Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Ole Løkken
Mathias Myrold
Håvard Tysland

Utvikling av Atea Pricing Calculator

En webapplikasjon for salgsansatte i Atea

Bacheloroppgave i dataingeniør
Veileder: Ole Christian Eidheim
Mai 2022

NTNU
Kunnskap for en bedre verden

Ole Løkken
Mathias Myrold
Håvard Tysland

Utvikling av Atea Pricing Calculator

En webapplikasjon for salgsansatte i Atea

Bacheloroppgave i dataingeniør
Veileder: Ole Christian Eidheim
Mai 2022

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU
Kunnskap for en bedre verden

Forord

Denne rapporten er en bacheloroppgave skrevet for Norges teknisk-naturvitenskapelige universitet av tre dataingeniørstudenter, våren 2022. Oppgaven er gitt av bedriften Atea og veiledet av Ole Christian Eidheim fra NTNU. I rapporten vil man ved videre lesing få en innsikt i oppgaven, hva som er blitt utviklet, hvordan det er utviklet og relevant teori.

For teamet var oppgaven interessant av flere årsaker:

- Teamet ønsket å prøve seg på å utvikle et større produkt.
- Bedriften var interessert i å videreføre prosjektet, og en god implementasjon kunne være med på å bidra til verdiskapning i selskapet.
- Friheten til å ta egne valg på teknologier som teamet så på som interessante.

Prosjektet har vært givende, og har gitt studentene et godt innblikk i arbeidshverdagen som systemutvikler. Teamet har blitt utfordret på et teknologisk, ingeniørfaglig og administrativt plan. Planlegging, diskutering og gjennomføring av et slikt prosjekt gir gode erfaringer som tas med videre. Teamet står igjen med et produkt som samtlige er fornøyde og stolte av.

Avslutningsvis ønsker vi å utrykke vår takknemlighet til:

- Ole Christian Eidheim fra NTNU for alle gode tips han har gitt i rollen som veileder.
- Joachim Stamnes Flatberg fra Atea for hans rolle som oppgavestiller og veileder.
- Alle som har bidratt til å svare på brukertester.

Ole Løkken	Mathias Myrold	Håvard Tysland
Trondheim, 19.05.22	Trondheim, 19.05.22	Trondheim, 19.05.22

Oppgavetekst

Ved oppstart av bacheloren var følgende oppgavetekst gitt av Atea, som også har vært stående gjennom hele prosjektet:

Hensikt:

Oppgaven skal kunne utfordre studentene innen moderne utviklingsmetodikk, foreta og begrunne vurderinger for en løsning basert på bedriftens ønsker og hva som måtte være ansett som best-practice, samt strategisk tankegang iht. gjeldende standarder for valgt teknologi.

Oppgaveforslag:

Oppgaven tar utgangspunkt i eksisterende webapplikasjon, «Atea Pricing Calculator» (APC) og tilhørende bakenforliggende struktur for AMS (Atea Managed Services). APC er en webapplikasjon med en komplett oversikt over tjenester, med tilhørende varelinjer som tilbys av AMS i dag.

AMS, avd. R&D (Research & Development) ønsker en full omutvikling av APC hvor modernisering og universell utforming/brukervennlighet står i fokus, sammen med tettere integrasjoner mot eksisterende systemer for å komplettere datagrunnlaget til APC.

En omutvikling av APC vil berøre både frontend- og backend-stack.

Utfyllende kommentarer:

AMS benytter i dag APC til oppslag av fakturerbare tjenester og utforming av priskalkyler, som igjen kan benyttes ved et eventuelt anbud. APC kjører i dag på en egen isolert instans, med en egen lokal database uten noen form for dynamikk, dette kan gjøre oppslagene på sikt utdaterte.

- Målet for re-utviklingen/moderniseringen av APC kan være, men ikke begrenset til følgende;
- Brukervennlig grensesnitt og brukeropplevelse
- Lett å navigere
- God tjenesteoversikt med tilhørende prisoversikt
- Gode søkemuligheter
- Mulighet til å enkelt bygge dynamiske tilbud
- Tett integrasjon med eksisterende oppslagsverktøy for AMS
- Datagrunnlaget APC skal benyttes oppdateres dynamisk vha. API
- Gode rapporteringsmuligheter for kjøring av eller opprette bl.a. priskalkyler for gitte fakturerbare tjenester.

Mulige teknologier for å løse utviklingen av oppgaven kan være, men ikke begrenset til følgende;

- React-basert frontend
- Golang-basert backend
- Mikrotjenestebasert arkitektur på kubernetes

Sammendrag

Denne bacheloroppgaven omhandler utvikling av webapplikasjonen Atea Pricing Calculator, et system brukt av salgspersoner i Atea. Bakgrunnen for prosjektet var et behov for re-utvikling av en eksisterende webapplikasjon med fokus på brukervennlighet og modernisering, samt tettere integrasjon mot Atea sine eksisterende systemer.

Atea Pricing Calculator innebærer en komplett oversikt for Atea sine tjenester og tilhørende varelinjer. Det tilbys muligheter for å hente ut rapporter, lage anbud som kan presenteres for mulige nye kunder og opprette spesialtilbud for allerede eksisterende kunder. Systemet baserer seg på data hentet fra eksisterende systemer i Atea, samt data lagt inn av salgspersoner.

Webapplikasjonen inkluderer en klient som er utviklet med JavaScript biblioteket React og komponentbiblioteket Material UI. Videre er det utviklet et API ved hjelp av C# og .NET 6 som, i tillegg til å kommunisere med den lokale databasen, oppdaterer faktureringsmotoren Atea benytter til å fakturere kunder.

Gjennom prosjektet har utviklerne utforsket forskjellige elementer ved å lage en webapplikasjon. Teamet har utforsket hvordan man kan utvikle en sikker, brukervennlig og dynamisk webapplikasjon som legger opp til videreutvikling. Dette innebærer sikkerheten rundt en identitetsleverandør, samt fordeler og ulemper med komponentbiblioteker som Material UI. Videre er det også sett på hvilke valg man bør gjøre for å utvikle et API som skal kommunisere med flere eksterne tjenester, og hvilke tiltak man kan ta for å legge opp til videreutvikling av et prosjekt.

Resultatet av utforskningen viser at fordelene ved å ta i bruk komponentbibliotek i stor grad overveier ulempene. Videre viser erfaringene gjennom prosjektet et positivt inntrykk av å ta i bruk identitetsleverandører for applikasjonens sikkerhet. Derimot, som for all annen programvare kan den ikke sørge for en sikker applikasjon på egenhånd. I tillegg, har erfaringer og annen kunnskap ført til forslag til hvordan man utvikler API med hensyn til flere eksterne tjenester, samt spesifikke tiltak som bidrar til enklere overgang mot videreutvikling av et prosjekt.

Abstract

This bachelor thesis deals with the development of the web application Atea Pricing Calculator, a system used by salespeople in Atea. The background of the project was a need for re-development of an existing web application, with focus on usability and modernization, as well as closer integration with Atea's existing systems.

Atea Pricing Calculator includes a complete overview of Atea's services and associated item lines, with options for retrieving reports, creating offers that can be presented for possible new customers and creating special offers for already existing customers. The system is based on data collection from existing systems in Atea, as well as data entered by salespeople.

The Web application includes a client developed with the JavaScript library React and the Material UI component library. Furthermore, an API has been developed using C# and .NET 6, which in addition to communicating with the local database, updates the billing engine Atea utilizes to invoice customers.

Throughout the project the developers have explored various elements of creating a web application. The team has explored how to develop a secure, user friendly and dynamic web application that provides for further development. This involves security connected to an identity provider, as well as advantages and disadvantages of component libraries such as Material UI. Furthermore, which choices should be made to develop an API that will communicate with several external services and what measures that can be taken to plan for further development of a project, have been focused on.

The results of the exploration show that the advantages of using a component library largely outweighs the disadvantages. Furthermore, the experiences through the project show a positive impression of using identity providers for the security of the application. On the other hand, as for all other software, it cannot provide a secure application on its own. In addition, experience and other knowledge have led to proposals for how to develop APIs with regard to several external services, as well as specific choices that contribute to an easier transition in to further development of a project.

Innhold

Figurer

Tabeller

Akronymer

AAD – Azure Active Directory

AMS – Atea Managed Services

APC – Atea Pricing Calculator

API – Application Programming Interface

CRUD – Create, Read, Update, Delete

CSS – Cascading Style Sheets

DevOps – Development and operations

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

JSON – JavaScript Object Notation

ORM - Object-Relational Mapping

REST – Representational state transfer

SPA – Single Page Application

SSO – Single Sign-On

SQL – Structured Query Language

URI – Uniform Resource Identifier

WCAG – Web Content Accessibility Guidelines

Ordliste

I rapporten er det noen begrep som blir gjentatt, som kan være nyttig å forstå i konteksten av rapporten. Disse blir forklart under.

Atea Pricing Calculator Produktet som utvikles i gjennom prosjektet.

Atea Managed Services Avdelingen i Atea bacheloroppgaven skrives for.

Faktureringsmotor En tjeneste som blir brukt for å fakturere kunder.

Exivity En ekstern faktureringsmotor brukt av Atea Managed Services.

Tjeneste Et produkt som Atea selger til sine kunder.

Varelinje Et fakturerbart underprodukt av en tjeneste.

1 Introduksjon

I dette kapittelet blir prosjektets bakgrunn og problemstilling introdusert. De tre studentene som har gjennomført rapporten vil bli omtalt som teamet eller utviklerne.

1.1 Bakgrunn

Atea er et IT-selskap med 22 kontorer over hele Norge i tillegg til en rekke andre land. Via disse leverer de gjennom APC en komplett oversikt over tjenester med tilhørende varelinjer som tilbys av AMS. Hovedmotivasjonen for oppgaven var en re-utvikling av APC hvor modernisering og brukervennlighet sto i fokus. Tidligere versjon av APC var en statisk nettside uten noe kobling mellom nettsiden og faktureringsmotoren som ble brukt til å fakturere kundene av AMS. Dette kunne resultere i salgsfeil ved at ulike priser var gitt i APC og i faktureringsmotor. Det var ønskelig at den nye løsningen skulle ordne dette problemet. Videre var dette et prosjekt som kunne videreføres av Atea etter endt bachelorperiode, og det var ønsket at teamet tok hensyn til dette under utvikling.

1.2 Problemstilling

Utviklingen av selve webapplikasjonen er hovedfokuset under prosjektet. Oppgavestiller har ulike ønsker og krav til løsningen som teamet skal følge. Som beskrevet i oppgaveteksten ønskes en god brukeropplevelse ved at nettsiden tar hensyn til prinsipper innenfor brukervennlighet og universell utforming. Etter videre diskusjoner med oppgavestiller kommer det også frem at mulighet for videreutvikling settes pris på. I tillegg skal kun bestemte personer ha tilgang til nettsiden, og ulike funksjoner skal kreve spesialtilganger. Til slutt er det også ønskelig med en dynamisk applikasjon, hvor varelinjer oppdateres i nettsiden samtidig som de oppdateres i faktureringsmotoren. I den sammenheng vil teamet under utviklingen ta hensyn til følgende problemstilling:

«Hvordan lage en sikker, brukervennlig og dynamisk webapplikasjon som legger til rette for videreutvikling?»

Det er en bred problemstilling med mange ulike aspekter. Det er dermed valgt å konkretisere ned til følgende forskningsspørsmål som skal besvares i konklusjonen basert på empiri og teori som er samlet i løpet av prosjektet:

1. *Fører bruken av en identitetsleverandør til en sikker applikasjon på egenhånd?*
2. *Hva er fordelene og ulempene ved å bruke et komponentbibliotek for å utvikle en brukervennlig nettside?*
3. *Hvilke valg bør tas hensyn til når man utvikler et API som jobber sammen med flere eksterne tjenester?*
4. *Hvilke tiltak kan man ta for å legge opp til videreutvikling i et prosjekt?*

1.3 Rapportstruktur

Rapportens struktur er delt inn slik:

Kapittel 1 - Introduksjon gir en beskrivelse av prosjektets bakgrunn. I tillegg blir problemstillingen presentert.

Kapittel 2 – Teori presenterer relevant teori knyttet til problemstillingen og utviklingen.

Kapittel 3 – Valg av teknologi og metode beskriver de ulike valgene som har blitt gjort i løpet av prosjektet. Det innebærer blant annet valg av utviklingsmetode og teknologier.

Kapittel 4 – Resultater inneholder de produktrelaterte, administrative og vitenskapelige resultatene som har fremkommet.

Kapittel 5 – Diskusjon består av drøfting rundt prosessen og resultatene i lys av problemstillingen.

Kapittel 6 – Konklusjon og videre arbeid inneholder konklusjonene som trekkes basert på resultatene og diskusjonen knyttet opp mot problemstillingen og det utviklede produktet. Det blir også lagt frem tanker om arbeid som kan bli gjort videre.

2 Teori

2.1 Teknologi

2.1.1 Single-page application

Single-page application eller SPA er en populær måte å lage nettsider på. De laster raskt uten å kommunisere med servere hver gang brukeren samhandler med nettsiden. En slik applikasjon består av ett stort webdokument som blir lastet inn én gang. Der samhandling fra bruker bare endrer på komponenter med data fra webservere. Motsetningen av en slik applikasjon er en *multi-page application*, hvor hele sider fra webserveren blir hentet ut og dermed øker innlastningstiden til nettsiden. Å lage applikasjonen som en *single-page application* vil resultere i god ytelse og en mer dynamisk opplevelse av nettsiden.

2.1.2 Komponentbiblioteker

Komponentbiblioteker for brukergrensesnitt består av ferdiglagde og designede komponenter med JavaScript, HTML og CSS. Slike bibliotek er ofte åpen kildekode og dermed tilgjengelige for alle. Når et brukergrensesnitt utvikles, kan utviklere importere komponenter til sitt prosjekt. Å lage komponenter fra grunnen av tar tid samtidig som det ofte fører til duplisering av kode. Bibliotekene inneholder ofte fleksible komponenter som kan brukes i flere sammenhenger og dermed reduserer man dupliseringen av kode.

Ved utvikling av et grensesnitt velger ofte utviklere ett bibliotek og holder seg til dette igjennom utviklingen. Dette bidrar til konsekvente løsninger for nettsiden og en bedre brukeropplevelse med sammenheng i designet og funksjonaliteten. Ettersom bibliotekene ofte er åpen kildekode og blir oppdaterte kontinuerlig når problemer eller utfordringer oppstår, blir komponentene mer komplette og kompatible. Et hyppig problem med selvutviklede komponenter kan være at de er inkompatible med noen nettlesere. Slike problemer kan utviklere unngå med å bruk av komponentbiblioteker.

2.1.3 Relasjonsdatabaser

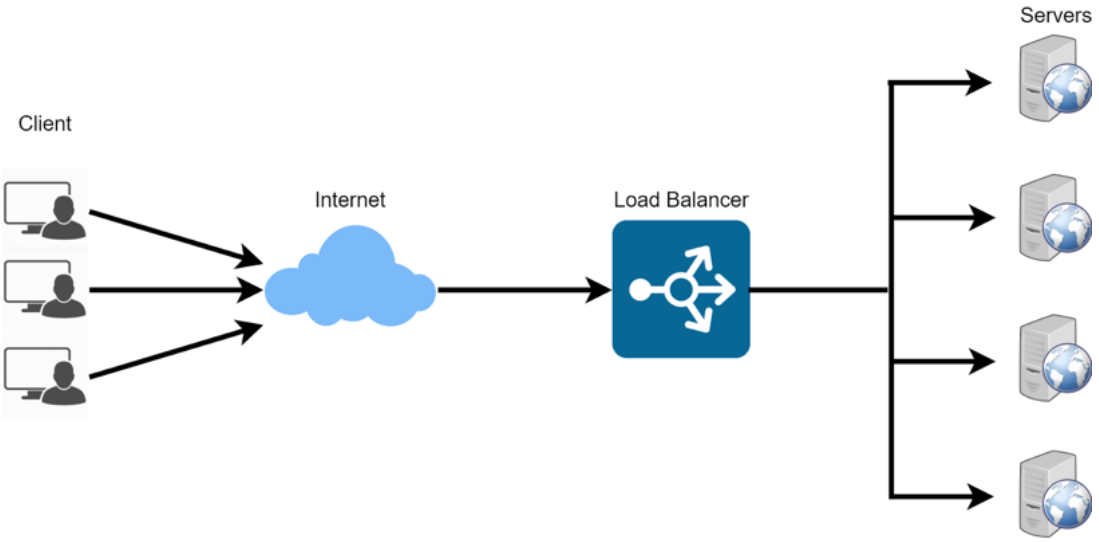
En relasjonsdatabase er en database hvor data er organisert som rader i tabeller. En rad vil normalt sett inneholde data for én identifiserbar enhet. I tillegg vil en rad gjerne inneholde relasjoner til andre tabeller. Fordelen med slike databaser er muligheten til å skape meningsfull data ved å kombinere to eller flere tabeller. Slik kan man hente ut akkurat den dataen som trengs .

2.1.4 REST

Representational State Transfer er en arkitekturstil når det kommer til å lage API-er. REST ble først definert av Roy Fielding i år 2000. RESTful API-er har siden den gang vært en av de vanligste måtene å binde flere tjenester i en applikasjon sammen. Det kjennetegnes gjerne ved at ressursene organiseres som en rekke med unike URI-er. Kommunikasjonen skjer gjennom HTTP kall og CRUD operasjoner. Et RESTful API vil bruke en GET-request for å få informasjon, en POST-request til å lage informasjon, en PUT-request til å endre informasjon, og en DELETE-request til å slette informasjon. HTTP statuskoder spiller en viktig i rolle for å gi klienter informasjon om forespørselen. Noen typiske statuskoder er 200 (*OK*), 404 (*Not Found*) og 401 (*Unathorized*).

2.1.5 Belastningsfordeler

Når datamengden mot en server øker, så er en løsning å utvide til flere servere. For å distribuere datatrafikken på en effektiv måte, bruker man en belastningsfordeler. En belastningsfordeler kan både være i form av maskinvare eller programvare. Likevel er grunnene for å bruke en de samme. Den sørger for at applikasjonen er pålitelig og tilgjengelig ved at den sender forespørsler videre til tilgjengelige servere hvis en av de krasjer. Algoritmer som *round robin* eller *least connections* brukes til å fordele trafikken. For at de mest gjentatte forespørslene skal gå enda raskere, er det vanlig at belastningsfordeleren har et hurtigminne. Belastningsfordeleren lagrer dataen i de vanligste forespørslene, og klientene får raskere svar ettersom forespørslene ikke trenger å gå gjennom serveren.

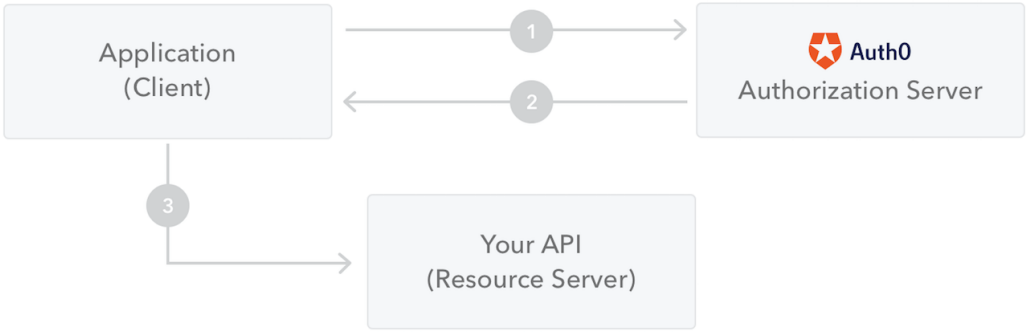


Figur 1: Eksempel på hvordan en belastningsfordeler arbeider.

2.1.6 Autorisasjon og autentisering

Autorisasjon og autentisering er begreper som brukes om hverandre, men representerer to forskjellige oppgaver. Autentisering er oppgaven med å finne ut hvem en bruker er,

mens autorisasjon er å finne ut hva en bruker har tilgang til. Autentiseringen foregår før autorisasjonen. Historisk har det vært normalt med en sesjonsbasert autentiseringsmetode. Her må serveren lagre informasjon om tilstanden til klienten. Med mange brukere kan dette kreve mye arbeid fra maskinvaren. JSON web tokens er en annen autentiseringsmetode som har blitt populær de siste årene. Det fungerer slik at når man logger inn får man utdelt et token som lagres hos klienten. Tokenet brukes når det skal sendes spørringer mot API-et tokenet tilhører. Tokenet er enkodet og hvis man prøver å endre på det vil det bli ugyldig. Det er vanlig å bruke autentiseringsrammeverk som OAuth 2.0. Da benytter man seg av en ekstern tredjepart som tar ansvar for registrering og innlogging, illustrert i figur 2.



Figur 2: Eksempel på rekkefølgen i en autentiseringsprosess som bruker protokollen OAuth 2.0.

2.1.7 Containerteknologi

For at en applikasjon ikke skal ha tilgang til data på host-maskinen kan det være hensiktsmessig med virtualisering. Det betyr å isolere et system, slik at det opplever å kjøre med helt egen maskinvare. Mellomlaget som sørger for virtualiseringen gjør det mulig å fordele ressurser etter behov . Dette er mer effektivt enn å kjøre flere applikasjoner på en server, uten å ha muligheten til å justere ressursbruken.

En normal fremgangsmåte for å oppnå dette er å bruke containere. Containere er en lettvekt form for tradisjonell virtualisering, som gjør de enkle å transportere mellom systemer. Siden en container ikke er sårbar for forskjellige miljøer, sparer man tid på å få driftsmiljø til å fungere.

2.2 Brukervennlighet

2.2.1 Universell utforming

Universell utforming er designing av produkter og miljøer for å kunne bli brukt av alle typer folk, i størst mulig grad, uten behov for et adaptert eller spesialisert design. Denne definisjonen ble gitt av Ron Mace ved North Carolina State University. I 1997, sammen

med en rekke andre arkitekter, designere og ingeniører kom han med følgende syv prinsipper for universell utforming:

1. **Like muligheter for bruk** – Designet skal være brukbart og tilgjengelige for brukere med ulike ferdigheter.

2. **Fleksibel i bruk** – Designet inneholder en stor rekkevidde med individuelle preferanser. Eksempelvis kan en bruker like et mørkt tema, mens en annen kan foretrekke et lyst tema.

3. **Enkel og intuitiv i bruk** – Designet er lett å forstå, uavhengig av brukerens erfaringer, kunnskaper og konsentrasjonsnivå.

4. **Forståelig informasjon** – Designet kommuniserer nødvendig informasjon til brukeren på en effektiv måte.

5. **Toleranse for feil** - Designet minimiserer konsekvensene for utilsiktede feil. Dette kan gjøres ved å stenge adgang til en bruker for steder den ikke skal være. I tillegg kan man gi brukeren advarsler og sikre funksjoner.

6. **Lav fysisk anstrengelse** – Designet kan bli brukt effektiv og komfortabelt med minst mulig utmattelse. Brukeren skal kunne bruke systemet lett og komfortabelt.

7. **Størrelse og plass for tilgang og bruk** – Hensiktsmessig størrelse og plass skal gjøre det mulig å bruke uavhengig av brukerens størrelse, holdning eller mobilitet.

2.2.2 Don Normans seks designprinsipper

Donald Norman er en av de mest betydelige forskerne innenfor feltet menneske-maskin-interaksjon og brukersentrert design. Basert på Normans idé om at enheter, datamaskiner og grensesnitt skal fungere korrekt og være intuitive, utarbeidet han seks prinsipper for design av grensesnitt :

1. **Synlighet** – En bruker skal kunne vite hvilke valg den har, og hvordan den utnytter dem, kun ved å se på et brukergrensesnitt. Hvis en knapp er gjemt bak en meny vil det være vanskelig å vite at den finnes.

2. **Tilbakemelding** – En bruker skal motta tilbakemeldinger når en handling blir utført, slik at den får beskjed om hvorvidt handlingen var vellykket. Dette kan være når en bruker sletter et objekt og får en tilbakemelding om slettingen var vellykket eller ikke.

