BACHELOROPPGAVE

Smart Ocean Data Visualisering Dashbord

Smart Ocean Data Visualisation Dashboard

Pål Rebnord Brügger og Eirik Gismervik Skare

DAT191
Fakultet for ingeniør- og naturvitenskap
Institutt for datateknologi, elektroteknologi og realfag
Informasjonsteknologi, Dataingeniør
Veileder: Lars Michael Kristensen
Innleveringsdato: 22.05.2023

Jeg bekrefter at arbeidet er selvstendig utarbeidet, og at referanser/kildehenvisninger til alle
kilder som er brukt i arbeidet er oppgitt, *jf. Forskrift om studium og eksamen ved Høgskulen på Vestlandet, § 12-1.*

Høgskulen på Vestlandet

Fakultet for ingeniør- og naturvitskap
Institutt for datateknologi, elektroteknologi og realfag

TITTELSIDE FOR HOVEDPROSJEKT

Rapportens tittel: *Smart Ocean Data Visualisering Dashbord*	*Dato:* *22.05.2023*
Forfatter(e): *Pål Rebnord Brügger & Eirik Gismervik Skare*	*Antall sider u/vedlegg: 36* *Antall sider vedlegg: 4*
Studieretning: *Informasjonsteknologi & Dataingeniør*	*Antall disketter/CD-er: Ingen*
Kontaktperson ved studieretning: *Volker Stolz & Lars-Petter Helland*	*Gradering: Ingen*
Merknader:	

Oppdragsgiver : *SFI Smart Ocean*	*Oppdragsgivers referanse:*
Oppdragsgiver kontaktperson: *Lars Michael Kristensen*	*Telefon:* *+4793866491*

Sammendrag:
Prosjektet hadde som mål å lage et visualiserende dashbord i form av en webapplikasjon, som viser data sendt fra undervanns sensorrigger.

Stikkord:

Webapplikasjon	*JavaScript*	*WebSockets*
Java	*NoSQL*	*HiveMQ*

Høgskulen på Vestlandet, Fakultet for ingeniør- og naturvitskap

Postadresse: Postboks 7030, 5020 BERGEN Besøksadresse: Inndalsveien 28, Bergen

Tlf. 55 58 75 00 Fax 55 58 77 90 E-post: post@hvl.no Hjemmeside: http://www.hvl.no

FORORD

Denne rapporten dokumenterer bacheloroppgaven Smart Ocean Data Visualisering Dashbord. Prosjektet er gjennomført av Pål Rebnord Brügger og Eirik Gismervik Skare, ved Høgskulen på Vestlandet våren 2023.

Vi ønsker å takke SFI Smart Ocean for å ha gitt oss dette prosjektet og Lars Michael Kristensen for veiledning underveis i prosjektet.

INNHOLDSFORTEGNELSE

1 INNLEDNING ... 1
1.1 Kontekst ... 1
1.2 Motivasjon ... 1
1.3 Prosjekteier ... 2
1.4 Problembeskrivelse og mål ... 2
1.5 Oppbygging av rapporten ... 3
2 PROSJEKTBESKRIVELSE ... 4
2.1 Praktisk bakgrunn ... 4
2.1.1 Tidligere arbeid ... 4
2.1.2 Initielle krav ... 4
2.1.3 Initiell løsnings-idé ... 5
2.2 Avgrensninger ... 6
2.3 Ressurser ... 6
2.4 Litteratur om problemstillingen ... 6
3 DESIGN AV WEBAPPLIKASJON ... 8
3.1 Forslag til løsning ... 8
3.1.1 Alternativ løsning 1 ... 8
3.1.2 Alternativ løsning 2 ... 10
3.1.3 Diskusjon av alternativene ... 11
3.2 Valgt løsning ... 11
3.3 Valg av verktøy ... 11
3.4 Prosjektmetodikk ... 12
3.4.1 Utviklingsmetodikk ... 12
3.4.2 Prosjektplan ... 12
3.4.3 Risikovurdering ... 12
3.5 Evalueringsplan ... 13
4 DETALJERT LØSNING ... 14
4.1 Simulator ... 14
4.2 Filtrere ut sensordata fra XML-filene ... 16
4.3 Opprette database ... 18
4.4 Utvikle hovedsiden til dashbordet ... 19
4.5 Oppdatere sensordata på dashbordet ... 20
5 RESULTATER ... 23

5.1 Evalueringsmetode 23
5.2 Evalueringsresultat 23
5.3 Prosjektresultat 24
5.4 Prosjektgjennomføring 24
6 DISKUSJON 25
6.1 Fremgangsmåte 25
6.2 Konsekvens av fremgangsmåte 26
7 KONKLUSJON OG VIDERE ARBEID 28
7.1 Konklusjon 28
7.2 Videre arbeid 29
8 REFERANSER 30
9 VEDLEGG 33
9.1 Fremdriftsplan 33
9.2 Risikoanalyse 35

1 INNLEDNING

I dette kapittelet vil det bli gitt en oversikt over prosjektets kontekst, motivasjon, prosjekteier og mål.

1.1 Kontekst

SFI Smart Ocean (Smart Ocean, 2023) er et forskningssenter som har en visjon om å utvikle og implementere ny teknologi som vil forbedre bærekraften til havbaserte næringer, som offshore olje og gass, skipsfart, fiskeri og annet havbruk. Prosjektet er finansiert av Norges Forskningsråd og forskningen utføres av et konsortium av forskningspartnere: HVL (HVL, 2023), NORCE (NORCE, 2023), UiB (UiB, 2023), Havforskningsinstituttet (Havforskningsinstituttet, 2023) og Forsvarets forskningsinstitutt (Forsvaret, 2023). I tillegg er det flere industripartnere som W Sense (W Sense, 2023), Aanderaa (Aanderaa, 2023), AkerBP (AkerBP, 2023), Bouvet (Bouvet, 2023), Tampnet (Tampnet, 2023), TSC Subsea (TSC Subsea, 2023), Reach Subsea (Reach Subsea, 2023), Metas (Metas, 2023) og Kongsberg (Kongsberg, 2023).

SFI Smart Ocean startet i desember 2020 med et mål om å utvikle et undervanns sensor- og kommunikasjonssystem som skal bidra til å sikre bærekraftig drift av havindustri og faktabasert havforvaltning. Så langt har forskningssenteret utplassert to testrigger i Austevoll som har begynt å sende sensordata. Dataene sendes i form av kompliserte og proprietære XML-filer, og det er avdekket et ønske om å visualisere disse sensordataene på en oversiktlig måte.

1.2 Motivasjon

Motivasjonen for dette bachelorprosjektet var å utvikle en webapplikasjon som gir tilgang til viktig havmiljø data i sanntid. En webapplikasjon vil gi et bedre brukergrensesnitt for brukere å se og analysere data samlet inn av sensorer utplassert på ulike lokasjoner, og gi innsikt i havforhold, forurensingsnivåer, værmønstre og andre nøkkelfaktorer.

Ved å gjøre sanntidsdata om havforhold enkelt tilgjengelig, kan webapplikasjonen brukes som grunnlag i forbindelse med beslutningstaking i havindustrien og forskning. For eksempel kan offentlige etater bruke dataene til å spore helsen til havets økosystem og overvåke overholdelse av regelverk.

Webapplikasjonen kan også være et nyttig verktøy for forskere som studerer havmiljøet, ved å la dem få tilgang til og analysere store mengder data i sanntid. Dette kan forenkle samarbeid og mulig bidra til å akselerere vitenskapelige oppdagelser. Dette kan videre gi dypere innsikt i hvordan havets økosystem fungerer og bidra til å finne løsninger på

miljøutfordringer, samt kalibrere eksisterende modeller av havet. Oppsummert ville webapplikasjonen ha et bredt spekter av bruksområder.

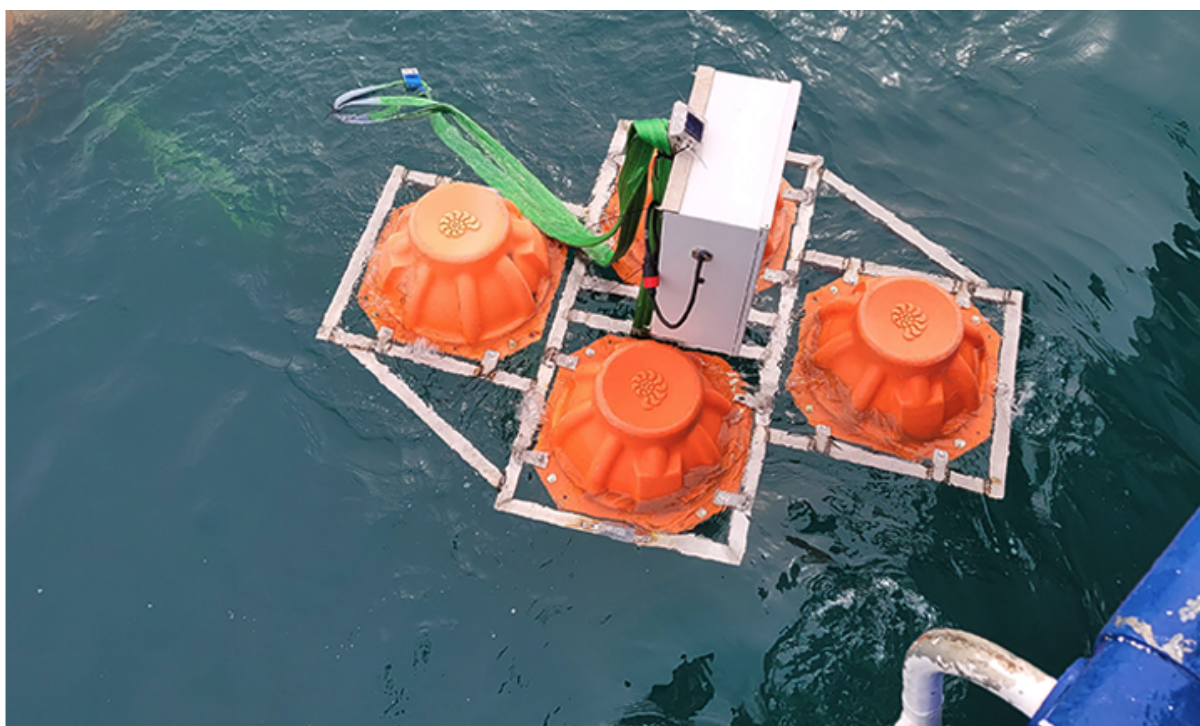
1.3 Prosjekteier

Dette prosjektet er en del av SFI Smart Ocean og er et offentlig prosjekt. Prosjektet i seg selv er ikke ment til å generere inntekter, men å være et tilbud til SFI Smart Ocean, deres partnere og offentligheten. Derfor er ikke eierskap et viktig aspekt av dette prosjektet.

