Ørjan Trulsen & Frode Pedersen

Vessel Doc

Applikasjon for erstatning av papirdokumentasjon om bord i skip

Bacheloroppgave i Ingeniørfag (Data)
Veileder: Mikael Tollefsen

Mai 2020

Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Institutt for IKT og realfag

NTNU
Kunnskap for en bedre verden

Ørjan Trulsen & Frode Pedersen

Vessel Doc

Applikasjon for erstatning av papirdokumentasjon om bord i skip

Bacheloroppgave i Ingeniørfag (Data)
Veileder: Mikael Tollefsen
Mai 2020

Norges teknisk-naturvitenskapelige universitet
Institutt for IKT og realfag

NTNU
Kunnskap for en bedre verden

NTNU
Kunnskap for en bedre verden

TITTEL:
VESSELDOC - Paper replacement app

KANDIDATNUMMER(E):
10071, 10073

DATO:	EMNEKODE:	EMNE:	DOKUMENT TILGANG:
30.01.2020	IE303612	Bacheloroppgave	

STUDIUM:	ANT SIDER/VEDLEGG:	BIBL. NR:
Bachelorstudium, Data	102/8	

VEILEDER(E) :
Mikael Tollefsen, Hans Georg Schaathun

SAMMENDRAG:

I dagens løsning for kvalitetssikring/dokumentering om bord i skip blir skjemaene fylt ut med penn og papir. Formålet med denne rapporten er derfor å utforske ulike alternativer for en digital løsning av papirdokumentasjon ombord i skip, med mål om å utvikle den beste løsningen for brukerne.

For å løse denne problemstillingen blir det sett nærmere på ulike alternativer som inkluderer metoder for utførelse og tilgjengelige modeller/arkitekturer. Det blir tatt høyde for flere utfordringer, som mangel på offentlig nettverk om bord, enhetsstøtte for en alternativ løsning, og samtidig følge alle krav som burde høre med en slik løsning.

Gjennomføringen av dette prosjektet har resultert i et fungerende produkt/løsning som kan erstatte papirdokumentasjon innenfor den maritime industrien. I motsetning til bruk av penn og papir argumenter gruppen for at denne løsningen er raskere, sikrere og mer miljøvennlig.

Denne oppgaven er en eksamensbesvarelse utført av studenter ved NTNU i Ålesund.

Postadresse
NTNU i Ålesund
Postboks 1517
N-6025 Ålesund

Besøksadresse
Larsgårdsvegen 2
Internett
www.ntnu.no

Telefon
73 59 50 00
Epostadresse
postmottak@ntnu.no

Bankkonto
7694 05 00636
Foretaksregisteret
NO 974 767 880

Obligatorisk egenerklæring/gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

Du/dere fyller ut erklæringen ved å klikke i ruten til høyre for den enkelte del 1-6:		
1.	**Jeg/vi erklærer herved at min/vår besvarelse er mitt/vårt eget arbeid, og at jeg/vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.**	☑
2.	**Jeg/vi erklærer videre at denne besvarelsen:** • ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • ikke refererer til andres arbeid uten at det er oppgitt. • ikke refererer til eget tidligere arbeid uten at det er oppgitt. • har alle referansene oppgitt i litteraturlisten. • ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	☑
3.	**Jeg/vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§14 og 15.**	☑
4.	**Jeg/vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert i Ephorus, se Retningslinjer for elektronisk innlevering og publisering av studiepoenggivende studentoppgaver**	☑
5.	**Jeg/vi er kjent med at høgskolen vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens studieforskrift §31**	☑
6.	**Jeg/vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider**	☑

Publiseringsavtale

Studiepoeng: 20

Veileder: Mikael Tollefsen, Hans Georg Schaathun

Fullmakt til elektronisk publisering av oppgaven

Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven §2).
Alle oppgaver som fyller kriteriene vil bli registrert og publisert i Brage HiM med forfatter(ne)s godkjennelse.
Oppgaver som er unntatt offentlighet eller båndlagt vil ikke bli publisert.

Jeg/vi gir herved NTNU i Ålesund en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:	☑ja	☐nei
Er oppgaven båndlagt (konfidensiell)? (Båndleggingsavtale må fylles ut) - Hvis ja: **Kan oppgaven publiseres når båndleggingsperioden er over?**	☐ja ☑ja	☑nei ☐nei
Er oppgaven unntatt offentlighet? (inneholder taushetsbelagt informasjon. Jfr. Offl. §13/Fvl. §13)	☐ja	☑nei

Dato: 17.05.2020

FORORD

Både rapporten og prosjektet som er vist til her, er skrevet og utført av to dataingeniørstudenter hos NTNU i Ålesund. Selve oppgaven gir 20 studiepoeng, og markerer fullførelsen av vårt studium.

Vår motivasjon for å velge dette prosjektet kom fra tidligere erfaringer, da medlemmer av gruppen har jobbet innenfor den maritime industrien. Under denne tiden ble det oppdaget et behov for en slik løsning.

Vi vil gjerne få takke noen nøkkelpersoner, som ikke bare har bidratt til vår fullførelse av dette prosjektet, men samtidig vært med på å aktivt forme vårt studium hos NTNU Ålesund:

- **Universitetslektor Mikael Tollefsen hos NTNU i Ålesund.**
  Hovedveileder gjennom oppgaven, og vår tidligere lærer.
  Tusen takk for god oppfølgning og hjelp!

- **Førsteamanuensis Girts Strazdins hos NTNU i Ålesund.**
  Tusen takk for å at du er en slik sterk figur for instituttet for IKT og realfag. Du motiverer oss gjennom måten du brenner for faget ditt og evnen til å bry deg.

- **Alle våre tidligere lærere gjennom studiet.**
  Vi vil takke tidligere lærere for all deres hjelp.

- **Våre familier**
  Tusen takk for deres støtte, tålmodighet og motivasjonen dere har gitt oss.

INNHOLD

SAMMENDRAG

I dagens løsning for kvalitetssikring/dokumentering om bord i skip blir skjemaene fylt ut med penn og papir. Formålet med denne rapporten er derfor å utforske ulike alternativer for en digital løsning av papirdokumentasjon ombord i skip, med mål om å utvikle den beste løsningen for brukerne.

For å løse denne problemstillingen blir det sett nærmere på ulike alternativer som inkluderer metoder for utførelse og tilgjengelige modeller/arkitekturer. Det blir tatt høyde for flere utfordringer, som mangel på offentlig nettverk om bord, enhetsstøtte for en alternativ løsning, og samtidig følge alle krav som burde høre med en slik løsning.

Gjennomføringen av dette prosjektet har resultert i et fungerende produkt/løsning som kan erstatte papirdokumentasjon innenfor den maritime industrien. I motsetning til bruk av penn og papir argumenter gruppen for at denne løsningen er raskere, sikrere og mer miljøvennlig.

TERMINOLOGI

Begreper

Leider	Type stige i skip
Offiser	Overordnet om bord i skip
Synkronisering	Oppdatere informasjon mellom flere enheter.
Peer-to-peer	Kommunikasjon direkte mellom klienter i et nettverk.
Server	En maskin som i hovedsak kommuniserer og serverer informasjon til klienter.
Zegeba	Bedrift som tilbyr en løsning for utfylling av skjema.
Prosess	En serie med instruksjoner for å få en jobb gjort i en datamaskin.
Programvare	Filen som inneholder instruksjonene til prosessen.
Sprint	Et tidsrom der en rekke bestemte gjøremål skal utføres.
Produktkø	En prioritert liste over det som er nødvendig for produktet.
Backlog	Liste over gjøremål.
Inkrement	Øke eller legge til.
IP adresse	Adresse som gjør at datamaskin kan nå en annen innenfor et nettverk.
Variabel	Et symbol som kan holde en eller flere dataverdier som kan forandres.
Attributt	Egenskaper som legges ved metoder og datamodeller.
Datatype	Varianter av informasjon som boolean(true/false), integer(tall), string(tekst og karakterer).
Web	Et system som gjør informasjon tilgjengelig over internett.
Java EE	Java Enterprise Edition er et sett med spesifikasjoner som utvider java med funksjoner som webtjenester.
Bibliotek	Samling av funksjoner og klasser som kan forenkle programmering.
Commit	Oppdatere en endring i versjonskontrollsystemet git.
Pull	Hente oppdateringer i versjonskontrollsystemet git.
Kildekode	Forståëlig versjon av programvare som kan redigeres.
Brukergrensesnitt	Hvordan en applikasjon framstår for brukeren.
Klient	En applikasjon som kommuniserer over et nettverk og blir brukt av en bruker.
Brukersystem	Et system som autentiserer brukere og gir informasjon om brukere av en tjeneste.
Rammeverk	Et programvaremiljø som har standarder som oppbygging av applikasjoner og mer.
Backend	Den delen av applikasjonen som gjør tunge kalkulasjoner og kommuniserer med databasen. I dette prosjektet er serverapplikasjonen backend.
Frontend	Den delen av applikasjonen som brukeren kan samhandle med.
Stackoverflow	Et nettsted der utviklere kan spørre og svare på spørsmål.
Databasetabell	Et sett med verdier i rader og kolonner der kolonner er et bestemt datatype med regler, og rader er en samling med verdier av disse kolonnene som anses som et objekt.
Maven	Et verktøy som behandler biblioteker og bygging av programvare prosjekt.
Operativsystem	Hovedprosessen i en datamaskin som fordeler hardware-ressurser til andre prosesser.
Virtuell Maskin	En programvare som kan kjøre et operativsystem.

Shell skript	En rekke kommandolinje-kommandoer som blir utført fra en fil.
Vertsnavn	Også kjent som domene. Dette erstatter IP-adresser med navn og et toppdomene.
Cache-lagring	Et område i lagringsminnet som er ment til å holde på midlertidig informasjon.
Autentisering	Bekrefte at en bruker er det den utgir seg for å være.
TIOBE	Programmeringssammfunsindeks

Forkortelser

SMS - Sikkerhetsstyringssystemer
ISM - International Safety Management
DNS - Domain Name Server
DBMS - DataBase Management System
SQL - Structured Query Language
RDBMS - Relational DataBase Management System
OOP - ObjektOrientert Programmering
JDK - Java Development Kit
SDK - Software Development Kit
JRE - Java Runtime Environment
REST - Representational State Transfer
HTTP - HyperText Transfer Protocol
URI - Uniform Resource Identifier
UI - User Interface
UX - User Experience
JSON - JavaScript Object Notation
CRUD - Create, Read, Update, and Delete
API - Application Programming Interface
VS -Visual Studio
MVC - Model-View-Controller
JWT - JSON Web Token
RFC - Request For Comments
MAC - Media Access Control
HMAC - Hash based MAC
RSA - Ron Rivest, Adi Shamir og Leonard Adleman (oppfinnere av RSA krypteringsalgoritme)
ECDSA - Elliptic Curve Digital Signature Algorithm
RAM - Random Access Memory
CPU - Central Processing unit
UML - Unified Modeling Language
mDNS - Multicast DNS
SAML - Security Assertion Markup Language
PDF - Portable Document Format
XML - Extensible Markup Language

1 INNLEDNING

Sikkerhet er, og vil alltid være i høyt fokus om bord i skip. Dette er på bakgrunn av at arbeidet deres ofte innebærer høy risiko, hvor sannsynligheten for en ulykke er høyere sammenlignet med andre yrker. Det er flere tiltak som er satt til verks for å ivareta denne sikkerheten, blant annet gjennom rutiner for kvalitetssikring. Med kvalitetssikring menes det ulike prosedyrer knyttet til arbeidet som dokumentering og/eller rapportering. I dagens løsning blir majoriteten av denne dokumenteringen/rapporteringen utført ved hjelp av penn og papir. Det vil si at alle skjemaene de skal bruke til dokumenteringen blir skrevet ut, fylt ut på papir, og lagret i permer om bord i skipet. Ikke nok med at denne løsningen er både tungvint og ineffektiv, den lider også av lav sikkerhet. Dette er et resultat av hvordan skjemaene blir behandlet og oppbevart i nåværende løsning. Målet blir derfor i denne oppgaven å utrede en bedre løsning enn den nåværende.

Opp igjennom tidene har det vært store skipsfartsulykker hvor de har avdekket store mangler i administrasjonen om bord i skip og i rederienes landsorganisasjoner. Skritt for skritt har de prøvd å øke sikkerhetskulturen innenfor skipsfarten ved å vedta flere store konvensjoner. Disse konvensjonene innebærer ulike krav som må opprettholdes for å bidra til å øke sjøsikkerheten. Et av det store kravene fra den internasjonale skipsfartsorganisasjonen er funksjonskravet om sikkerhetsstyringssystemer (SMS). Det ble fastsatt i ISM-koden at alle rederier skal utvikle, gjennomføre og vedlikeholde et SMS.[1] Kjernen i dette er gjerne dokumentering, rapportering og analysering. Gjennom å digitalisere og erstatte papirdokumentasjon om bord i skip kan man gjøre sikkerhetsstyringen bedre på flere måter. Blant annet kan digitalisering

[1] https://lovdata.no/dokument/SF/forskrift/2014-09-05-1191/KAPITTEL_1#KAPITTEL_1

gjøre det mer effektivt å fylle ut skjemaer, sikrere på grunn av sikkerhetskopiering, og mer oversikt for administrasjon.

For å gi et praktisk eksempel kan man dra frem en typisk arbeidsdag for en av dekksarbeiderne på et skip. En matros skal opp i en kran for smøring. Her må matrosen utnytte seg av en leider for å komme seg opp da det er flere meter over bakken. Arbeidet blir sett på som "arbeid-i-høyden" siden det er en fare for å falle ned og skade seg. Før noe arbeid starter har matrosen nødt å fylle ut to individuelle skjemaer, arbeidstillatelse og en "toolbox-talk" (risikoanalyse). Det ligger med to slike eksempel-skjemaer som hvert sitt vedlegg til rapporten. Matrosen fyller ut skjemaene med penn og papir, og må deretter på broen for å få arbeidstillatelsen godkjent og signert av en offiser. Ved bruk av en digital løsning kan hele denne prosedyren utføres mer effektivt og tryggere på f.eks. et nettbrett eller på arbeidernes egne smarttelefon.

Et nøkkelproblem ved å ha en digital løsning om bord i skip er begrenset tilgang til internett. Da skipene er mange mil ut i havet over lengre perioder kan det medføre til mangel på internett eller lav internetthastighet. Dette er fordi de eneste måtene for tilgang til internett er å få kontakt med mobilnettverk fra kysten, eller satellitter som er kjent for å ha begrenset overføringshastighet. I tillegg til den begrensede hastigheten, blir dette fordelt på arbeiderne ombord i båten, noe som gjør internett-hastigheten enda tregere. Selv om internettet er begrenset så er det fortsatt stabil tilgang til det lokale nettverket i båten. Det gjør det mulig for en digital løsning å synkronisere ved hjelp av dette nettverket. Det er et par alternativer for synkroniserings-metoder. Det ene alternativet er å synkronisere direkte mellom de mobile enhetene (peer-to-peer). Et annet alternativ til direkte synkronisering er synkronisering med en server. Siden det ofte er sensitiv informasjon som overføres må en ta høyde for dette når man velger synkroniserings-metode.

I denne rapporten blir det utforsket ulike muligheter for digital løsning av papirdokumentasjon ombord i skip, med formål om å utvikle den beste løsningen for brukerne. Her blir det tatt høyde for mangel på offentlig nettverk ombord, og hvordan dette kan bli løst. Til slutt blir det sett på utbygging av den digitale løsningen som kan gjøre SMS systemet til en sikrere, og mer oversiktlig løsning.

2 TEORETISK GRUNNLAG

2.1 Eksisterende løsninger

Det finnes allerede noen eksisterende løsninger på lignende problem, hvor et av de nærmeste er utviklet av det lokale firmaet Zegeba. Gjennom prosjektet har Zegeba lansert en ny løsning som de kaller for "Offshore Forms" som er siktet mot samme marked som oppgaven tar for seg. I sin løsning har Zegeba to forskjellige applikasjoner, en for utfylling og en for oppretting av skjemaer. I denne løsningen vil dette utføres i en og samme applikasjon så brukerne ikke trenger å bytte applikasjoner for oppretting av nye skjemaer. Siden det ikke er noe mer spesifikk informasjon om produktet til Zegeba, ligger det ikke mye til grunn for ytterlige sammenligning med denne løsningen.

2.2 Smidig utvikling (agile metode)

I oppgaven er det satt fokus på å jobbe agilt. Det handler om å jobbe mer smidig og kunne raskt respondere på endringer. Metodikken er en tilnærming for organisering og arbeidsmetoder innad i prosjektet, som tar

hensyn til dagens marked, hvor endringer skjer raskt. Det finnes flere ulike agile metoder å velge mellom som ofte deler mange av de samme elementene, og har samme filosofi bak. Den agile utviklingsmetodikken baserer seg på verdiene og prinsippene i det agile manifestet:

"Vi finner bedre måter å utvikle programvare på
ved å gjøre det selv og ved å hjelpe andre med det.
Gjennom dette arbeidet har vi lært oss å verdsette følgende:
***Personer og samspill** fremfor prosesser og verktøy*
***Programvare som virker** fremfor omfattende dokumentasjon*
***Samarbeid med kunden** fremfor kontraktsforhandlinger*
***Å reagere på endringer** fremfor å følge en plan"* [2]

I manifestet konstaterer de også at selv om punktene til høyre har verdi, så verdsetter de punktene til venstre enda høyere.

2.3 Scrum

Scrum ble først presentert i 1995 av Jeff Sutherland og Ken Schwaber på Oopsla-konferansen i Austin, Texas (USA). Det har siden blitt tatt i bruk av en enorm mengde programvareselskaper over hele verden. I dag anerkjennes det som det mest anvendte rammeverket for smidig/agile programvareutvikling, hvor over 1000 bøker har blitt publisert om det. Metoden har imidlertid også blitt vellykket brukt innen andre domener, f.eks. produksjon, markedsføring, drift og utdanning. [3]

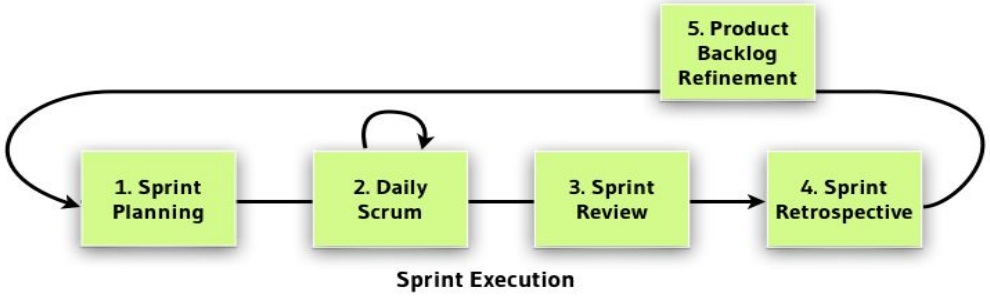
2.3.1 Scrum hendelser/møter

Scrum inneholder noen foreskrevne hendelser som brukes for å skape regelmessighet, og for å minimere behovet for møter som egentlig ikke er definert i Scrum. Alle hendelser er tidsbestemte på forhånd. Når en sprint begynner, er varigheten fast og kan verken forkortes eller forlenges. De forskjellige hendelsene er som følger:

- Sprint

[2] https://agilemanifesto.org/iso/no/manifesto.html
[3] https://en.wikipedia.org/wiki/Agile_software_development

- Sprint planlegging
- Daglig scrum
- Sprint gjennomgang (review)
- Sprint retroperspektiv



(Figur 1: Oversiktlig figur over scrum hendelser/møter [4])

En **Sprint (iterasjon)** definerer oppfinnerne som den grunnleggende utviklingsenheten i Scrum.[5] Det er en tidsboks/iterasjon på gjerne en måned eller mindre hvor det blir utviklet et "ferdig", brukbart, potensielt produktinkrement. Hver sprint starter med et sprint-planleggingsmøte hvor man etablerer et sprintmål og tar for seg de nødvendige elementene fra produktkøen. Teamet vil bli enige om disse elementene å omgjøre dem til mindre gjøremål for sprintkøen inkludert en prognose for arbeidet som kreves. I slutten av hver sprint vil man ha en sprintgjennomgang og retroperspektiv, der man går gjennom fremdriften som skal vises for interessentene, og identifisere lærte materialer og forbedringer for senere sprinter. [6]

Sprint planleggingsmøte skjer i starten av hver nye sprint. Dette er et møte mellom produkteier og scrum utviklingsteamet. Her skal det være en gjensidig diskusjon hvor de skal bli enige om omfanget av arbeidet som er ment å bli utført i løpet av denne sprinten. Scrum Master sitter på ansvaret for å sikre at arrangementet finner sted, og at de fremmøtte

[4] https://staragile.com/scrum-meetings
[5] Schwaber, Ken (February 1, 2004). Agile Project Management with Scrum. Microsoft Press. ISBN 978-0-7356-1993-7
[6] Sutherland, Jeff (October 2004). "Agile Development: Lessons learned from the first Scrum".

forstår hensikten. Møtet er forøvrig tidsbegrenset hvor den anbefalte varigheten er fire timer for en to ukers sprint. I løpet av denne tiden skal møtedeltakerne dratt elementer fra produktkøen, brutt de ned i mindre gjøremål og samtidig grovestimert hvor mange "storypoints" hvert gjøremål burde være. De skal med andre ord definere en sprintkø/backlog, og et overordnet mål for sprinten. [7]

Daglig scrum, bedre kjent som "Stand Up møter" (skal helst stå oppreist under møtet), arrangeres hver dag gjennom sprinten. Det er et internt møte for utviklingsteamet, og de er selv ansvarlige for å gjennomføre det med sin egen valgte struktur. Møtet er på samme måte som de andre hendelsene, tidsbegrenset. I løpet av maks 15 minutter skal hvert medlem av utviklingsteamet rapportere kort til hverandre innenfor teamet's valgte struktur. Strukturen burde inkludere spørsmål som:

- Hva jobbet jeg med i går som bidro til at teamet møter sprintmålet?
- Hva har jeg planlagt å jobbe med i dag?
- Ser jeg noe hinder som kan hindre meg i å utføre mine gjøremål? [8]

Sprint review/gjennomgang holdes på slutten av en sprint for å inspisere inkrementen og tilpasse produktkøen om nødvendig. I det tidsbegrensede møtet samarbeider Scrum utviklingsteamet og interessentene om hva som ble gjort og eventuelt ikke gjort gjennom sprinten. Man har som hensikt å få uformelle tilbakemeldinger fra produkteier og andre som deltar i møtet. Dette er slik fordi gjennomgangen skal virke som en naturlig slutt på sprinten og for å fremme samarbeidet. [7]

Sprint retroperspektiv er en mulighet for Scrum utviklingsteamet til å inspisere seg selv og sammen lage en plan for forbedringer som skal

[7] Ken Schwaber; Jeff Sutherland. "The Scrum Guide" Scrum.org.
[8] "What is a Daily Scrum?". *Scrum.org*.

presumeres til neste sprint. Dette møtet skal skje etter sprint review og før neste sprint planlegging. [7]

2.3.2 Scrum artefakter

Scrum artefakter representerer arbeid/verdier for tilføring av gjennomsiktighet og muligheter for inspeksjon og tilpasning. Artefakter definert av Scrum er spesielt designet for å maksimere flyt av nøkkelinformasjon slik at alle har samme forståelse av artefakten.[9] Det er tre hoved artefakter i Scrum:

- "Product backlog"/produktkø
- "Sprint backlog"/sprintkø
- "Increment"/inkrement

Ifølge "The Scrum Guide" [10], som er utviklet av selve oppfinnerne av scrum, forklarer de forskjellige artefaktene på en slik måte:

Produktkøen er en prioritert liste over alt som er kjent for å være nødvendig i produktet. Det er den eneste kilden til krav over endringer som skal utføres ved produktet. Produkteieren er kjent som den ansvarlige for køen, inkludert dens innhold, tilgjengelighet og bestilling. Køen er aldri sett på som fullstendig, og det er i den tidligste utviklingen av den at man legger de opprinnelig kjente og best forståtte kravene. Produktkøen utvikler seg etter hvert som produktet og miljøet det skal brukes i utvikler seg. Den er dynamisk, og vil hele tiden endres for å identifisere hva produktet trenger. Hvis et produkt eksisterer, eksisterer også en produktkø.

Sprintkøen er et sett med produktkø-elementer("user-stories/bugs") som er valgt for en spesifikk sprint, pluss en plan for å kunne levere produktets

[9] https://www.scrumguides.org/
[10] https://www.scrumguides.org/scrum-guide.html

inkrement og samtidig realisere sprint målet. Sprintkøen er en prognose fra utviklingsgruppen om hvilken funksjonalitet som vil være i neste inkrement og arbeidet som trengs for å kunne levere den funksjonaliteten til en "ferdig" inkrement. Køen synliggjør alt arbeidet som utviklingsteamet sammen har identifisert som nødvendig for å nå sprint-målet.

Inkrement er summen av alle elementene fra produktkøen som er fullført under en sprint. Dette inkrementet skal fungere uavhengig av om produkteieren bestemmer seg for å gjøre den tilgjengelig for brukerne.

2.3.3 Scrum roller

Det er tre definerte roller innenfor Scrum-rammeverket:

- Scrum master
- Produkteier
- Utviklingsteamet

Disse rollene har hvert sitt sett med ansvarsområder. At ansvarsområdene blir fulgt opp og gjennomført er viktig for at det blir et suksessfullt resultat av prosjektet. I tillegg er det nødvendig at arbeiderne med de forskjellige rollene jobber tett sammen for at arbeidet blir fullført så effektivt som mulig.

Scrum master er ansvarlig for at Scrum blir forstått og fulgt på en skikkelig måte. Dette overholder master ved å sikre at Scrum-teamet følger både scrum-teorien, sin praksis og sine regler. Dette er den mest misforståtte rollen i Scrum. Scrum masteren har ikke noen ledelse

autoritet og eier ikke en prosjektlederrolle. Det er rett og slett en tilrettelegger med en rekke andre ansvarsområder som:

- Å hjelpe produkteieren med å vedlikeholde produktkøen på en måte som sikrer at teamet kan kontinuerlig gjøre framgang.
- **Tilrettelegger** for teamarrangementene.
- Fremme selvorganisasjon i teamet.
- Hjelpe Scrum-teamet med å unngå/fjerne hindringer for deres fremgang.
- Utdanne sentrale interessenter om Agile og Scrum-prinsipper.[11]

Produkteieren representerer produktets interessenter og kundens stemme. Rollen fokuserer på *hva* og ikke *hvordan,* og på samme måte som de andre rollene har produkteieren et sett med dedikerte ansvarsområder:

- 'Singlehanded' ansvarlig for håndtering av produktkøen:
    - Prioriterer elementer fra produktkøen.
    - Forsikre seg om at produktkøen er synlig, gjennomsiktig og tydelig for alle.
- Siste dommer i kravspesifikasjons-spørsmål (tar siste beslutning).
- Optimalisering av verdien i arbeidet utviklingsteamet utfører.