3. **Signaler** – Linken mellom hvordan noe ser ut og hvordan det er brukt. En kaffekopp kan man vite hvordan man skal holde ved å kun se på den. På samme måte bør et grensesnitt være intuitivt slik at en bruker vet hvordan den skal finne ønsket informasjon kun ved å se på grensesnittet.

4. **Overføring** – Sammenhengen mellom kontroller og deres effekter burde samstemme. Eksempel på dette kan være ved volumkontroll med knapper over hverandre. For at det skal være god overføring vil da den øvre knappen øke volumet og den nedre senke volumet. Andre eksempler kan være når brukeren holder musen over en knapp vil fargen eller musepekeren endres. Slik vil brukeren kunne forstå at den er klikkbar.

5. **Begrensninger** – Begrense en brukers interaksjonsmulighet ved et grensesnitt. Dette er for å forhindre at brukeren blir overveldet over alle muligheten den tilbys. Å forhindre at brukeren kan legge inn bokstaver i et tekstfelt for telefonnummer er et eksempel på gode begrensninger.

6. **Konsistens** – Mennesker lærer nye ting fortere når de gjenkjenner mønster. Konsistens i form av at like løsninger utgir like resultat, er viktig for at en bruker skal gjenkjenne disse mønstrene. Dersom en knapp med et pluss-ikon ett sted betyr å legge til noe, bør den gjøre det i resten av grensesnittet også.

2.2.3 Web Content Accessibility Guidelines

Web Content Accessibility Guidelines, også kjent som WCAG, er utviklet med mål om en delt standard å gjøre nettinnhold så tilgjengelig som mulig for alle, uavhengig av funksjonsevne. Retningslinjene hjelper utviklere å lage et universelt utformet grensesnitt. WCAG er bygd opp på fire prinsipper :

1. **Mulig å oppfatte** – Nettstedet skal kunne brukes av alle type mennesker uavhengig av de kunnskaper og erfaringer brukeren besitter.
2. **Mulig å betjene** – Brukeren skal kunne navigere med det utstyret de benytter, enten det er mus eller kun tastatur.
3. **Forståelig** – Det skal være mulig for brukere å forstå hvordan sider brukes, samt den informasjonen som blir presentert.
4. **Robust** – Løsningen skal fungere på ulike nettlesere og hjelpemiddelteknologi som finnes. Dette kan for eksempel være skjermlesere.

2.3 Utviklingsprosess

2.3.1 Smidig utvikling

Smidig utvikling er en iterativ tilnærming til prosjektstyring og systemutvikling som hjelper team å levere verdi til kundene deres raskt og smertefritt. Et team som utvikler smidig jobber i kortere og mer overkommelig perioder. Mot slutten av hver periode evalueres resultatene, gjerne sammen med kunden. Slik blir det mulig å tilpasse seg til tilbakemeldingene underveis i prosjektet, og man unngår dermed ofte å forkaste store planer fra en tidlig fase .

Smidig utvikling baserer seg på verdiene og prinsippene som er utrykt i det agile manifestet for systemutvikling . Tidlig i 2001 i Utah møttes 17 personer en helg for å

diskutere fremtiden for systemutvikling. Samlingen ble enige om at problemet i den tid var at bedrifter var for fokusert på å planlegge og dokumentere, at de glemte å tilfredsstille kundens behov. Det agile manifestet gjorde sin entré etter denne helgen, og endret systemutvikling for alltid. Manifestet bygger på 4 følgende verdier:

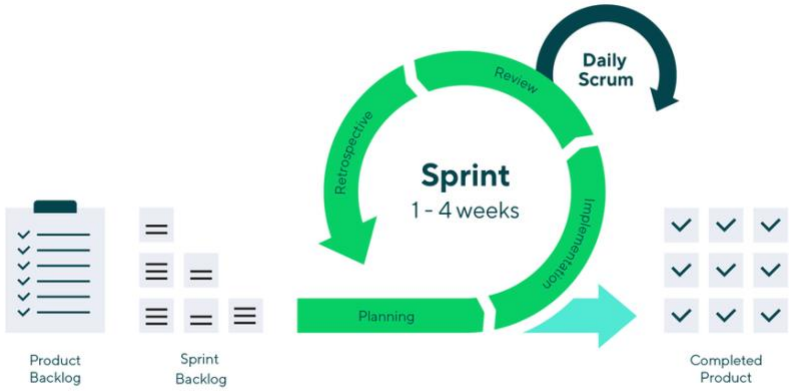
- **Individer og interaksjoner** fremfor prosesser og verktøy
- **Fungerende programvare** fremfor omfattende dokumentasjon
- **Kundesamarbeid** fremfor kontraktforhandling
- **Respondere på endring** fremfor å følge en plan

2.3.1.1 Scrum

Scrum er en av de vanligste smidige metodene innen systemutvikling. Viktige egenskaper innenfor scrum er tilpasning, læring og refleksjon gjennom erfaringer for å skape kontinuerlig forbedring av arbeidet. I scrum er det tre såkalte *artifacts*: produkt *backlog***,** sprint *backlog* og et sprint mål. Produkt *backlog* inneholder en dynamisk liste med alle krav og forbedringer. Sprint *backlog* består av en liste med funksjoner som skal bli implementert i en sprint. Disse funksjonene er hentet fra produkt *backlog* under sprint planlegging. Sprint mål er det definerte produktmålet for en sprint, og kan variere basert på et teams definisjon på å være ferdig. Noe av det mest essensielle i scrum er de gjentagende møtene som kan finne sted: sprint planlegging, sprint*,* daglig *stand up*, sprint review og sprint retrospektiv. Under sprint planlegging blir arbeidet for den kommende sprinten planlagt av hele teamet. Sprinten er den tidsperioden hvor teamet arbeider sammen for å fullføre sprint målet, og en typisk lengde er mellom en til fire uker. En daglig *stand up* er et kort møte som har som mål å sørge for at alle i teamet jobber mot å oppnå sprint målet. Her går hvert enkelt teammedlem gjennom disse tre spørsmålene:

- Hva ble gjort i går?
- Hva skal gjøres i dag?
- Er det noen hindringer?

Under sprint *review* samles teamet for en demonstrasjon av sprint målet og *backlog*. Sprinten avsluttes med en retrospektiv. Her samles teamet for å dokumentere og diskutere hva som fungerte og hva som ikke fungerte. Slik kan man sørge for å tilpasse, lære og reflektere slik at man har en kontinuerlig forbedring.



Figur 3: Eksempel på utvikling med scrum .

2.3.1.2 Kanban

Kanban er et annet populært rammeverk innenfor smidig utvikling. Å holde en stabil fremgang, samt at utviklingsteamet til enhver tid har noe å jobbe med er viktig. For å oppnå dette brukes ett kanban-brett, som er et verktøy for å visualisere arbeid og maksimere effektivitet. Brettet bruker kort, kolonner og kontinuerlig forbedring for å oppnå dette, illustrert i figur 4. Kolonnene består av nåværende status for et kort, som teamet selv kan definere. Disse brettene kan fremkomme både fysisk og digitalt, men essensen er fortsatt den samme .

Figur 4: Eksempel på Kanban-brett.

2.3.2 Versjonskontroll

Versjonskontroll er en nyttig programvare under utvikling i team som muliggjør at flere kan jobbe med samme prosjekt samtidig. Utviklere kan dermed jobbe hver for seg med kopier av prosjektet uten å forstyrre eller lage problemer i andres arbeid. Når kopiene skal slås sammen til et prosjekt har versjonskontrollsystemet ofte verktøy for flette de sammen på en trygg måte. Under en slik fletting kan det oppstå konflikter, der flere utviklere har gjort endringer som kolliderer med hverandre. Hvis dette skulle skje vil versjonskontrollsystemet oppdage og hjelpe med å løse konfliktene. Skulle det oppstå større feil i prosjektet, er det også mulig tilbakestille prosjektet til en tidligere fungerende versjon.

Noen versjonskontrollsystemer tilbyr også *pull requests*, som er et hjelpemiddel for trygg samling av kode til hovedprosjektet. Når et teammedlem ønsker å slå sammen kopien sin til hovedprosjektet er det mulig å opprette en forespørsel om dette. For at sammenslåingen skal gjennomføres må et eller flere de av andre teammedlemmene gå igjennom endringene og godkjenne sammenslåingen.

Figur 5: Eksempel på en forløpet i en pull request

2.3.3 Development and operations

Development and operations er en arbeidsmetode som skal sørge for stabilitet og effektivitet mellom utviklere og driftere. Automatiserte prosesser skal sørge for å oppdatere systemer til nyeste versjoner, og samtidig gi tilbakemeldinger om noe er galt. Kontinuerlig integrasjon og kontinuerlig utrulling er viktige begreper innen *DevOps*. Kontinuerlig integrasjon går ut på at utviklere kontinuerlig oppdaterer versjonskontroll når de har utført endringer eller laget ny funksjonalitet. Da settes det i gang en prosess

med å bygge og teste applikasjonen som inneholder ny kode. Prosessen gir tilbakemelding til utviklerne om prosessen er vellykket eller mislykket. Kontinuerlig utrulling vil si at når versjonskontroll endres, settes det i gang en prosess som setter ny kode ut i produksjon. Rekkefølgen er ofte slik at prosessen med kontinuerlig integrasjon kjører først, og om den går gjennom vil kontinuerlig utrulling starte.

2.4 Vitenskapsteori

2.4.1 Kvalitativ metode

Kvalitativ metode er forskningsmetoder som brukes ved innsamling og analyse av kvalitative data. Denne type data forekommer oftest i tekstlig form. En slik studie omfatter ofte få deltakere, og har som hensikt å oppnå dybdekunnskap og helhetlig forståelse av noe .

2.4.2 Empiri

Empiri er data som understøttes eller grunner seg på erfaring . Dette kan samles inn ved hjelp av kvalitative metoder som brukertester, eller igjennom observasjoner.

3 Valg av teknologi og metode

3.1 Metode

3.1.1 Utviklingsmetodikk

I denne seksjonen vil valg av utviklingsmetodikk og planlegging bli presentert.

3.1.1.1 Scrumban

Ettersom arbeidet var lagt tett opp mot oppdragsgiver for å oppfylle krav og behov som kom fortløpende, opplevde teamet det som nødvendig med en smidig utviklingsmetode. Ved valg av utviklingsmetodikk ble derfor *Scrum* i en kombinasjon med *Kanban* valgt, også ofte kalt *Scrumban*. Den kombinerer strukturen med *artifacts* og de gjentagende møtene fra *Scrum*, med de mer flytbaserte metodene fra *Kanban* . Valget med å gå for en slik kombinasjon var først og fremst for å kunne kontinuerlig forbedre arbeidet basert på tilbakemeldinger gjennom sprint review og retrospektiv. I tillegg var visualiseringen og fleksibiliteten som tilbys av *Kanban* noe som ble satt i fokus. Dermed kunne teamet kontinuerlig hente oppgaver fra produkt *backlog* under en sprint om det var nødvendig. Videre var teamets tidligere erfaringer med de to metodene en påvirkende faktor. En alternativ utviklingsmetode kunne vært *Extreme Programming* , men praksiser som test-drevet utvikling ble ansett som ugunstig for å oppnå mest mulig fremgang under systemutviklingen.

Ved å benytte Azure DevOps sitt planleggingssystem, kan man sette opp *backlog* med arbeidsoppgaver som for hver sprint ble fordelt basert på den spesifikke sprintens mål. Det innebygde Kanban-brettet ble brukt for å fordele arbeidsoppgaver etter ønske og behov. Slik sikret teamet at alle hadde en oppgave å ta på seg. På denne måten ble også sprintens fremgang visualisert. Brettet og en demonstrasjon av produktet ble gjennomgått i sprint review sammen med produkteier slik at fremgang gikk som forventet og produkteier kunne tilføye med nye krav og tilpasninger.

3.1.1.2 Prototying og brukertester

Teamet ble i oppstarten presentert med en *wireframe* fra oppdragsgiver, som ble brukt som en skisse over ønsket utforming av nettsiden. Fordi produktets brukervennlighet var satt i fokus ble det dermed dannet flere *wireframes* for å skissere resten av nettsidens funksjonalitet. Disse ble tatt i bruk som et grunnlag for brukergrensesnittet under utviklingen av produktet.

Wireframene ble brukt til å gjøre brukertester på medstudenter, samt venner og familie. Hensikten var å få et dypere overblikk over det planlagte brukergrensesnittets positive og negative sider. Senere i prosjektet ble det utført brukertester av sluttproduktet på

xtilbakemeldinger av brukerne som skal ha nytte av nettsiden, ettersom de innehar kjennskap til innholdet i APC.

3.1.2 Prosess

Teamet startet tidlig med å planlegge fremgang for prosessen. Ettersom teamet valgte *Scrumban* som utviklingsmetode var det naturlig å planlegge systemutviklingen i sprinter hvor hver sprint hadde ett spesifikt mål. Teamet valgte tidlig å komme seg i gang med utvikling, og det var dermed naturlig å sette sprintmål ved starten av hver sprint. Det ble da mulig å tilpasse målene etter fremdriften, fremfor å måtte følge nøye lagt plan fra starten av. Årsaken til å nedprioritere overflødig planlegging i starten, var at teamet ville komme tidlig i gang med utviklingen. Dette bygger på prinsipp fra smidig utvikling. Den planlagte prosessen så slik ut:

1. **Oppstart (12. januar – 28. januar)**
   Oppstartsmøter, forprosjektplan, visjonsdokument og oppsett av utviklingsmiljø

2. **Sprint 1 (31. januar – 11. februar)**
   Mål: Sette opp en layout for nettsiden med navigasjon og sidemeny etter wireframe fra oppdragsgiver. Hente ut data fra ekstern database og sende videre til frontend.

3. **Sprint 2 (14. februar – 4. mars)**
   Mål: Lage dashbord side og oversikt over tjenester.

4. **Sprint 3 (7. mars – 18. mars)**
   Mål: Koble sammen spørringer til backend og opprette, redigere og slette tjenester.

5. **Posterpresentasjon (21. mars – 25. mars)**
   Eksamen i INGT2300 og presentasjon av problemstilling ved hjelp av poster

6. **Sprint 4 (28. mars – 8. april)**
   Mål: Distribuere nettsiden og vise kundeoversikt sammen med tilhørende spesialtilbud. I tillegg kunne legge til, endre og slette spesialtilbud.

7. **Sprint 5 (11. april – 29. april)**
   Mål: Generere rapporter for en kunde eller tjeneste til en gitt tidsperiode.

8. **Dokumentasjon (2. mai – 19. mai)**
   Mål: Dokumentasjon og hovedrapport

Hva	Fra dato	Til dato	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Dokumentasjon	12.01.22	20.05.22																				
Forprosjektplan	12.01.22	28.01.22																				
Utviklingsmiljø	24.01.22	28.01.22																				
Backlog	28.01.22	28.01.22																				
Sprint 1	31.01.22	11.02.22																				
Sprint 1 review og sprint 2 backlog	11.02.22	11.02																				
Sprint 2	14.02.22	04.03.22																				
Sprint 2 review og sprint 3 backlog	04.03.22	04.03.22																				
Sprint 3	07.03.22	18.03.22																				
Sprint 3 review sprint 4 backlog	18.03	18.03																				
Lage poster og presentasjon	21.03.22	25.03.22																				
Sprint 4	28.03.22	08.04.22																				
Sprint 4 review og sprint backlog 5	08.04.22	08.04.22																				
Sprint 5	11.04.22	29.04.22																				
Sprint 5 review	29.04.22	29.04.22																				
Hovedrapport og innlevering	02.05.22	20.05.22																				
Presentasjon av oppgaven	23.05.22	27.05.22																				

Figur 6: Gantt-diagram som ble skissert i oppstartsfasen med de ulike milepælene.

3.2 Teknologi

Valg av teknologier for frontend ble preget av teammedlemmers erfaringer og kompetanser. Ved å velge kjente teknologier sparer man teamet tid og ressurser, der alternativet ville være å lære seg nye teknologier fra grunnen av. Tiden som blir spart her kan heller brukes til implementering av flere funksjonaliteter i applikasjonen eller til å utnytte den valgte teknologien til sitt fulle potensiale. For oppdragsgiver var det som nevnt ønskelig med en brukervennlig og dynamisk nettside som kunne videreutvikles. Dette er tatt til betraktning i valg av teknologier.

3.2.1 Frontend

3.2.1.1 React med TypeScript

Det ble besluttet å utvikle brukergrensesnittet med JavaScript-biblioteket *React*, men med programmeringsspråket *TypeScript*. React blir i dag regnet som et av de mest populære frontend-bibliotekene og har dermed en stor brukerbase. Som resultat av dette finnes det flere tredjepartsbiblioteker og støtte, som bidrar til enkle løsninger og god tilgang på informasjon. Fra tidligere prosjekt har samtlige teammedlemmer kjennskap til *React* og har gode erfaringer rundt biblioteket. React baserer seg på prinsippet *single-page application* som bidrar til en dynamisk webapplikasjon. Dette ble også tatt i betraktning i valget.

Det finnes også andre gode alternativ til slike biblioteker, som for eksempel *Vue.js* og *Angular*. Alternativene er teknologisk relativt like, da flere tilbyr mange av de samme funksjonalitetene. Avgjørelsen ble derfor tatt med tanke på teamets erfaring, samt den store støtten rundt biblioteket.

I tillegg til *React* ble som nevnt *TypeScript* benyttet, et programmeringsspråk som bygger på *JavaScript*, men med ekstra syntaks for typesjekking. Ved bruk av dette

språket kan utviklere jobbe tettere mot utviklingsmiljøet, samt utvikle raskere og tryggere. Med *TypeScript* får du støtte under utvikling for hvor du skal bruke de forskjellige typene, noe som kan luke bort feil som ellers kan være vanskelig å oppdage.

3.2.1.2 Material UI

Material UI er et omfattende komponentbibliotek for *React* som bidrar til en raskere utvikling og et konsekvent design av webapplikasjoner. Komponentene er bygget opp av eksisterende komponenter fra *React* som er videreutviklet og inkluderer mer omfattende funksjonaliteter. Disse er også utviklet med hensyn til universell utforming og WCAG. Komponentbiblioteket er stadig under utvikling og blir vedlikeholdt, noe som fører til at utviklere kan bruke komponenter som er kvalitetssikret og testet. For å utnytte dette i størst mulig grad, ble det besluttet å bruke dette komponentbiblioteket.

Likesom for *React* finnes det også andre gode alternativer til *Material UI*, som for eksempel det mye brukte biblioteket *React Bootstrap*. Teknologiene her bygger på mange av de samme verdiene, men ofte er det designet på komponentene som skiller dem. Siden teamet hadde gode erfaringer, samt tilgang til god støtte og dokumentasjon for biblioteket, ble det besluttet at *Material UI* skulle brukes.

3.2.2 Backend

3.2.2.1 .NET med C#

Fra oppdragsgiver var valget av rammeverk og kodespråk opp til teamet. Ettersom det skal bygges videre på, er det derimot gunstig at oppdragsgiver er kjent med teknologien. Valget falt etter diskusjon med oppdragsgiver på *.NET 6* som var den nyeste tilgjengelige versjonen av rammeverket med langsiktig støtte. Det var viktig for teamet at rammeverket skal kunne kjøres på alle de konvensjonelle plattformene, og at det er lite krevende å finne god dokumentasjon.

3.2.2.2 Dapper

Mesteparten av oppgavene til API-et er å legge til, endre, slette eller hente data fra en relasjonsdatabase. For å utføre disse operasjonen har vi brukt *Dapper* som er en micro-ORM. Dapper gjør det effektivt å skrive SQL opp mot en relasjonsdatabase. Alternativet er å bruke en ORM som gjør databaseoperasjonene med kode. I stedet for å lære seg syntaksen til en ORM, har teamet valgt å skrive SQL på grunn av tidligere erfaring. I følge Statista er SQL en av de vanligste språkene utviklere brukte i 2021. Det vil si at flere vil kunne forstå hva koden gjør, i forhold til om ORM hadde blitt brukt.

3.2.3 System

I denne seksjonen blir andre teknologier relatert til systemet presentert. Dette innebærer blant annet ulike rammeverk for testing, kodedokumentasjon og containerteknologier.

3.2.3.1 Azure Active Directory

Azure Active Directory er en identitesleverandør som administrer identiteter og brukere. Atea er en organisasjon med flere interne ressurser. For å gi en ansatt tilgang til de interne ressursene er det effektivt at den får tilgang til alt samtidig. Den ansatte har en konto som er tilknyttet katalogen til Atea i Azure Active Directory. Vedkommende bruker kontoen når man trenger tilgang til de interne ressursene, med mindre en global administrator har bestemt noe annet. Det samme gjelder når en ansatt slutter i jobben, og ikke lengre skal ha tilgang til ressursene. I stedet for å måtte fjerne brukeren fra alle interne tjenester, kan man fjerne den en plass. Dette kalles *Single Sign On*, og er mulig på grunn av OAuth 2.0. Innloggingen skjer med en tredjepart, slik at den samme innloggingen gjelder for ulike ressurser. Teamet har ønsket å gjøre applikasjonen enkelt tilgjengelig for de ansatte, og har derfor tatt i bruk AAD. Alternativet er at man må lagre brukere i databasen til applikasjonen. Det krever at brukere må huske på påloggingsinformasjonen sin til APC. For utviklerne sin del krever det blant annet at passord må lagres sikkert i databasen, og sørge for metoder for gjenoppretting av passord.

3.2.3.2 Docker og Kubernetes

Docker og *Kubernetes* er to teknologier som ofte brukes sammen. *Docker* er en tjeneste som hjelper til å pakke programvare i containere. Ettersom containere fungerer godt i forskjellige miljøer brukes de ofte når en applikasjon skal distribueres. *Kubernetes* er et system som gjør det lettere å distribuere, vedlikeholde og skalere applikasjoner. I Kubernetes har man mulighet til å orkestrere hvordan containere skal oppføre seg. Hvilken maskin de skal kjøre på, hvor mange kopier en container skal ha, og hva som skal skje om en container krasjer. Vi har valgt teknologien på grunn av at utviklerne i Atea er kjent med den. Samtidig er det viktig at teknologien er pålitelig, slik at ikke-tekniske ansatte enkelt kan teste applikasjonen.

3.2.3.3 Ingress NGINX Controller

For at AAD skulle fungere i den distribuerte versjonen av applikasjonen måtte nettsiden beskyttes med et SSL-sertifikat. I stedet for å beskytte backend og frontend hver for seg, brukes en Ingress NGINX Controller. Når klienter går inn på nettsiden sender de krypterte forespørsler til Ingress Controlleren. Den sender forespørslene videre til backend eller frontend. Samtidig er Ingress Controlleren med på å gjøre appen mer skalerbar. Den kan håndtere flere like servere av henholdsvis frontend og backend, og gjør applikasjonen klar for å ta imot flere klienter. Årsaken til at NGINX blir brukt er god dokumentasjon, og lett tilgjengelighet gjennom Docker.

3.2.4 Testing

I prosjektet ble det laget tester separat for både frontend- og backend kildekode. Testing av kodespråkene i begge prosjektdelene var nytt for teammedlemmene og innebar derfor utforskning av teknologier for å finne de beste løsningene.

For frontend falt valget på *Cypress*, et testverktøy for alt som kjører på nettlesere og inkluderer ende til ende, integrasjon og enhetstester. Testene i *Cypress* blir skrevet med *JavaScript*, som er et kjent språk for teamet og derfor en tidsbesparende faktor for prosjektet. Testverktøyet har et enkelt og oversiktlig brukergrensesnitt med tilknyttet dokumentasjon for alle funksjonalitetene. I tillegg er *Cypress* kompatibelt med *React* og *TypeScript*.

Fordi kildekoden i API-et ikke inneholder noen avansert logikk har teamet vurdert at det er tilstrekkelig med integrasjonstester. Det er opprettet en egen database til testing i skyen, som speiler databasen som brukes i produksjon. At testmiljøet er mest mulig likt produksjonsmiljøet gir testene en større kredibilitet. Testene er laget slik at data postes til databasen. Deretter sjekkes det, ved hjelp av andre endepunkt, om dataen er lagret. Samtidig blir statuskoden til responsen sjekket opp mot forventet respons.