1.4 Problembeskrivelse og mål

Havnæringer og havforskning står sterkt i Norge i dag, men har fremdeles stort potensial. For å dra mest nytte av havet og samtidig legge til rette for en bærekraftig utvikling av fremtidig bruk, behøves det mye data. I dag gjøres det en del målinger i havet, hovedsakelig gjennom skip, drivbøyer og fjernstyrte farkoster. Til tross for dette, er det mangel på sanntids- og langtidsmålinger.

SFI Smart Ocean har som mål å skape et trådløst undervanns observasjonssystem med et tilhørende skysystem. Dette observasjonssystemet vil gjøre det mulig å overvåke havet på en mer systematisk og direkte måte. Som nevnt tidligere, ble det utplassert to undervannsrigger med en mengde ulike sensorer i Austevoll. Figur 1.1 viser en slik undervannsrigg. Denne riggen sender sensordata, gjennom et undervannsnettverk, til en data-bøye og deretter videre til et HiveMQ-cluster (HiveMQ, 2023).



Figur 1.1 - Undervannsrigg

Et HiveMQ-cluster er en samling av HiveMQ-meglere som jobber sammen som en enkel megler. En megler er en entitet som mottar meldinger fra en kilde og videresender meldingen til alle som abonnerer på det emne som meldingen ble sendt til. Smart Ocean prosjektet har sett et behov for en webapplikasjon som kan vise disse sensordataene på en oversiktlig måte. Denne problemstillingen dannet et grunnlag for gruppens bachelorprosjekt.

Prosjektet hadde som mål å utvikle en webapplikasjon som kontinuerlig viser sanntids-data fra valgte sensorer som temperatur, salinitet og konduktivitet. Applikasjonen skulle kunne visualisere ønsket data fra valgte undervannsrigger med jevne intervaller, og det skulle være mulig å legge til og administrere sensorer og datakilder.

Delmålet til prosjektet var å ha en fungerende webapplikasjon som kunne vise sensordata publisert av en simulator. Etter hvert når de grunnleggende funksjonene var implementert, var det mer passende å tenke på sluttbrukers behov og utvikle applikasjonen deretter.

Oppsummert omhandlet forskningsspørsmålene om et webapplikasjons rammeverk er egnet for å lage et visualiserende dashbord og hvordan designe for forskjellige datakilder og sensorer. Videre hvorvidt prosjektet klarte sanntids og feiltolerant databehandling gjennom HiveMQ's meldingstjeneste. Oppsummert er forskningsspørsmålene:

FS1: Hvilke webrammeverk er egnet for et dashbord som visualiserer sensordata?

FS2: Hvordan designe en webapplikasjon for flere forskjellige datakilder med ulike sensorer?

FS3: Hvordan kan webapplikasjonen vise sensordata i sanntid via et meldingssystem og være feiltolerant ved mangel eller feil på sensordata eller meldingssystem?

1.5 Oppbygging av rapporten

Oppbyggingen av rapporten er som følger:

1. Innledning	
2. Prosjektbeskrivelse	En mer detaljert og utdypende beskrivelse av prosjektet.
3. Design av webapplikasjon	Webapplikasjonens design, med dets planlegging og løsninger.
4. Detaljert Løsning	Detaljert beskrivelse av løsningen.
5. Resultater	Resultater fra prosjektet og evaluering.
6. Diskusjon	Diskusjon av fremgangsmåter og resultater.
7. Konklusjon og videre arbeid	Konkludering på forskningsspørsmålene og videre arbeid.
8. Referanser	
9. Vedlegg	Risikoanalyse og fremdriftsplan.

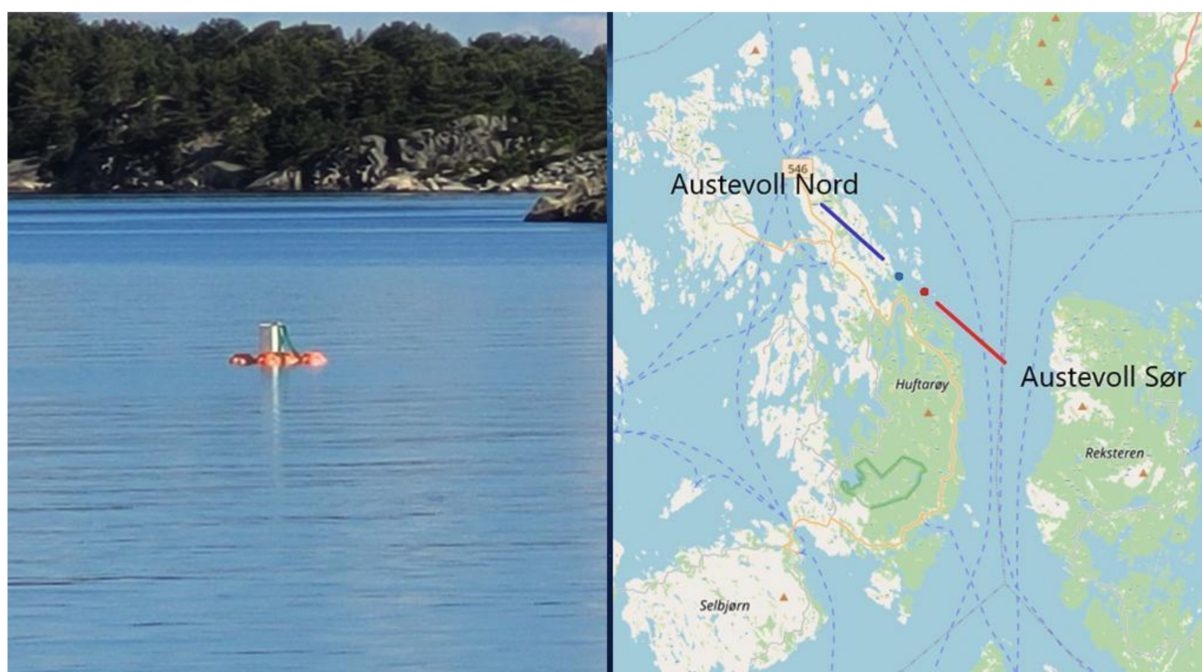
2 PROSJEKTBESKRIVELSE

Dette kapittelet beskriver bakgrunn for prosjektet og tidligere arbeid som er gjort. Videre presenteres teori, krav og ideer for problemstillingen.

2.1 Praktisk bakgrunn

2.1.1 Tidligere arbeid

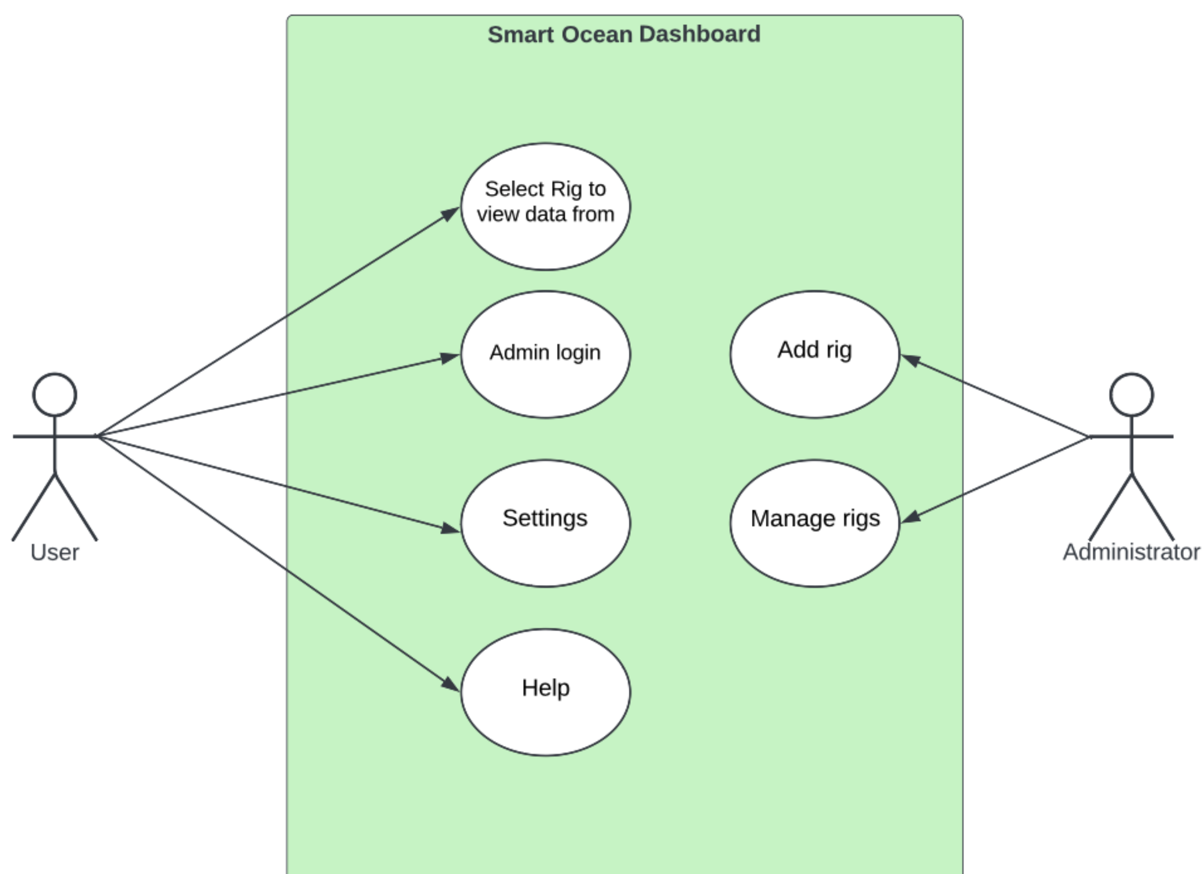
SFI Smart Ocean plasserte ut to testrigger i Austevoll i desember 2020. Prosjektet har fått tilgang til data som er samlet inn fra disse testriggene. Dette er nødvendig for å teste løsningen og for å bekrefte riktig håndtering av format på innkommende sensordata. Figur 2.1 viser testrigger utplassert i Austevoll.