Utviklingsteamet (Scrum dev team) består av fagpersoner som utfører arbeidet med å levere et potensielt inkrement av et "ferdig"-produkt på slutten av hver sprint. [12] Teamet har alt fra tre til ni medlemmer og er selvorganiserende. Ingen ledelse - eller mer som, ledelse utvikler seg gjennom teamet.

---

[11] Drongelen, Mike van; Dennis, Adam; Garabedian, Richard; Gonzalez, Alberto; Krishnaswamy, Aravind (2017). Lean Mobile App Development. Birmingham, UK: Packt Publishing Ltd. p. 43. ISBN 9781786467041.

[12] Morris, David (2017). Scrum: an ideal framework for agile projects. In Easy Steps. pp. 178–179. ISBN 9781840787313. OCLC 951453155

Ramakrishnan R, Gehrke J. Database management systems. McGraw Hill; 2000.

2.4 DNS-Service Discovery

DNS basert Service Discovery er en metode som lar klienter motta en liste over tjenester og knytte disse til vertsnavn ved hjelp av standard DNS tjenester. En av disse tjenestene kalles Multicast DNS som det vil være fokus på i dette prosjektet.

2.4.1 Multicast DNS

Multicast DNS er en nettverksprotokoll som forbinder vertsnavn med IP adresser.[13] Det gjør det mulig å oppdage tjenester uten å vite IP adressen til tjeneste-verten. Denne protokollen benyttes i mindre nettverk som ikke har en egen DNS server.

Multicast DNS er også en zero-configuration tjeneste, som betyr at det er et sett med teknologier som gir et brukbart nettverk uten at det er behov for manuell konfigurering.

2.5 Databasesystem

Et databasesystem består gjerne av to deler, databasehåndteringssystemet (DBMS), og selve databasen. En database er en strukturert samling av data, generelt lagret og elektronisk tilgjengelig fra et datasystem. [14] Basen blir vanligvis styrt av det nevnte databasehåndteringssystemet hvor informasjonen er organisert slik at den lett kan nås, administreres, oppdateres og kontrolleres. Det finnes flere ulike sorter databasesystemer, der den vanligste er basert på relasjonsmodellen.

2.5.1 Databasehåndteringssystem

I et databasesystem er databasehåndteringssystemet kjent som den sentrale programvaren. Systemet har ansvar for å ta seg av opprettelse,

[13] "RFC 6762 - Multicast DNS - IETF Tools." https://tools.ietf.org/html/rfc6762. Accessed 17 Mar. 2020.
[14] https://en.wikipedia.org/wiki/Database

endringer og lesing i databasen. Et slikt system kommuniserer oftest med et type spørrespråk (SQL).

2.5.2 SQL

Structured Query Language (SQL) er et domenespesifikt språk som brukes i programmering med databaser. Språket er designet for å håndtere data som holdes i et relasjonsdatabase-styringssystem (RDBMS). Det er spesielt nyttig i håndteringen av strukturert data, dvs. data som inneholder forhold mellom entiteter og variabler. [15]

2.6 Objektorientert programmering

Objektorientert programmering er et programmeringsparadigme basert på konseptet "objekter", som kan inneholde data, i form av felt (ofte kjent som attributter eller egenskaper), og kode, i form av prosedyrer (ofte kjent som metoder). [16] For å forstå dette paradigmet finnes det to konsepter som er nødvendige å forstå først: **objekter** og **klasser**. Disse danner basisen for all programmering i objektorienterte språk.

Objekter er forekomster av klasser. Vi kan ha mange objekter av hvilken som helst klasse. Gjennom klassedefinisjonene kan vi definere metoder til objektene som vi bruker for å kommunisere med de.

Klasser representerer det generelle begrepet om ting hvor de beskriver typen objekt. Objektene representerer individuelle instanser av klassen.

2.6.1 Konsept innenfor OOP

Foruten de tidligere nevnte konseptene som klasser og objekter, eksisterer det flere viktige konsepter innenfor objektorientert programmering. De hjelper oss med å produsere levedyktige produkt både i form av utvikling, vedvarende vedlikehold og testing. De fire viktigste er:

[15] https://en.wikipedia.org/wiki/SQL
[16] https://en.wikipedia.org/wiki/Object-oriented_programming

Innkapsling

Med innkapsling menes det å avgrense hva slags datatyper og hvilke behandlingsregler som gjelder for en bestemt klasse av objekter. Hvis en klasse ikke får tilgang til interne objektdata, fordi de er kun tillatt tilgang gjennom metoder, gir dette oss en sterk form for abstraksjon eller dataskjuling. Det er lagt til rette slik for å hindre utvendig interferens og misbruk.

Polymorfi

Polymorfi betyr at prosedyrer kan gjøres gjeldende for objekter hvis nøyaktige form eller type kan variere. Det er sett på som en annen type abstraksjon som forenkler kode utenfor klassens hierarki og muliggjør sterk separasjon av mulige bekymringer. [17]

Abstraksjon

Abstraksjon er en av løsningene som brukes for å gjøre et komplekst problem enklere å håndtere, både med tanke på forståelse og utførelse. Det handler om å fjerne unødvendige detaljer for å kunne fokusere på detaljer av en større betydning. Flere av de andre nevnte konseptene gir en form for abstraksjon.

Arv

I OOP er arv sett på mekanismen i å basere et objekt eller klasse på et annet objekt eller klasse. Siden klasser dannes hierarkisk, åpner det for underordnede klasser å arve egenskapene til de overordnede. F.eks. kan klassene "motorsykkel" og "bil" arve egenskapene til overklassen "kjøretøy".

[17] https://snl.no/objekt_-_IT

2.7 Java

Java er et sett med dataprogramvare og spesifikasjoner utviklet av James Gosling hos Sun Microsystems, som senere ble anskaffet av Oracle Corporation, som tilbyr et system for utvikling av applikasjonsprogramvare og muligheter for å distribuere det i IT-miljøer på tvers av plattformer. [18]

2.7.1 Programmeringsspråket

Java er et programmeringsspråk til generell bruk som er klassebasert, objektorientert og designet for å ha så få implementeringsavhengigheter som mulig.[19] De ble kunngjort i november 2006 at selskapet ville frigi Javakoden som åpen kildekode, og har siden den gang blitt verdens mest populære og brukte programmeringsspråk.

2.7.2 Java Development Kit (JDK) / SDK

Java Development Kit er en samling av programvareutviklingsverktøy for Javautviklere. Det er java sitt svar på en såkalt SDK(samling av utviklingsverktøy) som tilrettelegger opprettelse av applikasjoner ved bruk av kompilator, "debugger" og kanskje et programvare-rammeverk. Det er en overordnet versjon av JRE, og er sammen med språket et av de mest brukte SDK'ene.

2.7.3 Java Runtime Environment (JRE)

Java Runtime Environment er en fritt tilgjengelig programvaredistribusjon med innhold som dekker det nødvendige for å kunne kjøre et Java program. Dette inkluderer en frittstående JVM, standardbiblioteket til Java (JVC) og et konfigurasjonsverktøy.

[18] https://en.wikipedia.org/wiki/Java_(software_platform)
[19] https://www.oracle.com/java/

2.7.4 Java Virtual Machine (JVM)

En Java Virtual Machine er en abstrakt (virtuell) maskin som gjør det mulig for en datamaskin å kjøre Java-programmer; så vel som programmer, skrevet på andre språk, som er kompilert til Java bytecode.

2.8 Flutter

Flutter en en open-source brukergrensesnitt SDK laget av Google. Den brukes til å utvikle applikasjoner for Android, iOS, Windows, Mac, Linux, Google's Fuchsia og Web. [20] Verktøysettet er kjent for å kunne bygge pene, native kompilerte applikasjoner fra en og samme kodebase. Med Flutter legger Google vekt på tre hovedmål:

- Rask utvikling
- Ekspressivt og fleksibelt brukergrensesnitt
- Native ytelse

2.8.1 Dart

En flutter-app skrives i Dart programmeringsspråket. Dart er et klientoptimalisert programmeringsspråk for apper på tvers av plattformer, optimalisert for å bygge brukergrensesnitt. Språket er objektorientert, klassebasert med integrert garbage-kollektor(se kapittel 2.6) hvor syntaksen ligner C-stil. [21]

2.8.2 Widgets

Alt i flutter er en widget. Widgets er de grunnleggende byggesteinene i en flutter-app's brukergrensesnitt. Den sentrale ideen er at du bygger brukergrensesnittet ditt ut av en samling av widgets. Alle widgetene blir en del av en struktur (widget-tre) som representerer hvordan widgetene våre er organiserte. [19] Med andre ord, der hvor nettsider har en DOM, har flutter et widget-tre.

[20] https://flutter.dev/
[21] https://dart.dev/

2.8.3 Hot Reload

Hot reload er en av de beste funksjonene i rammeverket flutter. Ved en såkalt hot reload gir det utviklerne mulighet for å kunne ha en applikasjon kjørende, og samtidig utføre endringer i applikasjonen uten å måtte kompilere på nytt. Utviklere kan øyeblikkelig se endringene de har gjort, uten å miste tilstanden til applikasjonen.

2.9 Tilstandsløs programmering

Tilstandsløs programmering er det underliggende arkitektoniske prinsippet av webben. Det blir gjerne kalt for RESTful programmering, hvor REST står for Representational State Transfer. REST beskriver alle enkle grensesnitt som overfører data over et standardisert grensesnitt (for eksempel HTTP), uten et ekstra meldingslag. Det er en stil - ikke et verktøysett - som representerer et sett med designregler for å lage statsløse tjenester som igjen blir sett på som *ressurser*. Hver av ressursene kan identifiseres med sine unike URIer. [22] En klient kan da få tilgang til en ressurs ved å bruke URI og et standardisert sett med metoder. Det igjen viser til det fantastiske med nettet hvor faktumet er at klienter og servere kan samhandle på komplekse måter, uten at klienten vet noe på forhånd om serveren og ressursene den verter.

2.9.1 Spring Boot

Spring Boot er et java-basert rammeverk med åpen kildekode som brukes for å lage ulike typer mikrotjenester. Mange forbinder Spring Boot med Java EE, men i perspektiv er Java EE mer klassifisert som et verktøy hvor Spring Boot er gruppert som et fullverdig rammeverk. Spring Boot er kjent som en konvensjons-over-konfigurasjonsløsning for å lage frittstående, produksjonsbaserte applikasjoner som du kan bare "kjøre". Det er forhåndsinnstilt med beste konfigurasjon, fra Spring-teamet sitt

[22] https://docs.oracle.com/cd/E24329_01/web.1211/e24983/overview.htm#RESTF105

synspunkt, ved hjelp av Spring-plattformen og tredjepartsbiblioteker. Dette skal gjøre igangsettelse mindre smertefull, siden de fleste Spring Boot-applikasjonene trenger veldig lite konfigurasjon.[23]

2.10 Versjonskontroll

Versjonskontroll er et system som registrerer endringer i en fil eller sett med filer over tid, slik du kan huske spesifikke versjoner senere. [24] Behovet for en logisk måte å organisere og kontrollere revisjoner på har så å si eksistert like lenge som skrivingen har eksistert, men ble rask viktigere og ytterligere komplisert da databehandlings-tiden startet. For programvareutviklere er versjonskontrollsystemer nesten uvurderlige, hvor et slikt system i første omgang beskytter kildekoden fra både katastrofale, og tilfeldige forringelser av menneskelige feil og utilsiktede konsekvenser. [25] Ikke nok med dette, men bruken av et slikt system gir oss også muligheter for å:

- Gå tilbake i historien for å sjekke forandringer over tid.
- Tilbakestille filer eller hele prosjektet til en tidligere versjon/utgave.
- Se hvem som sist har forandret på noe, både når og eventuelt hva de har gjort.
- Bruke systemet for å kunne spore/måle diverse resultater, både med tanke på tidsbruk, arbeidsmengde, etc.

Det finnes flere typer forskjellige systemer innen versjonskontroll der noen av de viktigste er:

- **Sentralisert versjonskontroll**
    - Enkel "sentral" kopi av prosjektet på en server og "committer" endringene dine til denne sentrale kopien.
    - Du "puller" filene du trenger, men du har aldri en full kopi av prosjektet lokalt.

[23] https://spring.io/projects/spring-boot
[24] https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control
[25] https://www.atlassian.com/git/tutorials/what-is-version-control

- **Distribuert versjonskontroll**
  - I et slikt versjonskontrollsystem stoler man ikke på en sentral server for å lagre alle versjonene av prosjektets filer. I stedet kloner man en kopi av en "repository" lokalt slik at man har tilgang til hele historien av prosjektet. Dette betyr at i steden for å utføre en "checkout" av av en liten del av kildekoden, kloner man heller hele repositoryen.
  - To vanlige distribuerte versjonsstyringssystemer er Git og Mercurial. [26]

2.10.1 Git

Git er et gratis og åpen-kildekode distribuert versjonskontrollsystem som er designet for å håndtere alt fra små til veldig store prosjekter med hastighet og effektivitet. Sett fra et programvareutviklingsperspektiv er det designet for å spore endringer i kildekoden. Versjonskontrollsystemet ble i sin begynnelse utviklet av mannen bak Linux, Linus Torvalds. [27]

2.11 Digitale signaturer

En digital signatur er et matematisk skjema for å verifisere autentisiteten av digitale meldinger eller dokumenter. En gyldig digital signatur, der forutsetningene er tilfreds, gir mottakeren veldig sterk grunn til å tro at meldingen ble opprettet av en kjent avsender, og at meldingen ikke har blitt tuklet med under transport. [28] Digitale signaturer brukes ofte når det skal implementeres elektroniske signaturer, hvor de elektroniske dataene krever en eller annen form for signatur. Ifølge Adobe er denne signaturtypen faktisk kjent for å være noe av det mest avanserte og sikreste typene av elektroniske signaturer som finnes. De nevner også at

---

[26] https://confluence.atlassian.com/get-started-with-bitbucket/types-of-version-control-856845192.html

[27] https://git-scm.com/

[28] Paul, Eliza (12 September 2017). "What is Digital Signature – How it works, Benefits, Objectives, Concept".

man kan bruke dem for å overholde selv de strengeste lovene og forskriftene gitt av de høye nivået av sikkerhet fra signaturen. [29]

2.12 Brukergrensesnittdesign (UI design)

2.12.1 Don Norman's prinsipper for interaktivt design

Når det kommer til designprinsipper ved utforming av grafiske brukergrensesnitt har man mange alternativer tilgjengelig. Et av de mest kjente er Don Norman sine seks prinsipper for et godt interaktivt design:

Visibility (synbarhet)

Ved fokus på høy synbarhet av de ulike funksjonene i et grafisk brukergrensesnitt sikrer man at brukerne finner funksjonene, og dermed bruker produktet mer som planlagt. Det handler mye om prioritet over hvilke funksjoner som kommer til å bli mest brukt/er mest brukt som gjerne skal være lettere synlig/tilgjengelig.

Feedback (tilbakemelding)

Tilbakemelding ved en hendelse er viktig så brukeren forstår at noe har skjedd eller skjer. Dette kan utføres gjennom forskjellige metoder for ulike type hendelser. Som eksempel kan man bruke "switches" når noe skal på og av. Ved å endre tilstanden på bryteren gir man tilbakemelding til brukeren om at noe har skjedd. Dette kan også gjøres når man utfører asynkrone kall til serveren. Oppdaterer man brukergrensesnittet ved å vise en prosessindikator, vil det føre til at brukeren forstår at noe skjer i bakgrunnen.

Constraints (begrense tilgang)

Noe som kan være logisk å følge i et brukerdesign kan være å tilføre visse begrensninger for mulig funksjonalitet. Dette simplifiserer prosessen for brukeren, og heller gir dem klare tegn for hva man skal kunne gjøre. Om

[29] https://acrobat.adobe.com/no/no/sign/capabilities/digital-signatures-faq.html

designet blir for avansert, med for mye funksjonalitet, kan dette føre til at brukerne må ha opplæring eller at de ikke forstår hvordan de bruker produktet.

Mapping (forhold mellom element og effekt)

Med fokus på korrekt mapping så tenker man på forholdet mellom et element og effekten av å aktivere det elementet. Som tidligere nevnt kan man se for seg en bryter hvor man forventer at bryteren som hører til et lys vil skru på/av det lyset. Dette vil være lurt å følge, med tanke på forventningene til brukere.

Consistency (konsistent)

Når det kommer å gjøre designet konsistent tenker man gjerne på det å følge store anerkjente design retningslinjer som de fleste kjente applikasjonene prøver å følge. Dette inkluderer gjerne kjente tegn og symbol som utgjør lignende funksjoner i andre applikasjoner, som f.eks. en søkefunksjon oppdaget av et forstørrelsesglass.

Affordance (hvordan-brukes-den)

Gjennom designet vil man gi brukeren noen tips til hvordan det skal brukes. Man kan f.eks. ta i bruk pop-up bobler som informerer brukeren i startfasen sin. Er det første gang brukeren ser designet kan man utnytte seg av dette ved slike metoder. Prinsippet bak er å gi brukeren en idé om hvordan produktet fungerer.

2.12.2 Figma

Figma er et brukergrensesnitt-designverktøy som kan brukes til en rekke ting, inkludert UI-design, UX-design, appdesign og vektorillustrasjon. Hos figma har man tilgang til å opprettholde et levende dokument hvor brukergrensesnittet kan designes av flere brukere samtidig.

2.13 Javascript Object Notation (JSON)

JSON er et lettvekts åpent standard filformat til datautveksling. Fordelen med formatet er at det er både lett for mennesker å lese og skrive, samtidig som det er enkelt for maskiner å analysere og generere. Det er et format som er helt språkuavhengig, men som bruker konvensjoner fra C-familien av programmeringsspråk (C, C++, C#, Java, JavaScript, Perl, Python, osv). Disse egenskapene gjør JSON til et ideelt datautveklingsspråk.

JSON er bygget på to strukturer:

- En samling av navn / verdipar. På de forskjellige språkene realiseres dette som et objekt, struct, dictionary, osv.
- En ordnet liste over verdier. Realiseres som oftest gjennom en matrise, vektor, liste eller lignende. [30]

Eksempel på JSON:

```
{
    'key': 'name',
    'type': 'Input',
    'label': 'Vessel',
    'placeholder': "Vessel name",
    'required': true,
},
```

(Figur 2: Skjermdump av json)

3 MATERIALER OG METODE

3.1 Utviklingsmetodikk

I dette prosjektet har det blitt benyttet en utviklingsmetodikk kalt Scrum. Teorien bak denne metodikken er beskrevet nærmere i kapittel 2.3, hvor

[30] https://www.json.org/json-en.html

selve prosjektorganiseringen er beskrevet under kapittel 3.2.4.

Prosjektet må forholde seg til flere teknologiske utfordringer som følger av IT. En av hovedutfordringene var hvilken arbeidsmetodikk man skulle følge, og vedlikeholde i et IT-prosjekt. En stor del av dette ligger i å definere hvordan man skal jobbe i slike prosjekt, hvilken måte utfordringene takles på og med hvilke verktøy vil man løse disse utfordringene med. I dette prosjektet startet vi med blanke ark så her var det fritt fram for fastsetting av valgt arbeidsmetode/metodikk.

Når det skulle bli tatt stilling til hvilken arbeidsmetodikk man skulle følge i prosjektet ble tidligere erfaringer lagt mye vekt på. I gruppen var det felles enighet om at en agil tilnærming fungerer mye bedre enn mer tradisjonelle metoder når det kommer til systemutviklingsprosjekter. Når det er sagt, Scrum med sin inkrementelle utvikling gjennom korte iterasjoner, er den agile metoden som gruppen mente å passe best. Dette på bakgrunn av at vi har hatt faste møter med veileder og utviklingsteam hver andre uke gjennom hele semesteret. Prinsippet bak var da å bryte ned hele oppgaven i mindre gjøremål som ferdigstilles innen gitte tidsrom(en såkalt "sprint"). Med den nevnte iterative arbeidsmetodikken forventes det at gruppen produserer produktet delvis med spesifikke gjøremål for hver av disse "sprintene". Tankegangen gir et stort rom for eventuelle forandringer, samtidig som den forbedrer gruppens evne for tilpasning når noe først oppstår. Gjennom et slikt prosjekt vil der hele tiden være endringer, hvor vi som gruppe må ta stilling til hva og hvordan ting skal gjøres.

Siden gruppen kun består av to personer ble det jukset litt med rollene som er definerte i Scrum. Begge av gruppemedlemmene sitter i en dobbeltrolle, hvor både rollen Scrum Master og produkteier er medlemmer av utviklingsgruppen. Selv om dette er en realitet har gruppen hele tiden hatt fokus på å følge den tradisjonelle Scrum-metodikken på best mulig

måte. Dette innebærer at gruppen har opprettholdt en produktkø/backlog gjennom hele prosjektfasen hvor et møte ble holdt hver 14-dag for å prioritere noen av sakene inn i "sprintkøen". Møtet foregikk hovedsakelig over internett, hvor prosjektmedlemmene fastsatte hva som skulle gjøres iløpet av de neste to ukene/neste sprint. Sett vekk i fra dette første planleggingsmøtet, ble det gjennomført en rekke andre møter underveis i utviklingsperioden. Alle møtene gruppen utførte er listet opp under (der noen ble utført i samme forum, fortløpende):

- **Daglig Scrum**
    - Korte daglige "stand-up" møter der utviklingsteamet snakket om hva som ble gjort dagen før, hva som skal gjøres denne dagen, og eventuelt råd/tips om noen satt fast.
- **Sprint planlegging**
    - Møte som ble gjennomført i starten av hver sprint.
    - Mål i å estimere og planlegge innholdet i kommende sprint (det vil si bryte opp brukerhistorier fra produktkøen og plassere de i sprintkø som mindre gjøremål).
    - Hovedsakelig et møte for utviklingsteam hvor Scrum Master fungerte som en tilrettelegger.
- **Sprint gjennomgang / review**
    - Møte mellom prosjektmedlemmene (utviklingsteam og produkteier) etter hver endte sprint.
    - Løsninger ble diskutert med kommentarer, forslag for hverandre.
- **Sprint retroperspektiv**
    - Møter for utviklingsteamet hvor analysering av arbeidet i endt sprint ble sett på.
    - Muligheter for selvinspirering og hvor man kunne utarbeide en felles plan for forbedringer for neste sprint.
    - Ulike spørsmål ble diskutert og besvart som:

- Hva gikk bra denne sprinten og ikke?
- Kan vi gjøre noe bedre?
- Nådde vi målene våre?

- **Veiledningsmøter**
  - Møte med veileder hvor utviklingsteam kunne rådføre seg ved tekniske spørsmål som oppstod.
  - Utført av utviklingsteam og veileder.

3.2 Prosjektorganisasjon

3.2.1 Prosjektgruppen

Prosjektgruppen for dette prosjektet har bestått av to avgangsstudenter på bachelorstudiet i ingeniørfag-data.

Kandidatnummer(e)
10071
10073

3.2.2 Styringsgruppe

Studentenes hovedveileder under prosjektet, tildelt fra NTNU, var universitetslektor Mikael Tollefsen. Universitetslektor Hans Georg Schaathun bisto som Mikael's biveileder gjennom oppgaven. Siden det ikke er en spesifikk oppdragsgiver, er der heller ingen kontaktperson hos oppdragsgiver.

3.2.3 Oppdragsgiver

Dette er en egendefinert oppgave med ingen spesifikk oppdragsgiver, derfor er rollen som oppdragsgiver blitt smeltet inn i produkteierrollen.

3.2.4 Prosjektorganisering

Siden dette prosjektet har følgt utviklingsmetodikken Scrum, har prosjektmedlemmene hatt høyt fokus på rollefordelingen som følge av

dette. Scrum står beskrevet nærmere i kap.2.3, hvor rollefordelingen er et eget punkt. Tre type roller er sett på som de viktigste i scrum; produkteier, utviklingsteam og Scrum Master. Som nevnt tidligere er flere deltakere i gruppen tildelt dobbeltroller på grunn av få medlemmer. I tabellen under kan man se rollefordelingen i prosjektet.

Rollefordeling:

Rolle	Identifikasjon (navn/kand.nummer)
Produkteier	10071
Scrum Master	10073
Utviklingsteam deltaker 1	10071
Utviklingsteam deltaker 2	10073
Hovedveileder	Mikael Tollefsen
Biveileder	Hans Georg Schaathun

Oppgaver for utviklingsteam

- Ansvarlige for selve utviklingen av produktet.
- Ansvarlige for å planlegge sprint-backlog, hvor de skal ta elementer fra produktkøen å bryte de ned til mindre saker i sprint-backlog.

Oppgaver for produkteier

- Ansvarlig for å vedlikeholde produktkø, hva som skal være prioritert funksjonalitet.
- Fokus på hva, ikke hvordan.
- Skal **ikke** fungere som en prosjektleder.
- Fungere som en beslutningsdommer i kravspesifikasjon-spørsmål.

Oppgaver for Scrum Master

- Ansvarlig for tilrettelegging av den valgte arbeidsmetodikken.

- Har hatt ansvar for å sette opp møter, sikre at alle i scrum-teamet møter og at arbeidsformen fungerer som planlagt.
- Forbedre arbeidsformen og sørge for at refleksjonsmøtene leder til forbedring.

Siden veilederne kun skulle fungere som ekstern hjelp fra sidelinjen har de ikke hatt noen spesifikke oppgaver/ansvar i prosjektet.

3.3 Prosjektstyring

Prosjektstyring er et viktig verktøy for å håndtere hvordan prosjektet skal gjennomføres og hvordan man skal utføre ting for å oppnå ønskede gevinster. Dette inkluderer styring, koordinering, oppfølging og leveranser innenfor de gitte rammene.

3.3.1 Trello

For å kunne opprettholde god oversikt over prosjektet har gruppen tatt i bruk et prosjektstyringsverktøy for å gjøre hverdagen lettere. Ved bruk av det riktige verktøyet kan hele denne prosessen bli mer effektiv og enklere for alle. Dette er grunnen til at det har blitt tatt i bruk Trello.

Trello er kun ett av mange nettbaserte prosjektstyringsverktøy hvor de ligger til rette for å følge agil utviklingsmetodikk. Ved sin ekstreme enkelhet og elegante design skiller de seg litt ut fra de andre tilbyderne. Fra gruppens synspunkt var denne enkelheten den største faktoren, da vi slipper å kaste bort mye tid i opplæring for et mer avansert verktøy. Med andre ord; hadde det vært flere medlemmer i gruppen, som jobbet med en større oppgave, ville det nok ha blitt brukt et mer funksjonelt verktøy.

Trello er basert på at man skal kunne få full synlighet over alle relevante oppgaver, og gi oss fleksibilitet for å kunne omorganisere når prioriteringene våre endres. I det nettbaserte verktøyet har man mulighet for å opprette en scrum-tavle med forskjellige rader. I den første raden vil

man plassere en liste over ulike gjøremål som er nedbrutt fra produktkøen (dette blir kalt sprintkø). Deretter har man mulighet for å dra disse gjøremålene over til neste rad slik at hele teamet ser at dette gjøremålet er påbegynt (in progress). Når vedkommende er ferdig med gjøremålet kan man dra dette over i neste rad for å markere at det er ferdig eller klart for testing (done/integrate and test). Alle i prosjektgruppen har tilgang til denne tavlen, både med tanke på å se og bidra. Det gir oss en visuell oversikt over alle gjøremål som skal utføres, hvilke som er underveis og hvilke vi er ferdige med. I bunn og grunn har verktøyet blitt brukt for å vedlikeholde produktkø, sprintkø og tildele/organisere gjøremål mellom medlemmene.