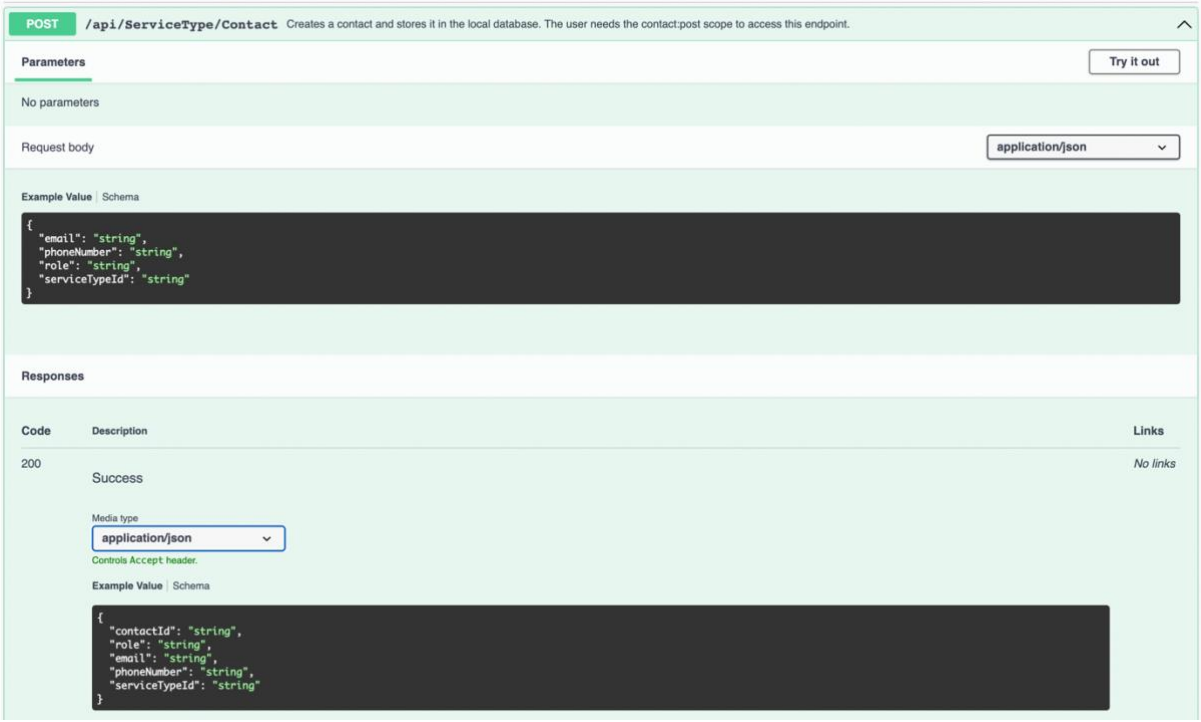
3.2.5 Kodedokumentasjon

Dokumentasjon av de brukte kodespråkene var nytt for teammedlemmene og førte til utforskning av alternativer. Siden sluttproduktet skal leveres til en bedrift som ønsker å kunne bygge videre på prosjektet, var målet å ta i bruk dokumentasjonsverktøy som oversiktlig presenterer kodens funksjon. Tidsrammen for prosjektet var også tatt i betraktning. Derfor var ikke målet en dokumentasjon med detaljer for alle elementer, men en mer overordnet stil.

For frontend finnes det flere alternativer av dokumentasjonsverktøy. Mer komplekse verktøy med visualiseringer av komponenter, eller verktøy med en mindre detaljert dokumentasjon som *JSDoc*. Med verktøyet *JSDoc* kan utviklere enkelt legge til kommentarer for funksjoner og komponenter. Ut ifra disse kommentarene blir det generert en enkel HTML fil som består av en klikkbar liste med dokumentasjon. Dette verktøyet passet dermed med våre forutsetninger og ble tatt i bruk for dokumentering av frontend-koden.

I dokumentasjonen av backend har teamet brukt verktøyet Swagger. Når man starter opp API-et lokalt blir det generert en side som beskriver alle endepunktene med en kort kommentar. Endepunktene er klikkbare, og om man trykker på det vil det se ut som på figur 7. Her ser man hva som er forventet som *body* når man sender en forespørsel, og hva man skal forvente i respons om man får statuskode 200 (OK). Lengre ned på siden vil man finne en oversikt over alle modellene som er brukt. Her kan man finne informasjon om alle feltene en modell har. Hvilken datatype feltet representerer og en

kort beskrivelse. Dokumentasjonen av både frontend og backend er beskrevet nøyere i Vedlegg C Systemdokumentasjon kapittel 9.



Figur 7: Eksempel på Swaggerdokumentasjon for å legge til en ny kontaktperson

3.3 Arbeids- og rollefordeling

Under prosjektperioden har det vært et stort fokus på å holde et tett samarbeid i teamet. Teamet har i all hovedsak vært samlet, fysisk eller over nett. Slik har det blitt sikret at ingen sitter med ett problem alene over en lengre periode. Ved å følge de smidige utviklingsmetodikkene ble det derfor ofte gjennomført parprogrammering for å løse slike problem. Videre har viktige valg blitt diskutert og tatt i plenum. I tillegg har samtlige blitt oppdatert om hva slags arbeid de andre utfører, og hvordan man ligger an.

Når roller skulle fordeles tok teamet en vurdering ut fra estimert arbeidsmengde. Teamet antok at den største andelen arbeid ville foregå på frontend og valgte dermed å plassere to av teamets tre medlemmer til å fokusere hovedsakelig på dette. Dette førte til en god flyt mellom frontend og backend. Til tross for de definerte rollene var samtlige av teamets tre medlemmer villige til å utføre arbeid hvor enn det var nødvendig. Utenfor utviklingen ble det også definert roller i sammenheng med møter og dokumenter. Disse rollene er spesifisert i Vedlegg D Prosjekthåndbok under arbeidskontrakten.

4 Resultater

Under dette kapittelet vil det bli presentert de resultater og observasjoner som er kommet frem i løpet av prosjektperioden. Disse er delt inn i produktrelaterte, administrative og vitenskapelige resultater.

4.1 Produktrelaterte resultater

Prosjektet omhandler i stor grad utviklingen av et produkt, og det er i den sammenheng aktuelt å presentere de produktrelaterte resultatene først. Disse tar utgangspunkt i de funksjonelle og ikke-funksjonelle kravene som er nøyere beskrevet i Vedlegg A Visjonsdokument, samt oppgavebeskrivelsen og problemstillingen. Videre vil det bli presentert illustrasjoner av sluttproduktet.

4.1.1 Funksjonelle krav

De funksjonelle kravene var i stor grad beskrevet av oppdragsgiver gjennom flere møter i oppstartsperioden. Teamet har derimot også hatt frihet til å legge til krav der det har vært hensiktsmessig. Et eksempel på dette er å velge lyst og mørkt tema. Teamet ønsket å ha med dette som et krav, slik at nettsiden kan være mer fleksibel i bruk.

4.1.1.1 Frontend

I denne seksjonen beskrives de funksjonelle egenskapene som er fremkommet under utviklingen av frontend på prosjektet. Disse består av funksjoner man kan fysisk gjøre eller se på nettsiden. I tabell 1 blir de funksjonelle kravene for frontend fra Vedlegg A Visjonsdokument presentert, samt hvorvidt de er oppfylt eller ikke.

Tabell 1: Funksjonelle krav for frontend.

Funksjonelle krav	Oppfylt	Ikke oppfylt	Kommentar
Logge inn med bruker som er registrert ved Atea	x		Laget egen innloggingsside med knapp som sender brukeren videre til nettsiden. Om brukeren ikke er registrert i Atea vil den ikke ha tilgang til siden.
Velge lyst og mørkt tema	x		Gjøres enten ved valg i brukerens enhetsinnstillinger eller ved hjelp av knapp i navigasjonsmenyen.
Navigere mellom de ulike sidene	x		Laget sidemeny med knapper som navigerer brukeren til ønsket side.

Søke etter en side, tjeneste eller kunde i navigasjonsbaren	x		Søkefelt i navigasjonsmenyen som gir brukeren mulighet til å søke på alle kunder, sider og tjenester.
Hente tjenester og tilhørende varelinjer	x		Blir presentert som en liste med tjenester og underliggende liste med varelinjer
Gi en varelinje ulike nivå (basic, plus, premium)		x	Kravet hadde lav prioritet, og etter diskusjoner med oppgavestiller ble det enighet om å forkaste det.
Opprette varelinje for en tjeneste	x		Knapp i listen over tjenester. Ved klikk på knappen sendes brukeren til et skjema med innskrivningsfelter
Redigere varelinjer for en tjeneste	x		Knapp i listen over varelinjer for en rask tilgang til redigering eller knapp på varelinjens informasjonsside
Slette varelinjer for en tjeneste	x		Knapp på informasjonssiden for varelinjen
Se en revisjon av tidligere endringer på en varelinje	x		Tabell med tidligere endringer på siden for varelinjen
Se kontaktinformasjon for en tjenesteeier	x		Liste med telefonnummer og eposter til kontaktpersoner på varelinjesiden. Eposter er klikkbare og sender brukeren til sin lokale eposttjeneste.
Hente spesialtilbud for en kunde	x		Hentes ut av bruker ved hjelp av en liste med spesialtilbud tilknyttet kunden
Redigere spesialtilbud for en kunde	x		Redigeres ved å trykke på redigeringsikon.
Slette spesialtilbud for en kunde	x		Slettes ved å trykke på knapp med søppelikon.
Tilganger skal være definert av brukerens rolle	x		Implementert ved at knapper blir deaktivert og bruker blir omdirigert hvis den ikke har riktig tilgang.
Administrator skal kunne endre roller for en bruker	x		Gjøres ved et skjema med roller som kan aktiveres eller deaktiveres for hver enkelt bruker.
Hente rapporter om en kunde eller tjeneste basert på en gitt tidsperiode	x		Rapportene viser relevant informasjon på valgte spørringer. Bruker kan velge mellom å hente for en gitt tjenestetype, en gitt

			kunde, eller en kombinasjon av disse.

4.1.1.2 Backend

I denne seksjonen beskrives de funksjonelle egenskapene som er kommet frem under utviklingen av backend. Disse består av handlinger som API-et utfører. I tabell 2 blir de funksjonelle kravene for backend fra Vedlegg A Visjonsdokument presentert, samt hvorvidt de er oppfylt eller ikke.

Tabell 2: Funksjonelle krav for backend.

Funksjonelle krav	**Oppfylt**	**Ikke oppfylt**	**Kommentar**
Hente informasjon om tjenester fra ekstern database	x		Hentet ut data fra Atea sitt data-API og fra lokal database. Mappet sammen dataen i kildekoden.
Opprette varelinjer for en tjeneste	x		
Redigere varelinjer for en tjeneste	x		
Slette varelinjer for en tjeneste	x		
Koble sammen med faktureringsmotor for å alltid ha oppdaterte og korrekte varelinjer	x		Faktureringsmotoren blir oppdatert når endepunktene blir kalt. For eksempel sørger endepunktet for å opprette en varelinje for å gjøre det i både i faktureringsmotor og i lokal database.
Hente rapporter fra faktureringsmotor	x		Henter ut rapporter om salg direkte fra faktureringsmotor. Serverer den dataen som behøves frontend i eget endepunkt
Oppdatere APC fra faktureringsmotor med jevne mellomrom		x	Sørge for at APC er oppdatert hvis noen oppdaterer priser i Exivity
Opprette spesialtilbud for en kunde	x		

Redigere spesialtilbud for en kunde	x		
Slette spesialtilbud for en kunde	x		
Redigere brukertilgang	x		

4.1.2 Øvrige implementasjoner

I tillegg til de kravene som ble satt i visjonsdokumentet har teamet implementert andre funksjonaliteter som ikke var en del av de opprinnelige kravene. Disse er presentert og beskrevet i tabell 3.

Tabell 3: Øvrige implementasjoner.

Funksjonalitet	**Beskrivelse**
Gjenopprette varelinje	Sette tilbake en varelinje til tidligere versjon basert på revisjonshistorie.
Søke i alle lister	Når en bruker blir presentert med en liste kan den søke etter det spesifikke objektet den leter etter.
Filtrering og sortering av revisjonshistorie	Når en bruker ser gjennom revisjonshistorien til en tjeneste, kan den filtrere basert på dato eller sortere etter dato.
Avslag for kunde	En bruker kan sette en prosentverdi i avslag for en varelinje slik at ny pris blir automatisk regnet ut.
Legge til og lagre et anbud	En bruker kan legge til et anbud med bestemte tjenestetyper og varelinjer. Brukeren kan også endre pris for varelinjene i anbudet.
Redigere et anbud	En bruker kan redigere et anbud.
Slette et anbud	En bruker kan slette et anbud
Generere PDF-fil med et anbud	En bruker kan generere en PDF-fil med et anbud som kan presenteres til en eventuell ny kunde.

4.1.3 Ikke-funksjonelle krav

I oppstartsfasen av prosjektet ble det også definert ikke-funksjonelle krav som systemet skulle oppfylle. Siden brukervennlighet og dynamikken av nettsiden var ønskelig fra oppdragsgiver ble dette hovedfokuset. Teamet satte også sine egne krav og egenskaper som skulle bidra til en sikker nettside som skulle kunne videreutvikles. Etter hvert ble

det ønskelig, både fra oppdragsgiver og utviklere, at applikasjonen skulle distribueres. Det ble dermed lagt til som et krav i Vedlegg A Visjonsdokument. Slik ble kravene implementert:

- **Sikkerhet**: Sikkerheten består av autorisasjon og autentisering. Autentiseringen ble løst ved hjelp av AAD. Autentiseringen består av to prosesser. Først må klienter logge seg inn på klientapplikasjonen og deretter hente et token til å bruke mot API-et. For at det skal fungere må appene være registrert i AAD, og klienten må sende forespørsler direkte mot identitetsleverandøren.
  Teamet har lagt til klientapplikasjonen i app registreringen til API-et. Dette har blitt gjort for at API-et skal verifisere tokenet det får fra klienten.

  Hvilke funksjonaliteter som skulle være tilgjengelig for hvem er lagret i APC sin database. Når en bruker logger inn for første gang, blir brukeren lagret i databasen. Da har brukeren ikke mulighet til å benytte seg av de funksjonaliteter som krever spesifisert tilgang. De brukerne som er lagret i databasen som administratorer, kan søke opp brukere for å gi eller fjerne tilganger.

- **Brukervennlighet**: Gjennom utviklingen av nettsiden jobbet utviklerne tett opp mot tre-klikk-regelen. Ved hjelp av en oversiktlig navigasjonsmeny, tydelige lister med søkefunksjoner og et overordnet søkefelt for hele nettsiden, kan brukeren raskt komme frem til ønsket informasjon. Knapper og innskrivingsfelt er store og tydelige, og har tilknyttede forklaringer til knappens funksjon eller hva feltene skal fylles ut med. Ved handlinger som fører til endring av data, må brukeren bekrefte endringen. I tillegg gir nettsiden en visualisering av at endringene blir utført gjennom en lasteskjerm. Videre blir tydelige tilbakemeldinger med fargekoder på hvorvidt handlingen var vellykket eller mislykket presentert for brukeren. I tillegg til tekstlige forklaringer blir ikoner benyttet for å visualisere handlinger eller informasjon på nettsiden.

  For å beholde konsistensen på nettsiden har teamet holdt seg til faste løsninger for komponenter. Knapper for redigering som er visualisert med en blyant og opprettelse av nye objekt med et pluss tegn, er to eksempler av faste løsninger. På samme måte brukes farger aktivt for å representere ulike handlinger. For eksempel er fargen grønn en indikasjon på at noe skal legges til, fortsettes på eller at noe er vellykket. Fargen rød derimot, blir brukt i sletting, å angre eller for å presentere feilmeldinger.

  Under utvikling er som nevnt komponentbiblioteket *Material UI* brukt. Ved å bruke et slikt komponentbibliotek sikrer man å bygge et konsistent grensesnitt. Biblioteket har blitt brukt av teamet som et hjelpemiddel til å bygge opp egne komponenter. I tillegg er brukes det for å sikre at ikoner og knapper har en konsistent stil. Videre tilbyr *Material UI* en måte å definere fargetemaer, som er benyttet for å gi brukeren mulighet til å velge mellom lyst og mørkt tema.

- **Dynamisk API**: Prosjektet besto av å utvikle et REST API som skulle jobbe mot både eksternt API hos Atea og en ekstern faktureringsmotor. Det ble valgt å samle alle kall til og mot eksterne tjenester i REST API-et. Klienten kommuniser derfor kun med REST API-et. Det er også valgt å kun ha ett kall til API-et for én

handling, slik at man unngår ett eget kall for oppdatering av database og ett eget kall for oppdatering av faktureringsmotor.

- **Legge til rette for videreutvikling**: Siden prosjektet skal avleveres var det ønskelig for utviklerne å legge opp til videreutvikling av applikasjonen. Dette har ført til forskjellige valg under oppbygging av prosjektet i sin helhet. For frontend er koden delt opp i mindre komponenter. Store filer som omfatter mange linjer kode har blitt delt opp og satt i egne filer. I tillegg har det blitt utviklet generelle komponenter og hjelpefunksjoner som kan gjenbrukes under videreutvikling av produktet.

  Backend består av flere prosjekter. Det er eget prosjekt for kommunikasjon med hver av de eksterne tjenestene. Hvert av disse prosjektene er selvstendige, slik at de enkelt kan byttes ut uten større implikasjoner i hovedprosjektets helhet. Prosjektstrukturen for både frontend og backend er nøyere beskrevet i Vedlegg C Systemdokumentasjon kapittel 3.

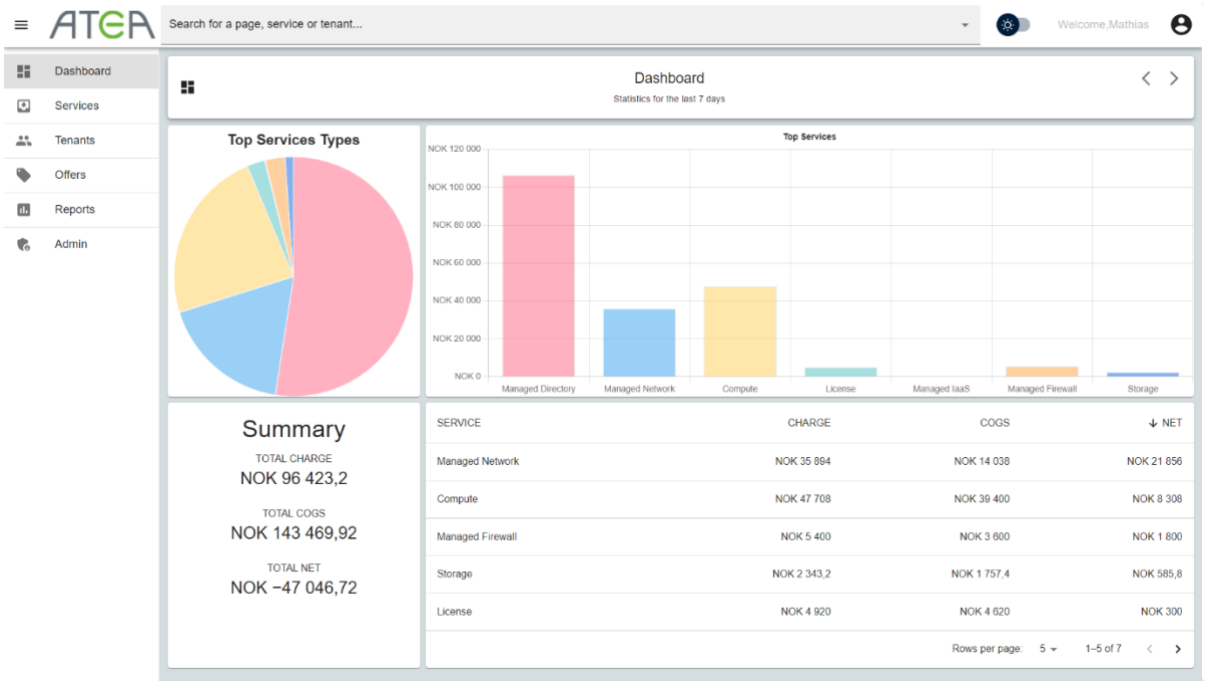
- **Distribuering**: Vi sørget for distribueringen av applikasjonen ved at hver gang hovedbranchen oppdateres, vil kontinuerlig levering starte. Da starter det to prosesser. Den ene bygger frontend i en egen container, mens den andre bygger backend i en egen container. Hvis byggingen fullfører, vil de to containerne bli lagret i et container-register. Kubernetes vil automatisk hente ut de nyeste versjonene som er i container-registeret, og distribuere de på verdensveven.

4.1.4 Illustrasjon av sluttprodukt

I denne seksjonen vil resultatet av det utviklede produktet bli illustrert i form av skjermutklipp. Det vil videre følge en kort forklaring til hver figur over hva som kan ses og gjøres på siden.

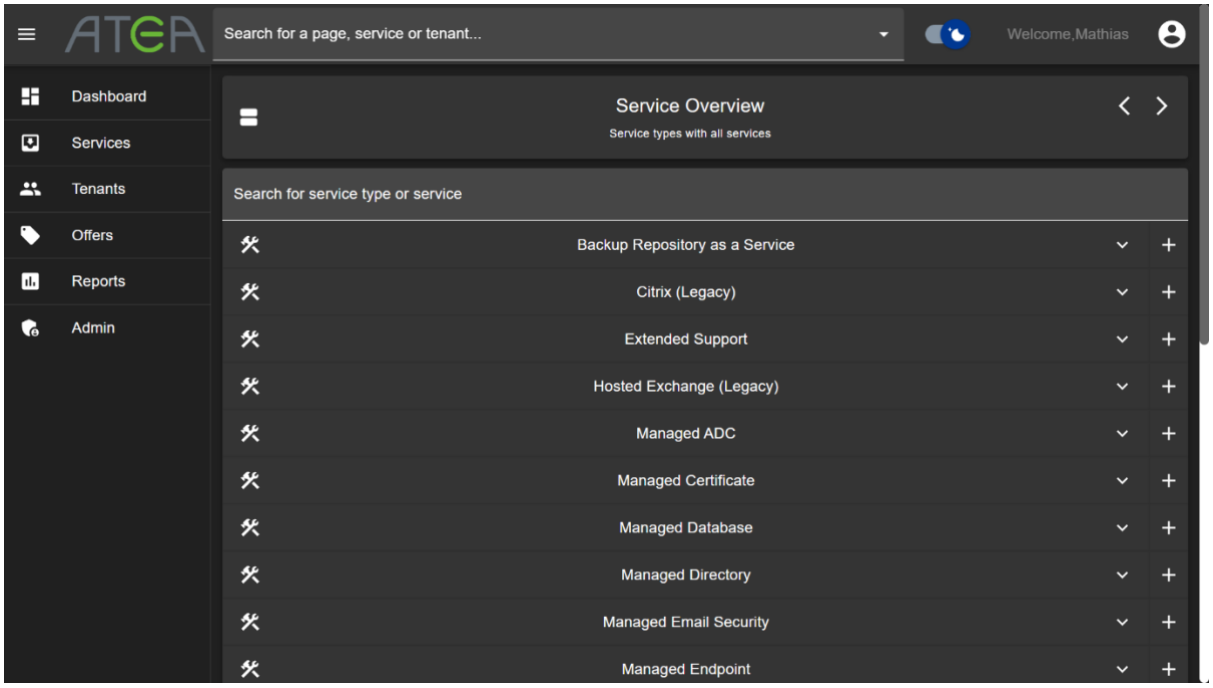
Første siden brukeren blir presentert med etter innlogging er et dashbord, illustrert i figur 8. Her kan brukeren se statistikk over inntekter og kostnader for de ulike tjenester de siste syv dagene. I tillegg er det en navigasjonsbar der brukeren kan søke på ulike sider, tjenester eller kunder som finner sted på alle steder i nettsiden. Der kan man også

velge å åpne eller lukke sidemenyen. Denne kan brukes til å navigere seg frem til ønsket informasjon.