Figur 2.1 - Undervannsrigger i Austevoll

2.1.2 Initielle krav

Løsningen sine initielle krav var en webapplikasjon som kan vise sanntidsdata hentet fra HiveMQ's meldingstjeneste. Løsningen skal kunne håndtere forskjellige datakilder og sensorer. Videre skal den være feiltolerant ved å sikre at riktige data visualiseres. Det ble utarbeidet et brukstilfellediagram for å illustrere brukerens interaksjon med webapplikasjonen, se
Figur 2.2.



Figur 2.2 - Brukstilfellediagram

2.1.3 Initiell løsnings-idé

Den initielle løsnings-idéen innebar en webapplikasjon som kunne abonnere på et emne i HiveMQ's meldingstjeneste, hvor data fra ulike sensorer blir sendt. For å gjøre dette ville webapplikasjonen koble seg opp til en HiveMQ-megler hvor den ville få tilgang til sensordata. Deretter skulle sensordata presenteres i en webapplikasjon i dashbord-stil. Nettsiden kunne bli bygget med et JavaScript-rammeverk slik som React (React, 2023) eller Angular (Angular, 2023). På nettsiden kunne det være et interaktivt kart hvor en kunne zoome inn på de forskjellige sensorene som er markert på kartet, og velge en for å se utvidet informasjon. Gjerne ved å integrere OpenStreetMap eller tilsvarende. I tillegg var det tenkt å bruke REST API-er, WebSockets og en NoSQL database i løsningen.

2.2 Avgrensninger

Løsningens avgrensninger var blant annet lite variasjon i testdata og et begrenset antall undervannsrigger. Dette ville begrense muligheten til å teste med mange ulike formater og inputs. Det kunne gjøre at løsningen ikke endte opp like robust for nye datakilder. Videre var det mulig at tid ble en avgrensing, grunnet prosjektdeltakerne sin begrensede erfaring ville ting ta tid og det kunne være vanskelig å beregne hvor lang tid de forskjellige implementasjonene ville ta.

2.3 Ressurser

For å gjennomføre prosjektet trengte gruppen flere ting. Gruppen trengte en brukerkonto hos HiveMQ messaging service, for å kunne bruke den til å lage en simulator til å teste applikasjonen. Simulatoren skulle i første del simulere hvordan undervannsrigger sendte XML-filer til en HiveMQ-megler og i andre del fikk filene tilsendt fra megleren til webapplikasjonen, hvor sensordataen ble hentet ut fra filene og vist frem. Dette ville hjelpe med å teste applikasjonen når det ble nødvendig og med selvvalgte tidsintervaller. I tillegg var det vikitg med en god mengde eksempeldata til testing. Videre var det også behov for tilgang til en database for lagring av data, initialt tenkt NoSQL. NoSQL databaser gir en fordel når det skal lagres store mengder med ustrukturerte data, uten å være bundet til noen modeller. Gruppen ville videre behøve veiledning fra veileder og eventuelt andre deltakere i Smart Ocean.

2.4 Litteratur om problemstillingen

UiB skrev i 2020, en artikkel om prosjektet til SFI Smart Ocean (UiB, 2020). Artikkelen forteller at OECD (The Organization for Economic Cooperation and Development) har indikert at havindustriene har potensial til å doble sin økonomiske vekst på ti år. Samtidig blir presset på arealene i havet og fjordene stadig større. SFI Smart Ocean skal skape et observasjonssystem for havet som vil bidra til at norske havnæringer skal dra nytte av dette økte markedet, og bidra til å løse de marine utfordringene Norge og verdenssamfunnet står overfor, på en bærekraftig måte.

Videre beskrives planen om å skape et trådløst nettverk av stasjonære sensorer for marine målinger og databehandling. Nettverket må være fleksibelt, robust, sikkert og energi- og kostnadseffektivt. Nettverket med trådløst Internett i sjøen vil gjøre det mulig å gjøre målinger der kablede nettverk er uaktuelle, av økonomiske og praktiske årsaker.

Hovedmålet til SFI Smart Ocean er å lage et "Internet of things" (IoT) under vann, der flere enheter i et undervannsnettverk kan kommunisere med hverandre og kan sende informasjon og målinger til land.

Denne artikkelen, i tillegg til en god del annet lesestoff, har hjulpet gruppen å forstå formålet til prosjektet og hvilke krav som vil bli stilt til vår løsning, om hvilke funksjonelle og ikke-funksjonelle egenskaper som vil være nødvendige. Mer litteratur om problemstillingen ligger i referanselisten.

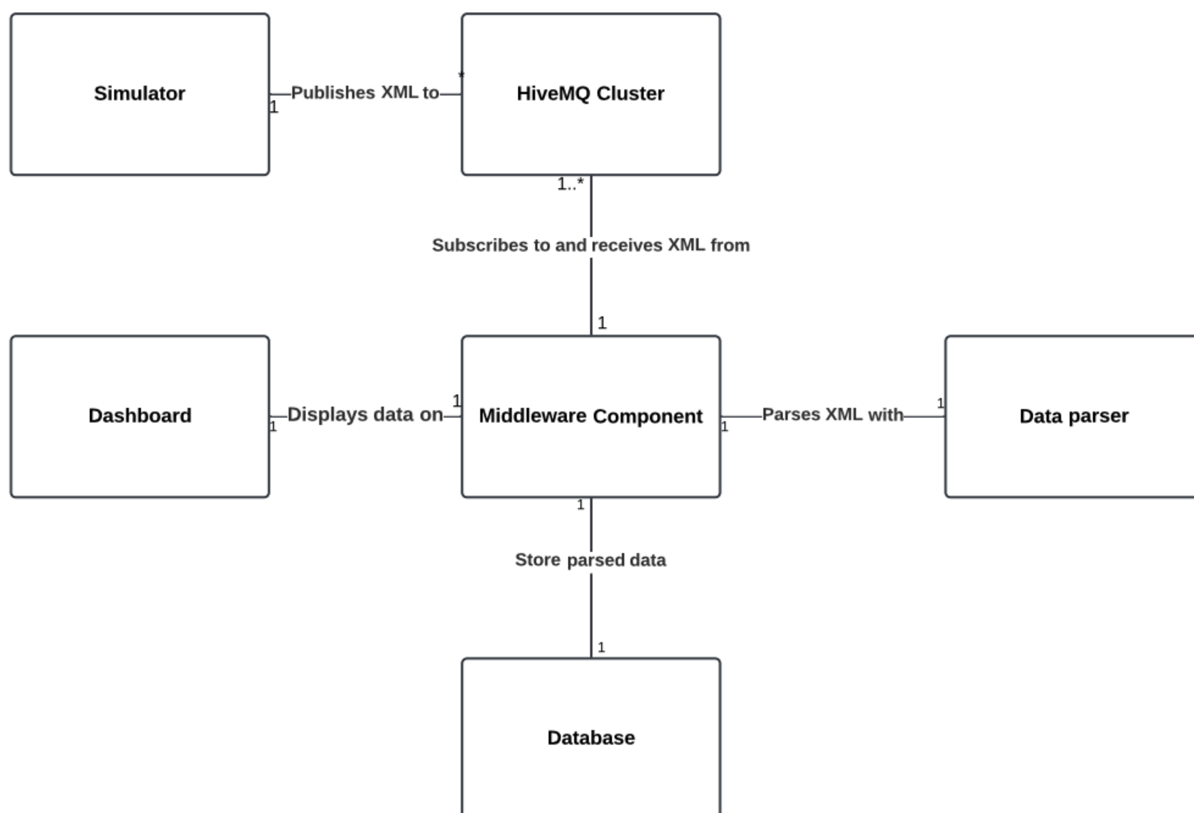
3 DESIGN AV WEBAPPLIKASJON

Dette kapittelet beskriver de ulike designtilnærmingene som ble evaluert av gruppen. Videre vil det bli gitt detaljer om hvilke verktøy som ble benyttet.

3.1 Forslag til løsning

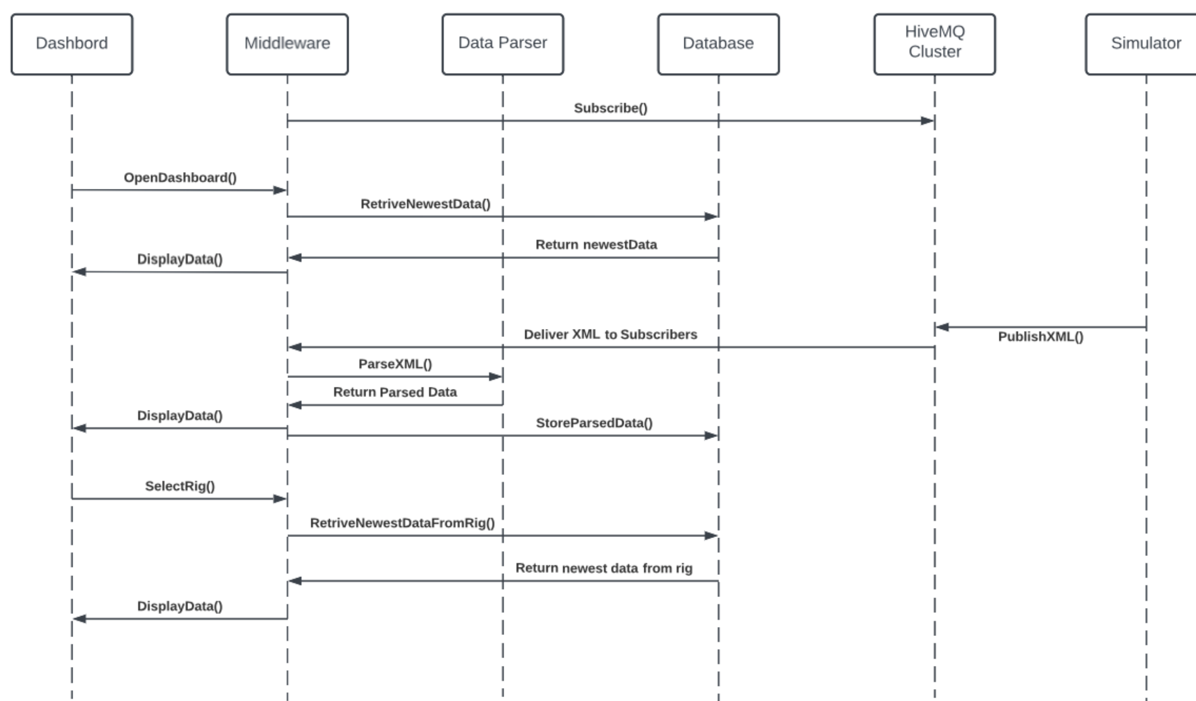
3.1.1 Alternativ løsning 1

Det første alternativet til en løsning involverte flere komponenter. En simulator som publiserer eksempel data til et emne på et HiveMQ-cluster. Dette ville være et cluster som gruppen opprettet for testing. Videre ville det være en komponent som kobler seg opp til samme cluster og abonnerer på samme emne og mottok sensordata. Simulatoren skulle bli brukt hovedsakelig til testing og demonstrasjon. Så skulle en komponent filtrere ut sensordata fra XML-formatet som sensordataen ble sendt med. Deretter ville det bli opprettet Java-objekter som inneholdt sensordataen og sendt til middleware-komponenten som sendte det videre til en NoSQL database og dashbordet. Ut ifra dette ble programvarearkitektur diagrammet i Figur 3.1 utarbeidet. Her gis det en oversikt over de forskjellige komponentene som var planlagt og forholdet mellom dem.