Beskrivelse av de ulike tilstandene for gjøremål i trello tavlen (radene):

Product backlog

- Listen over all kommende funksjonalitet og større gjøremål.

Next

- Listen over alle gjøremål som skal utføres i nåværende sprint.
- Fyllt opp under sprint-planleggingsmøtet.

In progress / Development

- Liste over alle gjøremål som er under utvikling/påbegynt.

On hold / Waiting

- Liste over gjøremål som ligger på vent for en grunn. Kan være alt fra en godkjennelse eller svar fra noen.

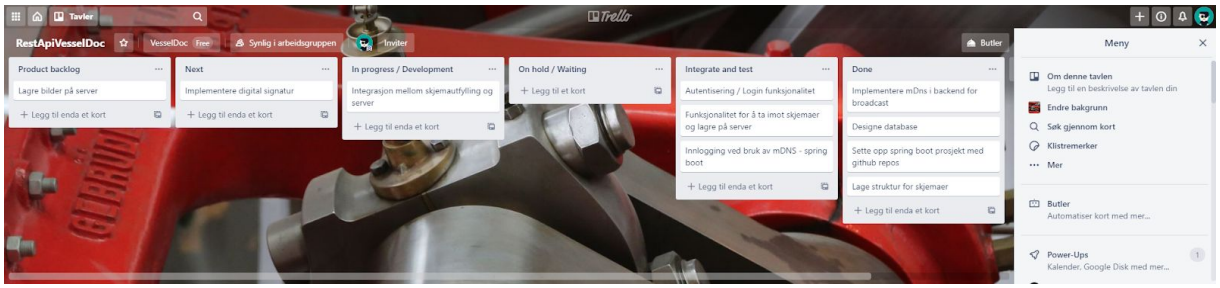
Integrate and test

- Liste over gjøremål som er ferdige, men må testes før det blir delt med resten av koden og ferdigstilt.

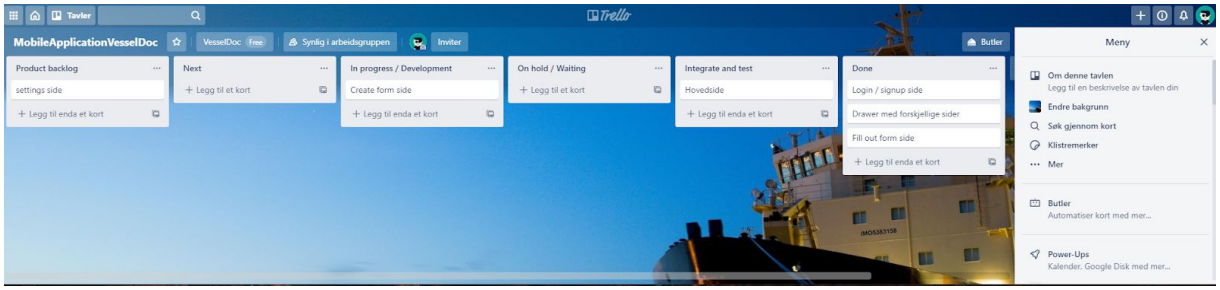
Done

- Liste over alle gjøremålene som er helt ferdige. I denne raden har man tilgang til historikk.

I starten av prosjektet ble det tatt en beslutning om å vedlikeholde to individuelle tavler fordi prosjektmedlemmene mente det var behov for det. Dette var for å hindre rot i mellom gjøremålene for app og server, derav en tavle for frontend og en for backend.



(Figur 3: Skjermdump av vår backend scrum-tavle i trello)



(Figur 4: Skjermdump av vår frontend scrum-tavle i trello)

3.4 Metode

3.4.1 Beslutning av mobil rammeverk

Prosjektet krever at at en kan benytte produktet under varierende situasjoner, blant annet med ustabil nettverkstilkobling og at digitale skjema kan utfylles fra en enhet som er så tilgjengelig som mulig. Derfor er det fokus på at det er en applikasjon som har støtte for flere mobile plattformer. Med dette har prosjektgruppen valgt å utvikle klienten til produktet med rammeverket flutter. Dette rammeverket er basert på

programmeringsspråket dart, og er kjent for å være kompatibelt med både Android og iOS. Flutter står beskrevet mer i detalj under kapittel 2.8.

3.4.2 Beslutning av applikasjonsserver

Serveren til applikasjonen trenger en rekke muligheter for å oppnå målet til prosjektet. Det skal kunne å ta imot rest forespørsler. Serveren skal også kunne legge til og hente ut data fra en database. I tillegg til dette må det være et brukersystem. Med disse mulighetene ville også det vært til fordel at prosjektarbeiderne er kjent med rammeverket til serveren. Rammeverket Spring-boot vil derfor bli benyttet fordi det går under alle kriteriene nevnt over. Flere detaljer om spring-boot står i kapittel 2.9.1.

3.4.3 Beslutning av type datapersistering

Når en skal persistere data, kan dette utføres gjennom serialisering til XML-filer eller å persistere det til en database. I gruppens beslutning av metode for prosjektets datapersistering, ble både tidligere erfaringer vektlagt samtidig som det ble sett på hvilke metoder som var anbefalt. Det som kom frem ved flere søk var at bruken av en database og et RDBMS, er en anbefalt måte å håndtere denne type persistering. Siden medlemmene også har mest erfaring og høyest kompetansenivå med en slik metode falt valget på dette. Bruk av en relasjonsdatabase gir teamet mulighet for å få enkel tilgang til dataen ved såkalte CRUD operasjoner (Create, Read, Update, Delete).

Beslutningen av hvilken type database og RDBMS denne løsningen skulle nytte, ble også basert på tidligere erfaringer og kompetansenivå i gruppen. Medlemmene hadde mest kjennskap til MySql, som er kjent for å være et mye brukt RDBMS, som både er åpent og gratis å bruke. MySql ble derfor valgt som løsningens RDBMS.

3.4.4 Beslutning av kommunikasjonsmetode

Et av hovedproblemstillingene til dette prosjektet er hvordan kommunikasjonen mellom enhetene skal utføres. Da en nødvendigvis ikke har tilgang til internett er det fortsatt noen alternative måter det kan utføres på.

Lokal server

En mulighet til å løse problemstillingen er å sette opp en lokal server ombord i båten som alle enhetene kan kommunisere med ved hjelp av det lokale nettverket. Med denne metoden er det mulig å benytte REST API som kan leses mer om i kapittel 2.9. For å oppdage denne serveren kan en bruke en metode kalt service discovery. Denne metoden benytter ruterens multicast DNS til å publisere hvilken IP adresse serveren er på, som nevnt i kapittel 2.4.

En fordel ved å bruke en lokal server er at serveren kan holde på innloggingsinformasjon, som fører til at klientene slipper å være koblet til internett for å logge inn. En annen fordel er at det er uavhengig av hvilken plattform klientene er på siden det er REST basert. Enda en fordel er at serveren håndterer all dataen som overføres, så klientene trenger kun å lagre og sende dataene en gang, i motsetning til alternativer der dataene synkroniseres mellom flere klienter(peer-2-peer). I tillegg til dette så er prosjektmedlemmene erfart med en slik løsning, som kan være tidsbesparende for prosjektet.

En av ulempene ved å bruke en lokal server framfor å synkronisere direkte med andre klienter er at det kreves mer vedlikehold av utviklerne, fordi da må de fokusere på å utvikle to applikasjoner istedenfor en applikasjon som gjør alt. En annen ulempe er at det krever IT-kompetanse til å sette opp en slik server, og med tanke på at det skal være en server i hver båt kan det virke ugunstig.

Nearby connectivity

Det finnes noen biblioteker for rammeverket flutter, som gjør det mulig for en applikasjon å kommunisere med andre enheter med samme applikasjon. Dette gjør den ved hjelp av wifi direct og bluetooth så det ikke er behov til å være tilkoblet et nettverk.

En fordel med å benytte nearby connectivity for dette prosjektet er at utviklerne trenger kun å fokusere på å utvikle en applikasjon, istedenfor å måtte jobbe med to som en ofte må med en klient server løsning. I tillegg til dette slipper det å være en person ombord i båten med IT-kompetanse for å ta i bruk applikasjonen. En annen fordel er at det ikke er behov for å ha et lokalt nettverk i båten.

En ulempe ved å benytte denne løsningen er at en enhet må synkronisere samme data flere ganger med andre enheter kan være strømkrevende. Et annet negativt resultat er at hver enhet må lagre mange dokumenter som kan kreve mye lagringsplass, noe som mobile enheter ikke burde være utsatt for. Enda en ulempe er at hvis prosjektet skal ha et brukersystem, så må det enten være på en sentral server så brukere kan logge seg inn kun når de har internett, eller så må det være et desentralisert brukersystem, noe som er komplisert å utvikle.

Peer 2 peer

En metode er å benytte det lokale nettverket til å kommunisere direkte til andre klienter. Fordelene og ulempene ved denne løsningen er for det meste det samme som nearby connectivity som det står om ovenfor. Eneste forskjellen er som sagt at det blir tatt i bruk et lokalt nettverk istedenfor direkte kommunikasjon mellom enhetene.

3.5 Programmeringsspråk

Komplette løsninger innen IT krever ofte ulike rammeverk og språk for å skape tjenester som skal servere forskjellige formål. For dette prosjektet var det akkurat sånn. Gjennom utviklingen av en løsning har det blitt tatt i bruk flere typer språk og rammeverk. I hvert av underkapitlene i inneværende kapittel begrunnes det hvorfor språket ble foretrukket over et annet, og hva det utnevnte språket ble anvendt til.

3.5.1 Dart

Dart er et klient-optimalisert programmeringsspråk siktet for utvikling av apper på forskjellige plattformer. Språket er utviklet av Google, og ble dekket mer i detalj tidligere i rapporten (kap 2.8.1).

Grunnlaget for bruk av dart i denne løsningen var litt tilfeldig. Gruppen hadde allerede bestemt seg for å bruke rammeverket flutter for utvikling av den grafiske delen av applikasjonen. Når en flutter-applikasjon skal utvikles, blir koden skrevet hovedsakelig i dart. Dette førte til at dart fulgte med mer som en pakke-avtale med flutter.

3.5.2 Java

Backend-delen av denne løsningen (rest-api) er skrevet i det objektorienterte programmeringsspråket Java (se kap. 2.7). I følge TIOBE er java det mest populære og brukte programmeringsspråket med sine gode 16.73 prosent av alle verdens brukere. [31] Java ble, som i flere andre avgjørelser i prosjektet, foretrukket av gruppemedlemmene på bakgrunn av kjennskapen og kompetansenivået i språket. Et annet vesentlig argument var identisk som i bruken av dart. Gruppen hadde tidligere bestemt seg for å anvende rammeverket spring-boot, derav fulgte java hånd i hånd med rammeverket.

[31] https://www.tiobe.com/tiobe-index/

3.5.3 SQL

For å kunne lagre unna skjemaene og annen nyttig informasjon i applikasjonen, måtte det bli tatt i bruk en eller annen form for database. Gruppen valgte å utvikle en MySql relasjonsdatabase, som kommuniseres med gjennom det kjente programmeringsspråket SQL (kap. 2.5.2).

3.6 Utviklingsverktøy

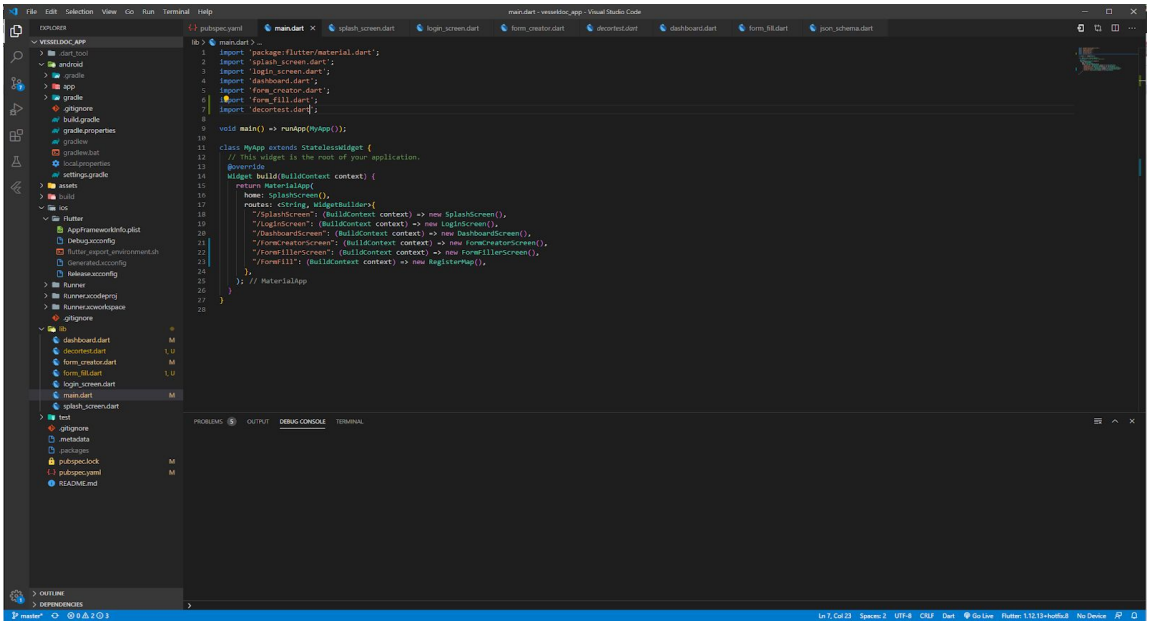
På samme måte som med ulike programmeringsspråk og rammeverk er utviklere også avhengige av en rekke forskjellige programvarer som støtter opp disse språkene/rammeverkene. Ikke nok med at programvarene legger til rette for at utviklerne kan produsere mer effektivt, men de minimerer også sløsingen av tid med ikke-relevante oppgaver. I dette prosjektet har det blitt benyttet flere av disse nevnte programvarene for å hjelpe gruppen med å oppnå prosjektmålene sine. Følgende underkapitler inneholder en kort beskrivelse av de forskjellige programvarene, og hvordan de har vært anvendt i prosjektet.

3.6.1 Frontend utvikling - Visual Studio Code

En av de mest brukte applikasjonene i dette prosjektet var Visual Studio Code. Visual Studio Code, bedre kjent som VS code, er en kildekode editor som kan tilpasses både med temaer, hurtigtaster og ved installering av diverse utvidelser. I en undersøkelse utført av stackoverflow i 2019 ble det kjent for å være det mest populære utviklingsmiljø-verktøyet, hvor over 50% av deltakerne valgte dette verktøyet.[32] Med samme utfall som resultatet i undersøkelsen ble VS code foretrukket av samtlige medlemmer av gruppen. Dette var på bakgrunn av alle tilpasningene VS code tillater, og dens brede støtte for en hel variasjon av ulike språk og rammeverk.

[32] https://insights.stackoverflow.com/survey/2019#development-environments-and-tools

Store deler av arbeidet ved utviklingen av prosjektets frontend-løsning ble utført i denne editoren. Med andre ord; alt av dart kode ble skrevet i dette verktøyet, ved bruk av en aktiv utvidelse, tilrettelagt for flutter/dart-utvikling. Under er der lagt med en skjermdump fra utviklingen i verktøyet.

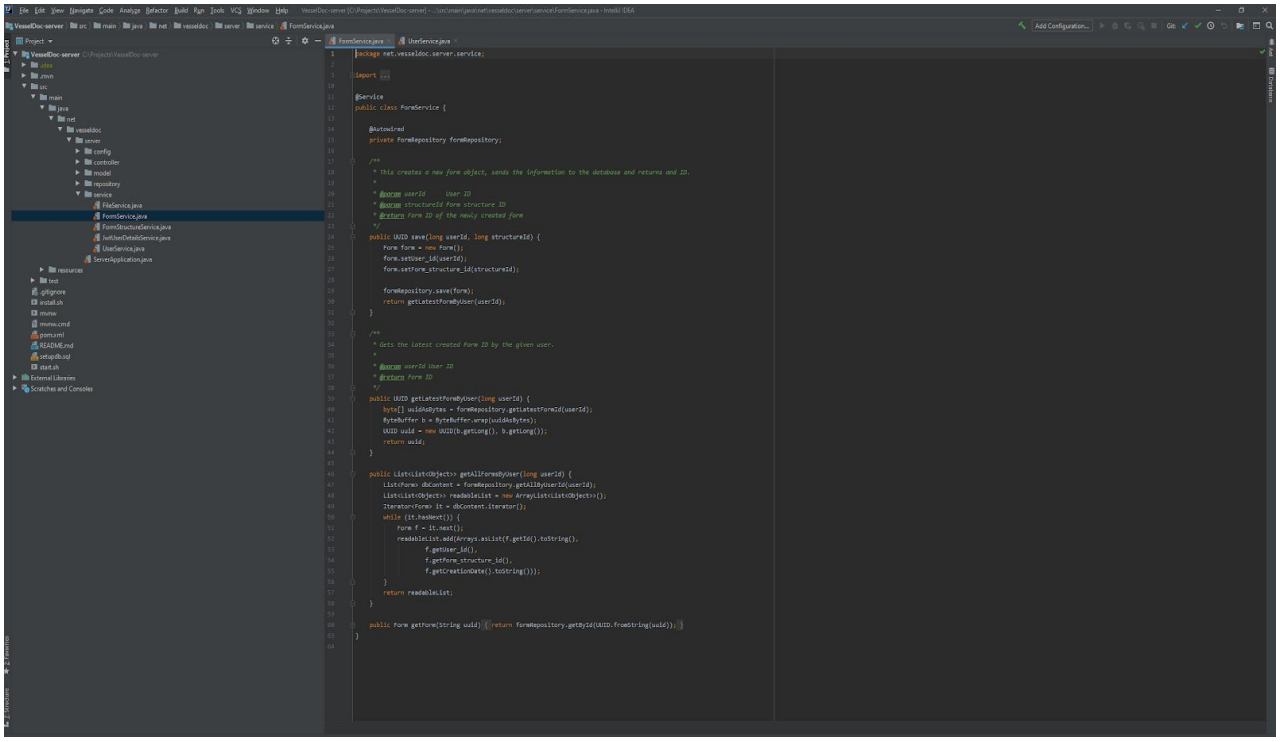


(Figur 5: Skjermdump fra VS Code)

3.6.2 Backend utvikling - IntelliJ IDEA 2019

Under utviklingen av prosjektets backend løsning ble det brukt et lignende verktøy som i frontend. Forskjellen ligger i at verktøyet brukt i backend gjerne er mer tilsiktet programmeringsspråket java, og krever derfor ikke utvidelser for å fungere optimalt med dette. Siden frontend tjenesten skulle skrives i dart og backend tjenesten i java, ble det tatt en beslutning om å bruke forskjellige utviklingsverktøy for hver av tjenestene. Valget falt på IntelliJ IDEA 2019. IntelliJ er på samme måte som VS code, et integrert utviklingsmiljø/verktøy som er laget for utvikling av programvare. En viktig faktor som påvirket dette valget var at intelliJ IDEA føltes mer tilrettelagt for utvikling av en tilstandsløs serverløsning enn VS code. Siden både IntelliJ og VS code har blitt aktivt brukt i studiet for gruppemedlemmene, var kjennskapen til disse applikasjonene høyere enn andre tilsvarende applikasjoner. Det faller naturlig å velge disse to

applikasjonene som de mest vesentlige hjelpemidlene i utviklingsarbeidet. Under har det blitt lagt med skjermdump av prosjektets programmering i IntelliJ IDEA 2019.



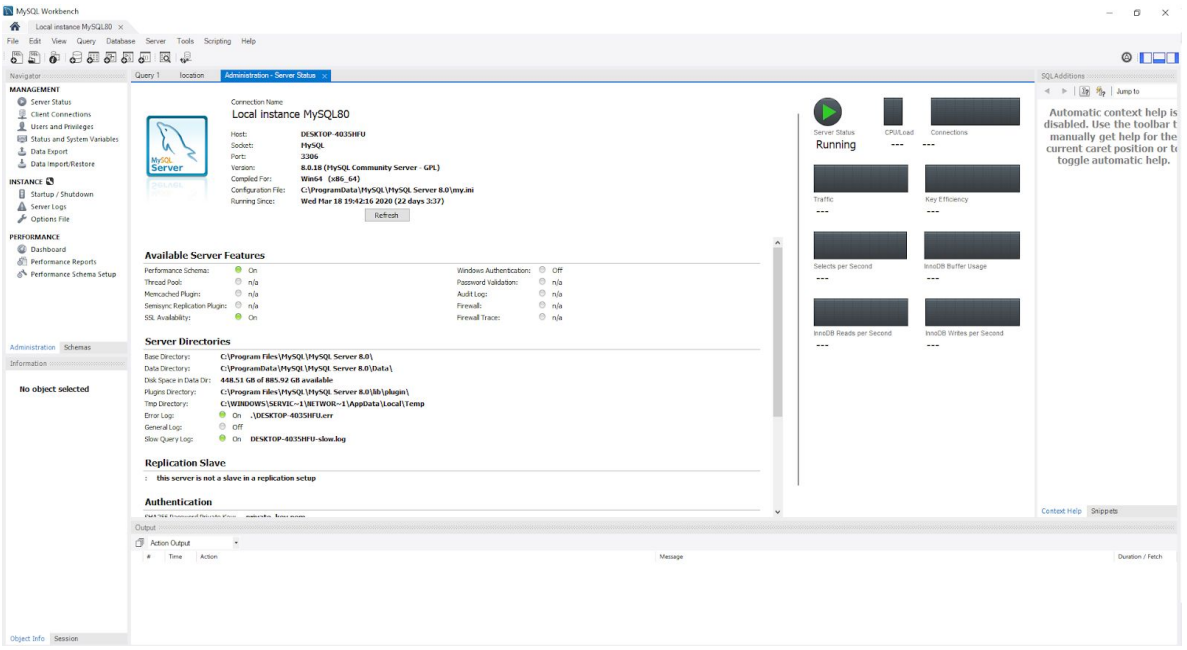
(Figur 6: Skjermdump fra IntelliJ IDEA 2019)

3.6.3 Databaseutvikling - MySql Workbench

Når de kommer til bruk av eksterne verktøy/grafiske-grensesnitt for databaseutvikling så er dette strengt tatt ikke nødvendig. Samtidig var det et faktum i gruppens tidligere erfaringer at et slikt verktøy gjør databasearbeidet mer oversiktlig, og blir lettere å jobbe med. Det ble derfor enighet om å anvende et slikt verktøy i utviklingen av databasen for denne løsningen.

Som nevnt tidligere, ble en del av valgene for utviklingsverktøy vektlagt på bakgrunn av prosjektmedlemmenes kompetansenivå og tidligere erfaringer. Dette gjaldt også valget av databaseverktøy. Da bort i mot ingen av prosjektmedlemmene var godt kjent med andre databaseverktøy, falt valget lett på MySql Workbench. MySql Workbench er et visuelt databaseverktøy som tilbyr SQL-utvikling, administrasjon,

databasedesign, oppretting og vedlikehold i ett enkelt integrert utviklingsmiljø for MySQL databasesystemer. [33] Verktøyet var sett på som mest tidsbesparende alternativet siden gruppemedlemmene da slapp å bruke tid på å lære seg et nytt verktøy, og kunne derfor fokusere mer på ren utvikling av databasen. En annen vesentlig faktor for dette valget var typen databasesystem som skulle anvendes. Det ble tidligere valgt å anvende et MySql databasesystem, hvor MySql Workbench ble tilrettelagt for utvikling av et slikt system.



(Figur 7: Skjermdump fra MySql Workbench)

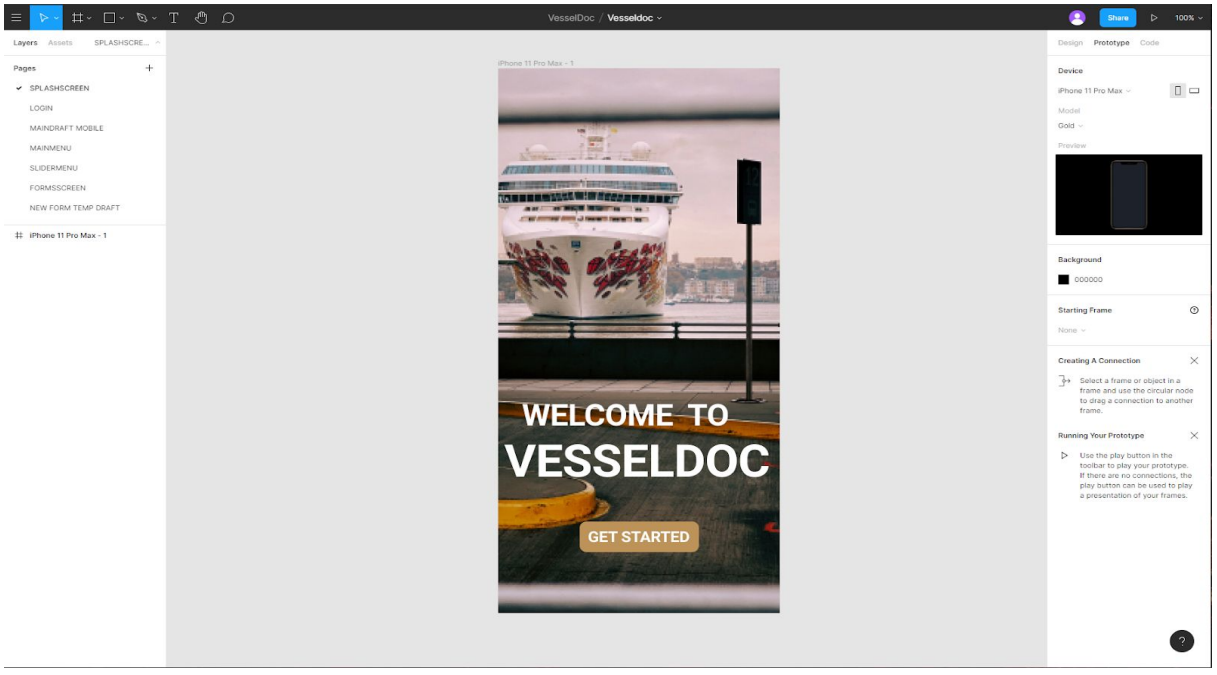
3.6.4 Brukergrensesnittdesign - Figma

For å gi gruppen bedre innsikt og høyere beslutningevne i valg av design, ble det bestemt å lage noen prototyper av ulike brukergrensesnitt. En slik prototype gjør det lettere for gruppen å finne ut av hva som fungerer/evt ikke-fungerer, hva ser fint ut og hvilke farger som passer sammen (fra gruppens ståsted). Hadde oppgaven hatt en ekstern oppdragsgiver kunne disse prototypene bli vist frem i en demonstrasjon,

[33] https://www.mysql.com/products/workbench/

hvor oppdragsgiver kunne gi gruppen feedback på eventuelle ønskede forandringer.