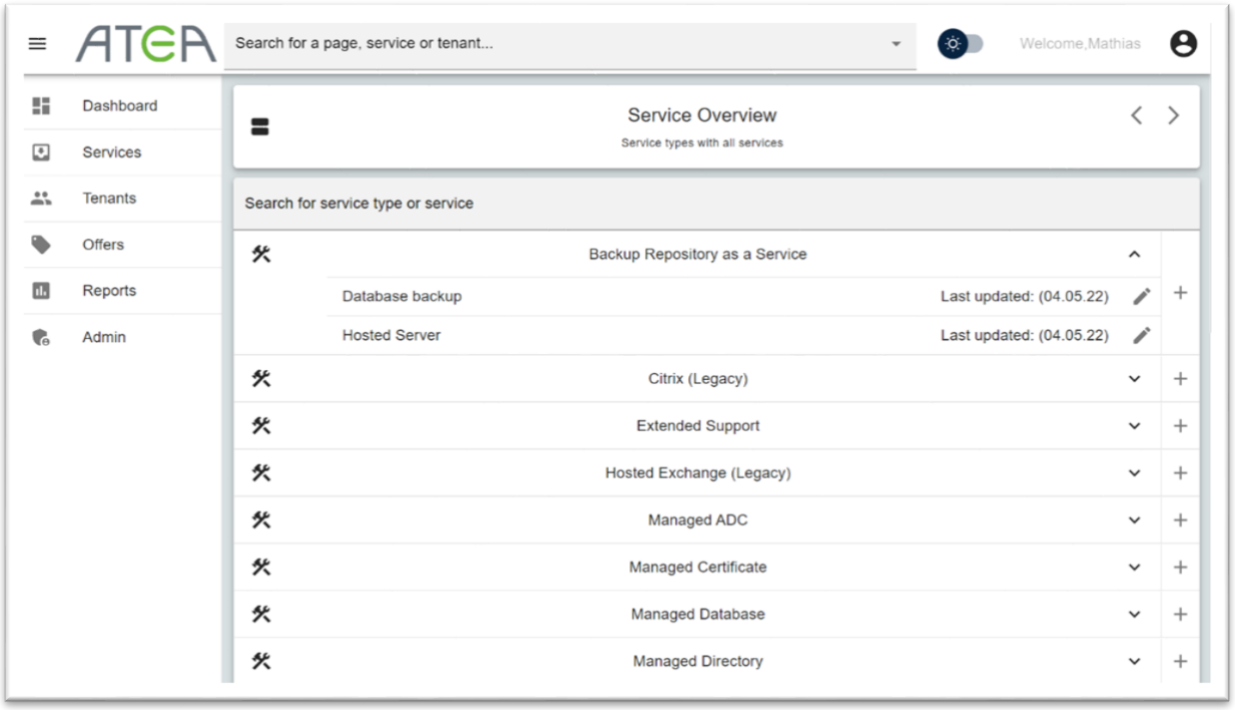


Figur 8: Dashbord.

Ved å trykke på den blå knappen øverst på siden kan brukeren velge mørkt eller lyst tema. Figur 9 illustrerer hvordan tjenesteoversikten ser ut i mørkt tema. Siden viser en oversikt over alle tjenester og gir bruker mulighet til å legge til nye varelinjer ved hjelp av å klikke på plussknappen.



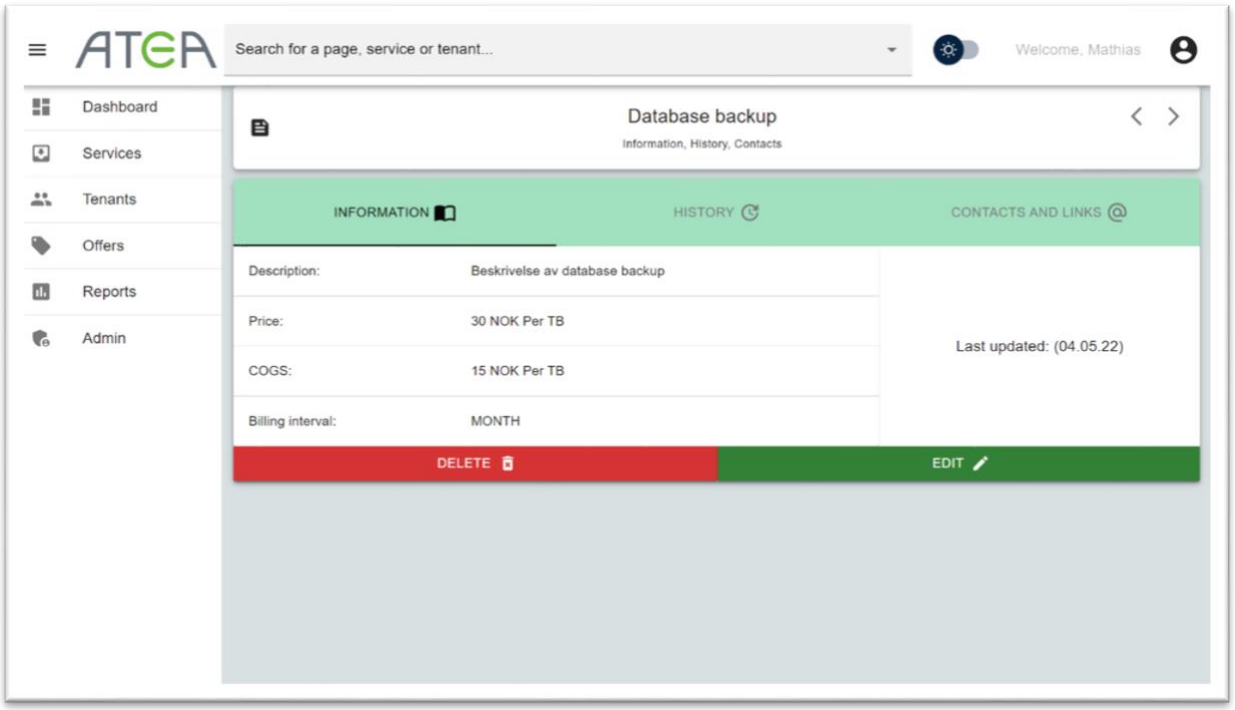
Figur 9: Tjenesteoversikt med mørkt tema.



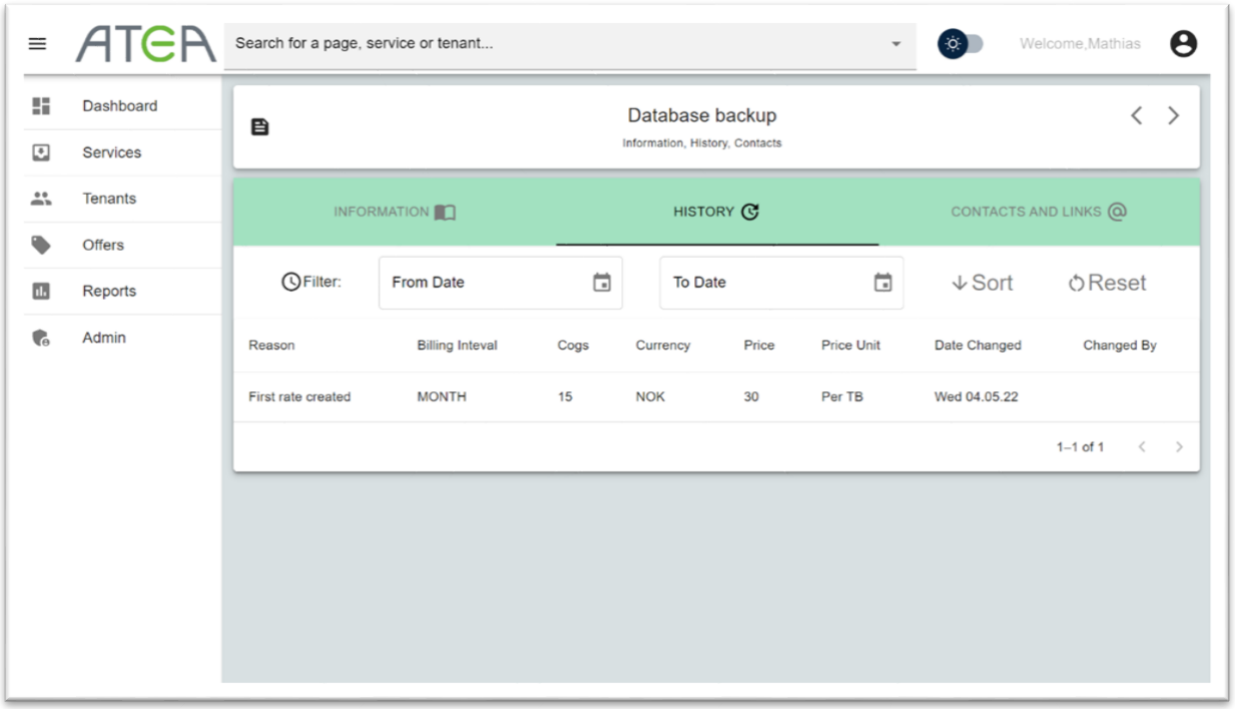
Figur 10: Tjenesteoversikt som viser underliggende varelinjer til en tjeneste.

Når man på tjenesteoversikten trykker på en tjeneste, kan man se de underliggende varelinjene til tjenesten, vist i figur 10. Her kan man trykke inn på en varelinje for å få mer informasjon om den. Ellers kan man velge å redigere varelinjen.

Når brukeren trykker på en varelinje fra tjenesteoversikten, kommer den til informasjonssiden illustrert i figur 11. Her kan brukeren se informasjon om varelinjen, samt velge å redigere eller slette varelinjen.



Figur 11: Informasjonsside for en varelinje.

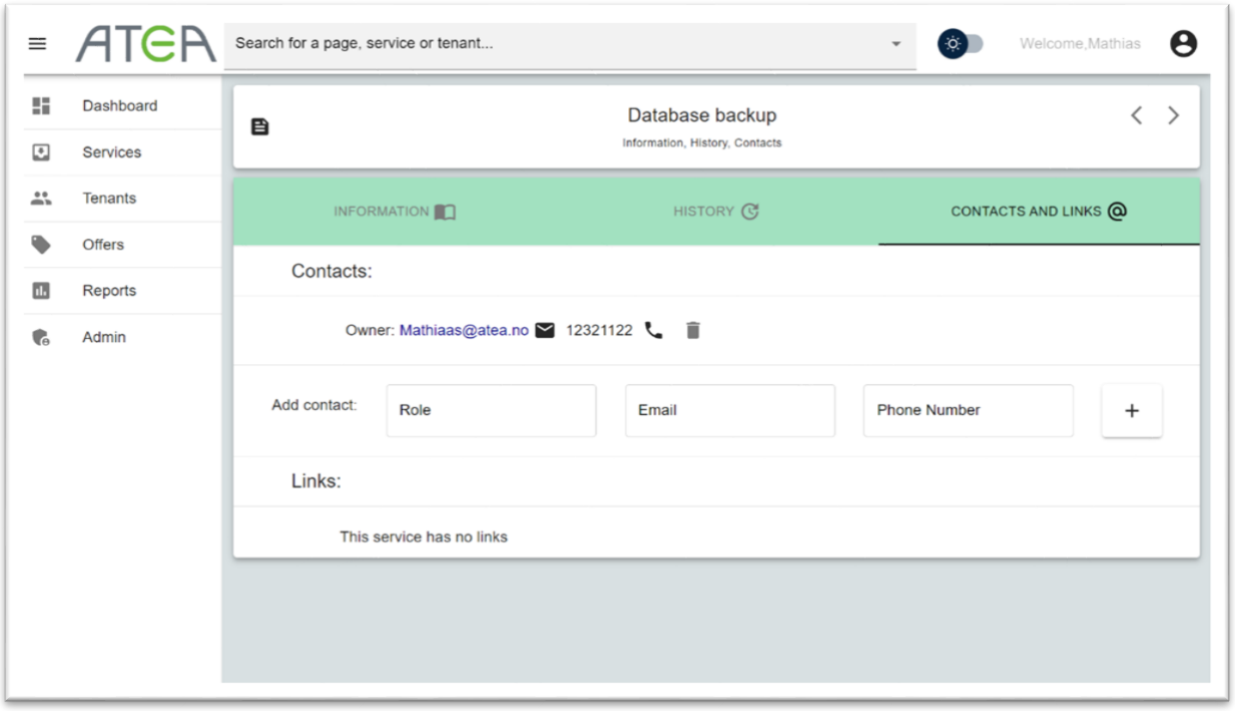


Figur 12: Revisjon på endringer av en varelinje.

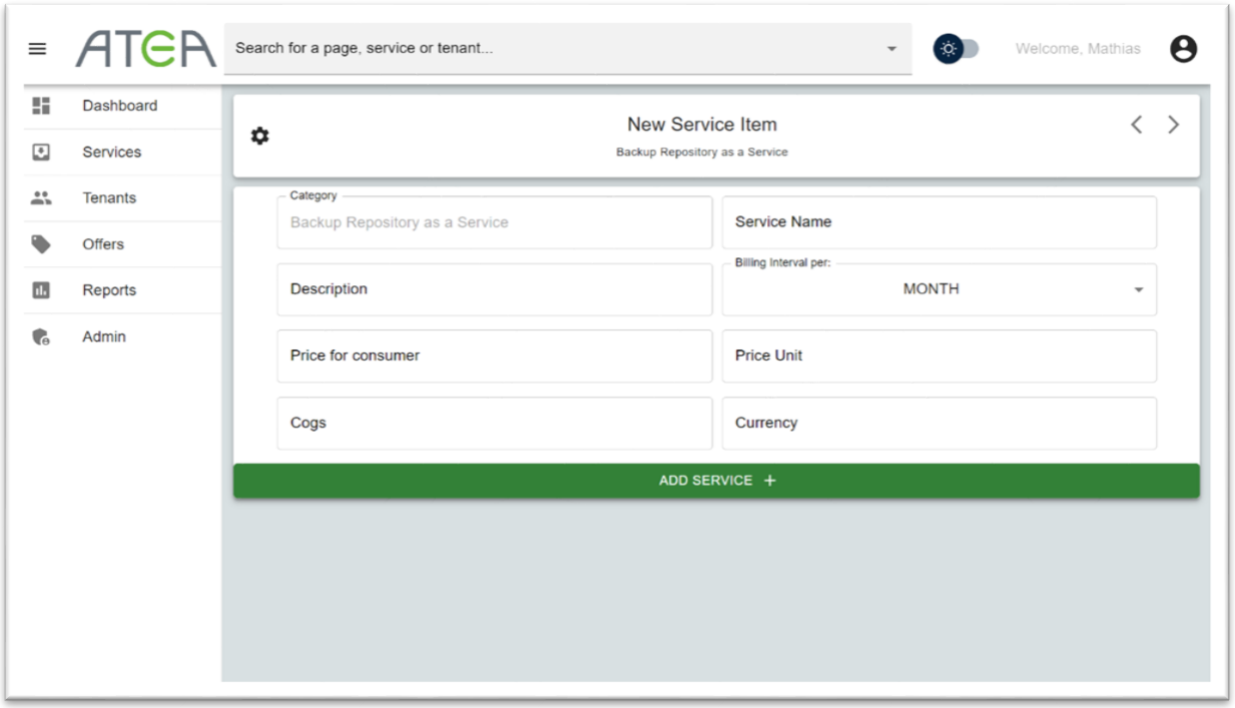
Videre kan man som illustrert i figur 12, se på revisjonshistorien til en varelinje. Denne består av relevant informasjon over hvorfor endringen ble gjort, hva som ble endret og

når endringen fant sted. Her kan brukeren velge å trykke på en tidligere endring for å tilbakestille til den versjonen.

Ved å navigere seg til kontakter og linker, kan man se eventuelle kontaktpersoner og linker tilknyttet tjenesten. Her har brukere med rette tilganger mulighet for å legge til nye kontaktpersonen, samt slette disse. Siden er illustrert i figur 13.



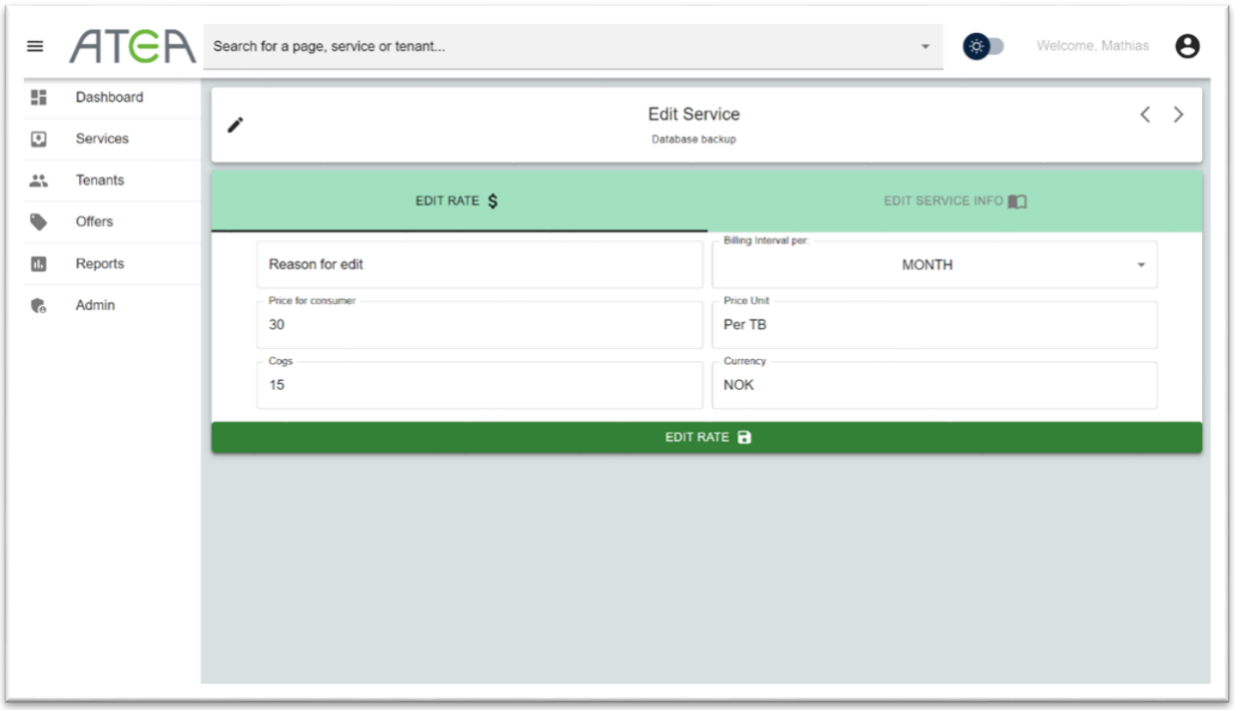
Figur 13: Se kontaktinformasjon og relevante linker for varelinjen.



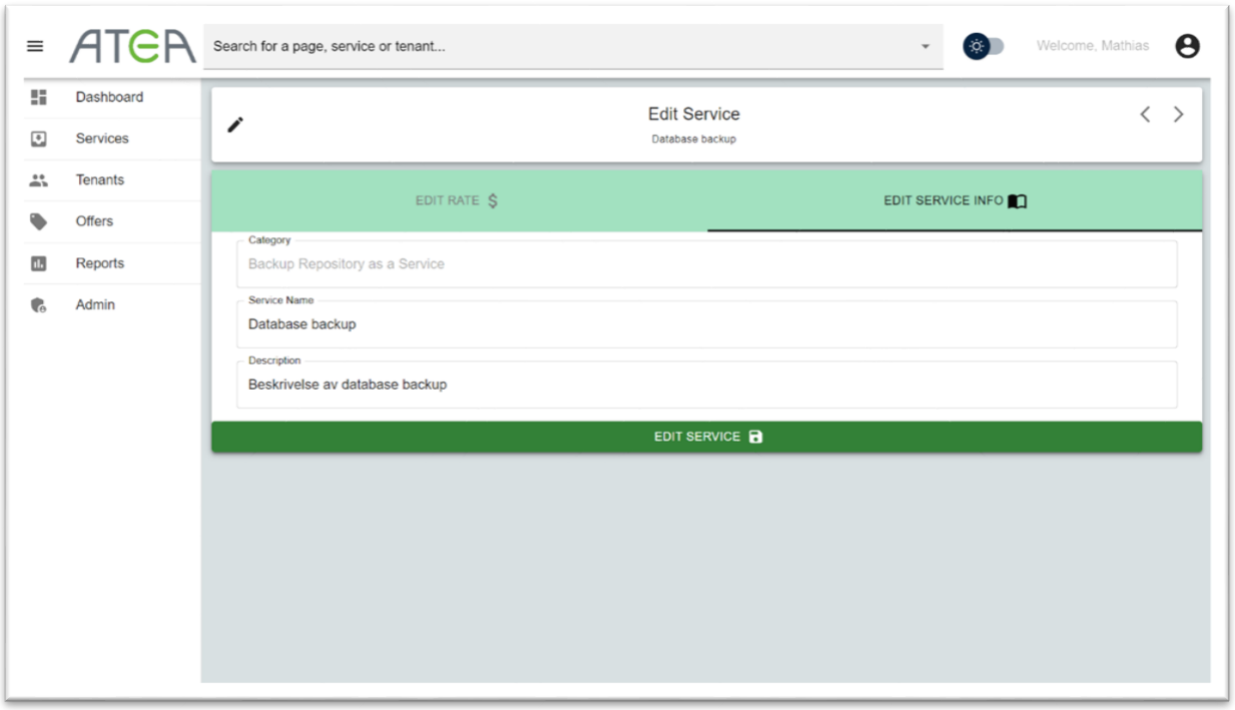
Figur 14: Skjema for oppretting av ny varelinje.

Når en bruker skal legge til en ny varelinje blir den presentert med skjemaet i figur 14. Her kan brukeren spesifisere relevant informasjon om varelinjen, og trykke på legg til knappen.

Skal en bruker redigere varelinjen får den opp et liknende skjema som for å legge til en ny varelinje. I tillegg til å endre verdier for pris og kostnad for bedriften, kan brukeren også legge inn hvorfor varelinjen blir endret. Skjemaet er illustrert i figur 15.



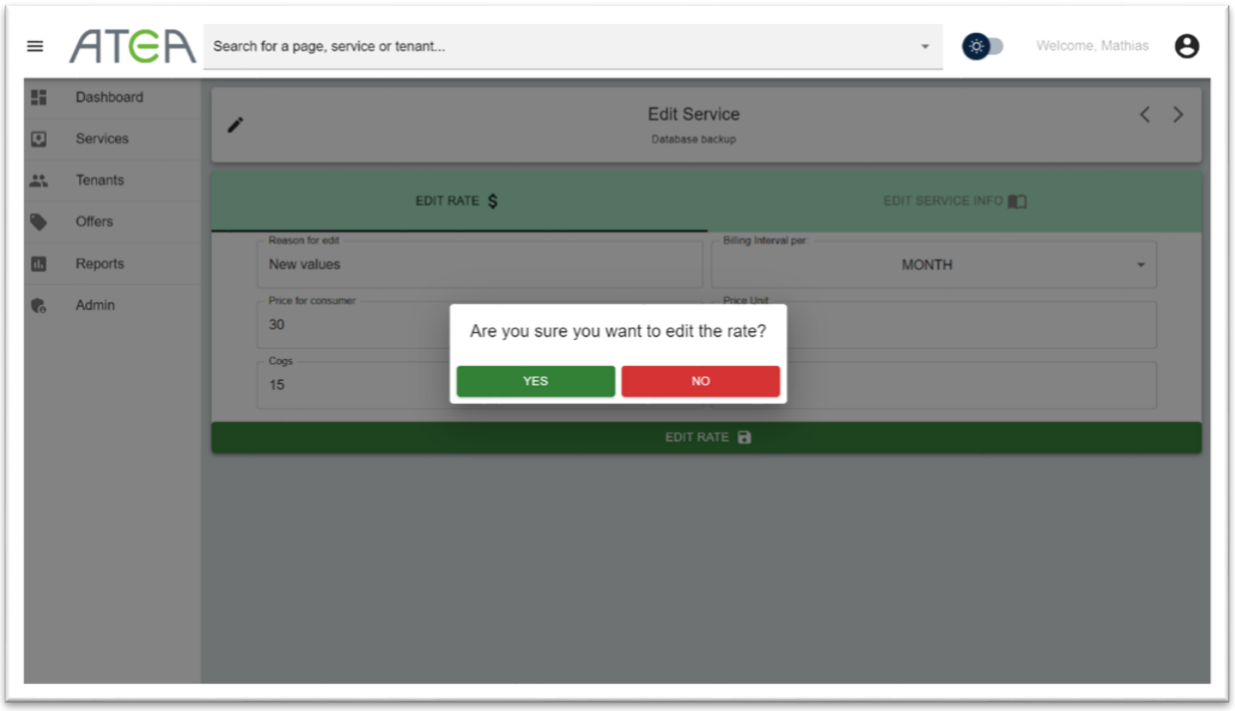
Figur 15: Skjema for redigering av varelinjers verdier.