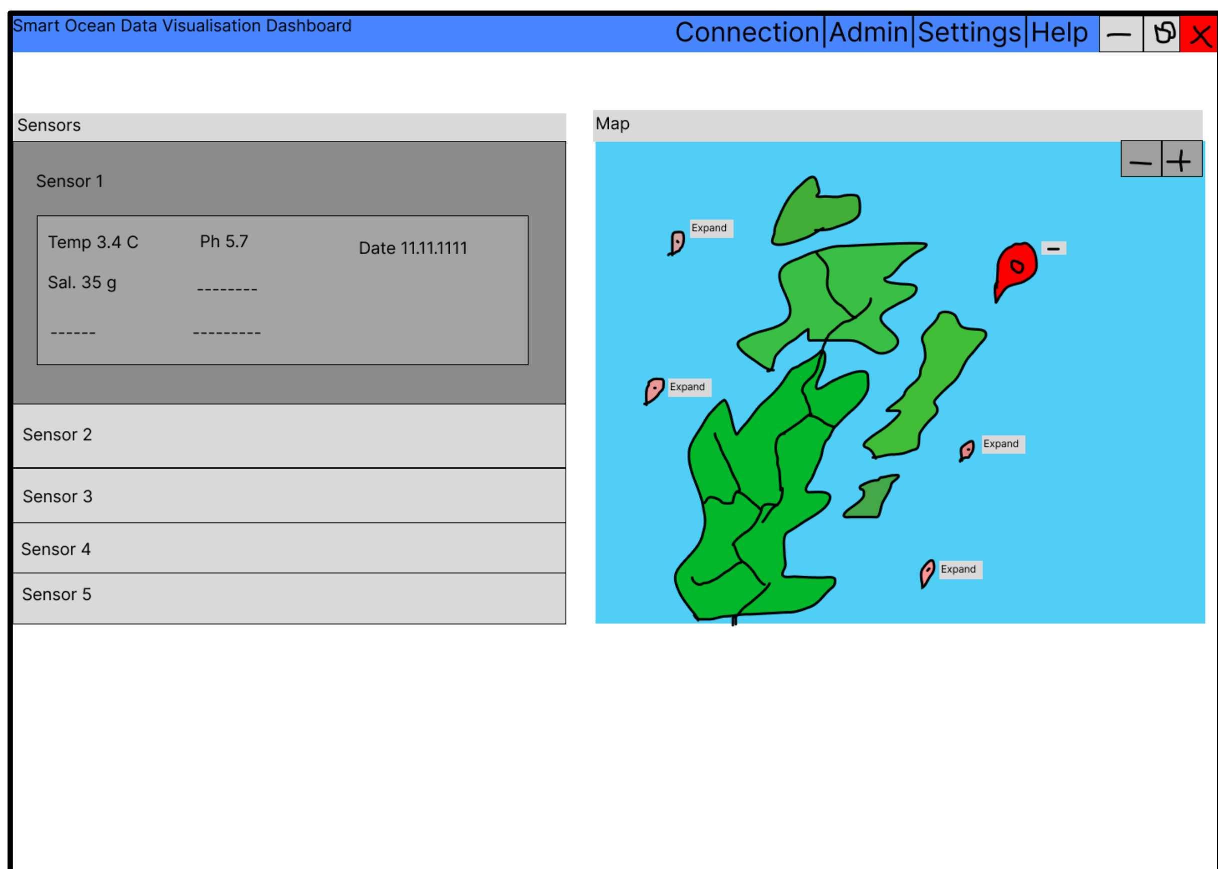
Figur 3.1 - Programvarearkitektur diagram

Sekvensdiagrammet i Figur 3.2 viser programflyten og interaksjonen mellom komponentene.



Figur 3.2 - Sekvensdiagram

Dashbordet vil ha en tabell på venstresiden som inneholder sensordata fra en rigg. På høyre side vil det være et interaktivt kart med riggene markert på kartet. Brukere skal kunne trykke på en av riggene på kartet og få vist de nyeste sensordata for den valgte riggen. Det ble også planlagt en admin side hvor en kan legge til nye rigger. Figur 3.3 viser et tidlig utkast til dashbordet, laget i Figma (Figma, 2023), et verktøy for design og prototyping av brukergrensesnitt.



Figur 3.3 - Utkast til dashbordet

3.1.2 Alternativ løsning 2

En alternativ løsning kunne vært å bruke verktøy til å utvikle en datavisualisering side. For eksempel Power BI (Microsoft, 2023), eventuelt sammen med en løsning for å håndtere innhenting av sensordata. Power BI kunne ha gjort visualiseringen av data relativt enkelt med alle de innebygde verktøyene, samt gjort det enkelt å bruke applikasjonen på forskjellige enheter. Et annet nyttig verktøy som kunne blitt brukt er ThingSpeak (ThingSpeak, 2023), som er en «Internet of Things» (IoT) plattform og et analytisk verktøy legger til rette for innsamling, analyse og visualisering av sensordata fra tilkoblede enheter. Det hadde gitt mulighet for å håndtere innhenting og visualisering av data med bruk av ett verktøy. Mens Power BI hadde funksjoner for å filtrere ut data fra et XML-format, hadde ikke ThingSpeak dette. Dermed ville en løsning basert på ThingSpeak fortsatt trengt en modul for å filtrere ut sensordata fra XML-filene.

3.1.3 Diskusjon av alternativene

Løsningsalternativene ble diskutert innad i gruppen og med veileder. Det ble lagt vekt på tidligere erfaring og mulighet for å utforme løsningen akkurat som ønsket. Gruppens erfaring talte sterkt for første løsning. Dette siden begge gruppemedlemmene har hatt diverse programmeringsfag som gikk ut på utvikling av webapplikasjon og tilhørende oppgaver, og gruppen hadde ingen tidligere kjennskap til verktøyene nevnt i den andre alternative løsningen. Videre virket det som det var mest rom for å utvikle løsningen som tenkt, ved å starte fra bunnen og bygge den opp selv med programmeringsspråk som gruppen var relativt erfaren i.

3.2 Valgt løsning

Det første alternativet ble valgt som løsning tidlig i prosjektet, og det ble tidlig laget en simulator som publiserer og abonnerer på en HiveMQ-megler. Videre ble det laget et program som henter ut sensordata fra XML-skjemaene og oppretter Java-objekter.

3.3 Valg av verktøy

HiveMQ ble brukt som megler for publisering/abonnering av sensordata. Intellij ble brukt som IDE.

Java ble valgt som programmeringsspråk til back-end delen av applikasjonen. På back-end delen ble det også brukt WebSockets, NoSQL database og en rekke rammeverk. Disse rammeverkene inkluderte: MongoDB API-et (MongoDB, 2023), HiveMQ API-et (HiveMQ, 2023), JSON (JSON, 2023), GSON (Google, 2023), Jetty WebSockets (Eclipse, 2023) og JAXP (Oracle, 2023).

Til dashbordet ble det brukt HTML, CSS og JavaScript. Vanlig JavaScript ble brukt sammen med Javascript-rammeverket Leaflet(Leaflet, 2023) for å integrere OpenStreetMap (OpenStreetMap, 2023). Innledningsvis ble det vurdert å bruke Google Maps API (Google, 2023) for å legge til Google Maps, men etter testing og vurdering ble OpenStreetMap valgt.

Github ble brukt til kodedeling. Google docs til rapportskriving og andre dokumenter, og Google sheets til fremdriftsplan. Lucidcharts (Lucid, 2023) og Figma ble brukt til modellering. Jira software (Atlassian, 2023) ble brukt som et verktøy for utviklingsmetoden Scrum.

3.4 Prosjektmetodikk

3.4.1 Utviklingsmetodikk

Gruppen benyttet Scrum (Scrum, 2023), som er et Agile-rammeverk (Agile Alliance, 2023). Ettersom det bare var to personer i gruppen ble det bestemt at applikasjonen ble utviklet i begge prosjektets hovedområder samtidig. En person ble gjort ansvarlig for front-end, mens den andre ble ansvarlig for back-end. I tråd med Scrum metoden, ble utviklingen gjort i sprinter, på 1-2 uker, med møter der fremgang og status ble gjennomgått. Gruppen jobbet sammen da tiden var inne for integrasjon og testing, dette virket som en logisk måte for å sikre godt samarbeid og effektivitet.

Det var også fokus på at koden skulle være modulær, vedlikeholdbar og skalerbar. Dette ville si at hver klasse eller modul i applikasjonene skulle ha ett enkelt ansvar og være fokusert på å gjøre én ting bra. Klasser som benyttet seg underliggende moduler skulle ikke trenge å vite hvordan de var implementert. Dette ville gjøre fremtidige endringer enklere, ved at moduler kunne oppdateres og eventuelt byttes ut.

3.4.2 Prosjektplan

En fremdriftsplan brukt som utgangspunkt for planlegging av arbeid. Se vedlegg 9.1.

3.4.3 Risikovurdering

Det ble identifisert flere ulike risikoer i prosjektet. En av de viktigste risikoene involverer feil og mangel på sensordata. Her var det viktig å utarbeide prosedyrer som kunne iverksettes hvis disse problemene oppsto. Videre ble det identifisert andre risikoer som inkluderte diverse software, hardware problemer, misforståelser i forhold til kravspesifikasjon og sykdom blant gruppemedlemmene. Når det kommer til software og hardware problemer var det viktig å ivareta en viss fleksibilitet hvis problemer skulle oppstå. Det samme gjaldt for misforståelser i forhold til kravspesifikasjon. Her var det viktig å ha god kommunikasjon samt være fleksibel hvis det skulle være behov for endringer. Hvis sykdom skulle oppstå var det viktig at gruppemedlemmene hadde god oversikt over viktige gjøremål og tidsfrister, samt at gruppemedlemmene prøvde å holde seg oppdatert på alle utviklingsområdene i prosjektet.