Ved gruppens valg av deres foretrukne designverktøy, ble en rekke forskjellige applikasjoner vurderte. Alle applikasjonene var relativt ukjente for gruppen; valget ble derfor tatt på bakgrunn av hvilke applikasjoner som var høyst anbefalt av andre utviklere og designere, på internett. Et av alternativene som ble repetert i utallige anbefalinger var Figma. Figma er et gratis* designverktøy, som kan leses mer om i detalj i kapittel 2.12.2. Hvor Figma skilte seg litt ut fra de andre alternativene, var ved bruk av levende filer (bedre muligheter for teamsamarbeid) og et mer familiært brukergrensesnitt. Figma ble derfor det foretrukne designverktøyet i denne oppgaven. Under har det blitt lagt med skjermdump av prosjektets utvikling av prototyper/designskisser i Figma.

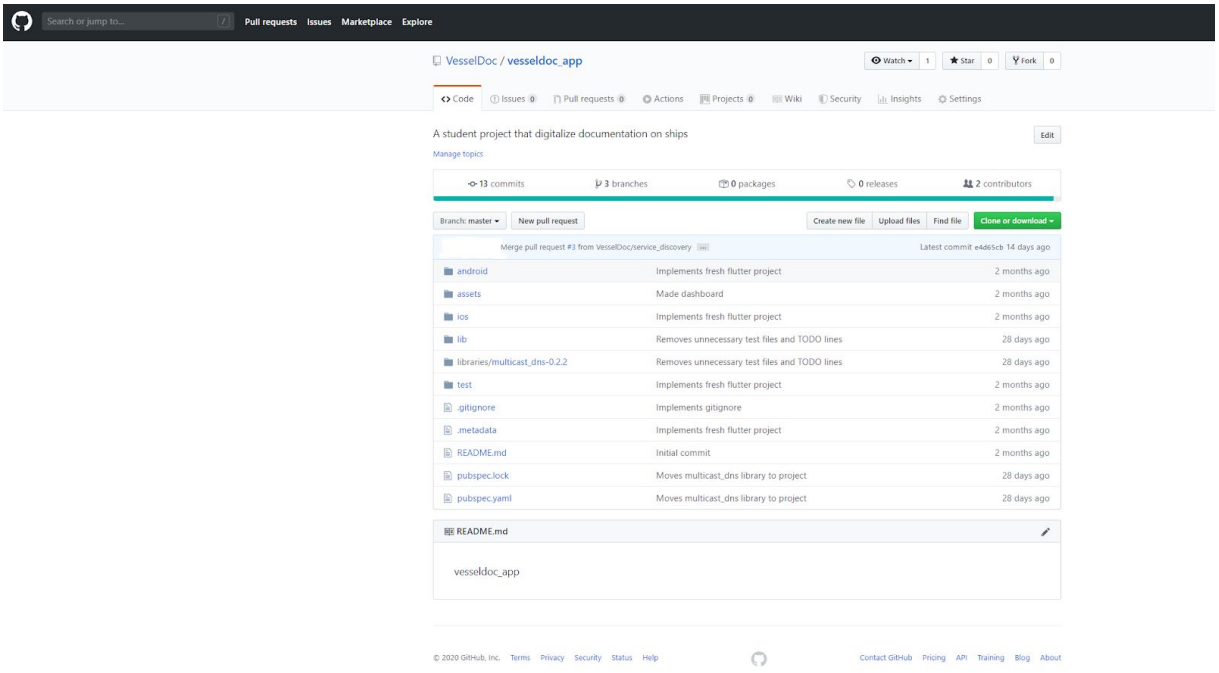


(Figur 8: Skjermdump fra Figma)

3.6.5 Versjonskontrollsystem - Github

Versjonskontroll er noe som de fleste utviklere ser på som helt nødvendig for å kunne opprettholde orden i programvare-koden. Det står mer detaljert om hva versjonskontroll egentlig er, og hvilke alternativer vi har

for det under kapittel 2.10 i rapporten. Valget av versjonskontrollsystem ble, på lik linje som en del andre beslutninger i prosjektet, tatt på bakgrunn av gruppens tidligere erfaringer. Gjennom bachelorgraden har gruppen fått testet ulike versjonskontrollsystemer, der det nettbaserte grafiske grensesnittet 'Github', ble foretrukket av prosjektmedlemmene. Siden Git kun er kjent for å være et kommandolinjeverktøy, ville gruppen ta i bruk et grafisk grensesnitt som github. Github var brukt for å forenkle styring og for å bedre oversikten, i versjonskontrollsystemet.

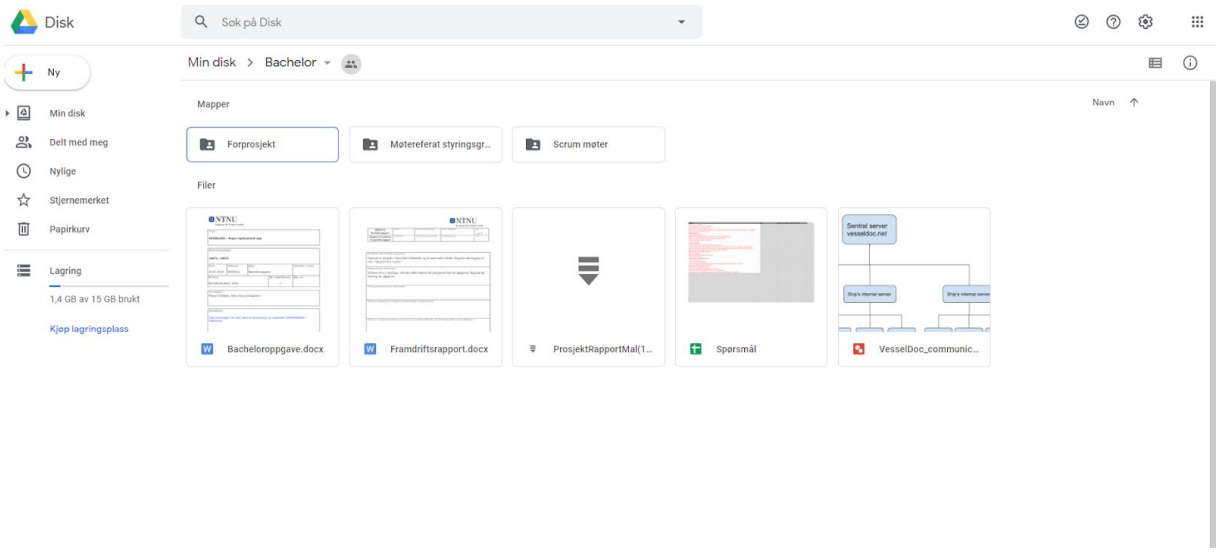


(Figur 9: Skjermdump fra Github)

3.6.6 Verktøy for rapportskriving/dokumentasjon - Google Drive

Når det skal skrives en rapport/dokumentasjon, lønner det seg å anvende gode redigeringsverktøy. I utformingen av denne rapporten var det flere typer redigeringsverktøy i bevegelse. Alle verktøyene som ble brukt var gratis inkluderte i en skybasert pakkeløsning fra Google. Denne pakkeløsningen består av ulike type redigeringsverktøy for redigering av forskjellige filtyper. Man kan blant annet redigere tekstfiler, regnearkfiler

og lignende. Hele pakkeløsningen blir tilbudt gjennom Google sin "Drive" tjeneste. "Drive" tjenesten er en fillagrings og synkroniseringstjeneste, som lar brukere gratis* lagre filer på google's servere, synkronisere filer på tvers av enheter og dele filer med andre brukere.[34] Løsningen tilbyr også (på samme måte som Figma) mulighet for levende dokumenter, hvor all endring skjer i "nåtid" hos de brukerne som har tilgang til dokumentet. Dette skaper flere fordeler for gruppen, hvor hovedfordelen er at gruppen kan redigere et og samme dokument til enhver tid. Vedlagt ligger med et bilde av gruppens mappestruktur, og deres filer, i Google Drive.



(Figur 10: Skjermdump fra Google Drive, her kan man se mapper for møtereferater med styringsgruppen, gruppens scrum møter og forprosjekt.)

3.7 Eksterne biblioteker og verktøy

3.7.1 Postman

I prosjektet er kommunikasjonen mellom klient og server, rest basert. Det vil si at klienten vil sende forespørsler til serveren for å få utført en jobb. Da en utvikler et slikt system, ville det vært tungvint å utvikle applikasjon og server samtidig. Derfor er Postman et nyttig verktøy. Dette verktøyet gjør at en kan teste forespørsler til en server uten å måtte kode en klient.

[34] https://www.google.com/intl/no_ALL/drive/

3.7.2 Spring-boot

Siden det var tidligere bestemt å utvikle en tilstandsløs tjeneste i backend, lå det i kortene å anvende et rammeverk for å hjelpe oss litt på vei. Fordelen ved bruk av et slikt rammeverk er å spare seg for en del unødvendig arbeid og tid. Spring-boot er et av de mest brukte rammeverkene til utvikling av tilstandsløse applikasjoner/tjenester.

I spring-boot blir det benyttet en rekke maven bibliotek som simplifiserer utviklingen av serverapplikasjonen til prosjektet.

Spring-boot "starter" er flere biblioteker som er tilpasset til å fungere med spring-boot. Disse bibliotekene er i flere varianter som web, security og test, der hver og enkelt av de har en rekke metoder for å komme i gang med spring-boot og få implementert ønskede funksjoner prosjektet. Et eksempel på dette er spring-boot starter web som gjør det mulig å benytte RESTful, spring MVC med tomcat som servlet container. [35] Det vil si at server-applikasjonen kjøres som en webserver og vil kunne ta imot REST requests og returnere responses. Et annet eksempel et spring starter bibliotek er spring security. Dette gir solide autentiserings klasser som gjør det enkelt og mer betryggende å utvikle et prosjekt med et brukersystem.

En annen serie med maven-bibliotek er hibernate. Hibernate er et verktøy for java som simplifiserer kommunikasjonen mellom en applikasjon og en database. Hovedsakelig gjør hibernate at en kan legge til, oppdatere, hente og fjerne data fra en database som om det var java-objekter. Dette

[35] "org.springframework.boot » spring-boot ... - Maven Repository." https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-web. Accessed 9 May. 2020.

gjør den ved å kartlegge java-klasser til databasetabeller, og java datatyper til SQL datatyper[36].

3.7.3 Xcode (testing)

For å kunne teste applikasjonen på en enhet med Apple sitt operativsystem, iOS, forventer flutter at xcode er installert på enheten brukt for kompilering.

3.7.4 JWT

JSON Web Token er en åpen standard (RFC 7519) som definerer en kompakt og selvforsynt måte for sikker overføring av informasjon mellom parter som et JSON-objekt. Siden denne informasjonen er signert digitalt kan den både bekreftes og klareres. JWTer kan signeres ved hjelp av en hemmelighet (HMAC-algoritmen) eller et offentlig/privat nøkkelpar ved å bruke RSA eller ECDSA.

JSON Web Tokens kan brukes til:

- Autorisering
    - Vanligste situasjonen å bruke JWT.
- Informasjonsutveksling
    - God måte å sikkert overføre informasjon mellom ulike parter.

Strukturmessig består en JWT av tre deler atskilt med prikker (.), som er:

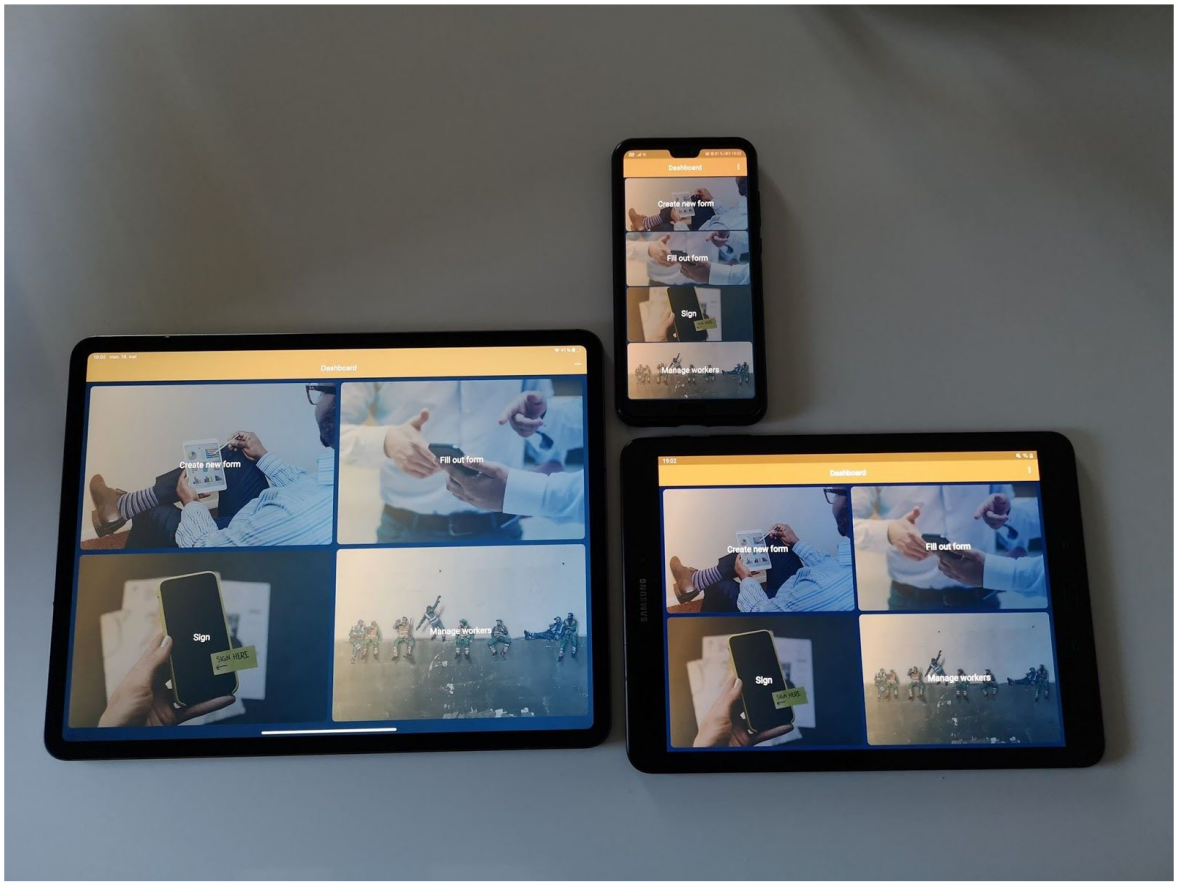
- Header
- Payload
- Signature

Derfor ser en JWT vanligvis ut som følgende: "xxxxx.yyyyy.zzzzz" [37]

[36] "Hibernate (framework) - Wikipedia." https://en.wikipedia.org/wiki/Hibernate_(framework). Accessed 13 May. 2020.
[37] https://jwt.io/introduction/

3.8 Materialer og utstyr

Dette kapittelet inneholder en ren opplisting av ulike materialer og utstyr som har blitt tatt i bruk ved gjennomføringen av prosjektet. Dette innebærer gruppemedlemmenes produksjon og test-enheter.



(Figur 11: Fysiske testenheter)

3.8.1 Samsung Galaxy Tab S3 (9.7 inch)

For fysisk testing av applikasjonen på et Android nettbrett, brukte gruppen en Samsung Galaxy Tab S3. Skjermstørrelsen er på omtrent det samme som flere tilsvarende nettbrett.

(Figur 12: Samsung Galaxy Tab S3)

Spesifikasjoner (Samsung Galaxy Tab S3)	
Modellnummer	SM-T820
Operativsystem	Android 7.0 (Nougat) 64-bit
Prosessor (CPU)	Qualcomm Snapdragon 820 APQ8096
Minne (RAM)	4096 MB
Lagring	32 GB
Skjerm	9.7 Tommer (2048 x 1536) OLED(AMOLED)
Minnekort	Ja

3.8.2 iPad Pro (12.9 inch)

På samme måte som med Samsung nettbrettet, brukte gruppen et annet nettbrett for testing av applikasjonen på iOS. Siden skjermen var litt større, så fikk gruppen testet applikasjonen både med større skjerm og annet operativsystem.

(Figur 13: iPad Pro 12.9)

Spesifikasjoner (iPad Pro 12.9)	
Modellnummer	A1895
Operativsystem	Apple ipadOS 13
Prosessor (CPU)	Apple A12X Bionic
Minne (RAM)	4096 MB
Lagring	64 GB
Skjerm	12.9 Tommer (2732 x 2048) LCD (IPS)
Minnekort	Ja

3.8.3 Huawei Matebook X Pro

For utvikling og/testing av applikasjonen har gruppen benyttet seg av en rekke ulike datamaskiner. En av de bærbare var Huawei Matebook X Pro.

(Figur 14: Huawei Matebook X Pro)

Spesifikasjoner (Huawei Matebook X Pro)	
ModellNr/navn	Huawei MateBook X Pro
Operativsystem	Windows 10
Prosessor (CPU)	Intel Core i7-8565U
Minne (RAM)	16 GB
Lagring	512 GB
Skjerm	13.9 Tommer (3000 x 2000) LTPS
Minnekort	Nei

3.8.4 Apple iMac 21.5

Som nevnt i forrige delkapittel har flere datamaskiner blitt bruk for både produksjon/testing i dette prosjektet. For å kunne teste applikasjonen på iOS (ipad), måtte en apple maskin bli tatt i bruk for kompilering i annet miljø.

(Figur 15: Apple iMac 21.5")

Spesifikasjoner (Apple iMac)	
Modellnr/navn	A1311
Operativsystem	macOS Mojave
Prosessor (CPU)	Intel Core i5 (I5-2400S)
Minne (RAM)	16 GB
Lagring	512 GB
Skjerm	21.5 Tommer (1920 x 1080) LED
Minnekort	Nei

3.8.4 Huawei P20 Pro

For fysisk testing av applikasjonen på en Android telefon, brukte gruppen flere enheter, blant annet en Huawei P20 Pro. Skjermstørrelsen er omtrent det samme som tilsvarende telefoner.

(Figur 16:Huawei P20 Pro)

Spesifikasjoner (Huawei P20 Pro)	
Modellnr/navn	P20 Pro
Operativsystem	Android 8.0
Prosessor (CPU)	Hisilicon Kirin 970
Minne (RAM)	6 GB
Lagring	128 GB
Skjerm	6.1 Tommer (2240 x 1080) OLED
Minnekort	Nei

3.8.5 Nvidia SHIELD

Som med de andre nettbrettene, blir SHIELD tablet brukt til testing. Dette nettbrettet har en liten skjerm i forhold til de andre, i tillegg har den et annerledes aspekt ratio, som kan være nyttig hvis en skal teste en app som har som mål å være kompatibel på så mange enheter som mulig.

(Figur 17: Nvidia SHIELD)

Spesifikasjoner (Nvidia SHIELD)	
Modellnr/navn	SHIELD Tablet
Operativsystem	Android 7.0
Prosessor (CPU)	Cortex-A15
Minne (RAM)	2 GB
Lagring	16 GB
Skjerm	7.96 Tommer (1920 x 1200) LED
Minnekort	Nei

3.8.6 Google Pixel 3a

Google Pixel blir brukt til å teste applikasjonen med en mobiltelefon. Dette gjør det mulig å teste om applikasjonen gir forventet resultat selv om skjermen er liten i forhold til nettbrett.

(Figur 18: Google Pixel 3a)

Spesifikasjoner (Google Pixel 3a)	
Modellnr/navn	Pixel 3a
Operativsystem	Android 10
Prosessor (CPU)	6 x Kryo 360
Minne (RAM)	4 GB
Lagring	64 GB
Skjerm	5.56 Tommer (2220 x 1080)
Minnekort	Nei

3.8.7 Lenovo ThinkPad T460

ThinkPad T460 er en av laptopene som blir brukt til utvikling av dette prosjektet.

(Figur 19: Lenovo ThinkPad T460)

Spesifikasjoner (ThinkPad T460)	
Modellnr/navn	ThinkPad T460
Operativsystem	Ubuntu 18.04 LTS
Prosessor (CPU)	Intel Core i7-6600U
Minne (RAM)	16 GB
Lagring	750 GB
Skjerm	14 Tommer (1920 x 1080)
Minnekort	Nei

3.8.8 Raspberry Pi

Raspberry Pi blir brukt til å teste serverapplikasjonen i prosjektet. Med denne kan en teste service discovery i det lokale nettverket, og i tillegg får en et bedre innblikk av hvordan oppsettet av en server ville blitt i en ekte situasjon.

(Figur 20: Raspberry Pi 4 Model B)

Spesifikasjoner (Raspberry Pi 4 Model B)	
Modellnr/navn	Raspberry Pi 4 Model B
Operativsystem	Raspbian buster lite
Prosessor (CPU)	Cortex-A72
Minne (RAM)	4 GB
Lagring	0
Skjerm	Ingen
Minnekort	32 GB

3.9 Design

Når det skal designes en løsning for ulike type brukere, er det viktig å følge gode designprinsipper. Å lage et tiltalende og intuitivt design handler ikke kun om farger og plassering av elementer, men er mer som en gjennomtenkt prosess med både forskning og testing. Det handler om å

bygge en løsning som gir best mulig brukeropplevelse og med et vakkert design. Ved bruk av Don Norman's sine seks prinsipper om produktdesign og en fornuftig fargepalett, har gruppen strukket seg for å få utformet et applikasjonsdesign med høyest mulig nivå (Don Normans prinsipper er brutt ned tidligere i rapporten, kap 2.12.1).

I IT-utvikling er det alltid fokus på to sentrale designelementer; Brukeropplevelse og brukergrensesnitt, bedre kjent som UX/UI. Med UX menes det brukeropplevelse og ens evne til å lage et grensesnitt som er forståelig og praktisk for brukerne. På den andre siden; UI handler hovedsakelig om brukervennlighet og estetikk. Disse to elementene har jobbet tett i underbevisstheten, for prosjektmedlemmene, ved utviklingen av denne løsningen.

Siden løsningen, i første omgang, var rettet mot det maritime markedet bestemte gruppen seg for et maritimt tema på applikasjonen. Med maritimt tema menes det en fargepalett ut i fra fargene til typiske materialer om bord i skip, og ved noen maritime-inspirerte symbol.

3.9.1 Fargepalett

(Figur 21: Fargepalett for løsningen)

Figuren over avbilder de tre primærfargene brukt i løsningen. Fargen til venstre skal etterligne fargen på et hampetau, hvor fargen i midten skal forestille fargen på en marineuniform. Det ble bestemt at applikasjonen skal ha et mørkere tema for å følge "dagens" mote.

3.10 Distribuering av applikasjonen

Ved eventuell ferdigstillelse og lansering, skal applikasjonen distribueres i to forskjellige distribusjonstjenester. Den offisielle appbutikken for

Android, og for iOS. Da kan så å si alle enheter med iOS og Android operativsystem laste ned løsningen vår.

3.11 Testoppsett

Etter at spesifikke mål innenfor en scrum iterasjon er utført, vil det bli gjort en test med en raspberry pi med nylig installert raspbian operativsystem. Dette gjøres for å sørge for at serveren fungerer som forventet rett etter at den er installert. Grunnen til at det blir gjort med en raspberry pi isteden for en virtuell maskin er fordi det benyttes multicast under testingen. Å benytte multicast i en raspberry pi er mye simplere å få gjort enn i en virtuell maskin. Det er fordi det er ferdig konfigurert i raspbian, men i en virtuell maskin er det enten ikke mulig eller komplisert å få konfigurert avhengig av hvilken virtualiserings-verktøy en bruker. Etter at maskinen settes opp, så hentes server-repositoryen ved å klones fra GitHub, og eventuelt skifter branch til det som har blitt jobbet med. Når det er gjort kan en kjøre shell skriptet install.sh som vil innstallere både database og serverapplikasjon. Den tilpasser også multicast så den oppdages av klienten. I tillegg til dette gjør skriptet sånn at serveren startes automatisk når systemet starter.

Når det kommer til klientapplikasjonen vil testingen foregå samtidig og etter at delmålene er arbeidet med. Testingen vil bli gjort ved å installere og kjøre applikasjonen ved hjelp av flutter toolkit. Installeringen vil bli gjort enten på en android enhet eller en android emulator. Etter at delmålene er utført vil det også bli testet på iOS enheter for å sjekke om det gir samme resultat som i android. Hva som testes kommer an på hva som er jobbet med, så om det er designutvikling vil alle knappene testes og sjekkes om alt er som forventet i vertikal og landskapsmodus. I tillegg vil dette vil det bli testet med flere skjermstørrelser. Hvis det har blitt jobbet med kommunisering med server vil det også være nødvendig å sette opp en server som er koblet til samme nettverk som klienten.

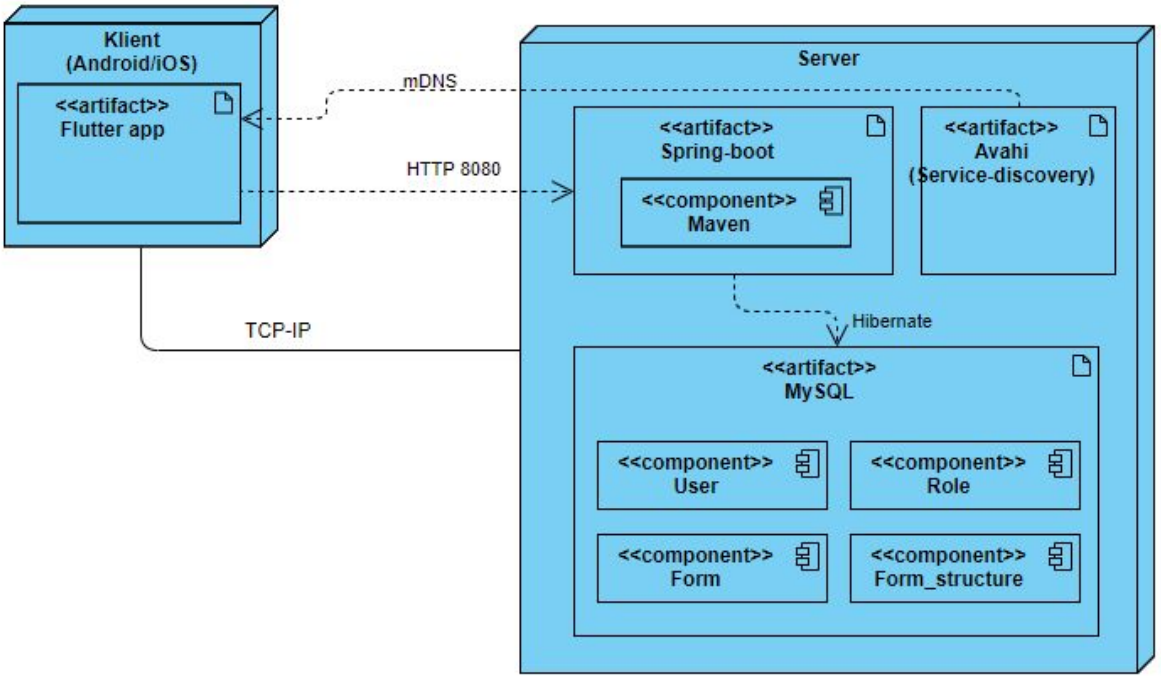
4 RESULTATER

I dette kapittelet er gruppens løsning på prosjektets problemstilling beskrevet. Dette skal utføres ved flittig bruk av skjermbilder fra applikasjonen, diverse UML-diagrammer og noe utdrag fra kildekode.