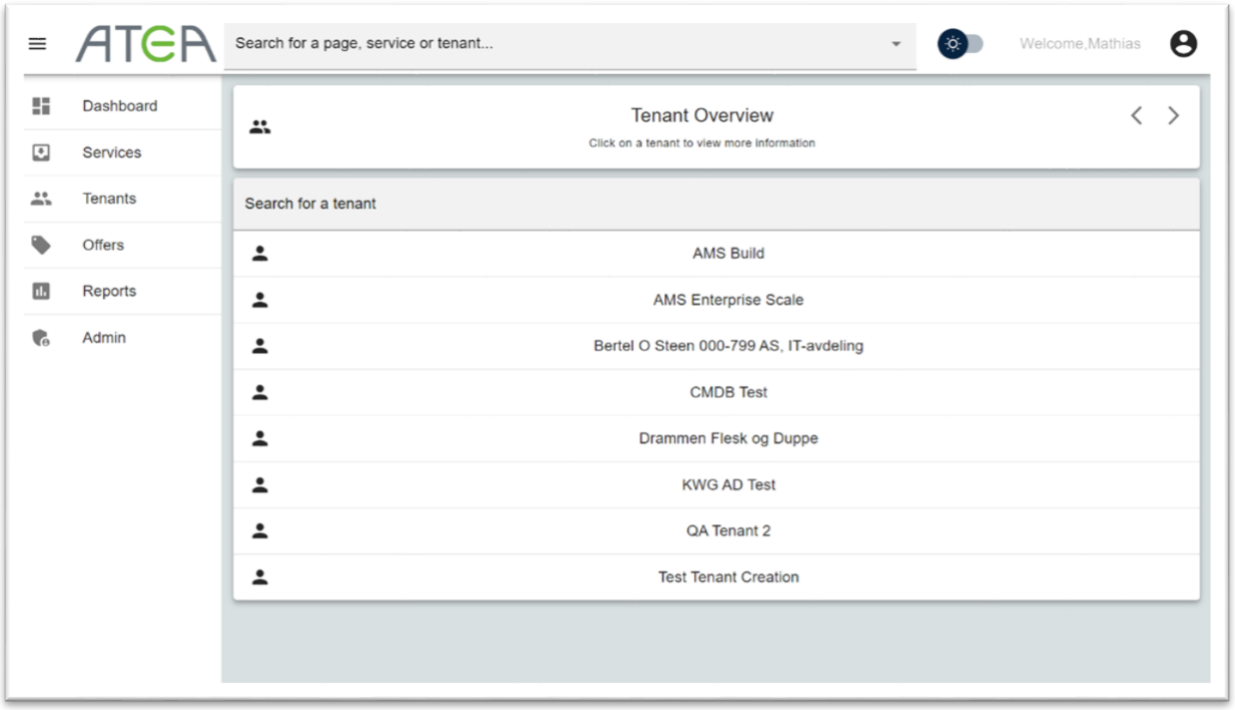
Figur 16: Skjema for redigering av varelinjers informasjon.

Brukeren kan endre navn og beskrivelse på varelinjen med skjemaet i figur 16.

Når en bruker trykker på knapper som innebærer å slette, redigere eller opprette vil et bekreftelsesvindu dukke opp, som vist i figur 17. Her kan bruker velge å fortsette eller avslutte handlingene ved hjelp av å klikke ja eller nei.



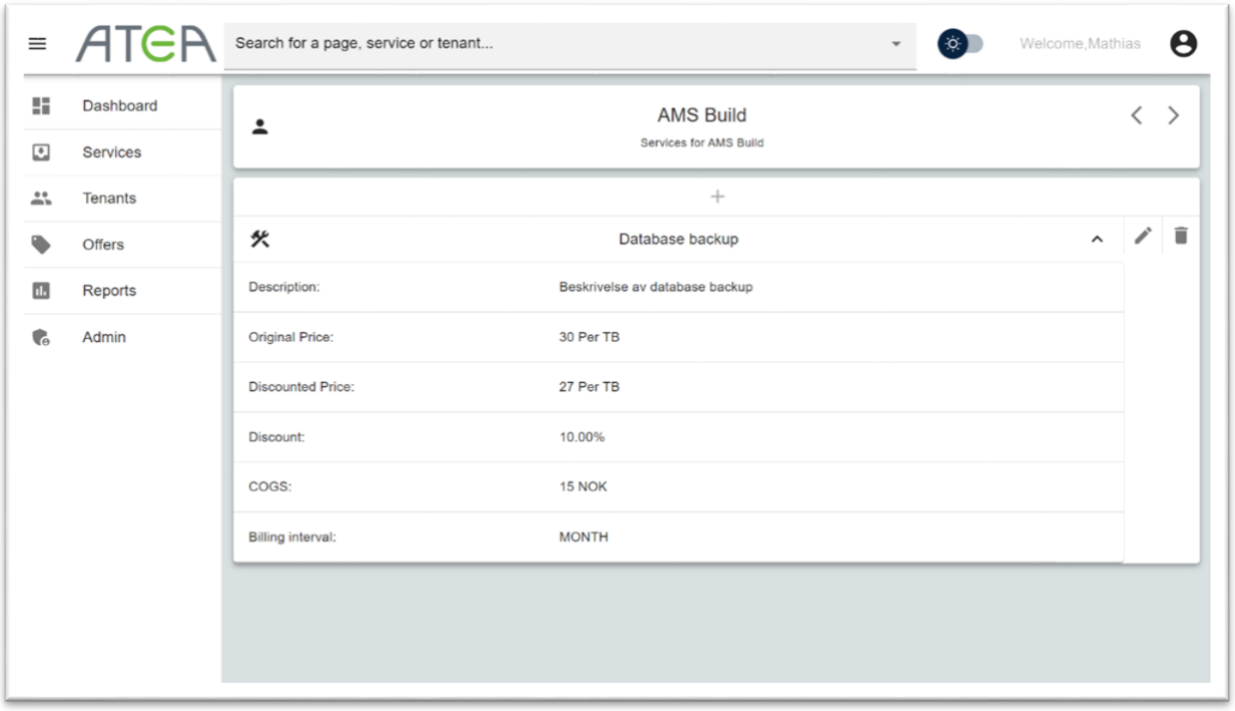
Figur 17: Bekreftelsesvindu.



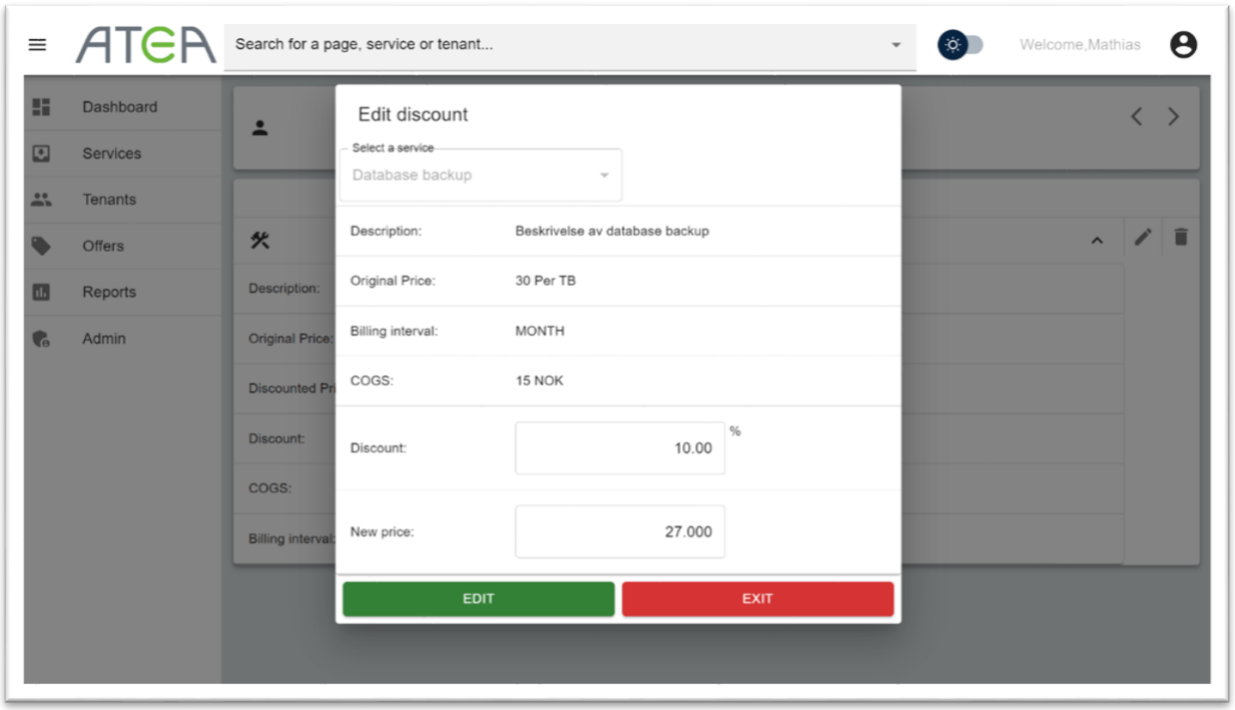
Figur 18: Kundeoversikt.

En bruker kan også navigere seg frem til kundeoversikten. Her blir den presentert med en liste, illustrert i figur 18, som består av et likt oppsett som oversikten for tjenestene.

Trykker brukeren inn på en kunde, kan den se en liste med varelinjer kunden har tilbud på i figur 19. Ved å trykke på varelinjen kan brukeren se informasjon om hvilket avslag kunden har, og annen informasjon om varelinjen.



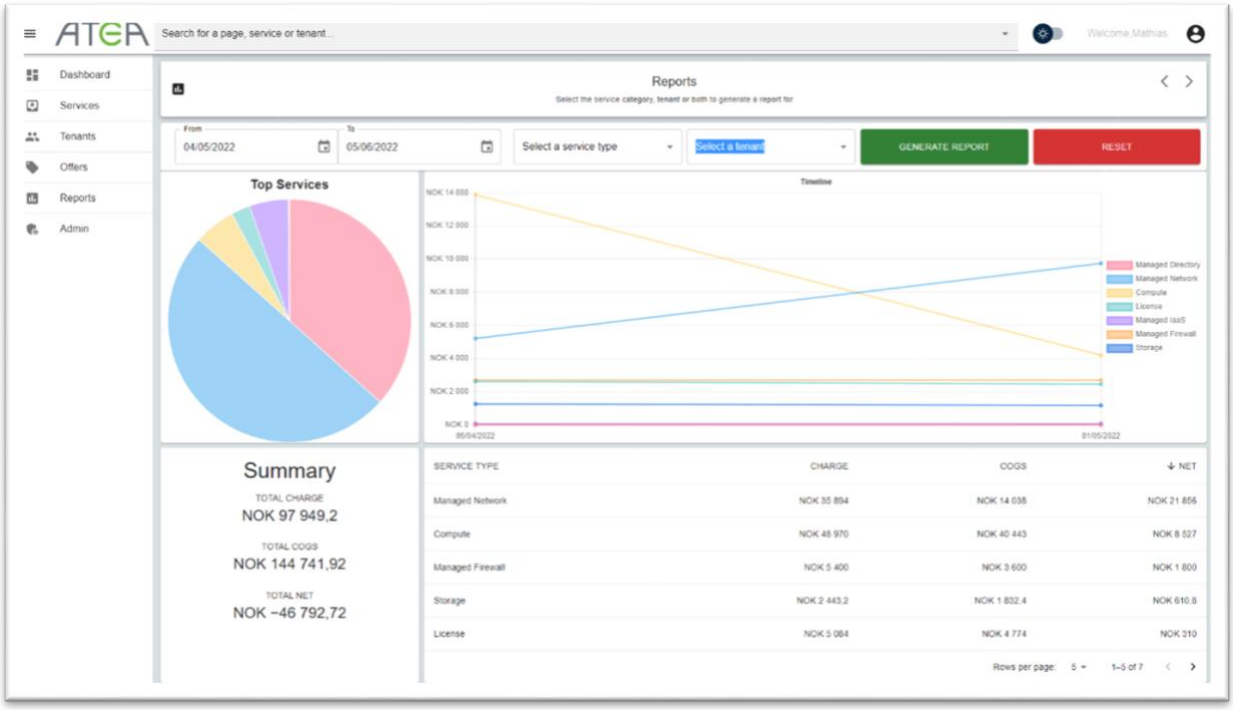
Figur 19: En kunde og tilhørende kunden har.



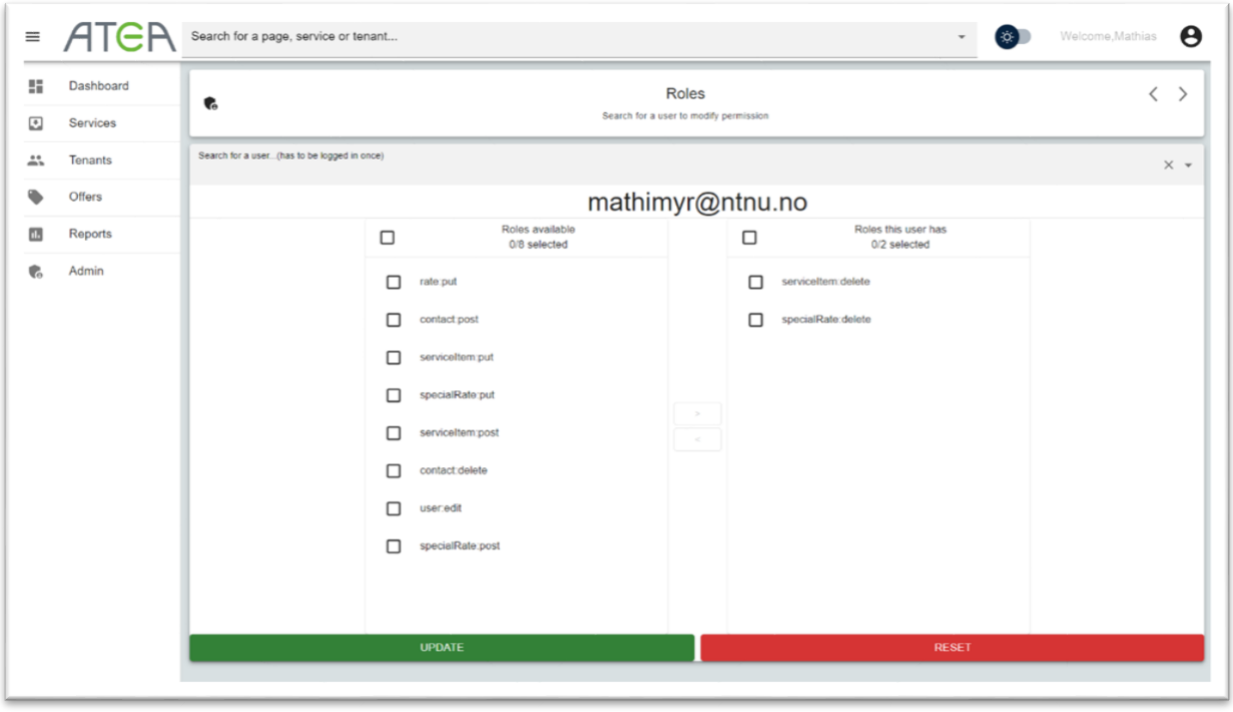
Figur 20: Endre et tilbud en kunde har på en varelinje.

Velger brukeren å redigere kan den ved skjema vist i figur 20. Dette kan gjøres ved å sette en prosent avslag, eller sette en spesifikk pris for varelinjen.

Brukeren har mulighet til å generere rapporter for en gitt kunde eller en gitt tjeneste innenfor et gitt tidsrom. I tillegg kan brukeren velge en kombinasjon av disse. Informasjonen blir presentert i en lik grad som på dashbordet, men som illustrert i figur 21, er det også et linjediagram som viser inntekt over tid for en tjeneste eller varelinje.



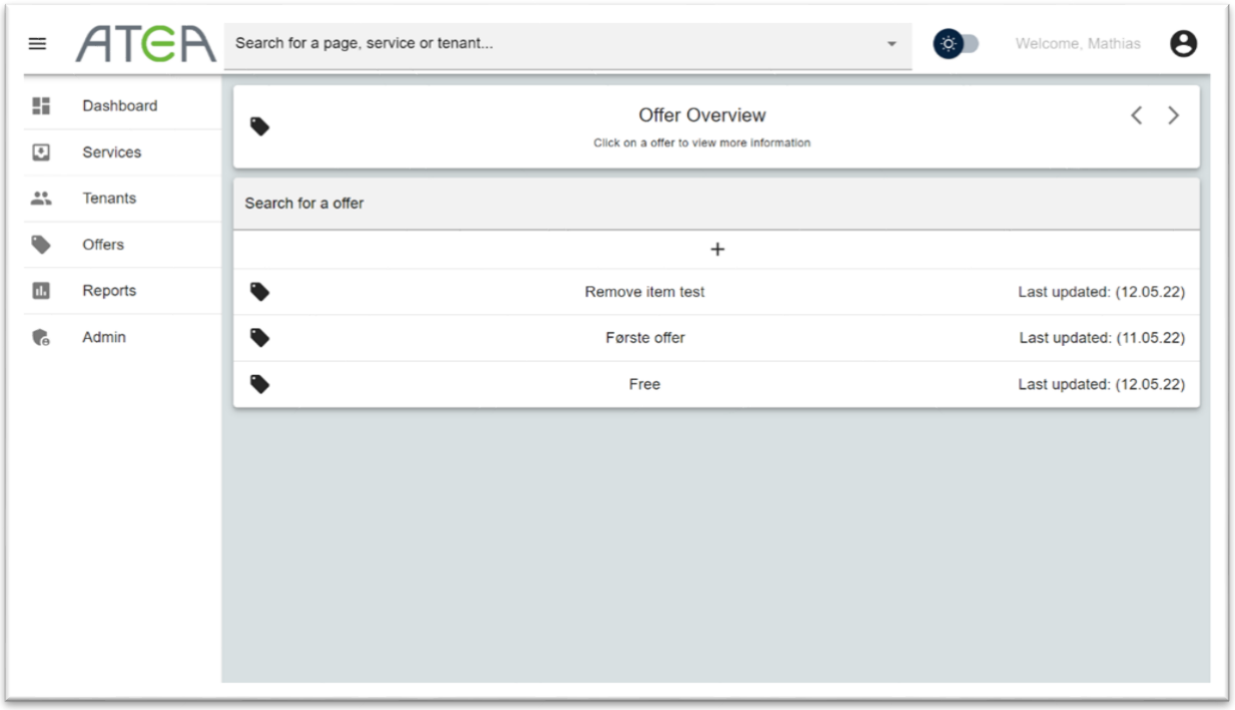
Figur 21: Rapporter.



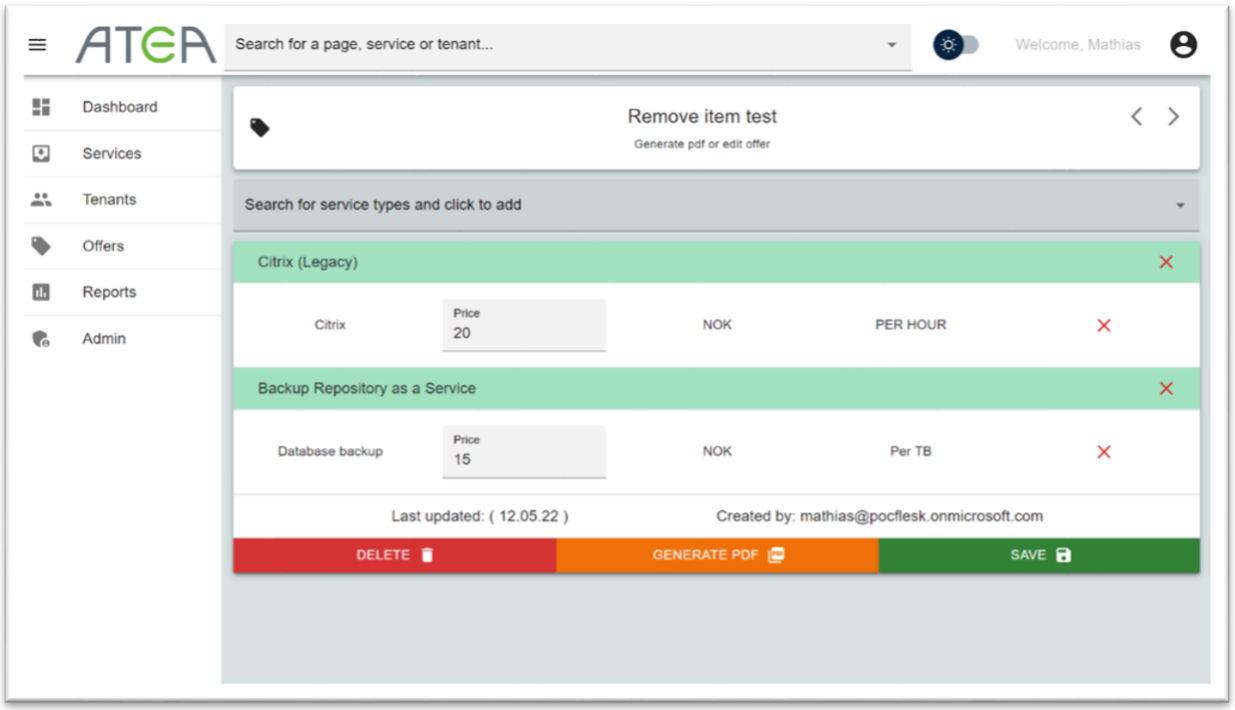
Figur 22: Administrator kan endre roller for brukere av applikasjonen.

En bruker kan som har administratortilgang kan endre andre brukers tilganger. Dette gjøres som vist i figur 22, ved at administratorbrukeren flytter over de rollene den andre brukeren skal ha, for så å klikke på knapp for å oppdatere.

Brukere kan også se en liste over de ulike anbudene som er lagret, illustrert i figur 23. Den kan her også velge å opprette et nytt anbud eller trykke inn på et anbud.