For en mer detaljert risikoanalyse, se vedlegg 9.1.

3.5 Evalueringsplan

I sluttfasen av prosjektet spilte simulatoren en viktig rolle ved evaluering av webapplikasjonen. Simulatoren ville kunne sende eksempeldata til webapplikasjonen i selvvalgte tidsintervaller. Dette ville tillate gruppen å teste hvilke arbeidsmengder løsningen kunne håndtere. I tillegg skulle unit-tester gi et innblikk i hvor feiltolerant løsningen var og hvordan løsningen håndterte forskjellige input. Veileders evaluering av løsningen ville også være sentral.

4 DETALJERT LØSNING

4.1 Simulator

Tidlig i prosjektet begynte gruppen å utvikle en simulator som kunne koble seg til en HiveMQ-megler fra et cluster av meglere. Simulatoren skulle være i stand til jevnlig å sende sensordata til en megler, slik som de virkelige undervannsriggene gjør. Men i motsetning til undervannsriggene, som publiserer data hver halvtime, så kunne simulatoren publisere data i egendefinerte intervaller. Dette gjorde testing mye enklere og fremtidige demonstrasjoner av webapplikasjonen ville bli bedre når den ble oppdatert hvert minutt, i motsetning til hver halvtime. Sensordataen som simulatoren publiserer, var eldre data som ble publisert av undervannsriggene noen måneder tilbake.

Simulatoren ble utviklet med HiveMQ sitt Java-API. Med dette API-et blir det opprettet et klient-objekt som kobler seg til en HiveMQ-megler, som håndterer MQTT tilkoblinger og meldinger. MQTT-protokollet (Message Queuing Telemetry Transport) er et publiser/abonner meldingsprotokoll som er designet for bruk i områder med begrenset båndbredde, prosessorkraft eller minne. Klient-objektet benytter MQTT-protokollet til å publisere sensordata til et emne hos megleren.

Videre er det en middleware-modul som oppretter et klient-objekt som kobler seg til en megler i samme cluster. Denne klienten abonnerer på de samme emnene som simulatoren publiserer sensordata til, med QoS (Quality of Service) satt til "at least once", som vil si at hver melding publisert av simulatoren vil bli mottatt minst én gang av klienten som abonnerer på det emne. Figur 4.1 viser en del av denne koden.

```
1 Create// Create the MQTT client2
2 Mqtt3BlockingClient client = MqttClient.builder()
3       .serverHost(hostString)
4       .serverPort(8883)
5       .sslWithDefaultConfig()
6       .useMqttVersion3()
7       .buildBlocking();
8
9 // Connect to the broker
10    client.connectWith()
11        .simpleAuth()
12        .username(username)
13        .password(UTF_8.encode(password))
14        .applySimpleAuth()
15        .send();
16
17    client.subscribeWith()
18          .topicFilter("Austevoll - Nord")
19          .qos(MqttQos.AT_LEAST_ONCE).send();
20    client.subscribeWith()
21          .topicFilter("Austevoll")
22          .qos(MqttQos.AT_LEAST_ONCE).send();
23    client.subscribeWith()
24          .topicFilter("Austevoll - Sør")
25          .qos(MqttQos.AT_LEAST_ONCE).send();
26
27    Mqtt3BlockingClient.Mqtt3Publishes publishes =
28    client.publishes(MqttGlobalPublishFilter.SUBSCRIBED)
29
30 // Receive sensordata XML-files
31    Mqtt3Publish sensorDataXML = publishes
32                    .receive();
```
Figur 4.1 - Kode som kobler til en megler, abonnerer på et emne og mottar sensordata

Linje 2-7 viser hvordan klienten blir opprettet med en «host» streng og et portnummer. Klienten blir bygget med HiveMQ API-et sin "builder" klasse. Linje 10-15 viser hvordan det opprettede klient-objektet kobler seg til megleren. Videre i linje 17-25, abonnerer klienten på tre emne der sensordata blir publisert. I linje 27-32 blir sensordata mottatt.

For å ivareta sikkerheten i både simulatoren som publiserer og i modulen som abonnerer, var det påkrevd med et gyldig brukernavn og passord. Disse ble opprettet på forhånd i HiveMQ sin webklient og var nødvendig for å få tilgang til HiveMQ-clusteret.

4.2 Filtrere ut sensordata fra XML-filene

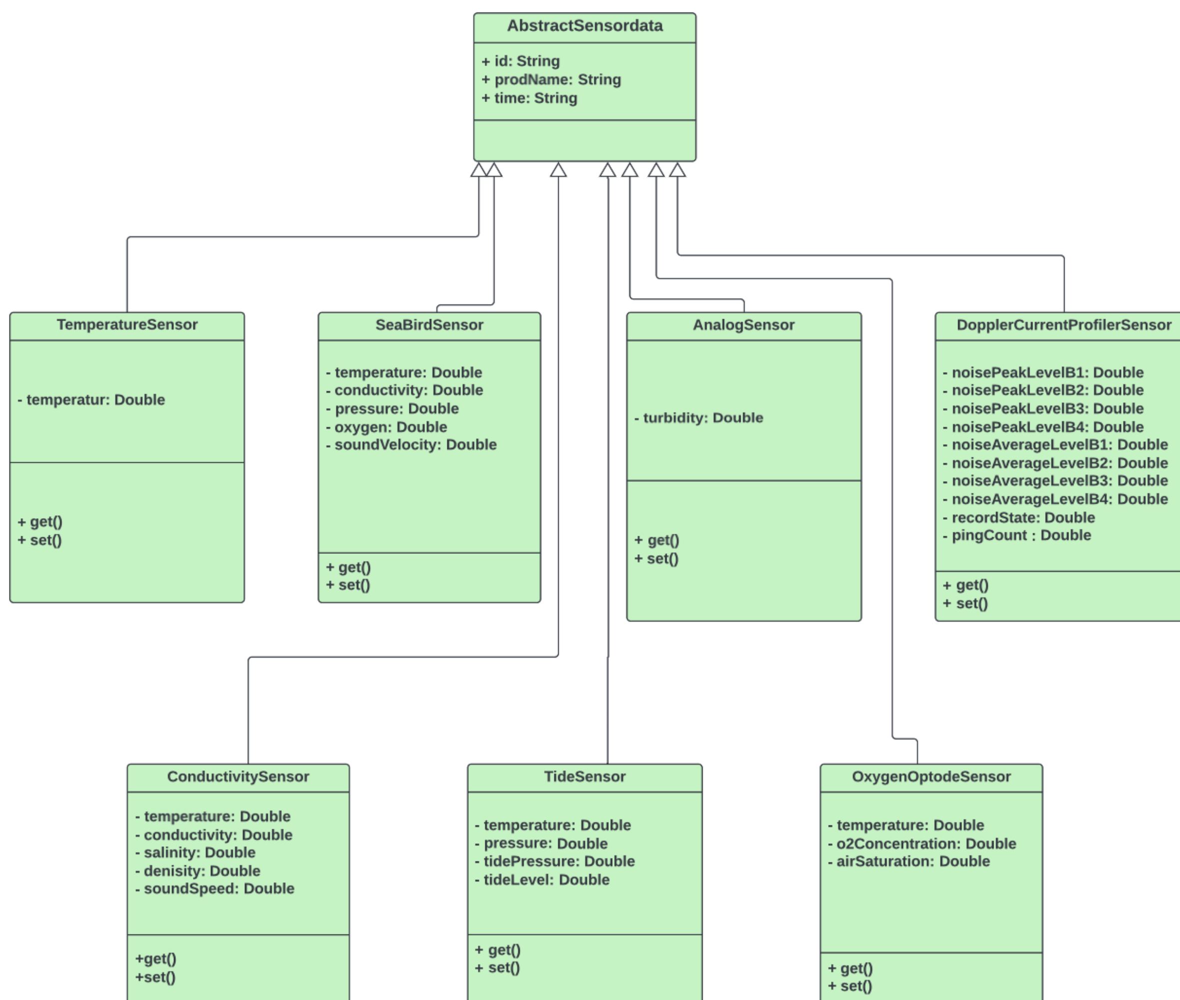
Formatet sensordataene blir sendt i, er i kompliserte XML-filer på flere tusen linjer. Hver sensor på undervannsriggen har forskjellige typer verdier og ofte ulikt antall verdier. For eksempel, har temperatursensoren en verdi som er temperatur, mens tidevannssensoren har fire verdier: trykk, tidevannstrykk, tidevannsnivå og temperatur.

En filtrerings-modul som kunne filtrere ut disse dataene ville bli nødvendig. DocumentBuilder klassen i JAXP-rammeverket (Java API for XML Processing), ble brukt til å utvikle denne modulen. DocumentBuilder klassen gir en praktisk og effektiv måte å filtrere ut data fra XML-filer, ved at metodene i klassen returnerer elementene ved gitt navn, som lister med verdier. Ved å bruke navnet til hver av sensorene på undervannsriggen, blir de ønskede verdiene filtrert ut av XML-filen. Figur 4.2 viser et utsnitt fra en av XML-filene.

```xml
<SensorData ID="5217-174" Descr="Tide Sensor #174" NodeTypeID="2020" SerialNo="174" ProdNo="5217"
ProdName="Tide Sensor" ProtocolVer="6">
  <Parameters>
    <Point ID="0" Descr="Pressure" Type="VT_R4" Format="%0.3f" Unit="kPa" RangeMin="0"
RangeMax="400">
      <Value>334.556</Value>
    </Point>
    <Point ID="1" Descr="Temperature" Type="VT_R4" Format="%0.3f" Unit="DegC" RangeMin="-5"
RangeMax="35">
      <Value>8.10294</Value>
    </Point>
    <Point ID="29" Descr="Tide Pressure" Type="VT_R4" Format="%0.3f" Unit="kPa" RangeMin="0"
RangeMax="400">
      <Value>334.5634</Value>
    </Point>
    <Point ID="30" Descr="Tide Level" Type="VT_R4" Format="%0.3f" Unit="m" RangeMin="-20"
RangeMax="20">
      <Value>-61.83512</Value>
    </Point>
  </Parameters>
</SensorData>
<SensorData ID="4835-435" Descr="Optode Sensor #435" NodeTypeID="2040" SerialNo="435"
ProdNo="4835" ProdName="Optode Sensor" ProtocolVer="6">
  <Parameters>
    <Point ID="1" Descr="O2Concentration" Type="VT_R4" Format="%0.3f" Unit="uM" RangeMin="0"
RangeMax="500">
      <Value>332.1638</Value>
    </Point>
    <Point ID="2" Descr="AirSaturation" Type="VT_R4" Format="%0.3f" Unit="%" RangeMin="0"
RangeMax="150">
      <Value>90.01277</Value>
    </Point>
    <Point ID="3" Descr="Temperature" Type="VT_R4" Format="%0.3f" Unit="Deg.C" RangeMin="-5"
RangeMax="40">
      <Value>8.101135</Value>
    </Point>
  </Parameters>
</SensorData>
```


Figur 4.2 - Utsnitt fra en av XML-filene.