4.1 Systemarkitektur

Arkitekturen for løsningen i prosjektet er en tredelt arkitektur. Løsningen er bygget på et velkjent designmønster kalt MVC (Model-View-Controller). For arkitekturen sin del vil dette si at løsningen er oppdelt i data (model), brukergrensesnitt (view) og et mellomliggende lag som er kjent for kommunikasjonen mellom de andre delene (gjerne kalt controller).

4.1.1 Diagram for produksjonssetting



(Figur 22:Diagram deployment)

4.1.2 Dataklasser - REST api

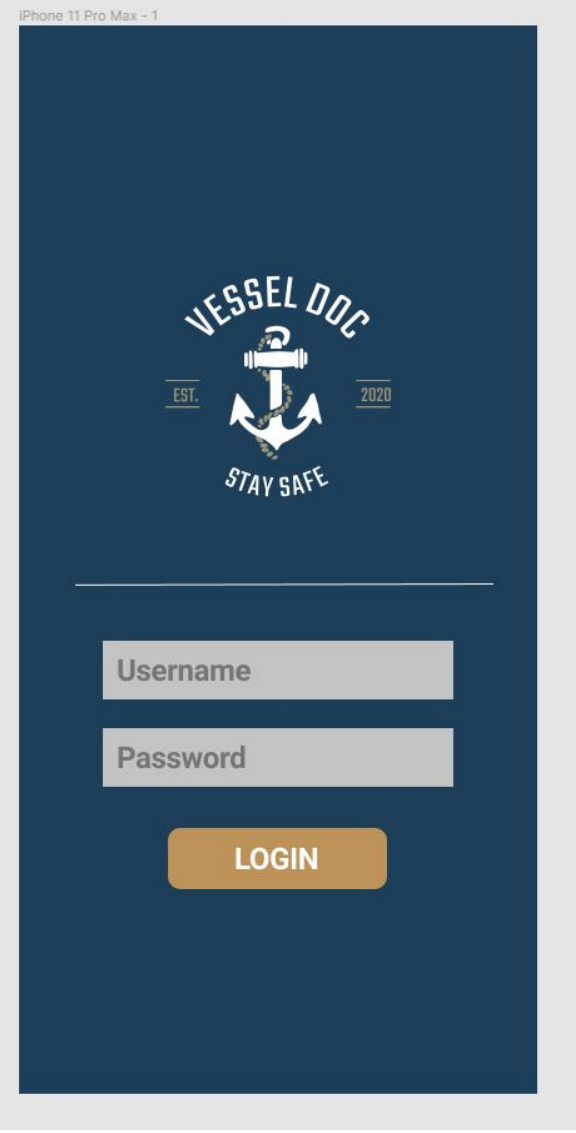
Se vedlegg 3 for en bedre oversikt.

4.1.3 Dataklasser - Grafisk grensesnitt

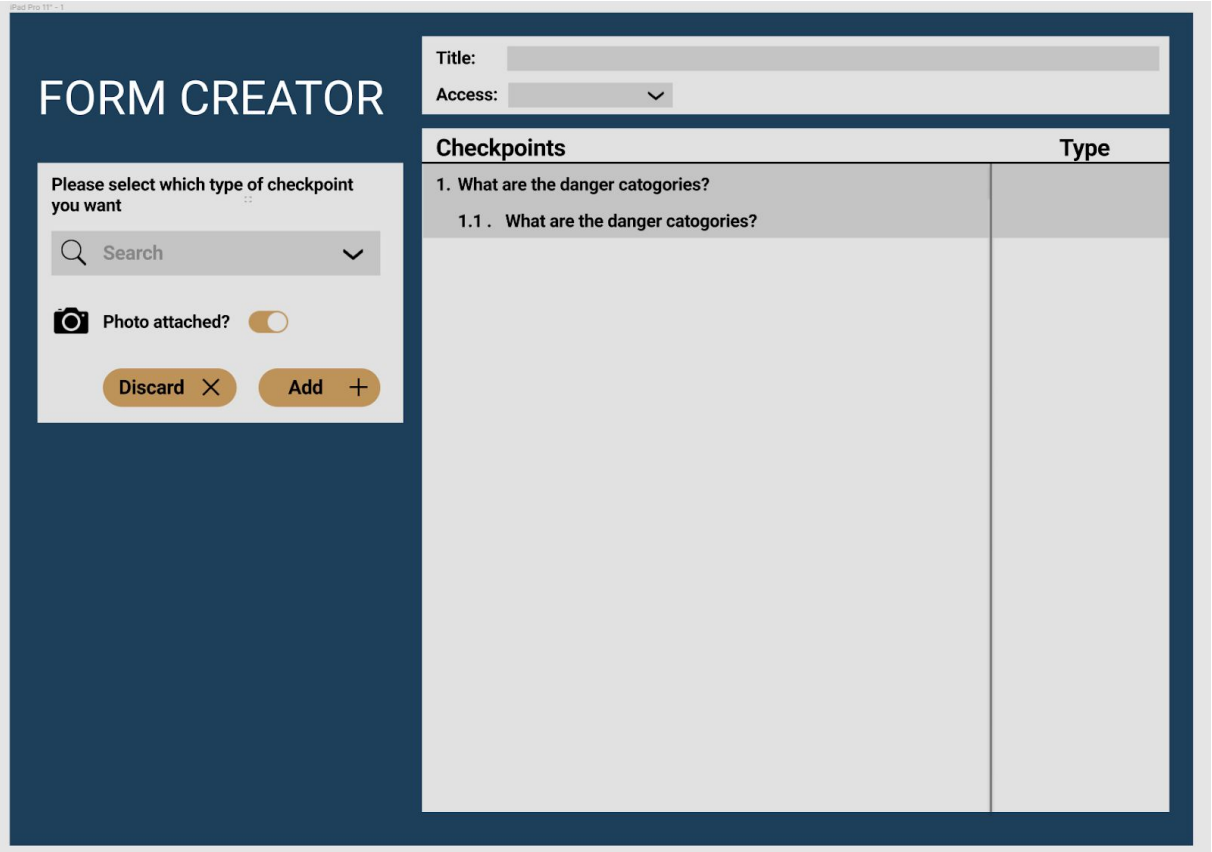
Se vedlegg 4 for en bedre oversikt.

4.1.4 Frontend struktur (UI/UX)

De fleste grafiske sidene i applikasjonen ble opprinnelig tegnet/designet i et designverktøy kalt Figma. Mer info om Figma står beskrevet i kapittel 2.12.2. Under er det lagt med diverse skjermdumper fra vårt designarbeid i figma.



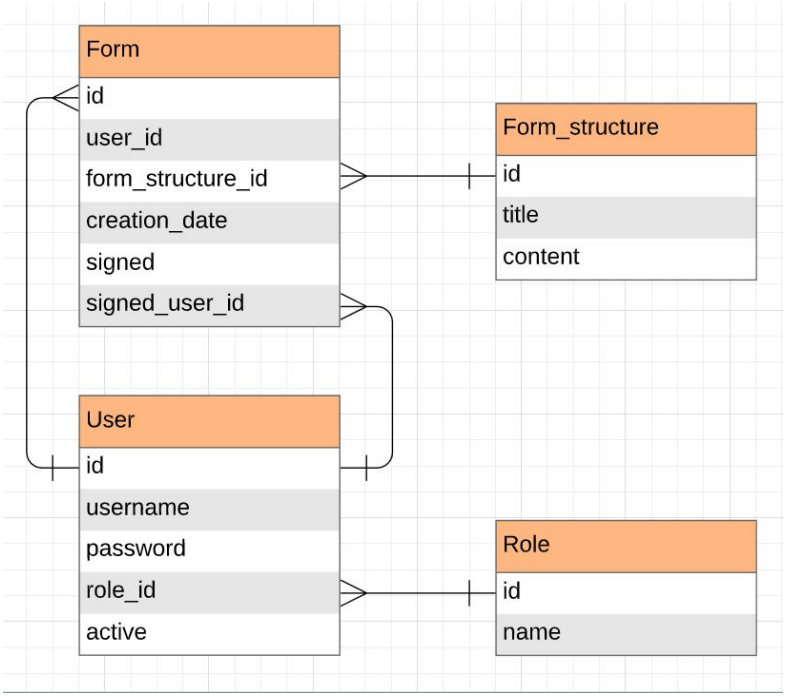
(Figur 23:UI/UX design login avbildet på Iphone 11 Pro Max)



(Figur 24 Opprinnelig UI/UX design "create form side" avbildet på Ipad 11')

4.2 Database design

4.2.1 ER Diagram



4.2.2 Entiteter

- User = Brukere av tjenesten. kan ha en rolle.
- Role = Rollene som brukerne kan ha.
- Form = Skjema som eies av en bruker. kan ha en form_structure. kan ha en bruker som har signert.
- Form_structure = Skjemastruktur som benyttes av skjema.

PK = Primary key
FK = Foreign key

User:

Attributt	DB Datatype	Java Datatype	Andre egenskaper
id(PK)	INTEGER	java.lang.Integer	Not null, unique, auto increment
username	VARCHAR	java.lang.String	Not null, unique
password	VARCHAR(61)	java.lang.String	Not null
role_id(FK)	INTEGER	java.lang.Integer	Not null
active	TINYINT(1)	java.lang.Boolean	Not null

Role:

Attributt	DB Datatype	Java Datatype	Andre egenskaper
id(PK)	INTEGER	java.lang.Integer	Not null
name	VARCHAR	java.lang.String	Not null

Form:

Attributt	DB Datatype	Java Datatype	Andre egenskaper
id(PK)	BINARY(16)	java.util.UUID	Not null, generert med org.hibernate.id.UUIDGenerator
user_id(FK)	INTEGER	java.lang.Integer	Not null
form_structure_id(FK)	INTEGER	java.lang.Integer	Not null
creation_date	DATETIME	java.uitl.Date	Not null, generert med hibernates CreationTimestamp annotation

signed	TINYINT(1)	java.lang.Boolean	Not null, default 0
signed_user_id(FK)	INTEGER	java.lang.Integer	

Form_structure:

Attributt	DB Datatype	Java Datatype	Andre egenskaper
id(PK)	INTEGER	java.lang.Integer	Not null, auto increment
title	VARCHAR	java.lang.String	Not null
content	MEDIUMBLOB	java.lang.Byte	Not null

4.3 Applikasjonens protokoll for lokalisering av tilgangspunkt

4.3.1 Server

Da det er avgjort at det skal være en lokal server i dette prosjektet ville det være fornuftig å benytte service-discovery istedenfor å måtte finne og skrive inn en ip adresse manuelt. Det viser seg at raspbian har dette innebygd. Raspbian er et operativsystem ment for Raspberry Pi som dette prosjektet har i fokus å ha støtte for. Systemet for service-discovery som raspbian benytter heter avahi, og dette bruker multicast DNS til å publisere tjenestene i nettverket.

For å få avahi til å publisere tilpasset informasjon i raspbian, må det utføres noen endringene i konfigurasjonsfilene. Det er implementert et installasjon-script(bash) som gjør dette automatisk. Dette scriptet gjør alt for å få serveren til å kjøre. For at den skal tilpasse avahi starter den med å forandre vertsnavnet til systemet, som vil bli brukt til å identifisere at dette er en VesselDoc-server:

```
sudo echo "vesseldoc" > /etc/hostname
sudo sed -i 's/127.0.1.1.*/127.0.1.1\tvesseldoc/g' /etc/hosts
sudo hostnamectl set-hostname "vesseldoc"
```

Deretter forandres en konfigurasjon for å aktivere en modul som gir ut vertsnavn til det lokale nettverket[38]. Vertsnavnet vil være under .local toppdomenet:

```
sudo sed -i 's/\[NOTFOUND=return\] dns/\[NOTFOUND=return\] resolve
\[!UNAVAIL=return\] dns/g' /etc/nsswitch.conf     (I en linje)
```

Etter dette skrives det en konfigurasjonsfil under /etc/avahi/services/ for å fortelle avahi at det er en tjeneste som skal publiseres:

```
sudo echo "<?xml version=\"1.0\" standalone='no'?><!--*-nxml-*-->
<service-group>
 <name replace-wildcards=\"yes\">%h</name>
 <service>
  <type>_http._tcp</type>
  <port>8080</port>
 </service>
</service-group>
" > /etc/avahi/services/vesseldoc-server.service
```

Til slutt restartes avahi som deretter publiseres som en tjeneste.

```
→ ~ avahi-browse -la | grep vesseldoc
+ wlp4s0 IPv6 vesseldoc                                  Web Site             local
+ wlp4s0 IPv4 vesseldoc                                  Web Site             local
```

(Figur 25: Terminal fra en annen maskin i nettverket. avahi-browse lister DNS-SD)

4.3.2 Klient

Etter at serveren er oppe og kjører, gjenstår det å la klienten finne den. Dette gjøres ved hjelp av service discovery. For at en flutter applikasjon skal finne tjenester publisert med service discovery, trenger en først å starte en multicast DNS klient.

```
final MDnsClient client = MDnsClient();

await client.start();
```

Deretter bruker den en DNS pointer. En sånn pointer gir et domene ut fra en IP-adresse[39]. Pointeren vil iterere gjennom IP-adressene den finner i

[38] "nss-resolve - Freedesktop.org." https://www.freedesktop.org/software/systemd/man/nss-resolve.html. Accessed 10 May. 2020.
[39] "What Is A DNS PTR Record? | Cloudflare." https://www.cloudflare.com/learning/dns/dns-records/dns-ptr-record/. Accessed 11 May. 2020.

det lokale nettverket. I hver iterasjon vil domenet bli plassert i en variabel kalt bundleId.

```
await for (PtrResourceRecord ptr in client
    .lookup<PtrResourceRecord>(ResourceRecordQuery.serverPointer(name))) {
  await for (SrvResourceRecord srv in client.lookup<SrvResourceRecord>(
      ResourceRecordQuery.service(ptr.domainName))) {
    await for (IPAddressResourceRecord ip
        in client.lookup<IPAddressResourceRecord>(
            ResourceRecordQuery.addressIPv4(srv.target))) {
      final String bundleId = ptr.domainName;
```

Etter dette vil variabelen med domenet sjekkes i et filter for å se om det blir identifisert som en VesselDoc server. Om den finner en match, vil den lagre IP-adressen i applikasjonens cache-lagring.

```
final String bundleId = ptr.domainName;

if (bundleId == 'vesseldoc._http._tcp.local') {
  print(
      "Found: ${ip.address.address}:${srv.port} with hostname ${srv.target}");
  sas.writeAddress("${ip.address.address}:${srv.port}");
  hasServer = true;
}
```

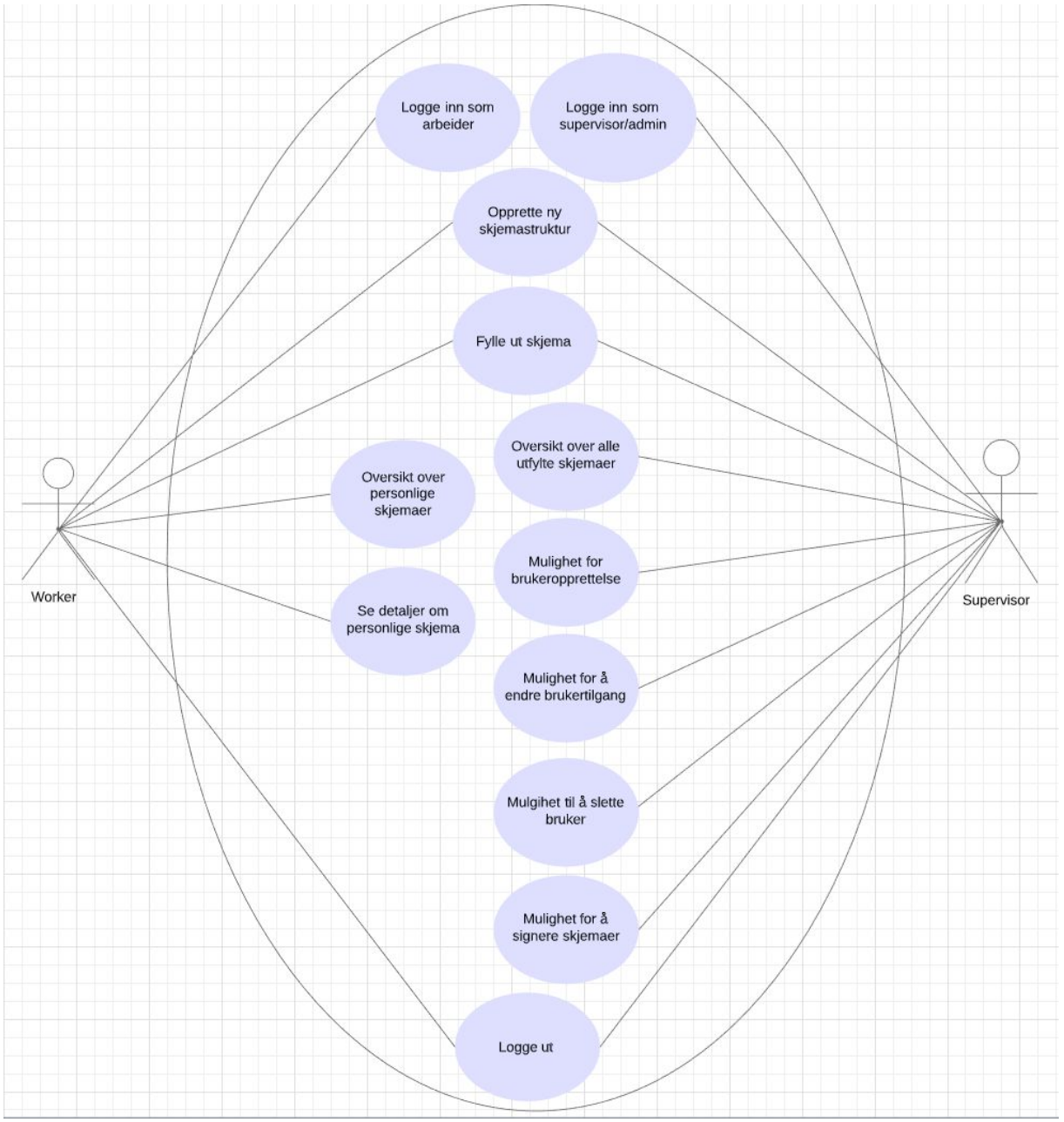
4.4 Skjema i applikasjonen (struktur og håndtering)

For selve skjemastrukturen i prosjektet bestemte utviklingsteamet seg for å nytte en tilleggspakke i flutter. Tilleggspakken het json_to_form, og ga teamet en utgangspunkt for en skjemastruktur standard. Ved slike tilleggspakker har man muligheter for å klone git repositoryen, å deretter legge til sine egne endringer for gitt pakke. Teamet gjorde det på denne måten for å kunne utførte de nødvendige endringene som måtte til for å oppnå ønsket skjemastruktur. Når skjemastrukturen var i boks, var det selve skjemahåndteringen som ble neste på listen. Her ble det gjort vurderinger om skjemaer skulle ha en egen tabell i databasen med sine felter, eller om skjemaer skulle lagres som filer i stedet. Siden det ikke lå noe til grunn for endring av data som allerede var sendt inn, falt beslutningen på å lagre det som filer.

4.5 Applikasjonens plattform kompatibilitet/responsivitet

Siden applikasjonen (VesselDoc) ble bygget i et såkalt "tverr-plattform rammeverk", fungerer denne løsningen like bra på alle enheter som kjører iOS eller android OS (uavhengig av operativsystemet). Dette er mulig fordi utviklingsteam bestemte seg for å bruke Flutter rammeverket i byggingen av applikasjonen, se kap. 2.8 for mer detaljer om rammeverket. Utviklingsteam argumenter for at dette er det beste rammeverket for tverr plattform utvikling, siden erfaringen tilsier at rammeverket fungerer sømløst på kryss av enheter og OS. En annen erfaring gruppen oppnådde gjennom prosjektet var med tanke på design-muligheter. Med andre ord; vakrere design er lettere å implementere i flutter enn i andre rammeverk. Løsningen er også bygget som en responsiv mobilapplikasjon hvor skjermbildene dimensjonerer seg ut i fra gitt skjermstørrelse/areal. Måten dette er utført på er ved prosentberegning av applikasjonens elementer, og ved bruk av egne skjermbilder som kun er ment for skjermer med nettbrett-størrelse. Gjennom kapittel 4.7 viser gruppen skjermbilder fra applikasjonen med ulike skjermstørrelser og operativsystemer, på testenhetene.

4.6 Applikasjonens funksjonalitet med roller (use case-diagram)



(Figur 26: Use Case-diagram for worker og supervisor/admin)

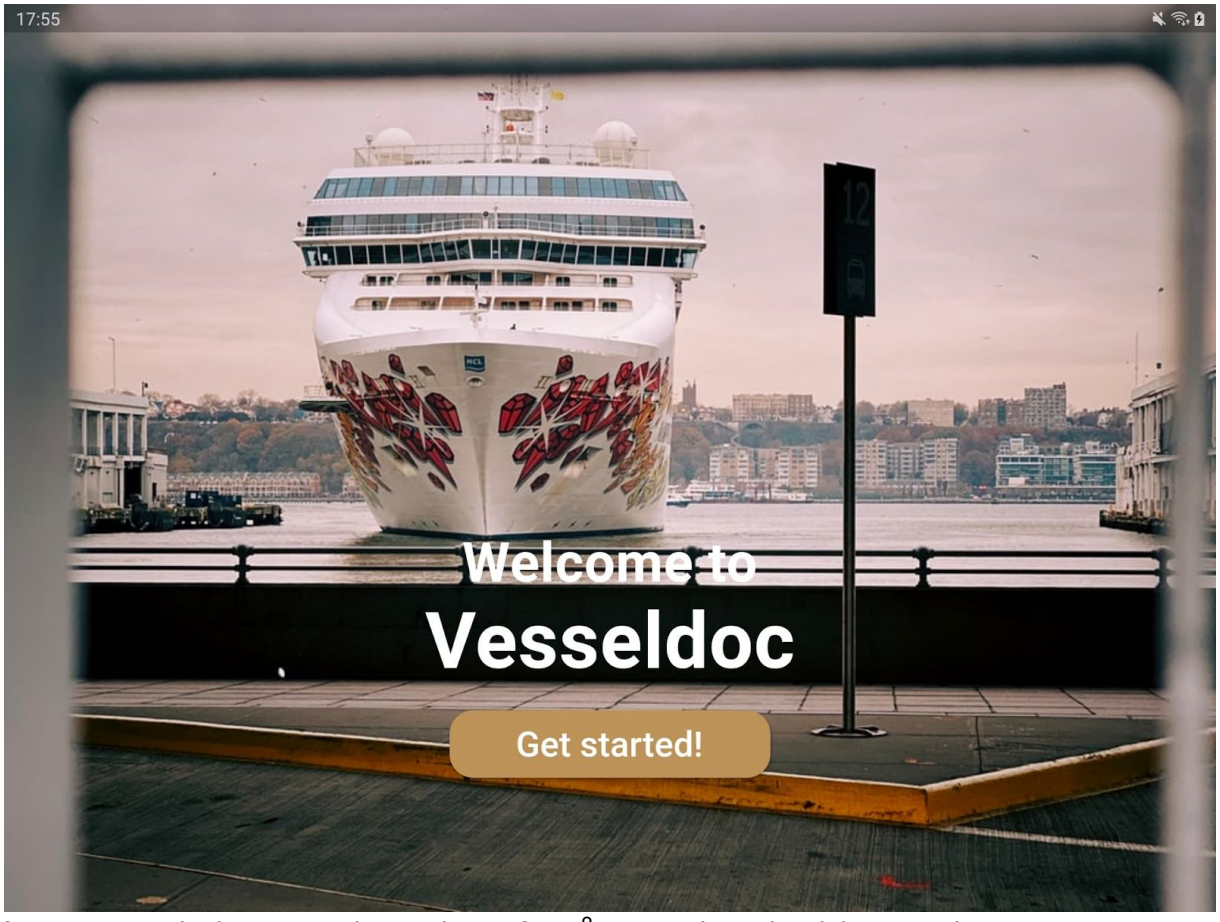
4.7 VesselDoc App - gjennomgang av løsning

Utarbeidet løsning etter en inkrementell utviklingsprosess er en tverr plattforms applikasjon, og et tilstandsløst api (REST API) hvor det blir håndtert data persistert i en relasjonsdatabase. Applikasjonen har mulighet for å hente eget tilgangspunkt til gruppens produserte api, og

blir også sett på som responsiv. I dette kapittelet vil utviklingsteam forklare selve løsningen, både visuelt og med tanke det som foregår bak teppet.

4.7.1 Splash side

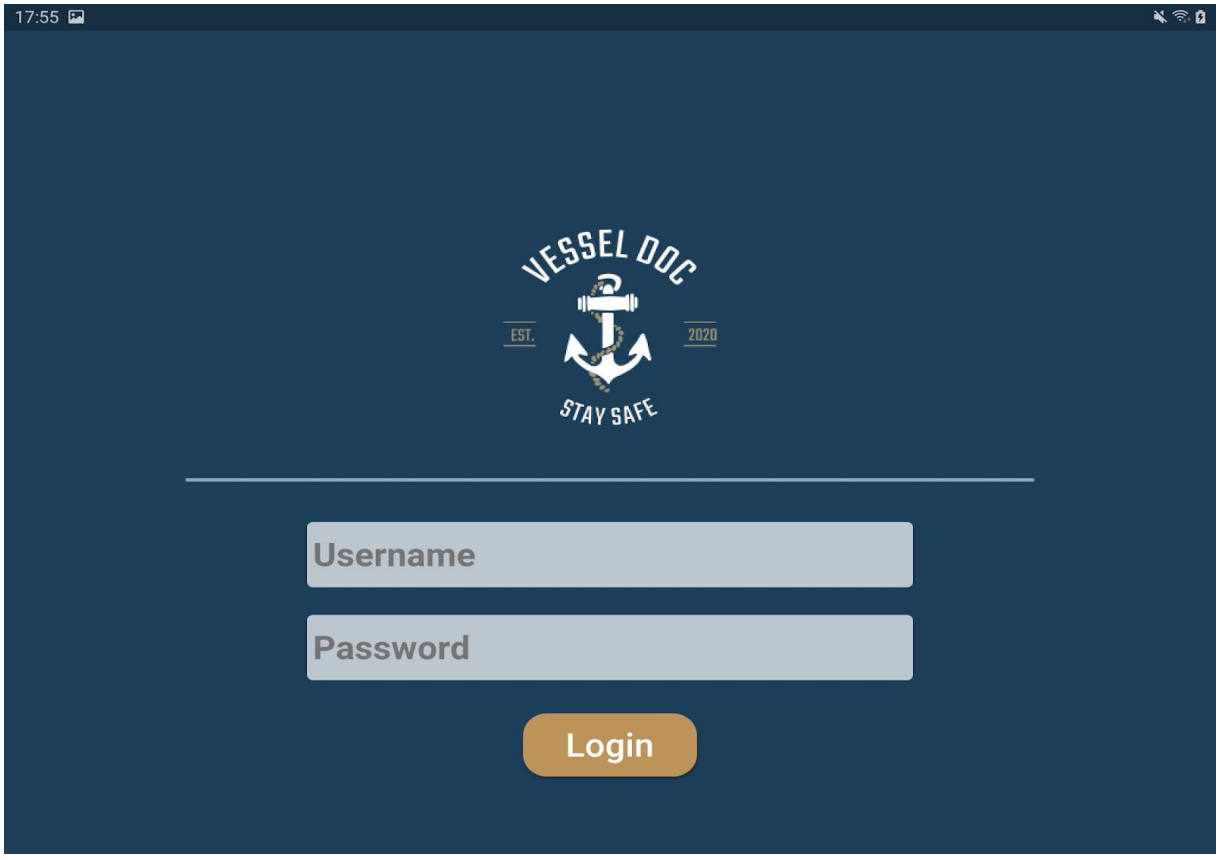
Det første man møter når man åpner applikasjonen er en enkel splash-side med en singel knapp for å "komme i gang". Når man trykker på denne knappen skjer det ikke så mye mer,rent grafisk, enn en ren omdirigering til login-siden. Det som folk ikke vet her er at det foregår mer under panseret. I bakgrunnen kjører applikasjonen en mDNS protokoll for å hente ut tilgangspunktet til det tilstandsløse API'et (som vertes via en enhet på det lokale nettverket ombord i skipet.) Under er det lagt med et skjermdump for visning av hvordan denne siden ser ut på en ekte enhet.