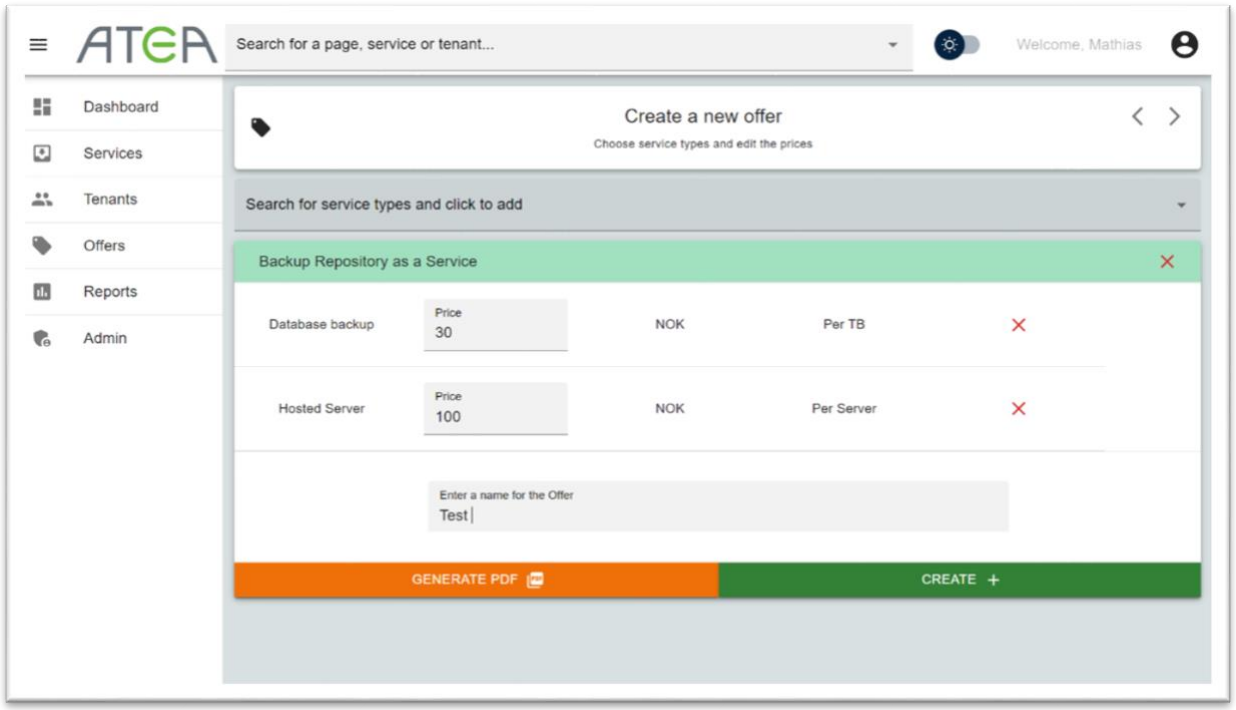
Figur 23: Anbudsoversikt.



Figur 24: Anbudsside.

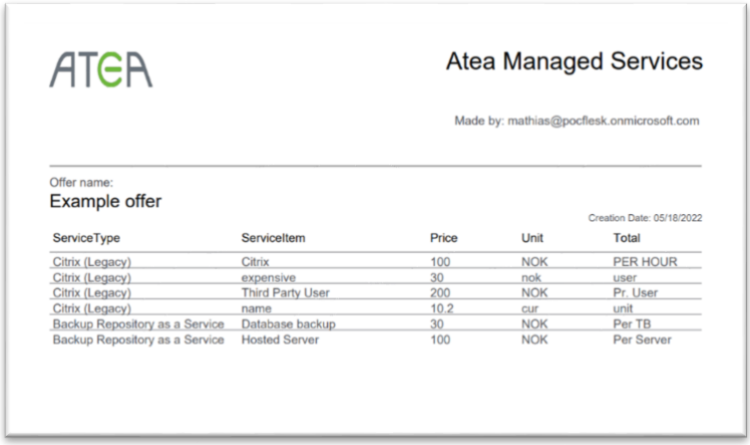
Trykker brukeren inn på et anbud vil den kunne se informasjon om hvilke tjenester og varelinjer som er inkludert i anbudet, illustrert i figur 24. Her kan brukeren endre prisen for varelinjen og lagre dette. Brukeren kan generere et PDF-dokument med anbudet. Hvis anbudet ikke skal brukes mer kan det også slettes.

Velger brukeren å lage et nytt anbud kan den søke etter en tjeneste og legge den til i anbudet, som vist i figur 25. Deretter kan brukeren endre pris, eller fjerne enkelt varelinjer som ikke skal være en del av anbudet. Videre kan anbudet lagres i databasen og et PDF-dokument kan også her bli generert.



Figur 25: Side for opprettelse av anbud. Muligheter for å legge til tjenester og varelinjer, samt fjerne dem. Det er også mulig å opprette PDF-dokument av anbudet.

Hvis en bruker velger å generere et PDF-dokument av et anbud vil det lastes ned på datamaskinen. I figur 26 kan man se et eksempel på hvordan et slikt dokument vil se ut.


Atea Managed Services

Made by: mathias@pocflesk.onmicrosoft.com

Offer name:
Example offer

Creation Date: 05/18/2022

ServiceType	ServiceItem	Price	Unit	Total
Citrix (Legacy)	Citrix	100	NOK	PER HOUR
Citrix (Legacy)	expensive	30	nok	user
Citrix (Legacy)	Third Party User	200	NOK	Pr. User
Citrix (Legacy)	name	10.2	cur	unit
Backup Repository as a Service	Database backup	30	NOK	Per TB
Backup Repository as a Service	Hosted Server	100	NOK	Per Server

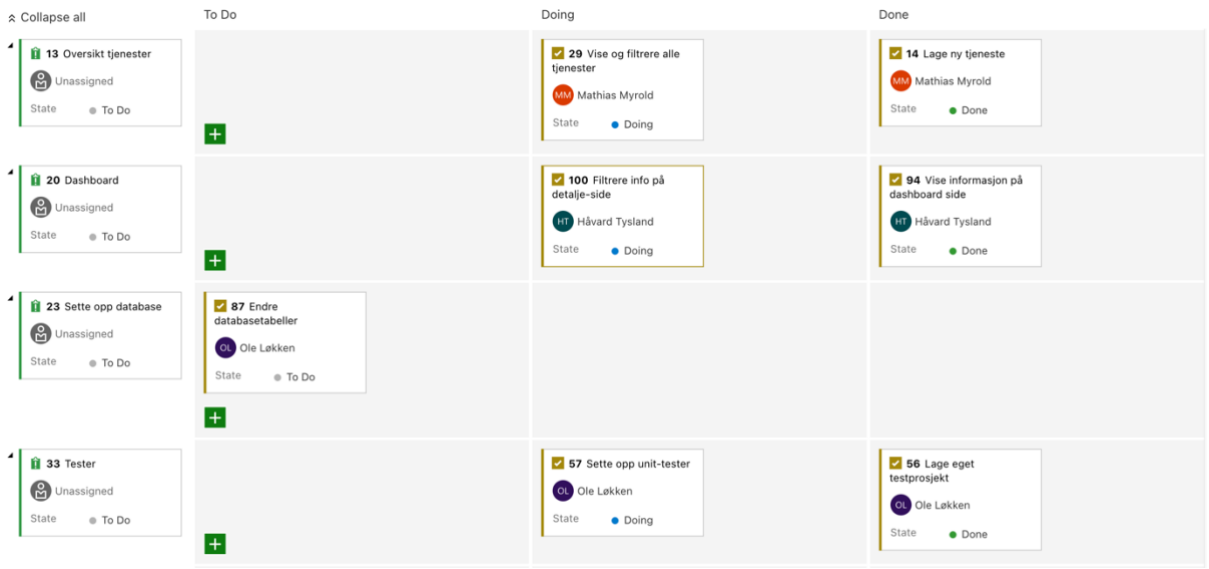
Figur 26: Eksempel på anbud i PDF.

4.2 Administrative resultater

Dette delkapittelet tar for seg de ulike administrative aspektene ved prosjektet.

4.2.1 Utviklingsmetodikk

Teamet brukte Scrumban, en kombinasjon av de to smidige metodene Scrum og Kanban. Alle de planlagte oppgavene ble lagt inn som *features* og *epics* i Azure DevOps sin innebygde *backlog*. Disse ble videreført inn i kanban-brettet som visualisering av arbeidsflyten. Oppgavene administrert i kanban-brettet for hver sprint hvor hovedstadiene var *To do, Doing* og *Done*. Eksempel på dette er vist i figur 27. Oppdragsgiver og veileder ble tidlig kalt inn til fremtidige sprint *review* gjennom Microsoft Teams. Møtemedlemmene besto av produkteier, utviklerne og veileder så fremt alle var tilgengelige. Etter disse møtene vurderte teamet sprinten i en *retrospektiv*, og planla neste sprint.



Figur 27: En del av Kanban-brettet på Azure DevOps som teamet brukte gjennom de ulike sprintene.

4.2.2 Fremdriftsplan

Fremdriftsplanen, som ble beskrevet i 3.1.2, utgjorde utgangspunktet i arbeidets fremdrift. Milepælene i Gantt-diagrammet besto av de ulike sprintene. Der ble det beskrevet fra- og til dato som en sprint skulle finne sted. Som nevnt ble sprintene gjennomført i de tidsrommene som originalt var planlagt. Hvert sprintmål ble oppnådd, med unntak av distribuering av nettsiden. Dette var mer krevende enn antatt, og gjorde at målet ikke ble nådd i sprint 4. Dermed måtte teamet ta følgende hensyn og gjøre nye vurderinger for fremdriften i den siste delen av prosjektet.

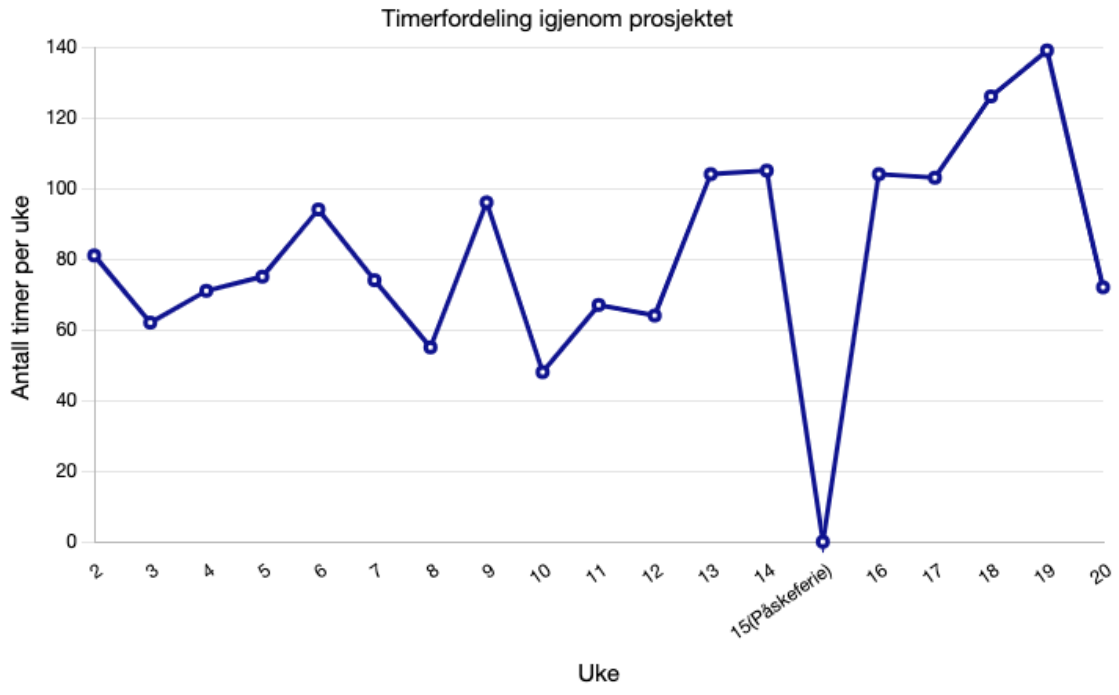
Videre kom det også nye ønsker fra oppgavestiller om en brukers mulighet til å opprette anbud. Teamet vurderte tidsbruken oppgaven ville ta, og anså det som verdifullt å sette av tid ved siden av dokumentasjon til å implementere funksjonaliteten. Originalt var planen å unngå koding etter siste sprint, men ettersom det nye ønsket oppsto ble det likevel nødvendig. Til tross for de små endringene i den skisserte planen, kom teamet i mål med ønsket arbeid.

4.2.3 Tidsforbruk

Teamet har ført timer for hver dag som er gjort arbeid på. Disse føringene inkluderte antall timer og en beskrivelse for hva man gjorde for den dagen, vist i figur 28. Timeliste for hver enkelt teammedlem ligger i Vedlegg D Prosjekthåndbok. Videre kan man i figur 29 se hvordan timene ble fordelt utover de ulike ukene.

Uke	Dag	Hva	Beskrivelse	Timer
17	Mandag	Rapporter	Starte på siden for å generere rapporter for tjenester eller kunder	8
	Tirsdag	Rapporter	Fortsette med å lage ulike diagrammer for å vise dataene	8
	Onsdag	Rapporter	Velge dato, tjeneste eller kunde å vise rapporten for	8
	Torsdag	Småfiks	Fjernet mye småting som ga warnings samt fikset andre ting som trengtes	8
	Fredag	Sprint 5 review	Fjernet siste endringer og sprint 5 review	8
	Lørdag			
	Søndag			
			SUM	40

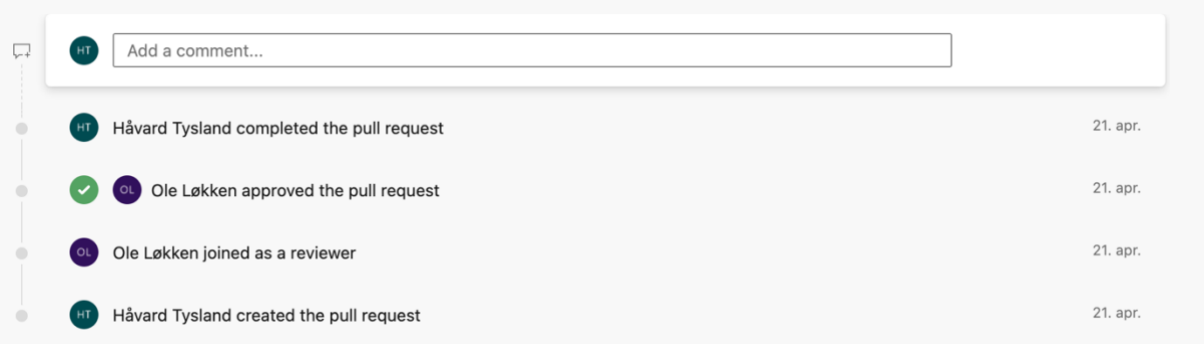
Figur 28: Eksempel på føring av timer i løpet av en uke.



Figur 29: Samlet timebruk per uke for hele teamet.

4.2.4 Versjonskontroll

Hver utvikler har jobbet i sin egen branch under utvikling. Når en utvikler etter hvert skulle flette sammen koden sin inn i hovedbranchen ble det opprettet en *pull request.* Et annet medlem av teamet gikk da inn og så gjennom koden og godkjente eller kom med muntlige kommentarer og spørsmål hvis det var nødvendig. Ble den godkjent ville da den som opprettet *pull requesten* fullføre den slik at koden ble flettet inn i hovedbranchen. Deretter ble det kjørt pipelines med automatisk bygging og tester, samt kontinuerlig utlevering.



Figur 30: Forløpet av en godkjent pull request.

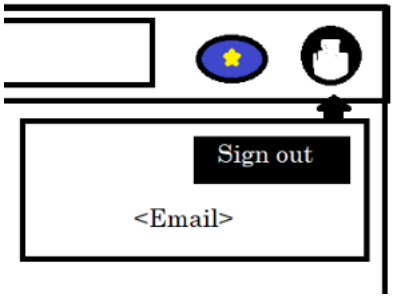
4.3 Vitenskapelige resultater

Denne seksjonen tar for seg de vitenskapelige resultatene som er kommet frem gjennom prosjektet. Innholdet baserer seg på brukertester som har blitt gjennomført ved bruk av kvalitativ metode. De første testene ble gjennomført på wireframes tidlig i prosjektet for å få et inntrykk av hvordan designet av nettsiden skulle være. Senere ble en brukertest for sluttbrukere gitt for å sjekke om det planlagte designet fungerer i praksis. I tillegg er det generert Google Lighthouse rapporter for å teste hvordan systemet er utviklet mot WCAG og universell utforming.

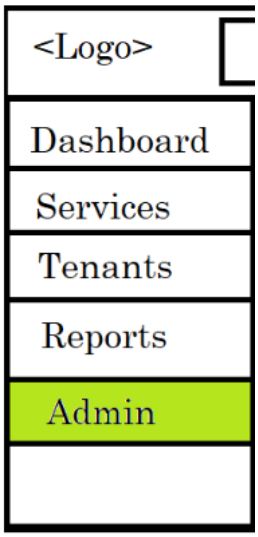
4.3.1 Brukertester på wireframes

Brukertester som ble gjort på wireframes ble gjennomført ved hjelp av et skjema utviklerne gikk igjennom muntlig med ulike personer. Skjemaet ligger under Vedlegg E Brukertester. Testene ble gjort på en mindre gruppe mennesker med forskjellig alder og ulik grad av teknologisk erfaring. Dette skulle gjenspeile en gruppe mennesker lik sluttbrukerne av produktet. Målet med testen var å bekrefte eller avkrefte at designet av nettsiden fremsto oversiktlig, forståelig og enkel å bruke.

Ut ifra testene fikk teamet bekreftet flere av løsningene som var planlagte. For eksempel, å plassere en knapp for å logge ut i en meny oppe i høyre hjørne av en nettside, ble sett på som intuitivt for de fleste testerne. På samme måte ble sidemenyen bekreftet som en god og kjent måte å navigere seg rundt på en nettside. Figur 31 og 32 illustrerer wireframes for utlogging og sidemeny.

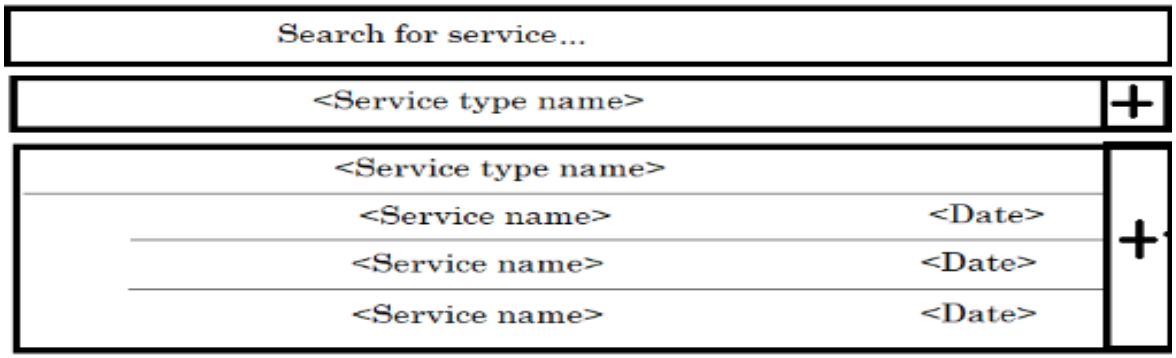


Figur 32: Utloggings design fra wireframes brukt i brukertest.



Figur 31: Sidemeny brukt til navigering på nettsiden.

Videre fikk utviklerne bekreftet at bruken av et konsekvent design på lister, knapper og innskrivingsfelt er en god måte å gjøre designet forståelig og lett å bruke. Det kom fram at dette bidrar til at brukeren kan bruke samme fremgangsmåte for å komme frem til ulike typer informasjon. Figur 33 viser den planlagte løsningen for lister.



Figur 33: Liste design fra wireframes

Fra brukertestene kom det også frem flere forslag til design. Flere brukere nevnte at bruk av symboler og farger i knapper eller symboler sammen med en tekstlig forklaring bidrar til en bredere forståelse av elementer på nettsiden. Det ble også stilt spørsmål angående tilbakemeldinger etter en handling er utført. Her svarte flertallet at det var ønskelig med en form for tilbakemelding for å vite om handlingen var vellykket eller ikke.

4.3.2 Brukertester på sluttprodukt

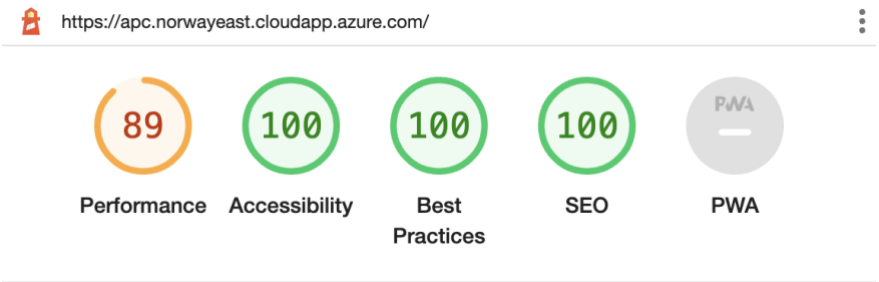
Brukertester på sluttproduktet ble gjort i slutten av prosjektet på en mindre gruppe ansatte i Atea. Her ble *Google Forms* tatt i bruk for å sette opp forskjellige oppgaver brukerne skulle gjøre. Testen gikk ut på at brukeren skulle gjøre oppgaver på nettsiden og deretter gi tilbakemeldinger på hvordan oppgaven gikk. For eksempel, var den ene oppgaven at brukeren skulle legge til en ny varelinje i en tjeneste. Etter oppgaven var gjort kunne brukeren krysse av om utførelsen gikk bra eller ikke, samt beskrive hvordan den gikk frem for å løse oppgaven. Til slutt kunne brukeren også komme med andre tilbakemeldinger om applikasjonen.

Resultatene fra brukertestene tilsier at oppgavene i stor grad var gjennomførbare og grensesnittet ble opplevd som intuitivt. Brukerne navigerte seg lett gjennom nettsiden og fant frem til ønsket informasjon. En bruker nevnte også at det var fint med store og tydelige knapper med farger. I tillegg var en bruker positiv over all funksjonaliteten som fant sted på nettsiden gitt tidsrammen av prosjektet.

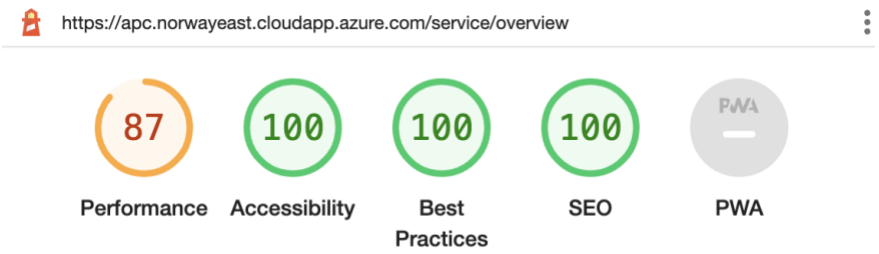
Det kom også noen tilbakemeldinger på forbedringspunkter. I oversikten over tilbud hos en kunde kunne varelinjene blitt sortert etter hvilken kategori de hører til. I tillegg var det ønsket nedtrekkliste på enhet og valuta når man legger til en varelinje. En bruker synes det ville vært naturlig å bli omdirigert etter man har lagt til en ny varelinje. Når man skal legge til tilbud for en kunde, observerte en bruker at listen med varelinjer kan bli veldig lang. Brukeren synes også det var vrient å forstå hvordan man skulle legge til et nytt tilbud for en kunde. Tilbakemeldingene på brukertestene kan leses nøyere i Vedlegg E Brukertester.

4.3.3 WCAG

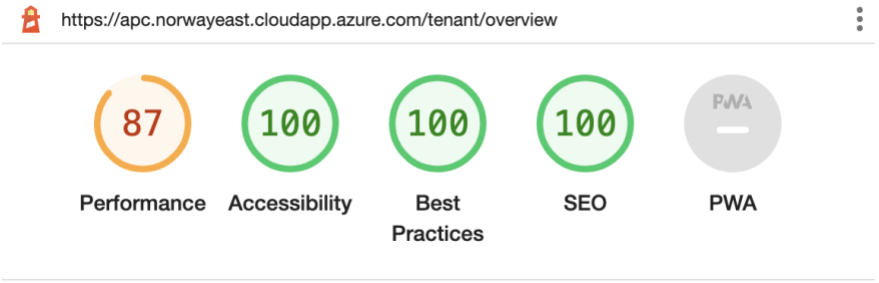
For å teste nettsidens ytelse, tilgjengelighet og korrekthet opp mot WCAG-prinsippene og universell utforming, er det generert Google Lighthouse rapporter. For eksempel vil det bli sjekket at bilder inneholder alternativ-tekster. I tillegg sjekkes det at knapper har nyttige *aria-labels*, som kan bli brukt av skjermlesere. Lighthouse rapportene er brukt av utviklerne til å sjekke produktet opp mot eventuelle mangler. Rapportene gir en tydelig beskrivelse av hva og hvor eventuelle mangler foreligger. Figur 34-37 viser genererte rapporter på ulike steder av nettsiden med en oversikt over nettsidens uttelling på de ulike punktene:



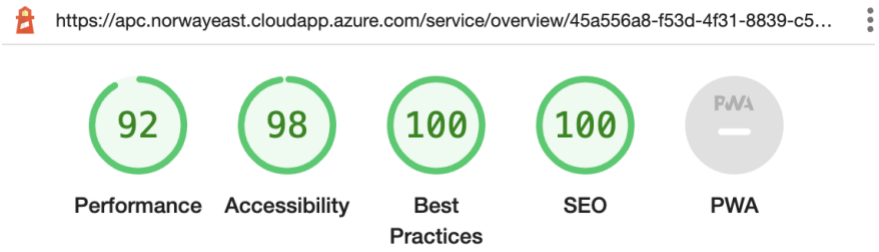
Figur 34: Dashboard



Figur 35: Tjenesteoversikt



Figur 36: Kundeoversikt



Figur 37: Informasjon om varelinje.

