha håndtert

annerledes. Avsnittene nedenfor beskriver noen av de områdene der vi ville gjort endringer.

Brukertesten

Ved ny gjennomføring av prosjektet ville vi etterstrebet å gjennomføre brukertestene fysisk. Den fysiske tilnærmingen gir oss bedre mulighet til å observere brukerens reaksjoner og samtidig få innsikt i eventuelle distraksjoner som testpersonen opplever underveis (Barnum, 2010). Vi mener også at testpersonen ville opplevd et mer avslappet miljø og en mindre formell setting.

Videre ville vi ønsket å gjennomføre testen tidligere i prosessen da vi kunne fått invitert testpersonene til en oppfølgingstest etter implementasjon av de forespurte endringene. Denne oppfølgingstesten ville gi oss svar og bekreftelse om at endringene var tilstrekkelige og slik som testpersonen ønsket. (Barnum, 2010)

Tidsstyring

Prosjektgjennomføringen ble tidlig definert av tidsfrister og innleveringer, både av dokumentasjon av prosjektet og kode. Dokumentasjonsprosessen var mer ressurskrevende enn antatt, dette var en risiko vi ikke forutså. Resultatet av dette er at gruppen mistet dyrebar tid i planleggingen til implementeringsfasen. Informasjon som var kritisk for implementeringsfasen som burde blitt etterspurt tidligere er oversikt over databasen og designet til brukergrensesnittet. Dette resulterte i utsatt oppstart av implementeringsfasen. Gjennom tilltak, hovedsaklig flere timer og nedprioritering av dokumentasjon, ble prosjektet gjennomført, men det ble satt ett unødvendig tidspress. Vi gikk 10 timer over budsjettert timeantall, men vi anser det som en liten overskridelse og anser prosjektet som fullført innenfor presentert budsjett.

7 KONKLUSJON OG VIDERE ARBEID

Dette kapittelet presenterer funnene og konklusjonene fra prosjektet, samt forslag til videre arbeid basert på disse resultatene. Vi vil først evaluere prosjektmålene og diskutere hvorvidt de ble oppnådd. Deretter vil vi identifisere eventuelle utfordringer og begrensninger som ble møtt underveis i prosjektet, og foreslå løsninger for å håndtere disse. Videre vil vi diskutere muligheter for videre utvikling og forbedring av løsningen, og presentere en plan for fremtidig arbeid. Til slutt vil vi oppsummere betydningen av prosjektet og dets bidrag til det overordnede målet.

7.1 Vurdering av prosjektmål

Basert på brukertestene og den tekniske verifiseringen vurderer vi applikasjonen som et godt svar på problembeskrivelsen. Denne vurderingen er basert på tilbakemeldingen på koden som var positiv, samt at de fleste av kravene til applikasjonen ble møtt. Dette er et godt utgangspunkt for videre utvikling og de viktigste behovene til selskapet vil være mulig å gjennomføre i vår applikasjon. Basert på tilbakemeldingene og evalueringsresultatene var det få endringer som må til for at produktet skal kunne deployes.

7.1.1 Hvorfor nådde vi ikke alle målene

Målene vi nådde ble presentert i 5.3, innenfor tiden som var til rådighet var det dessverre ikke mulig å utvikle all funksjonaliteten som var ønsket. Her etableres det hva som ikke ble ferdig og hvorfor.

Sletting av brukere

Det største målet vi ikke nådde, var sletting av brukere. Da vi startet prosjektet gikk vi ut ifra at det var en simpel operasjon for å slette en bruker. Dette viste seg ved ytterligere utforskning å ikke stemme. Brukerene hadde flere koblinger til annen data som var spredt utover tabeller i systemet, noe som gjorde dette til en større prosess å implementere enn forventet. Dette var en uforutsett blokk, men ikke en systemkritisk en. Dette er en funksjonalitet som ville kunne hjelpe selskapet med å frigjøre små mengder lagringsplass og gi bedre plass i totaloversikten. Dette er en funksjonalitet som heller ikke er på plass i dagens system, så selskapet vil ikke tape noe funksjonalitet på overgang til det nye systemet.

Søkefelt

Søkefelt på toppen av listen er eneste ønske etter brukertesten som ikke ble implementert. Dette var ikke en del av de opprinnelige funksjonelle kravene, og siden løsningen brukes i en nettleser, kan brukerne dra nytte av nettleserens innebygde søkefunksjonalitet.