(Figur 27: Splash screen skjermdump fra vår testenhet i landskapsmodus: Samsung Galaxy Tab S3)

4.7.2 Innlogging

Når appen har fullstendig hentet sitt kontaktpunkt med REST-api'et og omdirigert til login-siden vil man kunne se et påloggingsvindu. Her må bruker angi brukernavn og passord i de to tekst-feltene.



(Figur 28: Login skjermdump fra vår testenhet i landskapsmodus: Samsung Galaxy Tab S3)

Før en eventuell autentisering tar plass, vil en skjemavalidering sjekke om disse feltene er fylt ut, evt gir bruker beskjed om noen av feltene mangler.

(Figur 29: Skjemavalideringen ved innlogging vist med manglende passord)

Kommer man seg igjennom skjemavalideringen vil trykket på knappen starte en rekke med asynkrone oppgaver. Først av alt vil det grafiske grensesnittet gi bruker tilbakemelding om at knappen er trykket, og at autentiseringen er i gang. Denne tilbakemeldingen blir kommunisert ut ved bruk av en snackbar med en prosessindikator og en melding: "Signing in...", som dukker opp i bunnen av skjermen. Det ble valgt å løse det på denne måten siden et av hovedprinsippene i Don Normans 'prinsipper om interaktivt design', er tilbakemelding (beskrevet i kapittel 2.12.1). Ved bruk av en slik snackbar argumenterer utviklingsteamet for at bruker er gitt tilstrekkelig tilbakemelding om at noe er i ferd med å skje, på bakgrunn av deres trykk på knappen.

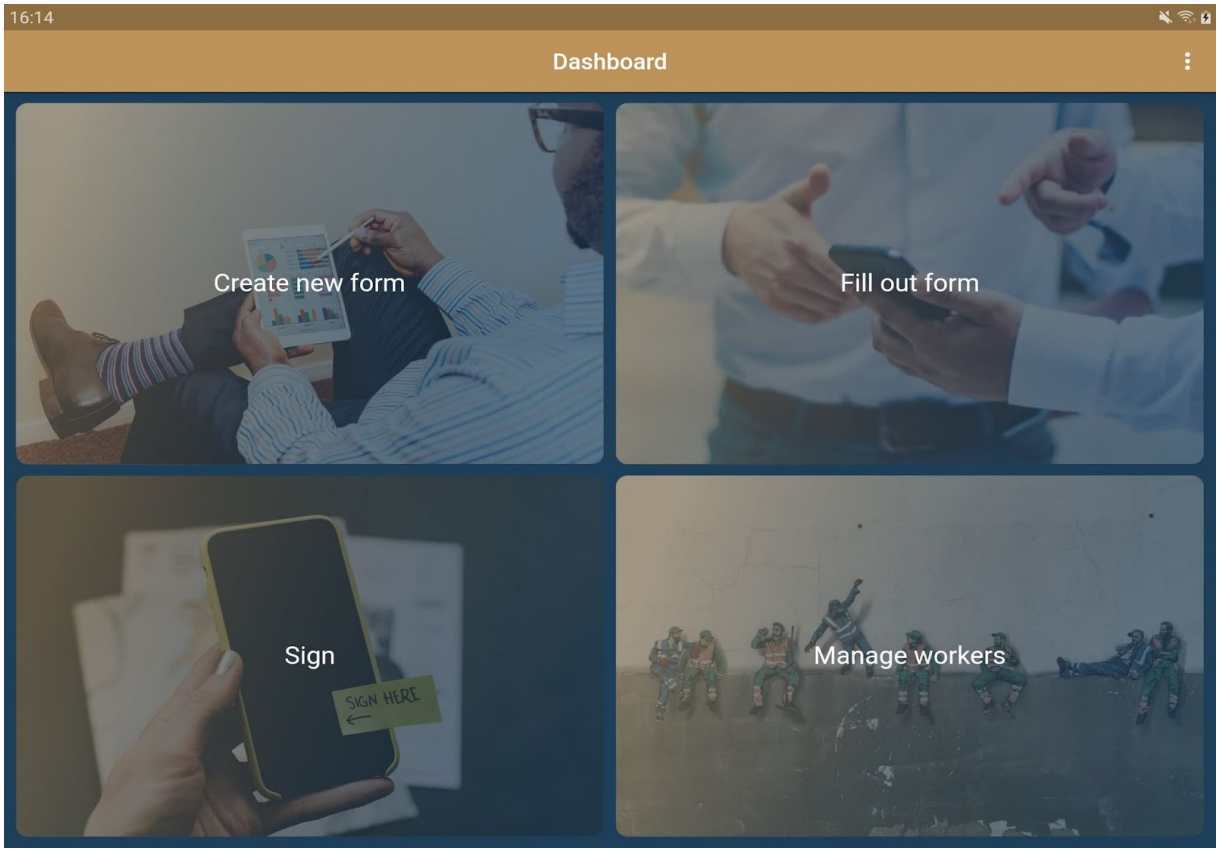
(Figur 30: Snackbar i login siden, som dukker opp ved autentiseringen)

Den neste oppgaven som tar plass bak teppet er en autentiserings-request til det tilstandsløse api'et. Denne forespørselen inneholder noen påkrevde parametere som de nylig inntastede brukernavnet og passordet fra brukeren. Forespørselen forventer så å få tilbake en statuskode med suksess eller ikke. Dette gjør den med å få svar som HTTP 200 OK, eller en type HTTP feilkode fra server. Om forespørselen gir 200 OK, kan det grafiske grensesnittet gå ut i fra at det var en suksess, og at bruker ble korrekt validert. For autentiseringsdelen i denne applikasjonen ble de valgt å bruke JWT (Json Web Token) som brukerens tilgangstoken. For mer info om JWT, se kap 3.8.4. Grunnen for at utviklingsteamet gikk for JWT i denne løsningen var fordi standarden blir sett på som en sikker metode for overføring av informasjon mellom parter. I tillegg til dette, er en slik token liten i størrelsen. Det vil si at den kan overføres raskt gjennom en URL eller som et forespørsel parameter. Sammenlignet med andre tilgangstoken (f.eks. SAML) er JWT både sett på som mer sikker, vanligere og lettere å prosessere.

Ved korrekt autentisering, vil server gi tilbake en slik JWT/tilgangstoken i sitt svar. Denne vil applikasjonen ta vare på, å bruke på samme måte som man en nettleser gjør med informasjonskapsler. Med andre ord vil alle videre forespørsler til api'et ha denne tilgangstokenen inkludert. Om tilgangstoken er ugyldig vil ikke api'et gi ut noe annet enn en feilkode.

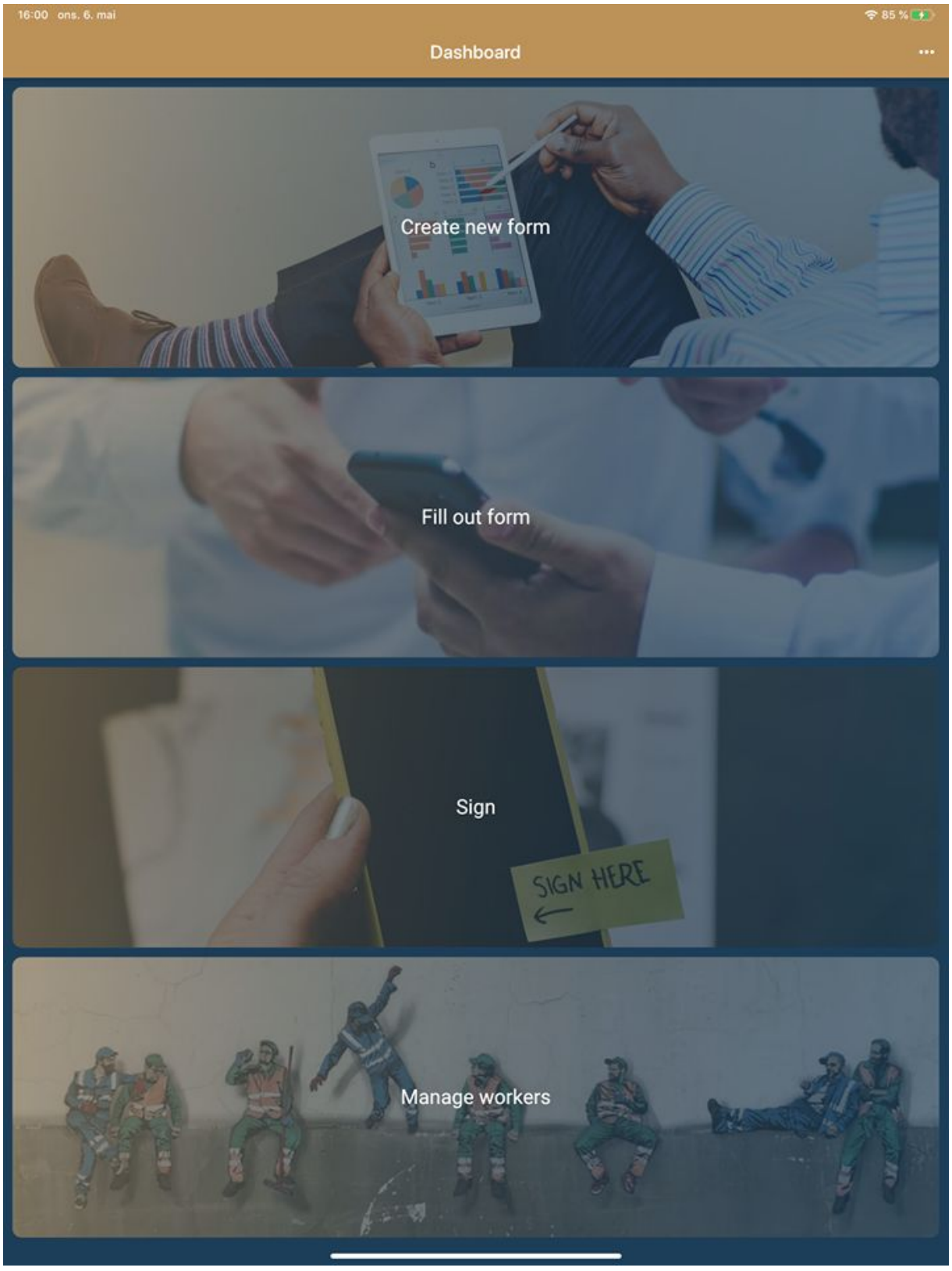
4.7.3 Dashboard

Om bruker ble korrekt logget inn, vil applikasjonen lede oss til en ny side. Dette er en dashboard side som er bygget ut i fra hvilken rolle brukeren har, og om man holder valgt enhet i landskap eller portrettmodus. Med en dashboard side, menes det en hovedmeny som kan ta bruker til hvilken som helst ønsket side i applikasjonen (som brukeren er autorisert til). Hos en overordnet ('admin'-rolle) vil dashboard siden inneholde fire store knapper, hvor hos en arbeider ('worker'-rolle) kun tre. Dette er fordi en arbeider ikke har tilgang til å styre brukertilganger, og eventuelle andre brukerendringer.



(Figur 31: Dashboard skjermdump fra vår testenhet i landskapsmodus: Samsung Galaxy Tab S3)

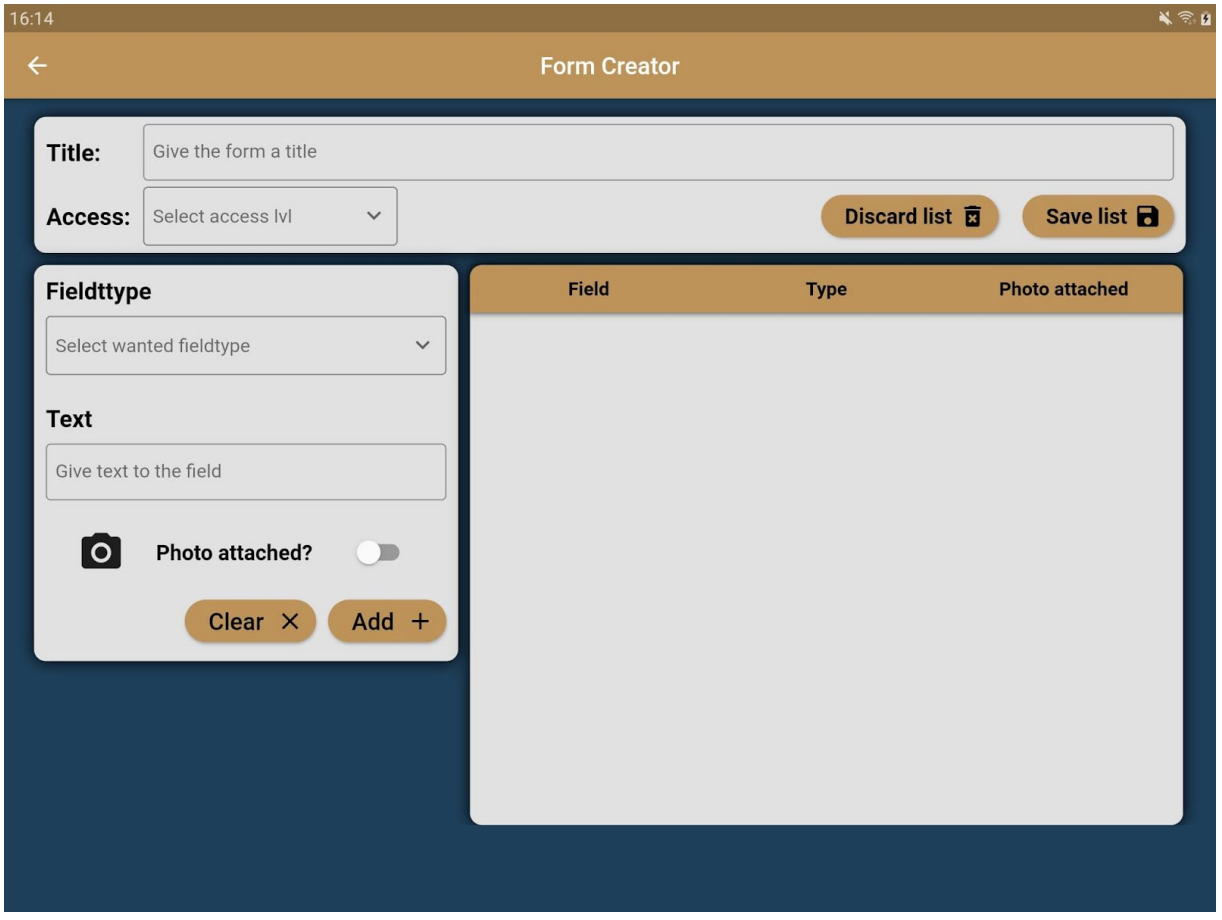
Man kan se at at knappene er laget med farger som veksler fra hampetau-farge til marineblått.



(Figur 32: Samme side som avbildet over, men på annen testenhet i portrettmodus: Ipad Pro 12.9)

4.7.4 Side for opprettelse av et nytt skjema

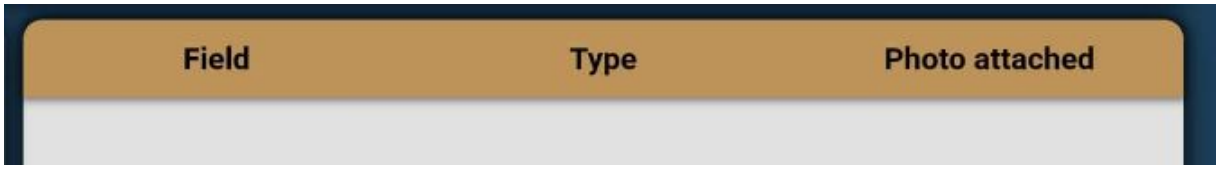
Denne siden i applikasjonen er for opprettelse av nye skjemaer. Det er her påtenkt at en bruker skal kunne opprette en ny utfyllbar skjemastruktur eller replikere en skjemastruktur fra en tidligere PDF (hvor utfyllingen har foregått på papir.) Siden blir kalt for "Form Creator", og blir dynamisk formet ut i fra brukerens enhetstype som i nettbrett eller telefon. På et nettbrett tegnes boksene ved siden av hverandre for å få mer utnyttelse av skjermstørrelsen, hvor de på mobil tegnes over hverandre.



(Figur 33: Form Creator skjermdump fra vår testenhet i landskapsmodus: Samsung Galaxy Tab S3)

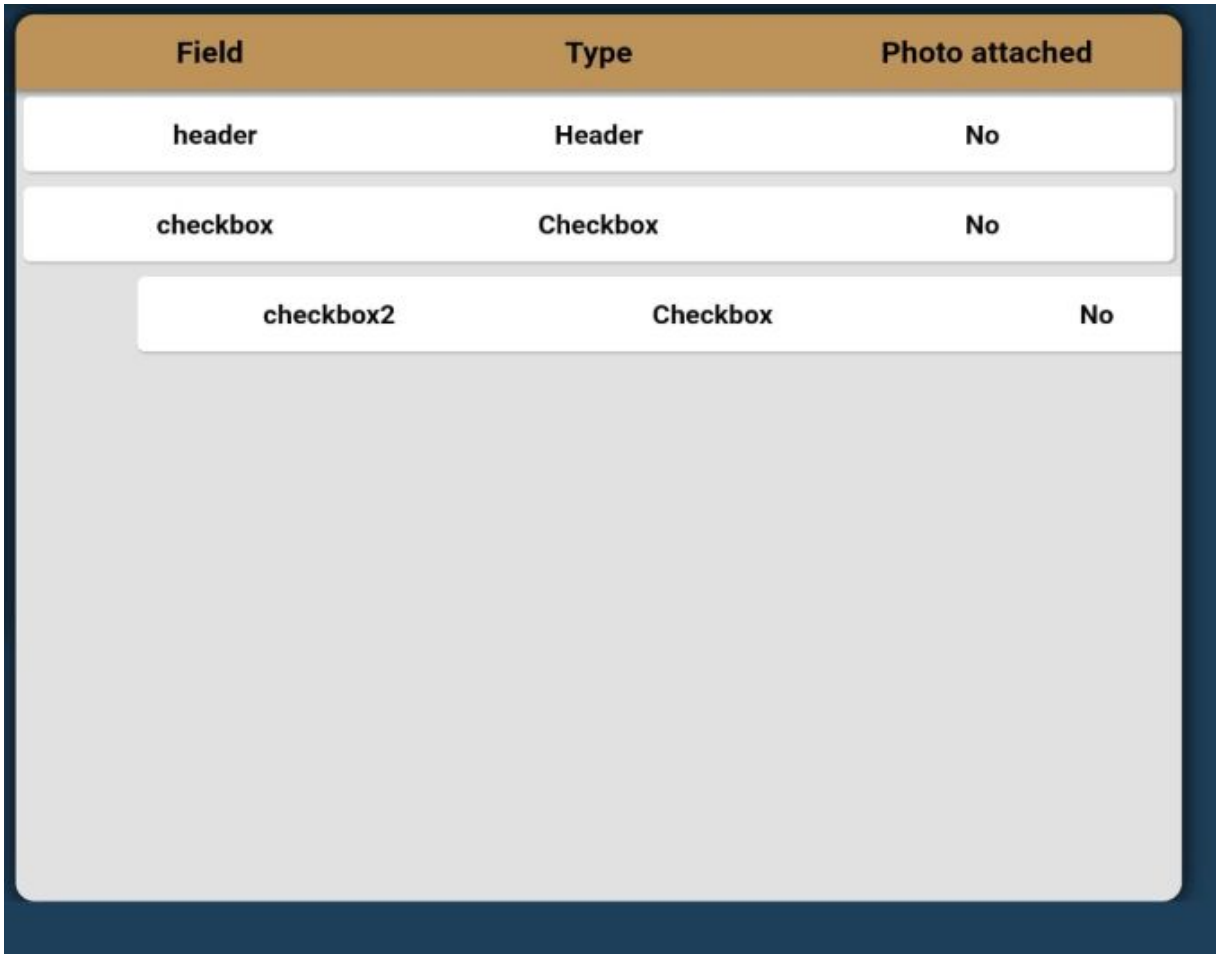
Det er tre bokser på denne siden; den første for å definere tittel på skjemaet og tilgangen for hvilke brukere. Den andre boksen er for å legge til ønsket felttype, og forandre teksten på den. Den tredje og siste boksen

er for nåværende oversikt over skjemaet som blir laget. Her fylles det ut en liste av ulike felter på bakgrunn av gitt brukerinput. Denne oversikten har tre kolonner: teksten på feltet, felttype og evt om bilder skal kunne legges ved feltet.

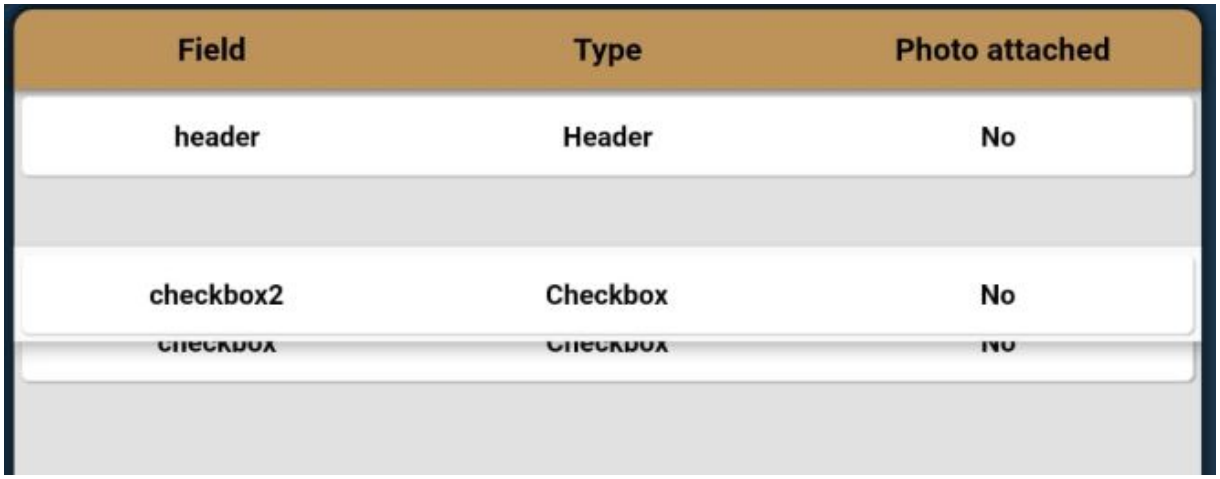


(Figur 34: Kolonner i skjemaoversikt)

Listen av felter er redigerbar. Om man vil slette et felt eller flytte det til en ny posisjon i listen kan man gjøre dette. Her kommer utviklingsteam tilbake til Don Normans prinsipper hvor det ene prinsippet er konsistens. Teamet tolker denne "sveip-for-sletting" funksjonen noe som er konsistent brukt i kommersielle anerkjente applikasjoner. Dette gjør at brukeren ubevisst vet hvordan man sletter et felt uten noen form for opplæring.

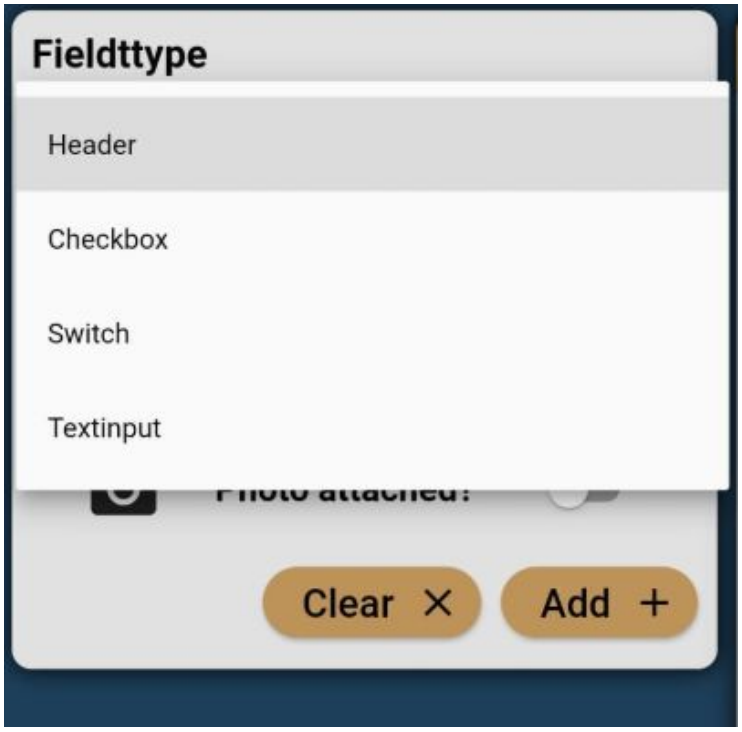


(Figur 35: 'Sveip for sletting' funksjon for sletting av valgte felt)



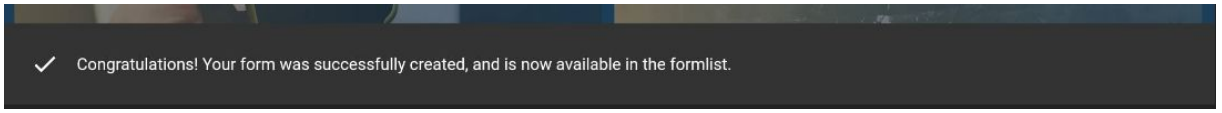
(Figur 36: Drag and drop i listen over tilgjengelige felter)

Når det skal velges felttyper i en skjemaopprettelse har bruker en dropdown-meny med fire ulike felttyper som valgfrie alternativer. Det inkluderer overskrift, sjekkboks, bryter, og tekstfelt. Samme boks inneholder også to knapper nederst, en for å legge til punkt, og den andre for tømming av utfylt data.