5 Diskusjon

I denne seksjonen diskuteres resultatene som ble presentert i kapittel 4.

5.1 Produktrelaterte resultater

I kapittel 4 ble de produktrelaterte kravene som var definert i Vedlegg A Visjonsdokument presentert og hvorvidt disse var oppfylt eller ikke. Siden oppgavebeskrivelsen ga en mindre detaljert beskrivelse om hva oppgaven skulle inneholde, hadde teamet flere møter med oppdragsgiver for å sette seg inn i funksjonalitetene og definere kravene. Teamet satte raskt et mål om å få med så mye funksjonalitet som mulig, som resulterte i en omfattende kravliste.

5.1.1 Funksjonelle krav

Under utvikling av produktet var teamet som nevnt delt inn i frontend- og backend-utviklere. Når man deler teamet inn slik kreves det god kommunikasjon og planlegging for hvilke funksjonaliteter som skal prioriteres. For at frontend-utvikleren skal kunne ferdigstille et krav eller funksjonalitet må tilhørende backend krav være implementert. Som resultat av dette ble prioriteringen av krav ofte bestemt ut ifra hvilke funksjonelle krav som bygger på hverandre.

Innledningsvis i prosjektet ble funksjonaliteter som omhandler tjenester og varelinjer prioritert, fordi flere av de andre kravene bygger på disse. Et eksempel på dette kan være at rapporter henter data fra lagrede tjenester. Videre prioritering av krav hadde mindre forskjeller, da utviklerne hadde et mål om å oppfylle så mange krav som mulig. For å oppnå dette ble prioriteringen av kravene gjort kontinuerlig i sprint reviews, slik at teamet kunne få en oversikt over hva som kan bygges videre eller begynnes på i neste sprint.

5.1.1.1 Frontend

I oppstartsfasen av prosjektet var prioriteringen av krav preget av arbeidsmengden relatert til å sette opp en fungerende backend. Tiden det tar å sette opp et utviklingsmiljø og tilhørende tjenester for backend kontra frontend, førte til at frontend-utviklingen startet med funksjonaliteter som ikke innebar kommunikasjon med API-et.

Utviklingen av frontend startet dermed med å implementere navigasjonen, samt muligheten til å velge lyst og mørkt tema til nettsiden. Samtidig som navigasjonen er en funksjonalitet som ikke krever kommunikasjon med backend, er det også essensielt for videreutvikling av nettsiden da flere komponenter har behov for navigasjonen. Lyst og mørkt tema hadde lavere prioritet, men ble implementert tidlig på grunn av ledig tid.

Som følge av prioritering av tjenester og varelinjer i starten av prosjektet, ble utviklingen rundt disse fokuset. Dette inkluderte først og fremst å opprette en varelinje for en tjeneste og hente ut disse. Når dette var implementert kunne frontend-utviklerne enten jobbe videre med resterende funksjonaliteter relatert til varelinjer eller jobbe med andre krav. Som sagt ble de resterende funksjonalitetene relativt likt prioritert. Dermed plukket utviklerne funksjonaliteter i henhold til hva som manglet. Dette ble gjort med et realistisk mål om å bli ferdig med alle kravene.

Kravet om at varelinjer skulle kunne opprettes med ulike nivå ble ikke implementert i produktet. Utviklerne var gjennom utviklingen usikker på denne funksjonaliteten sin nytte i applikasjonen, og bestemte etter møte med oppdragsgiver at den skulle nedprioriteres.

Senere i prosjektet hadde utviklerne selv forslag til øvrige implementasjoner som førte til flere funksjonaliteter for allerede utviklede komponenter. Lister med forskjellig innhold ble mye brukt på nettsiden og førte til implementasjoner for søkefelt i samtlige lister for å finne ønskelig informasjon raskere. At revisjonen med tidligere endringer i varelinjer kunne bli lang og uoversiktlig, førte til en implementering av sortering og filtrering av disse. Utviklerne kom også frem til at revisjonstabellen kunne brukes for å gjenopprette varelinjen til en tidligere versjon. En slik gjenopprettelse ble sett på som nyttig med tanke på eventuelle redigeringsfeil eller kun for å bruke tidligere verdier for varelinjer. Dermed ble dette også implementert.

I slutten av prosjektet kom oppdragsgiver med et nytt ønske til funksjonalitet. Brukere av produktet skulle ha muligheten til å sette sammen et anbud og ut ifra dette generere et PDF-dokument. Siden dette forslaget kom sent i prosjektet ble tidsbruken av en slik implementasjon vurdert, men ble besluttet gjennomførbar innenfor tidsrammen og dermed implementert i produktet.

5.1.1.2 Backend

Da teamet hadde forstått essensen i oppgaven, begynte prosessen med å definere hvilke modeller som var nødvendige, og hvordan de skulle samsvare seg imellom. Teamet startet med å utvikle funksjonalitet for de viktigste modellene, som var tjenester og varelinjer. Under den første sprinten tok det en del tid ettersom det ikke var tydelig for utviklerne hvordan man skulle hente data fra Atea sitt data-API. Dette skyldes en misforståelse mellom oppdragsgiver og teamet. Teamet ønsket å jobbe selvstendig, uten å forstyrre oppdragsgiver. Det ble derfor brukt unødvendig tid dette ble oppklart med oppdragsgiver. Tiden mistet her kunne gått utover sprint målet, men teamet jobbet hardt for å sørge for at det uansett ble oppnådd. Det ble erfart at selvstendighet ikke nødvendigvis alltid er positivt, særlig når man står fast med et problem.

Etter at utviklerne fikk tilgang til dokumentasjonen av Atea sitt data-API gikk utviklingen raskt. Operasjonene var stort sett å hente, redigere, slette eller legge til data. Teamet var gode til å kommunisere slik at backend jobbet på det neste frontend skulle implementere. Utviklerne fikk testet koden fra den hentes ut av databasen til den ble vist i brukergrensesnittet. På denne måten ble det forsikret at forespørslene og responsene var logiske. Samtidig sørget det for at utviklerne ferdigstilte en oppgave før de startet på den neste.

Et av de funksjonelle kravene var at applikasjonen skulle oppdatere den eksterne faktureringsmotoren Exivity. Det sørget for noen endringer som påvirket ferdigstilte oppgaver. Exivity har egne id-er som skiller objektene fra hverandre. Utviklerne valgte å lagre de eksterne id-ene i APC sin database, for at programmet ikke skulle trenge å sende flere forespørsler til Exivity når det skulle gjøres endringer. Endringer på oppgaver som er ferdige er ikke ønskelig, og reduserte tiden tilgjengelig for utvikling av ny funksjonalitet. Teamet kunne lest seg nøyere opp på faktureringsmotoren i oppstarten av prosjektet, men det ville igjen utsatt tiden før teamet hadde kommet i gang med utviklingen. Valget med å komme seg i gang tidlig sørget for at alle hadde arbeidsoppgaver fra en tidlig fase, som teamet mente var gunstig for den totale produktiviteten.

Et av de funksjonelle punktene som ikke ble gjennomført var at Exivity skulle oppdatere APC med jevne mellomrom. Denne implementeringen måtte skjedd i kildekoden til Exivity, og ikke APC. På grunn av tidsforbruket med å sette seg inn i kildekoden til Exivity, ble vi enige med oppdragsgiver om å ikke prioritere dette. Oppdragsgiver var kjent med hvordan man skulle implementere dette, og mente at det ville være fordelaktig at de implementerte dette selv.

5.1.2 Ikke-funksjonelle krav

I denne seksjonen diskuteres valgene tatt rundt de ikke-funksjonelle kravene for applikasjonen. Ikke-funksjonelle krav som inngår i de fire forskingsspørsmålene er ikke diskutert her, men i egne delkapitler.

Fra oppdragsgiver ble det stilt krav om at nettsiden skulle være brukervennlig og gi en god brukeropplevelse. For å oppnå dette har teamet gjort flere valg gjennom utviklingen som bygger på ulike prinsipper om universell utforming og brukervennlighet. Et eksempel på dette er bruken av ikoner sammen med tekst, for å sikre at all informasjon er forståelig, og at nettsiden er intuitiv i bruken. I tillegg har teamet basert seg på en gjennomgående struktur for lister av enten tjenestetyper, kunder eller anbud. Årsaken til at dette ble gjort var å holde en konsistent brukeropplevelse.

Videre ønsket oppgavestiller gode søkemuligheter, og i den sammenheng er det blitt implementert søkefelt både i navigasjonsmenyen og ved store lister. Siden lister i nettsiden kan bli lange over tid, har utviklerne vurdert implementasjon av paginering. Dette er et element som kunne redusert ventetiden under innlastningen av lange lister, og er noe som kan vurderes til videre arbeid. En av utfordringene med gamle APC var nemlig tiden brukt på å komme seg til ønsket informasjon. Valget på å ha et søkefelt i navigasjonsmenyen ble dermed gjort ettersom teamet anså det gunstig for brukere å kunne søke direkte etter ønsket side, varelinje eller kunde. Slik unngår brukere unødvendig navigasjon, og brukeropplevelsen kan oppleves mindre anstrengende.

Brukerens forståelse av hva som foregår før, under og etter handlinger har vært høyt prioritert med tanke på brukervennligheten. Hver handling fra bruker som fører til kommunisering med API-et, krever at brukeren bekrefter at den gitte handlingen skal gjennomføres. Under kommunikasjonen med API-et blir en animasjon med tekstforklaring presentert, slik at brukeren vet at kommunikasjonen pågår. Etter en slik handling vil tydelige tilbakemeldinger med fargekoder og forklarende tekst bli presentert.

På denne måten forsikrer teamet at brukeren vet hvilken handling den har gjort, samt om den ble gjennomført.

Synlighet er også et punkt i Don Normans seks designprinsipper, noe som er tatt i betraktning under designet av nettsiden. Et eksempel på dette er at handlingene brukeren kan gjøre med en varelinje er samlet i store synlige knapper på varelinjesiden. Nettsiden har også avvik i synligheten, da utloggingsknappen ikke er synlig for brukeren før profilmenyen oppe i høyre hjørne er åpnet. Dette valget ble tatt på grunn av ryddighetens skyld og begrunnet ved at dette er en konvensjonell plassering av en utloggingsknapp. Plassering av knappen ble også gjennom brukertester av wireframes beskrevet som intuitiv.

Videre var det ønskelig for både oppdragsgiver og utviklerne at applikasjonen skulle distribueres. Dette innebar å distribuere både frontend og backend i hver sin container, som var fordelaktig av flere grunner. Oppdragsgiver og andre ansatte i Atea kunne dermed følge fremgangen av prosjektet uten teknisk forståelse av hvordan koden blir kjørt lokalt. Samtidig var dette nyttig for frontend-utviklerne som med distribuering av applikasjonen kunne jobbe mot et og samme API uten å kjøre backend koden lokalt. For utviklerne førte dette til at frontend og backend kunne jobbe mer individuelt, da partene ikke var avhengig av å ha hverandres kode lokalt.

5.1.3 Fører bruken av en identitetsleverandør til en og sikker applikasjon på egenhånd?

Som beskrevet i kapittel 3.2.3.1, er SSO en av de store fordelene med å bruke en identitetsleverandør. Brukere trenger kun å huske på ett passord. *Password fatigue* er et begrep som forteller av følelsen når man har et overdrevent antall passord å huske på. Ifølge en studie fra NordPass gjennomført i 2020 må en gjennomsnittsperson huske omtrent 100 passord. En annen studie gjort av Beyond Identity sier at personer med høy *password fatigue* har dobbelt så stor sjanse for å bli hacket som de med lavere grad. I USA har 62% av de med høy grad av password fatigue blitt hacket. I motsetning er det 29% som har blitt hacket av de med lav *password fatigue*. Ulempen med SSO er om en uvedkommen vet innloggingsinformasjonen til en bruker, får den tilgang til mange tjenester samtidig. Derfor er det viktig at SSO er godt beskyttet med sikre passord, og tofaktorautentisering. De mest kjente identitetsleverandørene tilbyr tofaktorautentisering, og i AAD kan man skru det av og på.

En annen del av sikkerheten er autorisasjonen. Om alle brukere skal ha tilgang til de samme funksjonalitetene. I utviklingen av APC ønsket oppdragsgiver at applikasjonen skulle forbinde brukere med tagger som skulle gi bestemte tilganger. For eksempel kan en bruker ha tilgang til å gi en kunde et tilbud, men ikke til å lage nye varelinjer. Når en bruker prøver å gjøre en handling, vil applikasjonen sjekke opp mot APC sin database om brukeren har nødvendig tilgang. Hvis brukeren ikke har tilgang vil den få tilbakemelding om det. Ulempen med framgangsmåten er at det krever en ekstra databasetilkobling og spørring før brukeren får utført en operasjon. Det sørger for at ulike operasjoner tar lengre tid enn alternativet.

En alternativ måte å løse dette på er å lagre tilgangene til brukeren i tokenet. Dermed kan API-et lese tokenet for å sjekke om brukeren har tilgang, som vil være raskere enn å sjekke databasen. Hvis man skal lagre tilgangene i tokenet må man definere rollene i

identitetsleverandøren, ettersom det er den som oppretter tokenet. Teamet fant ut at i det i AAD kun var mulig for en bruker å ha én rolle samtidig. Ettersom applikasjonen har 10 forskjellige spesialtilganger, finnes det også veldig mange kombinasjoner av roller. Hvis rolletilgangen skal lagres i tokenet, konkluderte teamet med at man trenger enklere roller. For eksempel, kan man ha en rolle som gir brukeren muligheten til å endre data, men ikke legge til eller slette. Et alternativ er å ta i bruk en identitetsleverandør som tilbyr at en bruker kan ha flere roller samtidig. Dette var derimot ikke en mulighet ettersom Atea allerede brukte AAD.

5.1.4 Hva er fordelene og ulempene ved å bruke et komponentbibliotek for å utvikle en brukervennlig nettside?

Gjennom prosjektet har utviklerne møtt på flere fordeler med bruk av komponentbibliotek for å utvikle en brukervennlig nettside. Biblioteket som er brukt i prosjektet, *Material UI*, leverer ferdigdesignede, optimaliserte og kryssnettleserkompatible komponenter. Siden komponentene er optimalisert, også med tanke på universell utforming og WCAG, kan utviklerne bruke de i applikasjonen med sikkerhet om et brukervennlig brukergrensesnitt. Alternativet hadde vært å utvikle komponentene fra grunnen av. Dette kunne derimot ført til større usikkerhet rundt brukervennligheten, ettersom variasjonen i løsningene kunne vært større. Om et slikt bibliotek blir brukt konsekvent bidrar dette til konsistensen i nettsiden, der brukeren opplever like løsninger for både design og funksjonalitet. Samtidig åpner den konsekvente bruken av komponenter også for andre muligheter, som for eksempel å kunne velge fargetemaer for nettsiden. Dette er noe utviklerne har tatt i bruk for APC og er som nevnt i 2.2.1, et element som bidrar til en fleksibel nettside.

Som sagt er komponentene kryssnettleserkompatible, som vil si at de støtter flere typer nettlesere. Hadde utviklerne laget komponentene fra grunnen av kunne de funket på nettleseren man utvikler i, men kanskje ikke for andre. Hvilken nettleser en sluttbruker av produktet benytter seg av er noe som ikke kan forutsees. Det er derfor en fordel å benytte kryssnettleserkompatible komponenter, for å opprettholde fleksibiliteten for bruken av nettsiden. I tillegg, ville en utvikling fra grunnen av være tidskrevende og redusert tiden disponert til andre elementer som bidrar til brukervennlighet. Som bekreftelse om at bruk av komponentbiblioteker følger ulike krav for universell utforming og WCAG vises det til resultatene fra Google Lighthouse rapportene i 4.3.3.

Selv om slike komponentbibliotek har mange fordeler kan de også ha ulemper som kan gå ut over brukervennligheten. Bibliotekene har ofte flere gamle versjoner som ikke lenger har støtte, og dermed ikke er kompatible med nyere versjoner. I løpet av prosjektet støtte utviklerne på et problem som dette, der en av designfunksjonene i biblioteket ikke fungerte som den skulle. På grunn av dette problemet ble det oppdaget flere uregelmessige feil i designet av komponenter. Hvis slike feil ikke blir oppdaget kan dette gå ut over både funksjonaliteten og helheten av designet til nettsiden. Det er dermed viktig å sette seg inn i dokumentasjonen slik at man sikrer riktig bruk av biblioteket.

5.1.5 Hvilke valg bør tas hensyn til når man utvikler et API som jobber sammen med flere eksterne tjenester?

For å sørge for en sikker applikasjon ble det bestemt å samle alle kall til eksterne tjenester i REST API-et. Slik er det sørget for at alle kall kun blir gjort av brukere med riktige tilganger. Videre ble det bestemt at det kun skulle være nødvendig med ett enkelt kall for å hente eller oppdatere informasjon. Et eksempel på dette er når en bruker legger til en ny varelinje. Her unngår man å ha ett eget kall for å oppdatere faktureringsmotor og ett eget kall for å oppdatere databasen. En ulempe ved dette er at man gjerne svekker applikasjonens ytelse. Ønsket data må først sendes fra brukeren, deretter formateres slik at man kan oppdatere faktureringsmotor. Til slutt kan responsen sendes tilbake til brukeren.

Hvis et steg mislykkes bør responsen brukeren får inneholde en beskrivelse av hvilken tjeneste som var årsaken. Dette vil bidra med å gjøre brukeropplevelsen bedre. Applikasjonen oppfyller dette i noen tilfeller, men har også et forbedringspotensial her som bør tas hensyn til under videreutvikling. Videre bør utviklere som skal bygge et API med flere eksterne tjenester ta hensyn til rekkefølgen man oppdaterer tjenestene på. I teamets tilfelle sto valget mellom å oppdatere lokal database eller ekstern faktureringsmotor først. Erfaringene viste at den eksterne faktureringsmotoren var mer ustabil, og hadde en større sannsynlighet for å mislykkes under en operasjon. Dermed falt valget på å oppdatere faktureringsmotoren først, for deretter å oppdatere den lokale databasen. Et annet viktig punkt er å sørge for at om oppdatering av den siste tjenesten mislykkes, bør de tidligere tjenestenes oppdatering tilbakestilles. Dette var noe som ble ansett som lite sannsynlig i det utviklede produktet, og implementasjon av dette ble dermed ikke prioritert. Teamet ser dermed absolutt verdien av denne funksjonaliteten i et produkt som skal i produksjon, og ser på dette som et punkt som bør prioriteres under videreutvikling.

5.1.6 Hvilke tiltak kan man ta for å legge opp til videreutvikling i et prosjekt?

Når man skal sette opp et prosjekt som man vet skal videreutvikles er det ulike tiltak man kan ta for å gjøre denne prosessen mest mulig problemfri. Først og fremst bør man sette opp en tydelig og oversiktlig prosjektstruktur. For å sørge for dette ble det besluttet å dele frontend og backend opp i to ulike mapper. Frontend ble bygd opp av spesielle komponenter som skulle ha én spesifikk funksjon, samt generelle komponenter som kan gjenbrukes. Fordelen ved å ha spesielle komponenter er at det blir enkelt å finne og endre akkurat det man skulle ønske. Derimot kan det føre til mye lik kode hvis man har komponenter som er relativt like, men ikke kan tilpasses under en generell komponent. I backend ble det delt opp i egne individuelle prosjekter med hver sin funksjon. Nytten er at man da kan erstatte mindre deler av prosjektet i stedet for å starte helt på nytt.

Ved overlevering av et produkt til videreutvikling er det flere andre aspekter som bør rettes blikk mot. God kodedokumentasjon og forståelig kode er noe alle utviklere streber etter. Teamet har forsøkt å bruke forklarende og logiske navn på filer, metoder og variabler. I tillegg ble det prioritert en del tid på dokumentasjon av koden. Til tross for at dette tok tid som kunne blitt brukt direkte til utvikling, ble verdien sett på som høy nok

til å prioritere dette. Videre er dokumentering av domenemodeller og andre nyttige modeller viktig for å gi et overblikk av systemet man har utviklet. I tillegg valgte teamet teknologier som var kjente og oppdaterte med god dokumentasjon. Slik blir det mer sannsynlig at en utvikler som skal jobbe videre med prosjektet har vært borti de valgte teknologiene.

Videre er det med god dokumentasjon viktig å forklare hvordan man kan starte opp applikasjonen. Teamet har i tillegg til de mer konvensjonelle kjøremetodene valgt å bruke Docker. Dermed kan andre utviklere enklere starte opp prosjektet uten å måtte sette seg inn i all kildekoden.

Mot prosjektets avslutning er det som vist i Vedlegg E Brukertester, gjort brukertester på sluttproduktet. Selv om det nødvendigvis ikke er like vanlig å utføre brukertester mot slutten av et prosjekt, var det likevel nyttig ettersom tilbakemeldingene kan videreføres til Atea under overtaking av prosjektet. Fordelen ved å gjøre testene tidligere ville vært muligheten for å implementere disse endringene selv, men teamet prioriterte å implementere all funksjonaliteten slik at man kunne gjennomføre nøyere tester.

5.2 Administrative resultater

I denne seksjonen blir de administrative resultatene som kom frem diskutert.

5.2.1 Utviklingsmetodikk

Teamet fulgte den planlagte utviklingsmetodikken Scrumban uten særlige hindringer. Azure DevOps gjorde det enkelt å sette opp en backlog i starten av prosjektet. Slik ble arbeidet lagt frem på en oversiktlig måte slik at man lett kunne plukke oppgaver til sprint *backlogen* etter prioritet. Det innebygde kanban-brettet ga en visuell beskrivelse av hvordan teamet lå an under hver sprint, noe som opplevdes som veldig gunstig under utviklingsprosessen. Fleksibilitet ble også et viktig punkt, ettersom oppgavestiller kunne komme med nye krav for hver sprint under *sprint review*. Å ha definerte sprintmål var motiverende, og gjorde det tydelig hvorvidt teamet klarte å holde seg til planlagt fremdrift.

Til tross for at metodikken fungerte såpass bra fra start, var teamet innstilt på å kontinuerlig forbedre arbeidsprosessen. Etter de første sprintene ble det erfart at det ville være fordelaktig å dele oppgavene i enda mindre deler enn originalt satt opp. Slik ble hver oppgave mer overkommelig, og det ble da lettere å starte på en slik oppgave.

5.2.2 Fremdriftsplan

Som nevnt i 3.1.2, ble det ved starten av prosjektet skissert en fremdriftsplan for arbeidet som gikk hovedsakelig ut på å fordele sprintene til gitte datoer. Fordelen med dette var at både veiledere og teamet hadde definerte tidspunkt for møter igjennom hele prosessen. I tillegg kunne teamet ha en klar oversikt over om fremgangen gikk som

planlagt. Tidsrommene til disse sprintene ble fulgt som planlagt gjennom prosjektperioden. Samtlige av de planlagte sprint reviewene ble også fullført. Det oppsto enkelthendelser hvor produkteier eller veileder ikke kunne stille av ulike årsaker. Ved en slik hendelse ble det forsøkt å sette opp ett individuelt møte med vedkommende slik at samtlige var oppdatert til neste sprint.