I tiden etter brukertesten var det punktene vi så på som mest kritiske for løsningen som ble implementert. Søkefeltet var minst kritisk for funksjonaliteten til løsningen og var det ønsket som ville krevd mest tid å planlegge og å implementere. Derfor nedprioriterte vi denne.

7.2 Videre arbeid på prosjektet

Denne applikasjonen ble ønsket som et større prosjekt med flere funksjonaliteter nevnt i kapittel om initielle krav og løsnings ide (2.1.2 og 2.1.3). Dette er ideer som enda lever og selskapet ønsker denne løsningen videreutviklet. Det ble også etter evalueringsmøte med selskapets systemarkitekt utarbeidet en liten liste med forbedringer som kreves før produktet er helt produksjonsklart og kan kobles på det eksisterende systemet.

Forbedringene nevnt over vil kunne kvitte systemet med flere flaskehalser og manuelt ar-

beid, og føre til at de ansatte har flere muligheter og metoder for å løse drift, kommunikasjon og forespørsler.

7.3 Nytteverdi

Nytteverdien internt i selskapet er stor. Verdien ligger som beskrevet i visjonsdokumentet (vedlegg 1) i all tiden dette systemet vil spare for selskapet som en helhet. Ikke bare i ren behandlingstid for forespørsler, men også i verdien dette skaper i kunderelasjoner. Disse situasjonene som tidligere kunne ta alt fra en dag til flere uker vil nå kunne løses på minutter.

Kildeliste

Eidsmo, Arnstein (2001). *Relasjonsdatabaser og SQL*. Høgskolen i Nord-Trøndelag.

Watkins, John (2001). *Testing IT: An Off-the-Shelf Software Testing Process*. Cambridge University Press, s. 45–52.

Dahle, Knut Johannes (2007). *Praktisk bruk av parprogrammering – et industrielt studie*. URL: `https://www.duo.uio.no/bitstream/handle/10852/9766/dahle.pdf?sequence=3&isAllowed=y`. (Lest : 03.03.2024).

Barnum, Carol (2010). *Usability Testing Essentials : Ready, Set... Test!* Elsevier Science Technology.

Martin, Robert Cecil (2012). "Agile Software Development Principles, Patterns, and practices". I: Pearson Education International.

Spinellis, Diomidis (2012). "Git". I: Los Alamitos, CA: IEEE, s. 100–101.

Bacchelli, Alberto og Christian Bird (2013). *2013 35th International Conference on Software Engineering (ICSE)*. IEEE.

Oluwatosin, Haroon Shakirat (2014). "Client-server model". I: *IOSR Journal of Computer Engineering* 16.1, s. 67–71.

UIO (2017). *Fem tips til deg som skal gjennomføre brukertester*. URL: `https://www.usit.uio.no/om/organisasjon/ffu/ux/blogg/2017/brukertest.html`. (Lest : 16.04.2024).

Carullo, Giuliana (2020). *Implementing Effective Code Reviews : How to Build and Maintain Clean Code*. Apress : Imprint: Apress.

Clark, A (2021). *Evaluation and why it is important*. URL: `https://www.ebhoward.com/evaluation-and-why-it-is-important/`. (Lest : 12.04.2024).

Abreu, António (2022). *Project Risk Assessment and Corporate Behavior: Creating Knowledge for Sustainable Business*. MDPI - Multidisciplinary Digital Publishing Institute.

GitLab (2023). *What is a code review?* URL: https://about.gitlab.com/topics/version-control/what-is-code-review/. (Lest : 15.04.2024).

angular.io (2024). *Angular - what is Angular*. URL: https://angular.io/guide/what-is-angular. (Lest : 14.04.2024).

IBM (2024). *What is a REST API*. URL: https://www.ibm.com/topics/rest-apis. (Lest : 29.02.2024).

jenkins.io (2024). URL: https://www.jenkins.io/doc/. (Lest : 12.04.2024).

pgadmin.org (2024). *PGAdmin*. URL: https://www.pgadmin.org/. (Lest : 09.05.2024).

Postgres (2024). URL: https://www.postgresql.org/about/. (Lest : 12.04.2024).

Raeburn, A. (2024). *Extreme programming (XP) gets results, but is it right for you?* URL: https://asana.com/resources/extreme-programming-xp. (Lest : 15.04.2024).

spring.io (2024). *Spring Projects*. URL: https://spring.io. (Lest : 12.04.2024).

Docker (2024). URL: https://docs.docker.com/get-started/overview/. (Lest : 16.04.2024).

Vedleggsliste

1. Visjonsdokument
2. Prosjekthåndbok
3. Kravdokument