(Figur 37: Drop Down-meny i over valg av felttyper)

Man kan velge å enten lagre eller slette skjemastrukturen. Ved en eventuell sletting vil det helt enkelt tømme feltene i siden og omdirigere til dashboard-siden. Vil man heller lagre den nye skjemastrukturen, trykker bruker på lagre-knappen i stedet. Dette vil på samme måte som tidligere starte en rekke av asynkrone oppgaver. Rent grafisk vil applikasjonen gi tilbakemelding om at skjemaet lagres (i form av en ny snackbar), og etter en liten stund omdirigere brukeren til dashbordet. Samtidig vil det komme en ny tilbakemelding vedrørende innsendingen av skjemastrukturen.



(Figur 38: Snackbar med suksessfull innsending av ny skjemastruktur)

Tilbakemeldingen (som avbildet over) informerer brukeren om at den asynkrone sendingen til api'et gikk som forventet, og at den innsendte skjemastrukturen skal være tilgjengelig i listen over skjemastrukturer.

Når det er snakk om hva som skjer i bakgrunnen her, er dette en blanding av ulike hendelser. Skjemastrukturene blir dynamisk bygget som en liste av elementer i en JSON-fil. En funksjon blir kjørt for å bygge denne JSON-strukturen. Når funksjonen er ferdig å kjøre blir strukturen skrevet til en fil. Helt tilslutt blir den opprettede filen lagt med som et parameter i sendingen til api'et.

4.7.5 Side for utfylling av skjemaer

Siden for utfylling består av to individuelle skjermbilder; et for listing av tilgjengelige skjemastrukturer, og et for selve utfyllingen av valgt struktur. Det første som møter brukeren i denne siden er listen av tilgjengelige skjemastrukturer. For å få bygget denne listen må applikasjonen nok en gang utføre en asynkron forespørsel til api'et. Api'et

svarer på forespørselen med å levere tilbake listen av tilgjengelige skjemastrukturer. Brukeren får tilbakemelding parallelt med forespørselen, ved benyttelse av en prosessindikator.

Vedlagt under er en av funksjonene brukt i prosjektet, hvor man kan se hvordan rammeverket håndterer denne form for asynkrone hendelser. Funksjonen venter på en såkalt "Future", som er ganske selvforklarende hva er. Den representerer en verdi som vil være tilgjengelig en eller annen gang i fremtiden. Bruken av en "future" muliggjør det for utviklerne å bygge en prosessindikator så lenge den asynkrone oppgaven ikke er ferdig.

```
52
53    Widget buildList() {
54      return Container(
55        child: FutureBuilder<List<FormStructure>>(
56          future: _future,
57          builder: (BuildContext context, AsyncSnapshot snapshot) {
58            if (snapshot.connectionState == ConnectionState.done) {
59              return ListView.builder(
60                  itemCount: snapshot.data.length,
61                  itemBuilder: (BuildContext context, int index) {
62                    FormStructure formStruct = snapshot.data[index];
63                    return GestureDetector(
64                        onTap: () => Navigator.push(
65                            context,
66                            MaterialPageRoute(
67                                builder: (_) =>
68                                    FormFillerScreen(formStructure: formStruct))), // MaterialPageRoute
69                        child: Card(
70                          child: ListTile(
71                            title: Center(
72                              child: Text(formStruct.name),
73                            ), // Center
74                          ), // ListTile
75                        )); // Card // GestureDetector
76                  }); // ListView.builder
77            }
78            else{
79              return Center(child: new CircularProgressIndicator());
80            }
81          },
82        ), // FutureBuilder
83      ); // Container
84    }
85  }
86
```

(Figur 39: FutureBuilder er en av de tilgjengelige widget-komponentene i Flutter-rammeverket)

Som nevnt tidligere vil applikasjonen tegne ferdig skjermbildet med skjemastrukturer når fremtids-forespørselen har levert et svar. Når

skjermbildet er ferdig tegnet vil bruker se noe lignende som i bildet under. Hver skjemastruktur i listen er sitt egne individuelle element som også er trykkbart for å gå til neste side (utfylling av spesifikt skjema).

(Figur 40: Liste over tilgjengelige skjermstrukturer - skjermbilde fra vår testenhet i portrettmodus: Huawei P20 Pro)

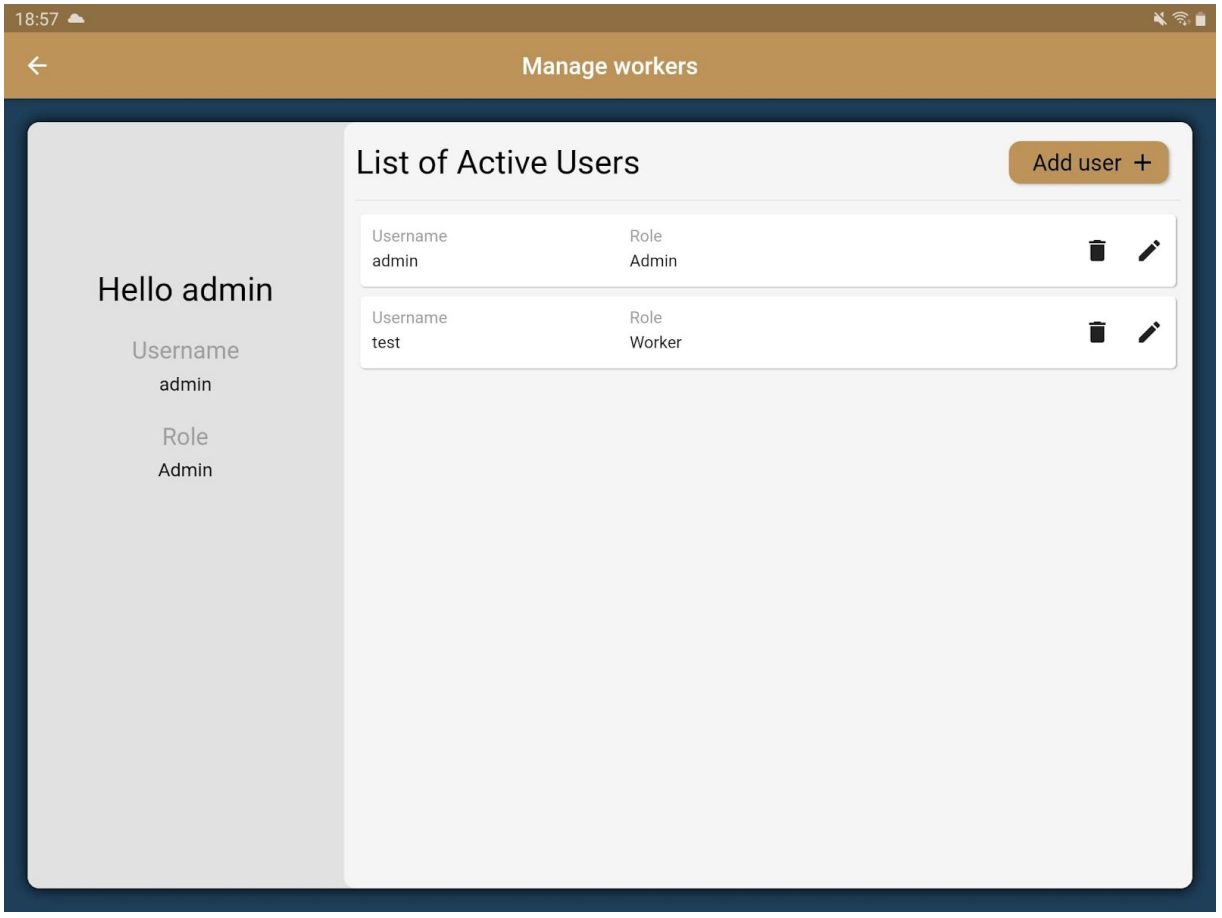
Når bruker har bestemt seg for et skjema kan en enkelt trykke på ønsket struktur, noe som vil lede til det andre skjermbildet av siden. I byggingen av neste skjermbilde vil applikasjonen gjøre en ny forespørsel, hvor den spør om brukerens ønskede skjemastruktur. Utfyllings-skjermbildet blir dynamisk bygget ut i fra valgt skjemastruktur, noe som demonstreres i figuren under.

(Figur 41: Spesifikt skjemastruktur med kun en overskrift og en sjekkboks - skjermdump fra vår testenhet i portrettmodus: Huawei P20 Pro)

Om skjemaet er ferdig utfylt kan bruker trykke på send knappen. En slik gest signaliserer til applikasjonen at den skal håndtere utfylt data, og deretter foreta en sending til api'et med det utfylte skjemaet lagt ved. I mellomtiden vil det grafiske i applikasjonen gi bruker tilbakemelding, og omdirigere til dashboard-siden igjen. Helt tilslutt vil det komme en tilbakemelding om sendingen gikk så den skulle, eller evt om noe gikk galt.

4.7.6 Administrasjonsside (styring av brukere, sette tilganger/roller, etc)

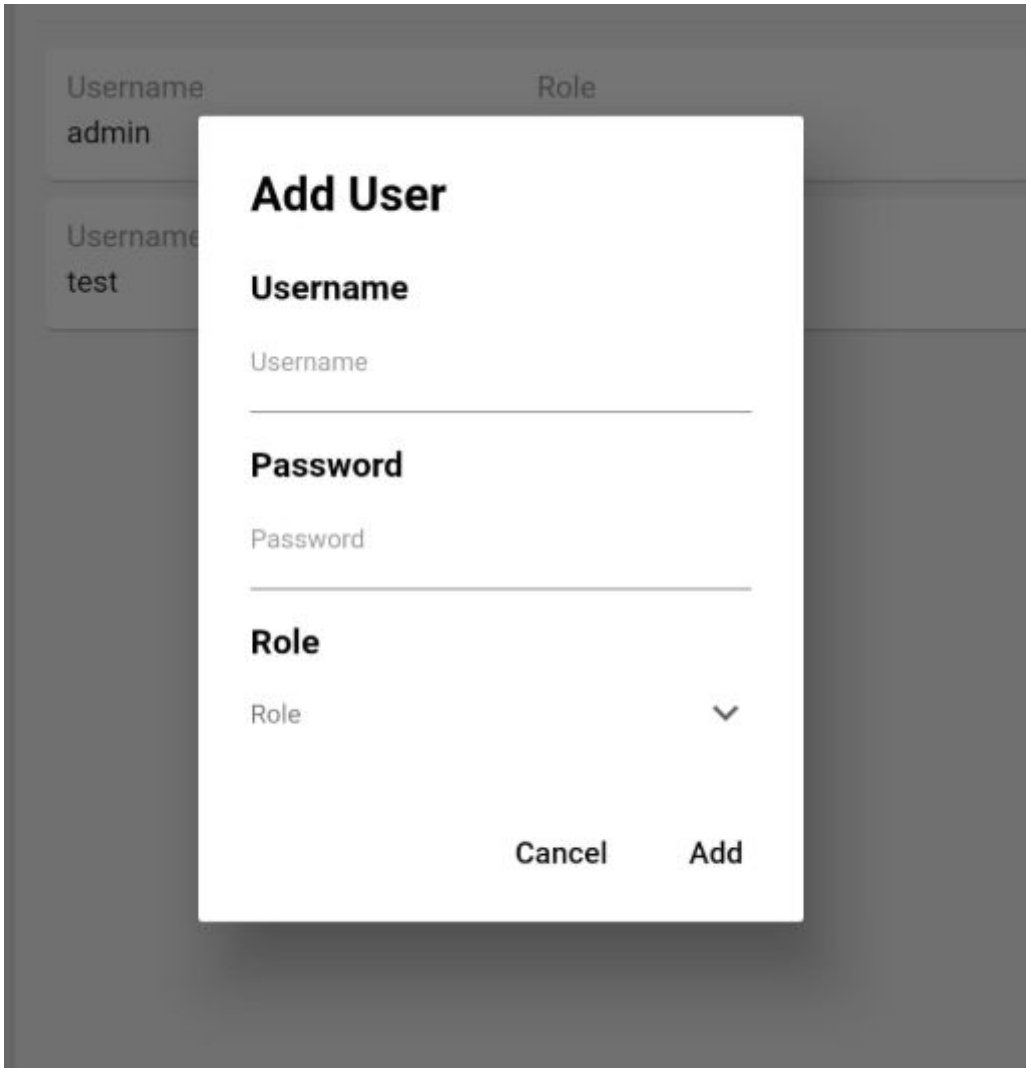
For administrering av brukere i applikasjonen har utviklingsteam laget en administrasjonsside hvor flere funksjoner er støttet. Blant annet funksjoner som: å legge til nye brukere, slette brukere eller eventuelt endre tilgangen for en bruker. Administrasjonssiden er kun tilgjengelig for "admin"-brukere som i de fleste tilfeller vil bestå av overordnede. I byggingen av denne siden hentes det noe informasjon fra api'et, som den nåværende listen av aktive brukere i databasen.



(Figur 42: Administrasjonsside skjermdump fra vår testenhet i landskapsmodus: Samsung Galaxy Tab S3)

Som det fremlegges i figuren kan man se at det er to brukere i nåværende applikasjon, hvor begge brukerne disponerer forskjellige roller. Skal det legges til en bruker vil man trykke på "add user" knappen.

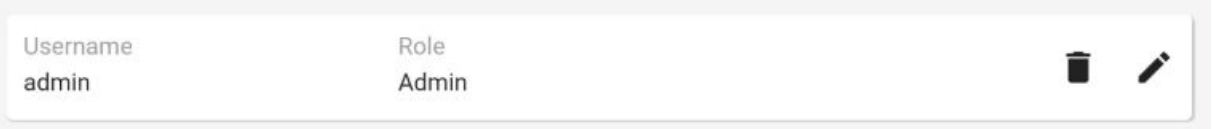
Ved trykk på "add user" knappen vil applikasjonen dra frem et popup vindu med tre utfyllbare felter. Her er det ment at overordnet skal fylle ut feltene for opprettelse av en ny bruker.



(Figur 43: Legge til ny bruker vindu)

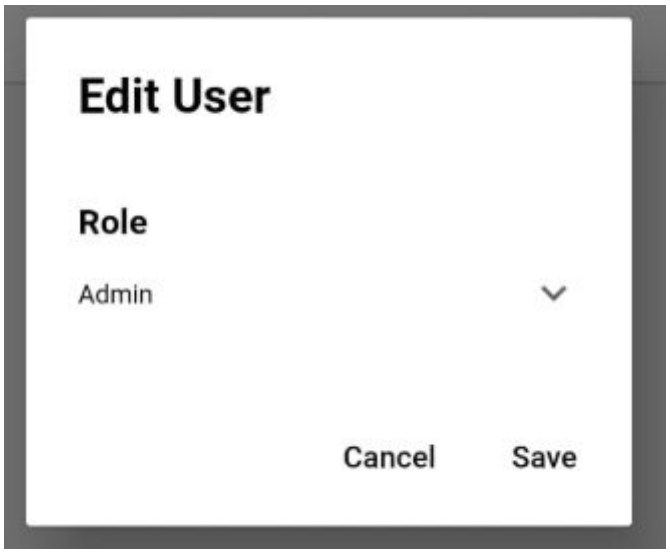
Når de tre feltene i popup vinduet er fylt ut kan en admin legge til en ny bruker ved å trykke på "add" - knappen. Da vil applikasjonen foreta en sending til api'et, og en ny forespørsel for den oppdaterte brukerlisten. Tilbakemelding vil her også skje i form av en snackbar, prosessindikator og en melding til brukeren.

Admin har også mulighet for å enten slette en bruker eller tildele den en annen rolle. Det to knappene på enden av brukerelementet i listen vil igangsette en av disse aktivitetene.



(Figur 44: Et brukerelement i listen over brukere, med to knapper på enden for redigering av en bruker)

Endring av brukertilgang/rolle vil bestå av en nesten identisk dialogvindu som ved registreringen av en ny bruker. Dette er vist i figuren under. På samme måte som ved registrering vil denne endringen igangsette en sending til api, og oppdatere brukerlisten enda en gang.



(Figur 45: Dialogvindu for endring av brukertilgang)

4.7.7 Side for signeringer

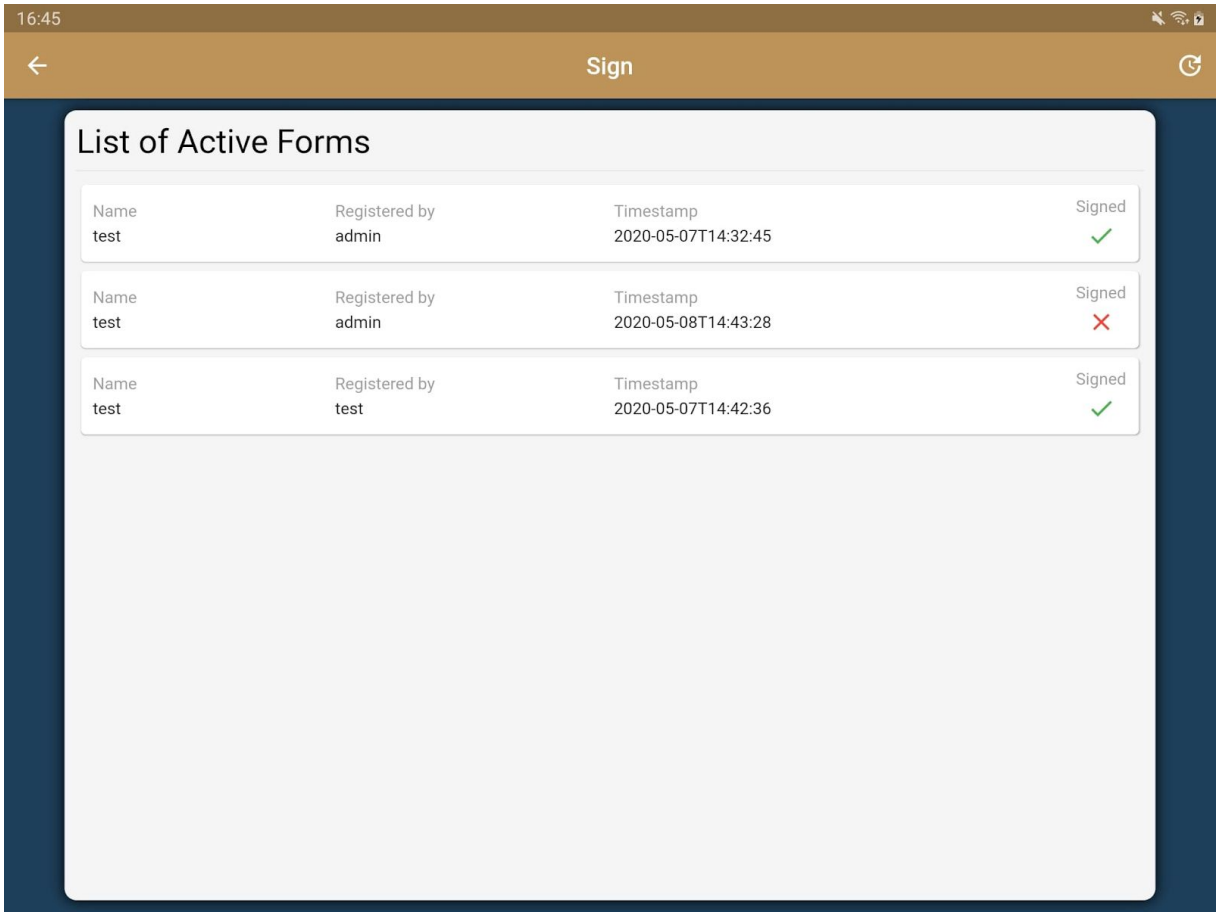
I flere situasjoner trenger arbeidere godkjenning fra overordnede for å kunne utføre en jobb. Godkjenningen består gjerne av en signatur eller lignende, og viser til felles samtykke i et dokument. For å gi brukerne en slik mulighet bestemte utviklingsteam seg for å implementere en signeringsside. Signeringssiden inkluderer en individuell side både for arbeiderne, og de overordnede brukere.

Selv om designet i signeringssiden for en arbeider og en overordnet er relativt like, ser man fremdeles noen forskjeller. For en arbeider, vil listen

av aktive sjekklister kun inneholde personlige registrerte sjekklister med relevant metadata for disse. Sammenligner man det med en overordnet, vil en slik bruker kunne se alle de registrerte skjemaene fra samtlige brukere, samtidig som en har mulighet for å godkjenne disse.

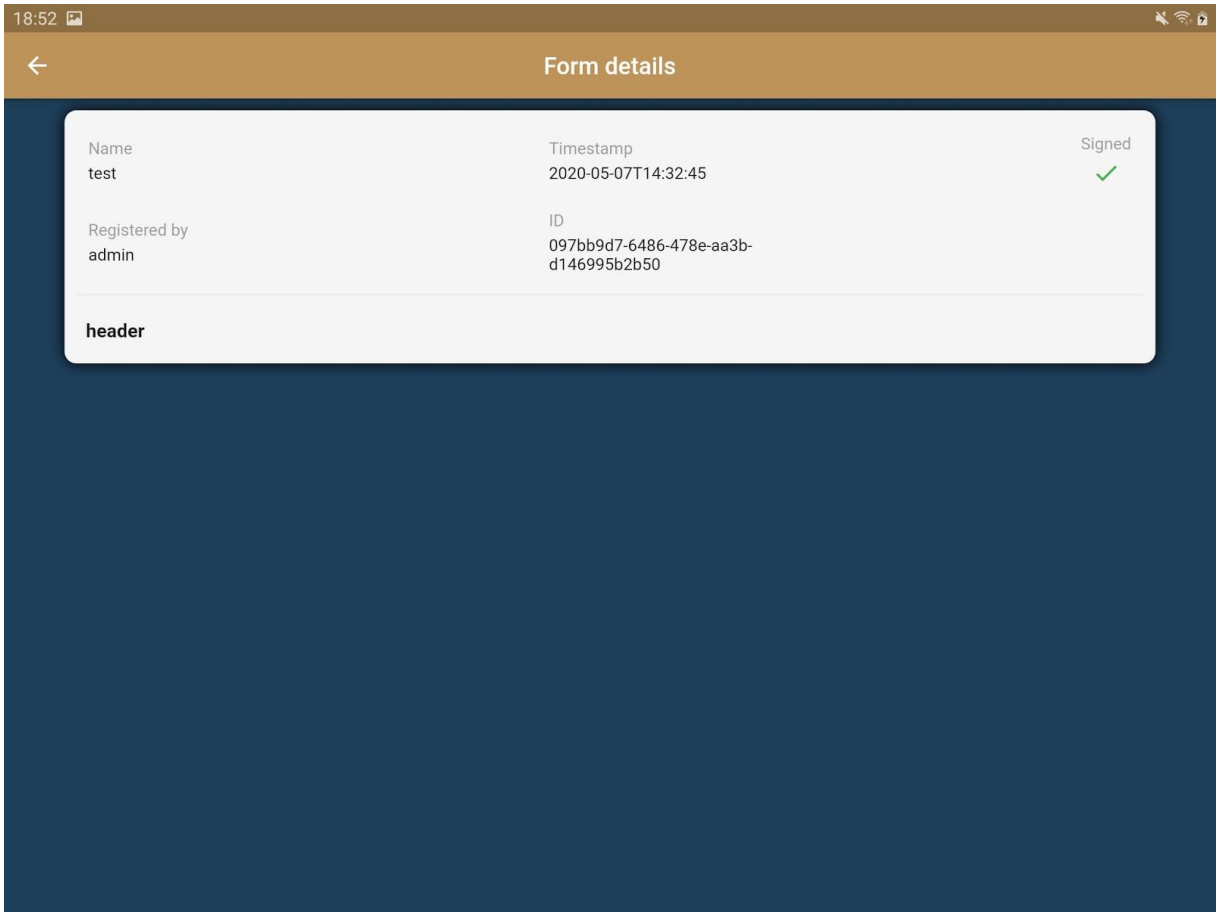
Vedlagt under er ulike skjermdumper tatt fra begge rollene innlogget. En liste hentet fra api'et gir brukerne en god oversikt over aktive sjekklister, med relevant informasjon om hver enkelt sjekkliste. Informasjon som blant annet: navn på sjekkliste, hvilken bruker registrerte den, tid ved registreringen, er sjekklisten signert/evt hvem har signert sjekklisten, og den utfylte sjekklisten.

Denne siden har også en refresh(forfriske) knapp i høyre hjørne av app bar, hvor det sendes en ny forespørsel til api'et som oppdaterer listen.

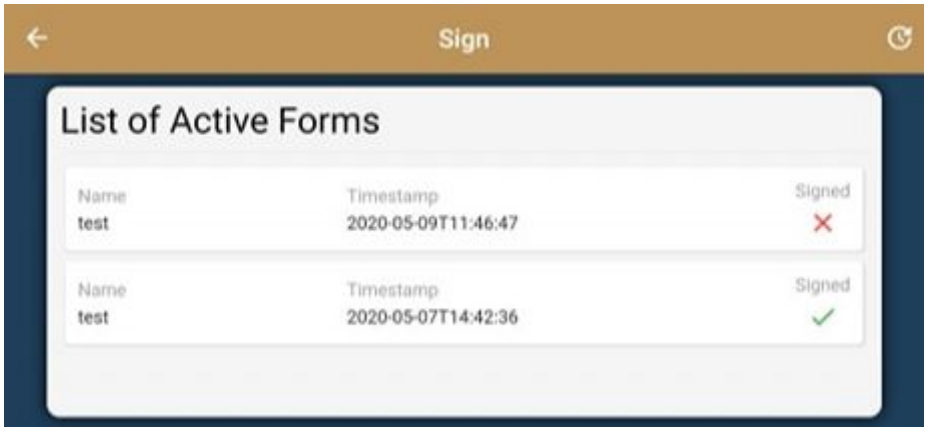


(Figur 46: Signeringsside for overordnede brukere/admin skjermbilde fra vår testenhet i landskapsmodus: Samsung Galaxy Tab S3)

Hvert element i listen av sjekklister er klikkbare for å kunne se detaljer i hver enkelt sjekkliste. Ved et klikk på en sjekkliste tar applikasjonen bruker til en nytt vindu, som inneholder all informasjonen beskrevet over.