Hver sensor på en undervannsrigg er plassert i en “SensorData” tag. Verdiene til hver sensor er nøstet inne i “Parameters”, “Point” og “Value” tagger. For hver av sensorene blir det opprettet Java-objekter som inneholder verdiene som er filtrert ut. Figur 4.3 under viser et klassediagram for Java-klassene det blir lagd Java-objekter av.



Figur 4.3 - Klassediagram

Klassediagrammet vise syv Java-klasser som arver fra en abstrakt super-klasse. Videre utvider Java-klassene med flere variabler enn de tre som er i super-klassen. Java-objektene blir samlet i en liste og returnert til komponentene som benytter seg av filtrerings-modulen.

4.3 Opprette database

Til lagring av sensordata ble det satt opp en NoSQL-database. En NoSQL database ble valgt fordi den er fleksibel og kan håndtere data med varierende antall verdier. I motsetning til tradisjonelle relasjonsdatabaser, er det ikke påkrevd med et fast skjema som krever at alle oppføringer har de samme feltene. Det tillater et dynamisk skjemadesign som kunne håndtere de ulike sensorene med de ulike verdiene.

Gruppen valgte å benytte seg av MongoDB Atlas, som er en skybasert databasetjeneste som gir en fleksibel og skalerbar plattform for lagring og spørring av data. MongoDB sin Java-API ble brukt til å lage en database-modul som kunne koble til databasen og ta seg av overføring

av sensordata. Koden under i figur 4.4 viser en del av hvordan sensordataen blir lagret i NoSQL databasen.

```
MongoClient mongoClient = MongoClients.create(CONNECT_URI)
database = mongoClient.getDatabase("SensorDataDemo");
MongoCollection<Document> collection =
database.getCollection(collectionName);


Document tideSensor = new Document("_id", id)
    .append("SensorID", sensorData.get(6).getId())
    .append("Time", sensorData.get(6).getTime())
    .append("Temperature",(sensorData.get(6)).getTemperature())
    .append("Pressure", ( sensorData.get(6)).getPressure())
    .append("TidePressure",(sensorData.get(6)).getTidePressure())
    .append("TideLevel", (sensorData.get(6)).getTideLevel());

collection.insertOne(analogSensor);
```
Figur 4.4 - Oppretting av dokumenter som lagres i databasen

Først blir det opprettet en “MongoClient” som kobler seg til databasen. Så blir en referanse til databasen hentet og lagret i en variabel. Videre blir en referanse til en “Collection”, med navnet gitt i parameteren “collectionName”, hentet og lagret i en variabel. En “Collection” er en samling i en MongoDB NoSQL database, som samsvarer med en tabell i en tradisjonell database, men uten krav om at alle oppføringene må inneholde spesifiserte felter. Neste steg er å opprette dokument-objekter ved å knytte nøkler til verdier. I koden vises det hvordan nøklene “_id”, “SensorID” og “Time” osv. knyttes til Java-objektet “TideSensor” som ligger i “sensorData”-listen, som er gitt i en parameter. Det opprettes slike dokumenter for alle Java-objektene som ligger i «sensorData»-listen og de blir lagret i samlingen “collection”. Det ble laget en samling for hver undervannsrigg.

Denne database-modulen ville gi de andre komponentene et abstrakt grensesnitt til å lagre og hente data fra databasen. I tillegg til lagring av sensordata, så ville database-modulen vil bli brukt til å oppdatere dashbordet med de nyeste sensordataene som er blitt lagret i databasen. På denne måten vil nye brukere som åpner nettsiden bli vist de nyligste dataene, istedenfor ingen sensordata i påvente av nye sensordata fra HiveMQ-megleren.

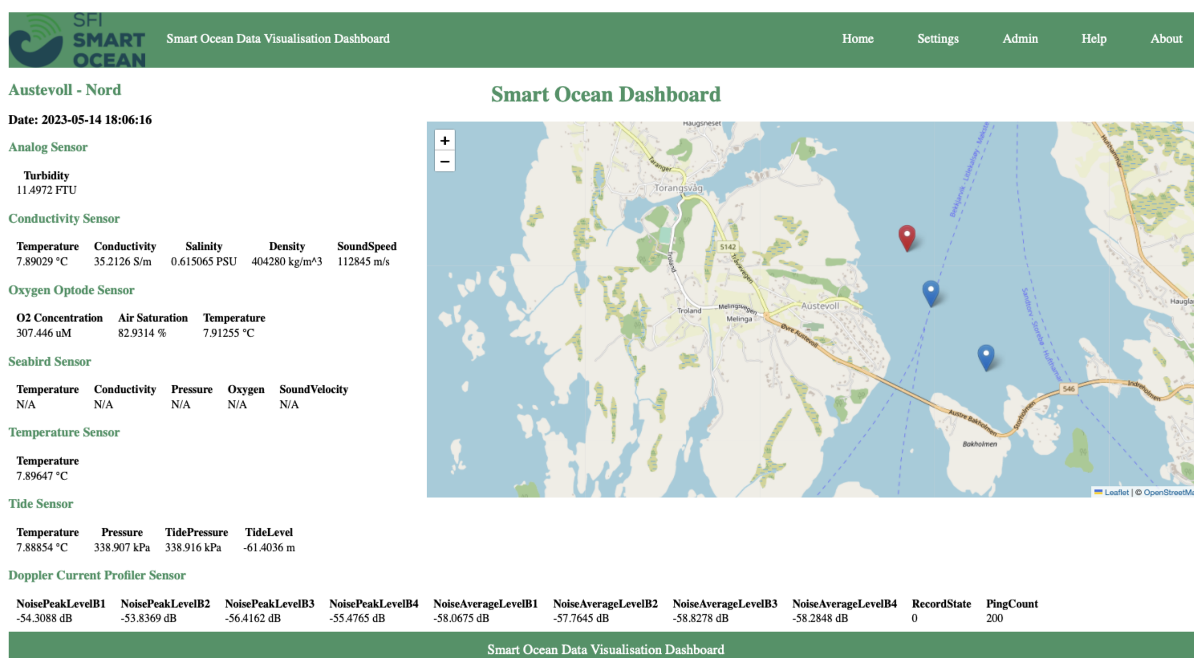
4.4 Utvikle hovedsiden til dashbordet

Dashbordet ble først utviklet med kun HTML5 og CSS fra bunnen. Andre metoder for webutvikling ble vurdert, men siden dashbordet skal være relativt enkelt, ble det besluttet

at det ikke var behov for noe rammeverk til å begynne med. Dette gjorde det enklere å ha full kontroll over dashbordet og dets kode.

Etter rammen til siden var bygd med navigasjonsbar på toppen ble det lagt inn et interaktivt kart. Som nevnt i Kapittel 3.3 ble det først brukt Google Maps, men etter revurdering og innspill fra veileder ble det besluttet å gå over til OpenStreetMap. Dette da det virket mer passende å bruke en gratis og fri karttjeneste som OpenStreetMap til dette prosjektet. Det dekket de samme behovene, men uten alle de ekstra kartmarkørene som det er mange av i Google maps.

For å implementere OpenStreetMap på dashbordet ble det brukt Leaflet, et Javascript-bibliotek for interaktiv nettkartlegging med åpen kildekode. Leaflet sitt API lar utviklere blant annet legge til kartlag og markører. Kartet er skalert til 65% vidde på høyresiden og med et panel til sensordata på venstresiden. Panelet er delt i rader med sensorer på hver rad. I tillegg er det lagt til tidsstempel over panelet for å vise når sensordataen ble registrert, samt navnet på hvilken undervannsrigg sin sensordata som blir vist. Figur 4.5 viser et skjermbilde av dashbordet.



Figur 4.5 - Dashbordet

4.5 Oppdatere sensordata på dashbordet

Sensordataen på dashbordet ble oppdatert med bruk av Javascript og WebSockets. For å hente data fra serveren, ble WebSockets benyttet. WebSockets er et protokoll for toveis kommunikasjon i sanntid mellom en klient og en server over en enkel, langvarig tilkobling. En WebSocket-server ble satt opp for å motta tilkoblinger fra klienter. Tilknyttet WebSocket-

serveren, er det en SocketHandler-komponent som håndterer tilkoblingene og administrerer kommunikasjonen mellom klient og server. På klientsiden er det et Javascript som oppretter en tilkobling til WebSocket-serveren. Koden i figur 4.5 under viser en del av SocketHandler-komponenten.

```
1 public static final List<Session> sessions = new ArrayList<>();
2
3  @OnWebSocketMessage
4   public void onMessage(Session session, String rigName) {
5     try {
6           ArrayList<AbstractSensorData> sensorDataList =
7           DAO.RetrieveRecentSensorData(rigName);
8           Gson gson = new Gson();
9           String data = gson.toJson(sensorDataList);
10          session.getRemote().sendString(data);
11      } catch (Exception e) {
12           e.printStackTrace();
13      }
14  }
15   public static void sendToAll(String messageJSON) {
16      for (Session session : sessions) {
17           try {
18               session.getRemote().sendString(messageJSON);
19           } catch (Exception e) {
20               // handle exception here
21          System.err.println("Error" + e.getMessage());
22               e.printStackTrace();
23           }
24       }
```

Figur 4.5 - SocketHandler-komponenten

Listen “sessions” inneholder referanser til alle klienter som er tilkoblet. Metoden “onMessage” henter de nyeste sensordataene til den etterspurte riggen, gitt i parameteren “rigName”. Etter at sensordataene er hentet fra databasen med, bruk av database modulen “DAO”, blir dataene omgjort til JSON formatet og sendt over WebSocket koblingen. JSON er et datautvekslingsformat som er mye brukt og støttes av de fleste programmeringsspråk. Et Gson-objekt, som er Googles API for omgjøring fra Java til JSON, blir brukt til dette formålet. Metoden “sendToAll”, blir brukt til å sende de nyeste sensordataene som akkurat har blitt publisert til HiveMQ-megleren. Den sender sensordataene til alle klientene, ved å bruke “sessions” listen.