I starten av et prosjekt må man avgjøre hvor mye tid man skal sette av til planlegging. Alternativt kunne teamet brukt lenger tid på å skissere en nøyere fremdriftsplan. Vurderingen teamet gjorde var derimot at overflødig planlegging ikke var hensiktsmessig ettersom en stor utviklingsprosess var i vente. I tillegg var det mye ny informasjon og teknologi å sette seg inn i, som heller ble prioritert.

Å respondere på endringer fremfor å følge en plan er et prinsipp fra smidig utviklingsmetodikk som teamet tok i bruk. Dette kom særlig frem mot slutten av prosjektet når ønsket om ny funksjonalitet ble presentert. Ettersom teamet var vant til å jobbe fleksibelt gjennom perioden, var det dermed lettere å endre planlagt fremdrift når det var nødvendig.

5.2.3 Tidsbruk

I starten av prosjektet ble teamet enig om å ha en så jevn arbeidsprosess som mulig. Grunnen var hovedsakelig at man ikke skulle sitte igjen med store mengder arbeid mot slutten av prosjektet. Som beskrevet i resultatene oppnådde teamet dette i stor grad. Fordelen ved dette var at teamet unngikk sene arbeidsøkter og kunne forholde seg til en fast arbeidsdag. Slik var det også lettere å holde motivasjonen oppe. Som man kan se i figur 29 i 4.2.3, ble det noen uker mindre arbeid enn andre. Dette kunne være et resultat av fag som gikk ved siden av, eller andre uforutsette ting som sykdom. Til tross for den jevne arbeidsprosessen var det derimot naturlig for teamet å sette inn et ekstra støt i slutten av prosjektet, som også gjenspeiles i tidsforbruket.

5.2.4 Versjonskontroll

For å ha kontroll på egen kode brukte teamet egne *branches* under utviklingen. Hver utvikler utviklet på sin egen branch og når man var ferdig med en oppgave flettet man den inn i hovedbranchen. Originalt var tanken å bruke *feature branches*, hvor hver funksjonalitet har sin egen branch. Som nevnt i 5.2.1, erfarte teamet at noen oppgaver ble for brede og manglet spesifisitet. Dette førte også til at det kunne ta lenger tid enn originalt ønsket å få flettet inn ny funksjonalitet til hovedbranchen. Dette var noe teamet tok vurderinger på, og forbedret i løp av prosjektperioden. For å sikre at all kode som ble flettet inn i hovedbranchen var levert til ønsket standard utnyttet teamet *pull requests* og *code reviews*. Hver pull request måtte godkjennes av et annet teammedlem før det kunne flettes til hovedbranchen. Her ble eventuelle spørsmål, endringer eller liknende ble tatt opp slik at teamet sikret ryddig, feilfri og effektiv kode. Det ble også implementert kontinuerlig integrasjon som kjører automatisk bygging og automatiske tester når en pull request blir opprettet. Fordelen med å kjøre disse var at man kunne oppdage alvorlige feil før man eventuelt godkjenner en pull request. Teamet kunne med

fordel vært tidligere ute med å lage flere tester, men dette ble i de tidlige fasene nedprioritert for å få utviklet så mye funksjonalitet som mulig.

5.2.5 Teamsamarbeid

Teamet har fra tidligere arbeid god erfaring med å jobbe sammen, og har hatt et veldig godt samarbeid gjennom hele prosjektperioden. For å sørge for at valgene som blir tatt representerer hele teamet, har det vært fokus på at alle meninger i teamet skal bære like tungt. Teamet har arbeidet sammen fysisk eller digitalt under de aller fleste arbeidsøktene, som har ført til at det har vært kort vei om ulike spørsmål skulle oppstå. På denne måten har også unødvendige misforståelser blitt unngått innad i teamet.

Rollefordelingen gjorde at frontend og backend kunne ha en kontinuerlig fremgang. Til tross for de definerte rollene, har samtlige vært behjelpelige der det skulle være nødvendig. Alt i alt er teamet svært fornøyd med samarbeidet.

5.3 Vitenskapelige resultater

5.3.1 Brukertester på wireframes

Brukertester på wireframes ble gjennomført tidlig i prosjektet for å få tilbakemeldinger på planlagt design. Siden brukertestene ble utført på en relativt liten gruppe mennesker er det vanskelig å trekke klare konklusjoner. Hensikten av testene var derimot ikke å finne en fasit, men heller få en pekepinn på hva som fungerer eller ikke. Som nevnt i 4.3.1, ble sluttbrukerne forsøkt gjenspeilet i testene. Uansett er det vanskelig å gjenspeile hvordan sluttbrukerne faktisk bruker nettsiden i praksis. Tilbakemeldingene fra brukertestene er dermed tatt til betrakting, men på ingen måte brukt som en fasit på hvordan nettsiden skulle gjøres brukervennlig.

Det skal også nevnes at wireframes som ble brukt under brukertester var statiske, uten klikkbare elementer. Med klikkbare wireframes er det mulig at testbrukeren hadde fått et bedre innblikk i hvordan nettsidens funksjoner og design var planlagt. Som resultat av dette kunne utviklerne fått flere og bredere tilbakemeldinger på funksjonaliteter, som videre kunne bidratt til planleggingen av nettsiden. Derimot ville utarbeidingen av klikkbare wireframes vært mer tidskrevende. Samtidig fungerte de statiske wireframesene til sin hensikt, nemlig å samle tilbakemeldinger på det planlagte designet av nettsiden.

5.3.2 Brukertester på sluttprodukt

I slutten av prosjektperioden ble det som nevnt gjort brukertester på sluttproduktet. Dette ble gjort til denne tiden av flere årsaker. Det var ønskelig å utføre testene på ansatte i Atea. Det var dermed nødvendig at nettsiden var distribuert og tilgjengelig slik at brukertestene kunne gjennomføres hvor som helst. Som nevnt i 4.2.2, tok

distribueringen lenger tid enn teamet først hadde antatt. I tillegg var det prioritert å få implementert all funksjonalitet før brukertestene ble gjort, slik at man kunne få testet flere ulike aspekter av applikasjonen. Hadde brukertestene blitt utført tidligere, ville teamet hatt mulighet til å følge opp tilbakemeldingene selv. Samtidig, kunne den ekstra tiden ført til at flere i Atea hadde hatt mulighet til å svare. Uansett, ble brukertestene ansett som verdifulle, ettersom tilbakemeldingene kan følges opp under videreutvikling.

I tillegg var det interessant for teamet å undersøke om løsningene fungerer som forventet. Ettersom brukertestene ble gjort på ansatte i Atea, var det som sagt ikke mulig å få inn like mange svar som ønsket. Likevel var det mest interessant å gjøre testene på ansatte, siden de vil være primærbrukerne av nettsiden. Alternativt kunne man også utført testene på andre, men teamet vurderte de ansattes kunnskaper om AMS som viktige for å kunne gi mer relevante tilbakemeldinger.

Resultatene som kom fram ga en pekepinn på at grensesnittet som er utviklet er forståelig og møter de forventninger som var gitt av Atea. Blant annet ble tydelige knapper med ikon og farge satt pris på, noe teamet hadde i fokus under utvikling av nettsiden. I tilbakemeldinger fra testbrukerne så man at alle oppgaver var gjennomførbare, men med noen tilbakemeldinger til forbedringspunkter. Disse tilbakemeldingene vil ikke bli implementert av teamet, men kan brukes veiledende under videreutvikling.

Å sortere varelinjene en kunde har tilbud på etter hvilken tjeneste den hører til, vil være hensiktsmessig når listen med tilbud blir veldig lang. Under utvikling av produktet har teamet jobbet med mindre data enn det som gjerne vil finne sted i produksjon. Dermed har teamet ikke vurdert en slik implementasjon, men dette er et punkt som ses på som nyttig. I tillegg var det ikke tatt hensyn til mengden data i listen for å legge til et tilbud for en kunde. Her kunne man i stedet implementere to nedtrekklister, hvor man først velger tjeneste, og deretter velger underliggende varelinje. Nedtrekkliste for enhet og valuta var noe teamet først implementerte. Det var derimot ikke tydelig hvilke valg listen kunne bestå av. Dessuten ble disse lagret som en åpen verdi i faktureringsmotoren, og det ble dermed naturlig for teamet med en lik implementasjon i APC. Ønsket om å omdirigeres etter man oppretter en varelinje kan variere. Ønsker brukeren kun å legge til én varelinje, vil dette være hensiktsmessig. Teamet har derimot tatt hensyn til at en bruker kan opprette flere varelinjer på en gang, og det vil da ikke være gunstig å bli omdirigert hver gang. I tillegg får brukere en god tilbakemelding på at handlingen er utført, som indikerer at de kan navigere seg videre. Alle tilbakemeldingene kan tas med under videreutvikling, og vil i stor grad være enkle implementasjoner, ettersom det kun kreves små endringer i enkelte komponenter.

I etterkant av denne brukertesten ser teamet at brukertester på faktiske sluttbrukere er veldig nyttig. På denne måten blir tilbakemeldingene fra testene direkte knyttet til hvordan nettsiden skal brukes i praksis. For utenforstående kan det være vanskelig å forstå innholdet av nettsiden og dermed vanskeligere å finne feil og mangler. Tilbakemeldinger fra sluttbrukere inneholder dermed mer informasjon om den faktiske bruken av nettsiden, som er ønskelig. Teamet ser derfor verdien i en slik brukertest, og i retrospekt er det en type test som kunne blitt gjennomført flere ganger i løpet av prosjektet.

6 Konklusjon og videre arbeid

I dette avsluttende kapittelet vil konklusjoner bli trukket rundt sluttproduktet og de fire forskningsspørsmålene som ble definert. Konklusjonene bygger på resultatene som er presentert og diskusjoner rundt disse.

6.1 Konklusjon

Problemstillingen som først ble definert var «Hvordan lage en sikker, brukervennlig og dynamisk webapplikasjon som legger til rette for videreutvikling?». I oppgaven har utviklingsprosess og valg av teknologi blitt presentert og diskutert. Hovedmålet har gjennom dette vært å svare på denne problemstillingen. Problemstillingen er åpen, og for å konkretisere er det dermed blitt delt inn i mindre forskningsspørsmål. Disse besvares under basert på empiri og teori samlet inn i løpet av prosjektet:

1. Fører bruken av en identitetsleverandør til en sikker applikasjon på egenhånd?

    - Siden identitsleverandører som alle typer programvare er utsatt for dataangrep, kan ikke en identitesleverandør garantere en sikker applikasjon på egenhånd. Likevel tilbyr de løsninger som kan gjøre systemer sikrere. Det finnes flere ulike leverandører, og man bør være klar over funksjonalitetene som tilbys før man velger. Teamet erfarte at kombinasjonen av å justere brukertilgang selv, og å bruke en ekstern tredjepart kan være gunstig. Når det er sagt anbefaler teamet andre å gjøre en grundig vurdering over hva som er hensiktsmessig, før man avgjør hvordan man skal sikre applikasjonen sin.

2. Hva er fordelene og ulempene ved å bruke et komponentbibliotek for å utvikle en brukervennlig nettside?

    - Erfaringer utviklerne sitter igjen med etter bruk av komponentbibliotek er i stor grad positive. Fordelene er flere og resulterer i en utvikling som direkte bidrar til brukervennligheten, samtidig som den frigjør tid til andre brukervennlige aspekter. Ulempene er få og kan også unngås ved riktig bruk av bibliotekene. Med tanke på at alternativet er utvikling fra grunnen av, ser teamet store fordeler for å ta i bruk komponentbiblioteker.

3. Hvilke valg bør tas hensyn til når man utvikler et API som jobber sammen med flere eksterne tjenester?

    - Hvorvidt man skal samle alle kall til det API-et man utvikler bør vurderes basert på om det er sikkerhet eller ytelse som er i fokus. Videre bør man vurdere hvilken rekkefølge man oppdaterer tjenestene i. Avslutningsvis bør man også vurdere hvordan man skal håndtere mislykkede oppdateringer i et steg av prosessen.

4. Hvilke tiltak kan man ta for å legge opp til videreutvikling i et prosjekt?

    - Først og fremst kan man sette opp en prosjektstruktur som er ryddig og oversiktlig. Videre er god og sammenhengende dokumentasjon av kode og system et viktig punkt som gjør det enklere for nye utviklere å gjøre seg kjent med systemet. Man kan med fordel også velge å bruke kjente teknologier og rammeverk, ettersom dokumentasjon er lettere tilgjengelig. Til slutt ble det erfart at brukertester på sluttproduktet kan være gunstig for å ha klare på punkter som kan forbedres når videreutviklingen skal i gang.

Videre har utviklingen og implementasjonen foregått uten store hindringer. Basert på resultatene presentert i 4.1, diskusjonen i 5.1, samt kravene fremstilt i Vedlegg A Visjonsdokument, vil teamet konkludere med et vellykket prosjekt.

6.2 Videre arbeid

Avslutningsvis er det relevant å diskutere hva som kan være veien videre for det utviklede produktet. Til tross for et vellykket prosjekt er det muligheter teamet ser for å videreutvikle applikasjonen.

6.2.1 Brukervennlighet for mindre skjermer

Det er tatt enkelte hensyn til bruk av applikasjon på mobil og mindre skjermer. Det anses derimot som aktuelt å se på muligheten for å gjøre applikasjonen fullt funksjonell på skjermer av ulik størrelse. Hvorvidt dette er noe som bør prioriteres i første omgang, kan vurderes basert på konteksten applikasjonen brukes i.

6.2.2 Faktureringsmotor oppdaterer produktet

Som beskrevet i 4.1.1.2, og diskutert i 5.1.1.2 ble dette kravet ikke oppfylt, og er noe som kan bli implementert ved videreutvikling. Dette kan bli gjort ved en jobbplanlegger som utfører denne oppdateringen i jevne mellomrom.

6.2.3 Tilbakestille feiloppdateringer

Som det skrives om i 5.1.5, bør applikasjonen sørge for løsninger som sikrer lik data i APC og faktureringsmotor.

6.2.4 Paginering av lister

I 5.1.2 nevnes det at paginering av lister som har potensiale til å bli lange er noe man kan ta hensyn til i en eventuell videre utvikling. Applikasjonen har flere lister som vokser etter hvert som brukere legger inn nye data, blant annet listen over anbud eller spesialtilbud for kunder. Dersom listene blir veldig lange, kan det gå utover ytelsen på nettsiden og dermed ventetiden brukeren opplever. Om mengden data blir så stor at det går utover ytelsen ser teamet fordelen med å implementere paginering.

6.2.5 Tilbakemeldinger fra brukertester av sluttprodukt

Tilbakemeldingene fra brukertester av sluttprodukt som er presentert i 4.3.2 og diskutert i 5.3.2, er noe som kan vurderes under videreutvikling av prosjektet.

7 Referanser

[1] Wikimedia Foundation, Inc, «Single page application,» Wikipedia, 11 Mai 2022. [Internett]. Available: https://en.wikipedia.org/wiki/Single-page_application. [Funnet 14 Mai 2022].

[2] UxPin, «Studio by UXPin,» UXPin, [Internett]. Available: https://www.uxpin.com/studio/blog/ui-component-library/. [Funnet 03 05 2022].

[3] K. Bratbergsengen, «relasjonsdatabase,» Store norske leksikon, 30 april 2019. [Internett]. Available: https://snl.no/relasjonsdatabase. [Funnet 25 april 2022].

[4] IBM, «What is a Rest API?,» [Internett]. Available: https://www.ibm.com/cloud/learn/rest-apis. [Funnet 5. Mai 2022].

[5] Mozilla, «Http response stauts codes,» [Internett]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#client_error_responses. [Funnet 5. Mai 2022].

[6] F5, «Load Balancer,» [Internett]. Available: https://www.f5.com/services/resources/glossary/load-balancer. [Funnet 5. Mai 2022].

[7] Auth0, [Internett]. Available: https://jwt.io/introduction. [Funnet 5. Mai 2022].

[8] H. Haugerud, 22. August 2020. [Internett]. Available: https://snl.no/virtualisering_-_IT. [Funnet 9. Mai 2022].

[9] Basefarm, [Internett]. Available: https://basefarm.no/blogg/flytter-kode-fritt-med-docker-containerteknologi/. [Funnet 9. Mai 2022].

[10] University of Washington, «What is universal design?,» University of Washington, 4 September 2021. [Internett]. Available: https://www.washington.edu/doit/what-universal-design-0. [Funnet 6 Mai 2022].

[11] educative, «What are Norman's design principles?,» educative, [Internett]. Available: https://www.educative.io/edpresso/what-are-normans-design-principles. [Funnet 5 Mai 2022].

[12] uutilsynet, «WCAG sortert etter prinsipp,» Tilsynet om universell utforming av ikt, [Internett]. Available: https://www.uutilsynet.no/wcag-standarden/wcag-sortert-etter-prinsipp/713. [Funnet 6 Mai 2022].

[13] C. Drumond, «What is agile project management?,» Atlassian, [Internett]. Available: https://www.atlassian.com/agile/project-management. [Funnet 3 Mai 2022].

[14] C. Drumond, «Is the Agile Manifesto still a thing?,» atlassian, [Internett]. Available: https://www.atlassian.com/agile/manifesto. [Funnet 3 Mai 2022].

[15] C. Drumond, «What is Scrum?,» atlassian, [Internett]. Available: https://www.atlassian.com/agile/scrum. [Funnet 3 mai 2022].

[16] wrike, «Guide to Scrum Sprints,» wrike, [Internett]. Available: https://www.wrike.com/scrum-guide/scrum-sprints/. [Funnet 9 Mai 2022].

[17] M. Rehkopf, «What is a kanban board?,» atlassian, [Internett]. Available: https://www.atlassian.com/agile/kanban/boards. [Funnet mai 3 2022].

[18] Cathy, «The Kanban Method in IT development projects,» bocasay, 29 Juli 2020. [Internett]. Available: https://www.bocasay.com/kanban-method-it-development-projects/. [Funnet 9 Mai 2022].

[19] S. Grønmo, «kvalitativ metode,» Store Norske Leksikon, 3 November 2020. [Internett]. Available: https://snl.no/kvalitativ_metode. [Funnet 4 Mai 2022].

[20] E. H. Olsvik, «empiri,» Store Norske Leksikon, 7 November 2021. [Internett]. Available: https://snl.no/empiri. [Funnet 2 Mai 2022].

[21] R. Lynn, «What is Scrumban?,» planview, [Internett]. Available: https://www.planview.com/no/resources/guide/what-is-scrum/lkdc-what-is-scrumban/. [Funnet mai 4 2022].

[22] Agile Alliance, «Extreme Programming (XP),» Agile Alliance, [Internett]. Available: https://www.agilealliance.org/glossary/xp/#q=~(infinite~false~filters~(postType~(~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'xp))~searchTerm~'~sort~false~sortDirection~'asc~page~1). [Funnet 10 Mai 2022].

[23] A. Ivanovs, «Stackdiary,» Stackdiary, 19 Februar 2022. [Internett]. Available: https://stackdiary.com/front-end-frameworks/. [Funnet 12 Mai 2022].

[24] Microsoft, «TypeScriptlang,» Microsoft, [Internett]. Available: https://www.typescriptlang.org/. [Funnet 12 mai 2002].

[25] Material UI SAS, «Mui,» Material UI SAS., 2022. [Internett]. Available: https://mui.com/. [Funnet 12 Mai 2022].

[26] Learn Dapper, 21 Mai 2019. [Internett]. Available: https://www.learndapper.com/. [Funnet 11 Mai 2022].

[27] Statista, [Internett]. Available: https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/. [Funnet 11 Mai 2022].

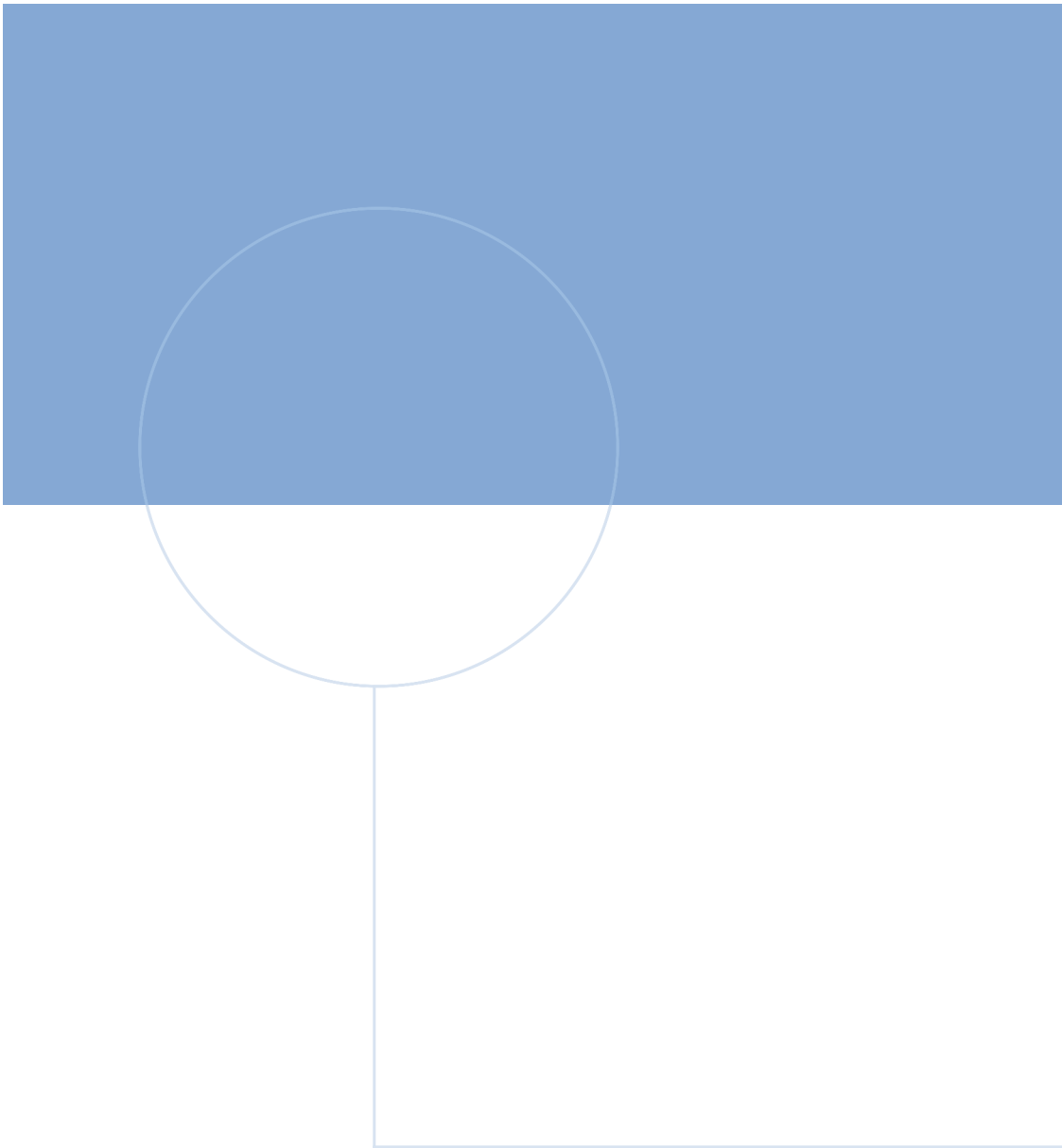
[28] Kubernetes, [Internett]. Available: https://github.com/kubernetes/kubernetes. [Funnet 12. Mai 2022].

[29] S. Williams, 21 Oktober 2020. [Internett]. Available: https://securitybrief.co.nz/story/average-person-has-100-passwords-study. [Funnet 14 Mai 2022].

[30] Beyond Identity, [Internett]. Available: https://www.beyondidentity.com/blog/measuring-password-fatigue. [Funnet 15. Mai 2022].

8 Vedlegg

- Vedlegg A – Visjonsdokument
- Vedlegg B – Kravdokumentasjon
- Vedlegg C – Systemdokumentasjon
- Vedlegg D – Prosjekthåndbok
- Vedlegg E - Brukertester



NTNU

Kunnskap for en bedre verden