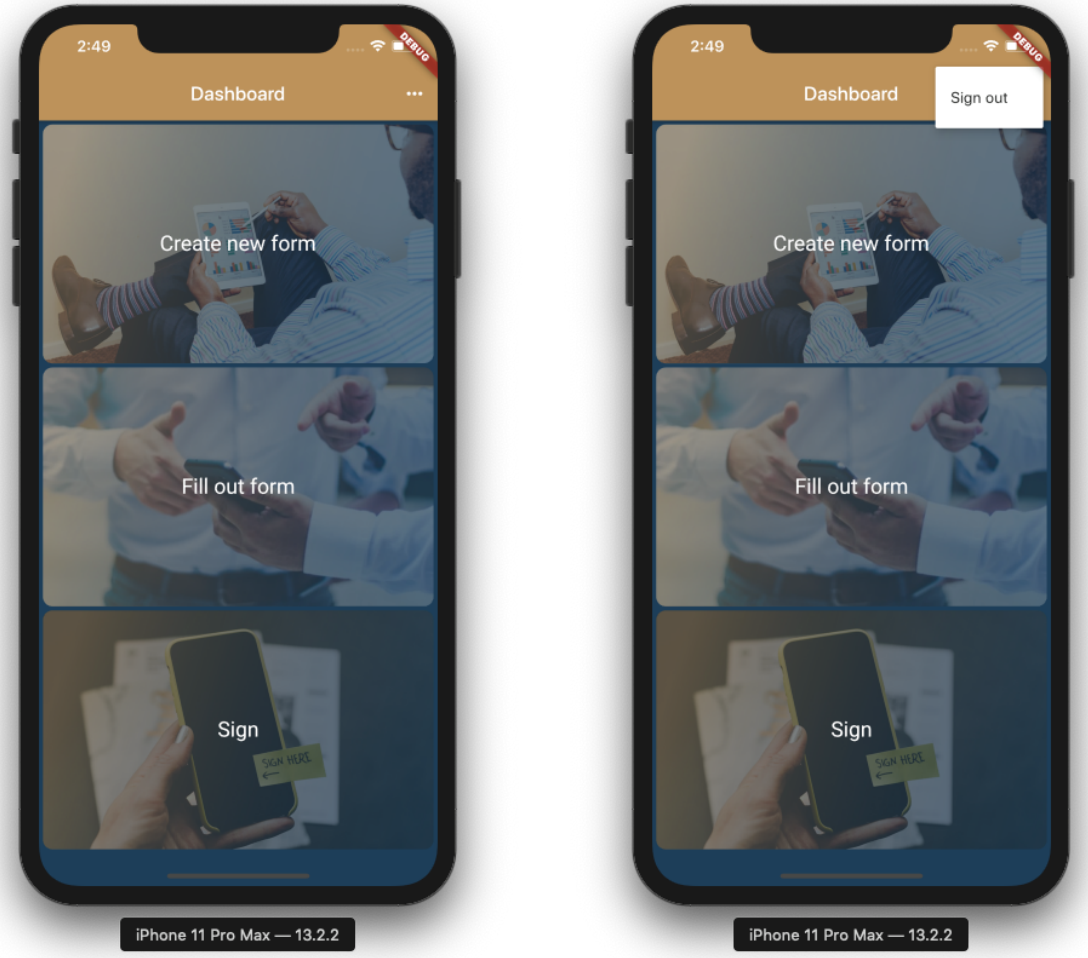
(Figur 47: Detaljer i en spesifikk fylt ut sjekkliste, testenhet i landskapsmodus: Samsung Galaxy Tab S3)



(Figur 48: Signeringssiden for arbeidere, testenhet i landskapsmodus:Huawei P20 Pro)

4.7.8 Utlogging

Utlogging foregår via dashboard-siden, og er rettet mot en nedtrekks-knapp i appbar. Ved en evt nedtrekk vil applikasjonen vise et alternativ, hvor de står "Sign out". En slik hendelse logger ut brukeren, og omdirigerer grafisk til splash-screen. Applikasjonen er da klar for å håndtere en ny brukerinnlogging.



(Figur 49:Utlogging i applikasjonen demonstrert på Iphone 11 Pro Max emulator i portrettmodus)

4.8 Kjente mangler/feil

4.8.1 Noe tilpasning mangler for enheter med lite skjermareal

For fullverdig ferdigstillelse av applikasjonen mangler der noe tilpasninger av den grafiske utformingen for enheter med lite skjermareal. De kom frem i slutten av prosjektet at noe tilpasning må til for å optimalisere applikasjonen for mindre skjermer. Dette ble ikke gjort noe med siden de fleste nå har telefoner med større skjermer, og fordi applikasjonen ble hovedsakelig tilrettelagt for nettbrett.

5 DRØFTING

5.1 Evaluering av oppnådd resultat

5.1.1 Evaluering av VesselDoc

Løsningens oppbygging

Oppbyggingen av løsningen i dette prosjektet besto av en iOS/Android applikasjon hvor et REST api serverte data persisterte i en relasjonsdatabase. Hovedgrunnen for denne type oppbygging var fordi gruppen såg for seg å utvikle et skalerbart produkt, som ikke skulle fungere kun på iOS/Android, men som senere kunne bygges ut til en nettside. Ansatte i rederiene stasjonerte på land kunne da ha fått tilgang til avviksskjemaer og lignende gjennom denne nettsiden, istedenfor å bruke dagens løsning ved sending av skjemaer på mail.

I tidligere studentprosjekter har gruppen brukt denne type oppbygging som er servert gjennom et tilstandsløst api, og mener det er en god løsning siden api'et kan benyttes ved flere plattformløsninger. Man har da mulighet for å benytte samme api for både nettside, og iOS/Android

applikasjon. En annen fordel er at man kan bytte ut 'frontend' rammeverket, uten å måtte gjøre noen vesentlige forandringer i api'et.

Løsningens design

Designet av VesselDoc skulle ta utgangspunkt i et maritimt tema med fargepalett deretter. Utviklingsteam argumenter for at det maritime temaet er opprettholdt gjennom hele applikasjonen, og er fornøyde med hvordan resultatet ble. Det blir også argumentert for at løsningen følger de seks designprinsippene til Don Norman, som vil bedre opplevelsen av løsningen.

Løsningens implementerte funksjoner

Det ble implementert en rekke av funksjoner for brukere av VesselDoc. Blant annet kan en bruker nå opprette et nytt skjema, fylle ut et skjema eller signere et skjema/få et skjema signert. Disse tre funksjonene i lag blir sett på som nøkkelfunksjonaliteten i applikasjonen. Sammen med desse, finnes det også en rekke av generelle funksjoner som må følge en slik løsning. F.eks: Brukersystemet med styring av brukere, sette tilganger, osv. Utviklingsteam argumenterer for at en applikasjon med slik funksjonalitet dekker det meste av kravene en slik løsning burde inneholde.

5.1.2 Evaluering av rammeverkene som ble brukt

Flutter

Bruk av flutter i prosjektet har kun resultert i positive erfaringer for gruppemedlemmene. Ved utvikling av en typisk mobil applikasjon må man ofte vedlikeholde to kodebaser, hvor man i flutter bare trenger å vedlikeholde en. Dette har spart utviklingsteam for mye ekstra arbeid.

Siden rammeverket er bygget for å kunne lett implementere vakkert design og effekter, har det gitt teamet en mulighet for å uttrykke seg selv.

Med å uttrykke seg selv menes det å implementere nøyaktig ønsket design, utført på en enkel måte. Teamet argumenter for at flutter er bedre sammenlignet med tilsvarende rammeverk på bakgrunn av deres predefinerte widgets, og rammeverkets mulighet for å se umiddelbare endringer i en kjørende applikasjon.

Andre fordeler med flutter er med tanke på videreutvikling av rammeverket og dens gode dokumentasjon. Med god dokumentasjon menes det at firmaet bak rammeverket (google) er svært aktive med å få kommunisert ut gode forklaringer av deres tilgjengelige funksjoner; samtidig som de ukentlig annonserer nye funksjoner, for en gitt bruk.

Spring Boot

Ifølge utviklerne bak spring boot er hele hovedmålet med rammeverket å lette utviklingen av produksjonsklare nettapplikasjoner, derav redusere utviklingstiden. Dette mener utviklerne at de har fått erfare ved anvending av spring boot i dette prosjektet. Utviklingsteamet argumenter for flere oppnådde fordeler ved bruk av rammeverket kontra tilsvarende, som:

- Å unngå XML-konfigurasjon
- Å unngå å skrive mange importuttalelser
- Har vært enkelt å komme i gang på grunn av de gitte standardene
- Teamet har mer erfaring med rammeverket
- Siden spring er det mest populære java-rammeverket per dags dato

Spring boot har gitt teamet en mulighet for å effektivt utvikle et tilstandsløst api som serverer data både ut og inn av applikasjonen.

5.1.3 Evaluering av responsivitet

Ved testing på kryss av enheter har teamet fått erfare at applikasjonens sin responsivitet har generelt vært bra. Applikasjonen har fungert sømløst uavhengig av operativsystem, og det med uten noe form for

plattform-spesifikke endringer. I tidligere kapitler kan man se skjermdumper fra hvordan designet forandrer seg i applikasjonen ut i fra hvilken enhet den kjøres på. Applikasjonen justerer seg både ut i fra gitt enhet, og ved hvilken vei man holder enheten. Selv om mye av løsningen ble tilrettelagt for nettbrett, kan man se at den fungerer ypperlig på telefoner også.

5.1.4 Evaluering av skjemahåndtering

Skjemahåndtering ble som tidligere nevnt utført ved å lagre en JSON fil i serveren med id som filnavn knyttet til et element i databasen. Det ble gjort på denne måten fordi at teamet så ingen behov for endring av spesifikke felter i et skjema. Teamet argumenter for at dette var den beste måten å utføre håndteringen på, både med tanke på enkelhet i systemet og for tidsbesparelser. En slik håndtering sparte teamet for en hel del utviklingsarbeid, og fungerte helt fint til ønsket formål.

5.1.5 Evaluering av versjonskontroller som ble brukt

Versjonskontroller er noe som blir sett på som essensielt ved slike gruppeoppgaver. Bruken av ulike versjonskontroller i denne oppgaven har muliggjort det for gruppemedlemmene å jobbe på flere filer samtidig, uten noen konflikter. Som tidligere nevnt har det blitt anvendt ulike versjonskontroller; både for rapporten sin del, og for utviklingsarbeidet i prosjektet.

Git, versjonskontrollen utviklingsteamet brukte for programmeringen, har fungert akkurat som ønsket og tidligere erfart. Selv om det finnes flere tilsvarende versjonskontrollsystemer der ute, er gruppen overbevist om at dette var det beste med tanke på ønsket behov.

Google Disk (Doc), versjonskontrollen som ble brukt for skriving av rapporten har også fungert som ønsket gjennom hele prosjektet. Gruppen har erfart at dette er et av de bedre versjonskontrollsystemene der ute.

5.2 Evaluering av selve prosjektet

5.2.1 Gruppens syn på valgt utviklingsmetodikk

Som tidligere nevnt valgte gruppen å bruke Scrum som sin ønskede utviklingsmetodikk. Metodikken har blitt fulgt gjennom hele prosjektoppgaven, både med tanke på tilnærminger og oppsatte møter. Erfaringen som gruppen sitter igjen med etter Scrum i prosjektet er positiv. Ved å følge slike iterasjoner med planlagte gjøremål, er det mer spesifikt hva som skal på plass, innen hvilket tidsrom. Det er en god metodikk å følge når de kommer til ny programvareutvikling, og kanskje ikke så bra metodikk når det gjelder vedlikehold av eldre programvare. Der kan det gjerne sitte kunder som venter på en reparasjon av noe kritisk, som ikke kan vente til neste sprint for en eventuell fiks.

Selv om erfaringen med scrum var generelt positiv ser man at det derimot kan til dels være overflødig når kommunikasjonen mellom medlemmene er såpass tett. Siden medlemmene i gruppen pratet sammen flere ganger på daglig basis, førte dette til at noen av møtene føltes overflødige. Likevel ser medlemmene hvordan denne praksisen har stor verdi i større prosjekter, med flere medlemmer i teamet.

5.2.2 Veiledningsmøter

Faste veiledningsmøter med en oppsatt veileder fra NTNU var et av kravene i oppgaven. Disse møtene skulle foregå mellom gruppemedlemmene og veileder, hver andre uke, så lenge det lot seg gjøre. Møtene ble holdt fysisk på NTNU Ålesund helt til det oppstod en

utfordring med tanke på den pågående pandemien Covid 19. Mer detaljer rundt dette finnes under kapittel 5.2.3.

Erfaringen gruppen sitter igjen med etter slike veiledningsmøter er positive, da medlemmene har hatt en plattform for å utveksle spørsmål og dele sine utfordringer til en mer kompetent person. Samtidig har møtene fungert som en slags oppkvikker, og som et lite krav-/press for å levere resultater.

5.2.3 Utfordringer

Gjennom dette prosjektet har gruppen måtte tatt stilling til ulike utfordringer. En av hovedutfordringene vi møtte var den pågående pandemien Covid 19. Denne pandemien førte til flere konsekvenser, da de mest relevante for prosjektet var at skolen stengte; noe som betyr at vi ikke fikk oppholde oss på campus i det hele tatt. Siden det tidligere i oppgaven hadde blitt inngått en felles enighet om å arbeide ved skolen sin datalab, førte dette til at gruppen nå måtte forandre sin arbeidsplass og heller jobbe hjemmefra. Selv om det meste av utviklingen gikk fint å utføre over nettverket, ga dette oss noen begrensninger rundt testingen. Det tilstandsløse apiet skulle vertes på en enhet av typen Raspberry Pi, som gruppen bare hadde en av. Måten dette ble løst på ble at all testing av api måtte utføres av et enkelt medlem.

En annen konsekvens vi møtte var med tanke på de fysiske veiledningsmøtene, som nå måtte gjøres digitalt. I tiden pandemien inntraff mistet vi derfor noen av veiledningsmøtene på grunn av tilpasning til den nye arbeidsmåten. Både lærere og studenter brukte tid for tilpasning.

5.2.4 Samarbeid i gruppen

Samarbeidet i gruppen var for det meste bra gjennom prosjektet. Det oppstod ingen konflikter mellom medlemmene. Derimot hadde medlemmene til tider ulike meninger om hvordan ting skulle utføres, men som til slutt ble løst av veileders syn på saken og hverandres argumenter.

En annen situasjon gruppen satt seg i var med tanke på samarbeidet rundt løsningen. Siden løsningen var tredelt med grafisk del, api og database, ble det en konsekvens at arbeidet ble dårlig fordelt. Gruppen hadde en felles enighet tidlig om å ikke dele arbeidet i forskjellige ansvarsområder, men hvor det ble litt sånn allikevel. Grunnen for dette lå antakelig i medlemmenes trygghet og kompetansenivåer for programmeringsspråkene.

5.3 Veien videre

Siden et lokalt firma annonserte at de skulle utvikle et lignende produkt, har gruppemedlemmene bestemt seg for at det ikke ligger noe til grunn for å lansere denne løsningen ved nær framtid.

5.4 Hva er lært

Gjennom dette prosjektet har gruppen lært mye både med tanke på samarbeid med personer, håndtering av nye utfordringer og selve prosessen med en stor prosjektoppgave. I tillegg til dette er medlemmene en erfaring rikere ved å ha skrevet sin første seriøse akademiske rapport av større omfang. Selv om det ble valgt å bruke teknologier som var tidligere kjente, fikk vi utvikle våre egne evner i språkene og samtidig forbedret våre ferdigheter som dataingeniører. Begge medlemmer har fått brukt mye av teorien lært på studiet i en mer praktisk situasjon.

6 KONKLUSJON

Målet med dette prosjektet har vært å utvikle en digital løsning for skjemautfylling innenfor den maritime industrien. Grunnlaget for målet er på bakgrunn av hvordan skjemautfylling foregår i dag; som blir sett på som både ineffektiv, utrygg og lite miljøbevisst.

Ved prosjektets oppstart ble det stilt spørsmål som "hvordan utvikle en digital løsning for skjemautfylling om bord i skip, som ikke bare er raskere å gjennomføre, men sikrere å bruke, samtidig som det er kompatibelt med det fleste enheter som folk ivaretar?". Noe som også er problemstillingen til prosjektet. Rapporten beskriver metoder og resultater som vår besvarelse på gitt problemstilling.

Resultatene i rapporten viser at prosjektmedlemmene har laget en løsning som:

- består av en applikasjon som er kompatibelt med de fleste mobile enheter (android og iOS) og en tilstandsløs serverløsning som er knyttet til en relasjonsdatabase
- er responsivt og brukervennlig brukergrensesnitt
- har et brukersystem der brukere har tilgang og rettigheter basert på hvilken rolle de har
- har funksjonalitet til å lage egendefinerte skjemaer (digitalt)
- kan koble til en server automatisk på et lokalt nettverk om den finner en
- kan fylle ut et skjema (digitalt)
- kan signere/få signert et skjema (digitalt)
- skjemaene blir lagret lokalt slik at de framtidig kan bli tatt backup av når skipene har internett

- sparer miljøet ved å muliggjøre skjemautfylling uten å måtte bruke papir
- sikkerhetsmessig er skjemaene kun tilgjengelig for brukerne som skal ha tilgang til de
- motiverer arbeidere med å bli flinkere i å fylle ut skjemaer på en daglig basis, som igjen kan bidra med å forebygge og forhindre ulykker/mangler

Siden det er utallige måter å løse en slik oppgave på, kan ikke gruppen si med sikkerhet at valgene som er tatt er de rette valgene. Derimot står gruppen bak det valgene som er gjort gjennom oppgaven, både med tanke på anvendelse av kjente teknologier og tilgjengelige ressurser. Med tilgjengelige ressurser menes det at gruppen besto av to medlemmer, og at tiden var begrenset.

Som de blir fremstilt i både drøfting og evaluering, er prosjektmedlemmene fornøyde med løsningen som er blitt utviklet. Løsningen er klar for å bli tatt i bruk av skip verden over, og kan fungere som en erstatning av dagens skjemaer i deres SMS systemer.

Medlemmene føler også at de har fått vist mye av kunnskapen som har blitt tilegnet gjennom studiet i en slik oppgave, samtidig som det har vært lærerikt. Det viser seg gang på gang at det å ha praksis oppgaver hvor man takler reelle problemstillinger hjelper oss å vokse. Ikke bare som individer, men også som ingeniører.

Gruppen konkluderer herved med at den fremstilte løsningen på problemstillingen i rapporten innfrir dens gitte krav, og at

problemstillingen er besvart ved våres syn på beste løsning, med ressursene vi har hatt tilgjengelig.

7 REFERANSER

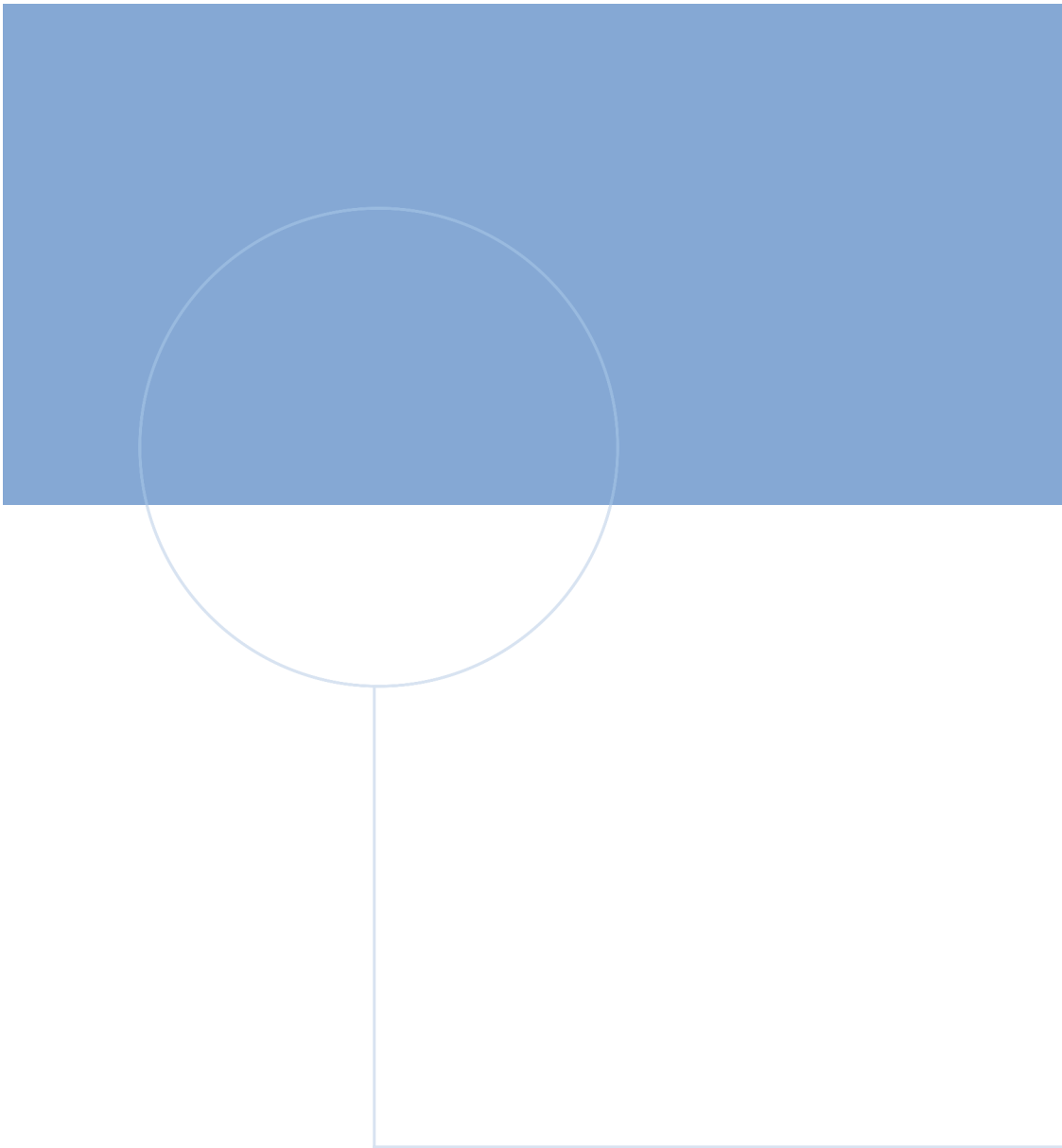
[1] https://lovdata.no/dokument/SF/forskrift/2014-09-05-1191/KAPITTEL_1#KAPITTEL_1, (Lest: 20.02.2020, publisert: 05.09.2014)
[2] https://agilemanifesto.org/iso/no/manifesto.html, , (Lest: 20.02.2020, publisert: 2001)
[3] https://en.wikipedia.org/wiki/Agile_software_development (Lest: 21.02.2020)
[4] https://staragile.com/scrum-meetings (Hentet: 21.02.2020)
[5] Schwaber, Ken. Agile Project Management with Scrum. Microsoft Press. ISBN 978-0-7356-1993-7 (Lest: 21.02.2020, Publisert: 01.02.2004)
[6] Sutherland, Jeff. "Agile Development: Lessons learned from the first Scrum". (lest: 23.02.2020 , publisert: Oktober 2004)
[7] Ken Schwaber; Jeff Sutherland. "The Scrum Guide" Scrum.org. (Lest: 23.02.2020, publisert/endret: 2017)
[8] "What is a Daily Scrum?". Scrum.org. (Lest: 23.02.2020, publisert/endret: 2017)
[9] https://www.scrumguides.org/ (Lest: 23.02.2020, publisert: uvisst)
[10] https://www.scrumguides.org/scrum-guide.html (Lest: 23.02.2020, publisert: 2017)
[11] Drongelen, Mike van; Dennis, Adam; Garabedian, Richard; Gonzalez, Alberto; Krishnaswamy, Aravind. Lean Mobile App Development. Birmingham, UK: Packt Publishing Ltd. p. 43. ISBN 9781786467041. (Lest: 01.03.2020, publisert: 2017)
[12] Morris, David (2017). Scrum: an ideal framework for agile projects. In Easy Steps. pp. 178–179. ISBN 9781840787313. OCLC 951453155; Ramakrishnan R, Gehrke J. Database management systems. McGraw Hill. (Lest: 01.03.2020, publisert: 2000)
[13] "RFC 6762 - Multicast DNS - IETF Tools." https://tools.ietf.org/html/rfc6762. (lest: 17.03.2020, publisert: Februar 2013)
[14] https://en.wikipedia.org/wiki/Database (lest: 12.05.2020, publisert/endret: usikkert)
[15] https://en.wikipedia.org/wiki/Object-oriented_programming(lest: 15.03.2020, publisert/endret: usikkert)
[16] https://snl.no/objekt_-_IT (lest: 15.03.2020, publisert/endret: 22.02.2019)
[17] https://en.wikipedia.org/wiki/Java_(software_platform) (lest: 16.03.2020, publisert/endret: usikkert)

[18] https://www.oracle.com/java/ (lest: 16.03.2020, publisert/endret: usikkert)
[19] https://flutter.dev/ (lest: 15.03.2020, publisert/endret: usikkert)
[20] https://dart.dev/ (lest: 15.03.2020, publisert/endret: usikkert)
[21] https://docs.oracle.com/cd/E24329_01/web.1211/e24983/overview.htm#RESTF105 (lest: 16.03.2020, publisert/endret: usikkert)
[22] https://spring.io/projects/spring-boot (lest:03.04.2020, publisert/endret: usikkert)
[23] https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control (lest: 03.04.2020, publisert/endret: usikkert)
[24] https://www.atlassian.com/git/tutorials/what-is-version-control (lest: 03.04.2020, publisert/endret: usikkert)
[25] https://confluence.atlassian.com/get-started-with-bitbucket/types-of-version-control-856845192.html (lest: 12.05.2020, publisert/endret: usikkert)
[26] https://git-scm.com/ (lest: 12.05.2020, publisert/endret: usikkert)
[27] Paul, Eliza . "What is Digital Signature – How it works, Benefits, Objectives, Concept". (lest: 03.04.2020, publisert/endret: 12 September 2017)
[28] https://acrobat.adobe.com/no/no/sign/capabilities/digital-signatures-faq.html (lest: 08.04.2020, publisert/endret: usikkert)
[29] https://www.json.org/json-en.html (lest: 08.04.2020, publisert/endret: 1999)
[30] https://www.tiobe.com/tiobe-index/ (lest: 10.04.2020, publisert/endret: 2020)
[31] https://insights.stackoverflow.com/survey/2019#development-environments-and-tools (lest: 15.04.2020, publisert/endret: usikkert)
[32] https://www.mysql.com/products/workbench/ (lest: 15.04.2020, publisert/endret: usikkert)
[33] https://www.google.com/intl/no_ALL/drive/ (lest: 15.04.2020, publisert/endret: usikkert)
[34] "org.springframework.boot » spring-boot ... - Maven Repository." https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-web. (Lest: 9 Mai. 2020, publisert: usikkert)
[35] https://jwt.io/introduction/ (lest: 12.05.2020, publisert/endret: usikkert)
[36] https://en.wikipedia.org/wiki/Hibernate_(framework).
[37] "nss-resolve - Freedesktop.org." https://www.freedesktop.org/software/systemd/man/nss-resolve.html. (lest: 10.05.2020, publisert/endret: usikkert)
[38] "What Is A DNS PTR Record? | Cloudflare." https://www.cloudflare.com/learning/dns/dns-records/dns-ptr-record/. (lest: 11.05.2020, publisert/endret: usikkert)

VEDLEGG

Vedlegg 1	Kildekode
Vedlegg 2	Forprosjektrapport
Vedlegg 3	Klassediagram Rest API
Vedlegg 4	Klassediagram Frontend (Flutter)
Vedlegg 5	Møtereferater | Veileder og gruppemedlemmer
Vedlegg 6	Eksempel av typisk skjema | Work permit
Vedlegg 7	Eksempel av typisk skjema | Tool Box Talk
Vedlegg 8	Logg over arbeidet | dag til dag

FIGURLISTE



NTNU

Kunnskap for en bedre verden