Når en klient åpner webapplikasjonen blir det sendt en melding til WebSocket-serveren om å hente de nyeste sensordataene til "Austevoll - Nord" riggen, som er satt som valgt rigg når en først åpner nettsiden. Hvis brukeren klikker på en av de andre riggene på kartet, blir den valgt riggens markør på kartet endret farge til rød. Dette klarte gruppen ved å legge til «*eventlisteners»* til Leaflet markøren på kartet. Når en «eventlistener» blir utløst blir det også sendt en melding til WebSocket-serveren om å hente de nyeste sensordataene. Etter at WebSocket-serveren har hentet de nyeste sensordataene fra databasen og sendt dem tilbake, blir Javascript brukt til å dynamiskt oppdatere tabellene på dashbordet.

Hver gang det kommer nye sensordata fra HiveMQ-megleren, blir filtrerings-modulen brukt til å få sensordata filtrert ut av XML-filen og laget en liste med Java-objekter. Videre blir listen omgjort til JSON formatet og sendt over WebSocket koblingen, med «sendToAll» metoden, til Javascriptet hos klienten som lytter på WebSocket koblingen etter oppdateringer.

Etter at sensordataen er blitt sendt til dashbordet blir sensordataen lagret i databasen. Det ble oppdaget at lagring til databasen var en relativt tidkrevende prosess. For å forbedre implementasjonen, ble det opprettet nye tråder for hver gang det skulle lagres sensordata i databasen. Med tråder, så kunne sensordata som skulle lagres i databasen, bli gjort parallelt. Dette gjorde denne delen av applikasjonen en god del raskere.

I Javascriptet hos klienten blir det også sjekket at ikke alle verdiene er null. Hvis alle verdiene er null, som er en indikasjon på feil eller mangel på sensordata, blir disse markert med "N/A", "Not Available".

Javascript, sammen med WebSockets, lar dashbordet bli dynamisk oppdatert uten at siden må lastes inn på nytt. Dette gir en jevn og responsiv opplevelse for brukeren.

5 RESULTATER

5.1 Evalueringsmetode

Under hele utviklingsperiodene har funksjonell testing blitt benyttet etter hver iterasjon. Simulatoren har vært sentral i den funksjonelle testingen, ved at ytelsen til webapplikasjon kunne testes under ulike tidsintervaller av innkommende sensordata. Simulatoren imiterer flere rigger ved at den publiserer data til ulike emner hos en HiveMQ-megler. Dermed fikk gruppen evaluert webapplikasjonen sin ytelse. Videre gir dette et godt bilde på hvordan løsningen ville fungere hvis den ble distribuert og fikk data i sanntid fra de virkelige undervannsriggene.

For å evaluere modulen som er ansvarlig for å filtrere ut data fra XML-filene ble det brukt Unit tester. Det ble laget flere tester som evaluerte modulen sin evne til å filtrere ut data for ulike XML-filer. Testene skulle sørge for at modulen var feiltolerant for de ulike formatene XML-filene kom i. Denne evalueringsmetoden, sammen med funksjonell testing, ville sørge for at produktet ble utviklet riktig.

Underveis ble løsningen evaluert av veilederen gjennom møter med demonstrasjon av løsningen, innspill og diskusjon. Det var viktig å validere løsningen underveis for å sikre at løsningen var på vei i riktig retning. Veileder hjalp med å avklare hvilke målområder som er viktig og hva som ville være gjeldende i en god løsning. Denne evalueringsmetoden sørget for at riktig produkt ble utviklet.

5.2 Evalueringsresultat

Webapplikasjonen som er blitt utviklet fungerer som tiltenkt. Bruker kan velge undervannsrigg fra kartet og får vist de nyeste sensordataene på en oversiktlig måte. For hver XML-fil som blir publisert fra simulatoren til HiveMQ-megleren, så blir sensordata filtrert ut og presentert på en intuitiv og oversiktlig måte på dashbordet. Oppdateringen av sensordataene på dashbordet skjer dynamisk ved bruk av Javascript og gir brukeren en sømløs og responsiv opplevelse. Inkluderingen av en "about"-side og en "help"-side forbedrer brukeropplevelsen ved å gi informasjon om webapplikasjonen og en brukermanual for dashbordet.

Unit testene var nyttig til å avdekke og rette opp feil i filtrerings-modulen. Gitt betydningen av denne modulen i applikasjonens funksjonalitet, var det viktig å sikre at den opererte riktig og nøyaktig.

Evalueringen av webapplikasjonen har gitt positive resultater, ved å underveis korrigere løsningen slik at den endte opp som ønsket. Veilederen har gitt positive tilbakemeldinger på løsningen, og vi tok det som en indikasjon på at webapplikasjonen oppfylte de viktigste kravene og viste tilfredsstillende ytelse.

5.3 Prosjektresultat

Evalueringsresultatet har vist at webapplikasjonen som er blitt utviklet har bestått de evalueringene den har gjennomgått. Løsningen viser kontinuerlig sanntids-data på en oversiktlig måte på et dashbord. Det var dette som var det viktigste målet i prosjektet og det har gruppen oppnådd.

5.4 Prosjektgjennomføring

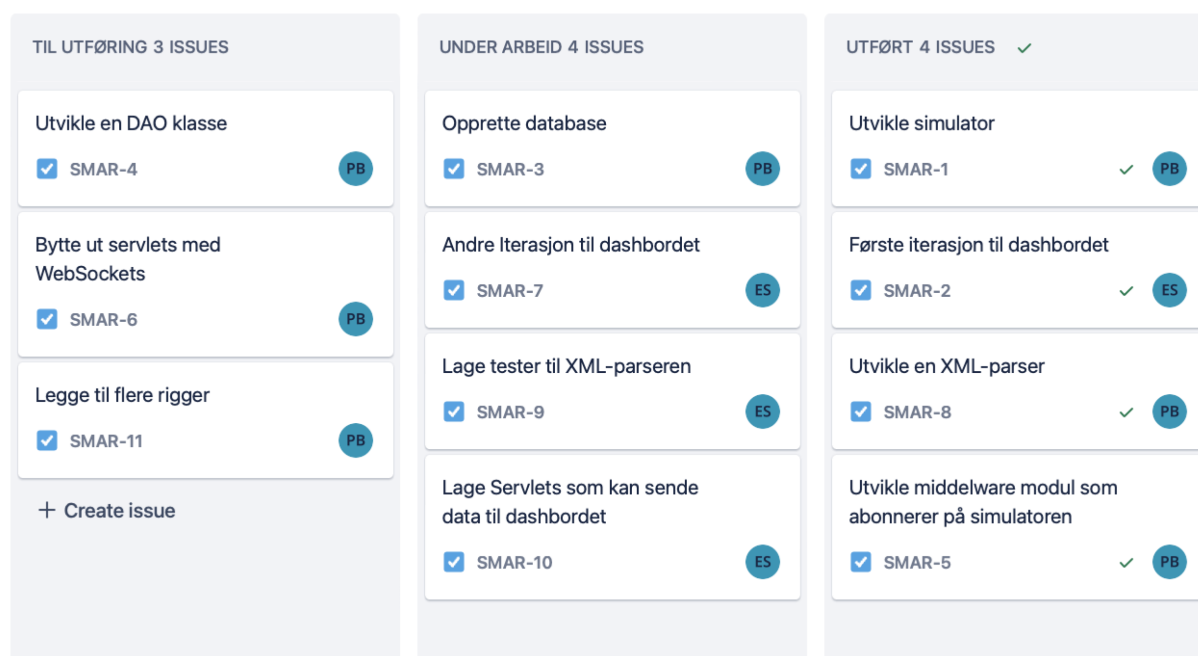
Prosjektet ble gjennomført i stor grad i henhold til planen som ble satt. Det var litt forsinkelser med enkelte ting som var mer kompliserte og dermed tok lengre tid enn antatt. Samt noe sykdom som og gjorde at gruppen havnet litt bak skjema. Dette resulterte i at det ikke ble tid til å utvikle et grensnitt for å legge til nye undervannsrigger i applikasjonen.

Planen som ble satt og fulgte fungerte i stor grad. Tid ble spart ved å ikke måtte bruke tid underveis på å finne ut hva neste steg var. Det var og til gruppens fordel å ha en helhetlig plan som gjorde det lettere å fordele arbeidsoppgaver underveis. Med bruk av Scrum, ble det ble lettere å holde oversikt over det andre medlemmet sine oppgaver. Valget om å dele opp store deler av utviklingsprosessen til hvert medlem gjorde prosessen mer effektiv, men samtidig mer delt. Mest sannsynlig hadde ikke en annen metode for oppdeling vært bedre, men det er noe å reflektere over.

6 DISKUSJON

6.1 Fremgangsmåte

Tidlig i prosjektperioden ble det utarbeidet en fremdriftsplan. Dette var for å skape en oversikt over de ulike arbeidsoppgavene. I tillegg ble Scrum-verktøyet, Jira, tatt i bruk for å holde oversikten over alle oppgavene som skulle utføres. Dette verktøyet ga en litt mer detaljert oversikt over prosjektstatusen. Med bruk av Scrum-verktøyet fikk gruppen delt opp oppgavene og fikk opprettholdt en klar oversikt over prosjektets status. Det ble holdt møter innad i gruppen, og møter med veileder, hvor fremgangen i prosjektet ble diskutert og de neste iterasjonene av webapplikasjonene ble planlagt. Ved at denne utviklingsmetoden ble utnyttet, ble gruppemedlemmene holdt oppdatert på hva som hadde blitt gjort, hva som fortsatt ble jobbet med og jobber som ikke var påbegynt. Figur 6.1 viser Scrum-tavlen underveis i prosjektet.



Figur 6.1 – Scrum-tavlen underveis i prosjektet

I utviklingen, oppsto det underveis flere problemer med å koble til HiveMQ-Megleren, samt problemer med MongoDB sitt Java-API. Det viste seg til slutt at det var Java-versjonen 14.1.1 som ikke var kompatibelt med HiveMQ og MongoDB sine API-er. Etter mye feilsøking, ble problemene utbedret, men det tok mer tid enn planlagt og dette resulterte at det ikke ble tid til å utvikle siden for innstillinger og et admin grensesnitt som kunne legge til nye rigger til webapplikasjonen. Gruppen innså i slutten av prosjektperioden at utviklingen av dette ble for tidkrevende og denne delen av applikasjonen gjenstår. Mens veileder hadde et ønske om

et slikt grensesnitt, ville det i mellomtiden være relativt enkelt å legge til nye rigger direkte i koden.

Oppdateringen av sensordataen på dashbordet ble først implementert med bruk av servlets og REST-API-et. Denne implementasjonen benyttet HTTP-protokollet med bruk av request og response. Men et problem med denne implementasjonen var at klienten alltid måtte spørre server etter oppdateringer. Selv om dette fungerte, så var det en unødvendig belastning på serveren med hyppige forespørsler etter nyeste sensordata. I tillegg ville nyeste sensordataene bare sitte å vente på at en klienter skulle spørre etter dem. Ettersom det viste seg å ikke være den beste løsningen, og etter forslag fra veileder, endret gruppen implementasjonen til å bruke WebSockets istedenfor. Denne prosessen ble mer tidkrevende enn planlagt ettersom gruppen støttet på flere problemer med versjoner av WebSockets som ikke var kompatible med Java-versjonen til applikasjonen. Gruppen kom til slutt fram til en løsning, og det var tydelig at WebSockets var bedre egnet til sanntidsoppdateringer. Nå kunne serveren sende ut de nyeste sensordataene til klientene med en gang de kom fra HiveMQ-megleren, uten at klienten måtte spørre etter dem.

Generelt sett gikk det mye tid til feilsøking og til å gjøre løsningen feiltolerant. Det tok også en del tid til å sette seg inn i og lære seg de forskjellige rammeverkene som ble brukt, ettersom det var flere rammeverk gruppen ikke hadde erfaring med fra før.

6.2 Konsekvens av fremgangsmåte

Simulatoren gjorde at det var mulig å teste løsningen fortløpende gjennom store deler av utviklingsprosessen. Den gjorde det mulig å kjøre løsningen relativt likt brukstilfellene den skal brukes i og dermed ga en god pekepinn på hva som fungerte og ikke.

Ved å jobbe på hvert sitt område fikk en mindre innsikt på den andre sitt område og utvikling. Det var og en del risiko forbundet med en slik oppdeling. I og med at gruppen var på kun to medlemmer som i store deler av utviklingen jobbet på hvert sitt område, var utviklingen sårbar hvis det skulle skjedd endringer. Hadde det oppstått lengre sykdom eller fravær ville det gått en del tid å sette seg inn i den andres oppgaver og løsninger. Dette ble ikke noe problem under utviklingen og tid ble spart underveis med en slik oppdeling, men gruppen erkjenner at det var en risiko.

En av risikoene som var klar for gruppen siden begynnelsen av prosjektet var at ting kan ta lengre tid enn antatt. Dette gjelder spesielt for implementering av funksjoner i løsningen. Gruppen satt generelt av god tid til de forskjellige implementasjonene og tid til å sette seg

inn i rammeverkene som skulle bli brukt. Dette ble gjort grunnet gruppens begrensede erfaring i å utvikle store løsninger. De fleste implementasjonene holdt fristene, men noen tok lengre tid.

En ting gruppen ville ha gjort annerledes hvis de kunne gjort arbeidet på nytt, ville vært å planlegge implementasjonen bedre og tenke litt mer langsiktig. Fordi gruppen ikke planlagte å bruke WebSockets til å begynne med, så ble brukt mye tid på implementasjonen av REST-API-et, som til slutt ikke ble brukt. Hadde gruppen brukt mer tid på å undersøke mulige teknologier, så kunne gruppen ha valgt riktig teknologi fra starten av og spart mye tid.

Men til slutt fikk gruppen utviklet en løsning som veileder var fornøyd med. Med unntak av grensesnittet for å legge til nye rigger, så oppfylte applikasjonen de viktigste kravene. En svakhet ved løsningen var at den var avhengig av at XML-filene ikke avviker fra formatet som applikasjonen er tilpasset. Hvis det ble for mye endringer ville ikke sensordataen bli oppdaget og ville ikke bli sendt til dashbordet. Applikasjonen ville derimot ikke krasje, men bare hoppe over en XML-fil som hadde et ukjent format. Men av de hundrede tidligere XML-filene gruppen har testet applikasjonen med, så har de vært konsistente. En annen svakhet med applikasjonen var hvis sensordata blir publisert til samme rigg oftere enn hvert sekund, så ville løsningen få problemer med å lagre sensordata i databasen, da det ble for lite tid. Sensordata ville bli sendt til dashbordet, men noe av sensordataen ville ikke bli lagret i databasen ved så intense tidsintervaller. Men slike tidsintervaller var urealistiske hvis webapplikasjonen skulle bli utplassert, ettersom de virkelige riggene sendte sensordata hver halvtime.

7 KONKLUSJON OG VIDERE ARBEID

7.1 Konklusjon

Som nevnt i begynnelsen på rapporten omhandlet forskningsspørsmålene om et webapplikasjons rammeverk var egnet for å lage et dashbord som visualiserer sensordata og hvordan designe for forskjellige datakilder og sensorer. Videre handlet forskningsspørsmålene om hvorvidt prosjektet klarte sanntids og feiltolerant databehandling gjennom HiveMQ's meldingstjeneste.

Så for å konkludere på forskningsspørsmålene som ble satt opp i starten av prosjektet:

FS1: Hvilke webrammeverk er egnet for et dashbord som visualiserer sensordata?

Til webdelen av løsningen ble det brukt HTML, CSS, Javascript og Leaflet. De tre første utfyller hverandre og utgjør et rammeverk som egnet seg bra til prosjektets løsning. Gitt løsningens enkle utforming og få funksjoner var det ikke nødvendig med noe stort og komplisert rammeverk. Siden gruppen valgte å bruke dette som rammeverk ble det spart tid på å ikke måtte sette seg inn i dokumentasjonen til større rammeverk. Videre er HTML, CSS og JS noe gruppens medlemmer er godt kjent med og effektivt kunne jobbe med. Rammeverket Leaflet ble brukt for å integrere OpenStreetMap. Det fungerte bra til å visualisere sensorriggene som markører på kartet.

FS2: Hvordan designe en webapplikasjon for flere forskjellige datakilder og sensorer?

Ved å bruke HiveMQ sitt Java-API, kunne middleware-modulen abonnere på flere emner samtidig. For hver datakilde, undervannsrigg, var filtrerings-modulen i stand til å hente ut de relevante verdiene fra de ulike sensorene i XML-filene. Brukeren av dashbordet kunne klikke på en av markørene på kartet og få vist de nyeste sensordataene.

FS3: Hvordan kan webapplikasjonen vise sensordata i sanntid via et meldingssystem og være feiltolerant ved mangel eller feil på sensordata eller meldingssystem?

Ved å bruke HiveMQ sitt Java-API, en NoSQL database, WebSockets og Javascript utviklet gruppen en webapplikasjon som var i stand til å vise sensordata i sanntid, samtidig som applikasjonen var feiltolerant ved mangel eller feil på sensordata. Med MongoDB sin database og tilhørende Java-API, sammen med Javascript, var løsningen feiltolerant ved mangel eller feil på sensordata. Ved mangel på sensordata fra meldingssystemet, så ville brukere alltid blitt vist den nyeste sensordataen som var lagret i databasen. Et Javascript sjekket for feil og mangler i sensordataen som var blitt sendt til klienten, og informerte bruker om disse.

7.2 Videre arbeid

Videre arbeid vil stort sett basere seg på å gjøre løsningen mer robust og åpen for fremtidige rigger med forskjellige sensorer. Det er mulig det blir tatt i bruk utstyr som vil kreve endringer i webapplikasjonen. På de to riggene som var utplassert, var det noen forskjeller i formatet på XML-filene som krevde noe endring i koden. Så hvis XML-filene de neste riggene publiserer blir for ulik de første riggene, så vil det kreve en oppdatering av noen av komponentene i applikasjonen.

Eventuell ekspandering av webapplikasjonen kunne innebært å utvide med brukere. At en kan logge seg inn og ha forskjellige rigger/sensorer en abonnerer på og kunne se historien/grafer til disse.

Videre var det planlagt diverse administrative egenskaper, noe som uteble grunnet mangel på tid. Disse egenskapene innebar å kunne logge seg inn som administrator hvor en da har mulighet til å legge til og fjerne ulike rigger, samt legge til andre administratorer. Muligheten for å legge til flere rigger gjennom webapplikasjonen er noe som bør være en prioritet ved videre utvikling. Slik løsningen er nå må nye rigger legges til gjennom endringer i koden til løsningen.

8 REFERANSER

Aanderaa, 2023. *Aanderaa About.* [Online]
Available at: https://www.aanderaa.com/about
[Accessed 01 02 2023].
Agile Alliance, 2023. *Agile Alliance.* [Online]
Available at: https://www.agilealliance.org/agile101/
[Accessed 20 01 2023].
AkerBP, 2023. *AkerBP.* [Online]
Available at: https://akerbp.com/
[Accessed 05 05 2023].
Angular, 2023. *Angular.* [Online]
Available at: https://angular.io/
[Accessed 01 02 2023].
Atlassian, 2023. *Atlassian Jira.* [Online]
Available at: https://www.atlassian.com/software/jira
[Accessed 13 03 2023].
Bouvet, 2023. *Bouvet About.* [Online]
Available at: https://www.bouvet.no/om-bouvet
[Accessed 05 05 2023].
Eclipse, 2023. *Eclipse Jetty.* [Online]
Available at: https://www.eclipse.org/jetty/documentation/jetty-11/programming-guide/index.html
[Accessed 01 03 2023].
Figma, 2023. *Figma.* [Online]
Available at: https://www.figma.com/
[Accessed 16 02 2023].
Forsvaret, 2023. *Forsvarets forskningsinstitutt Hjem.* [Online]
Available at: https://www.ffi.no/
[Accessed 02 05 2023].
Google, 2023. *Google Gson.* [Online]
Available at: https://sites.google.com/site/gson/gson-user-guide
[Accessed 07 03 2023].
Google, 2023. *Google Maps Developers.* [Online]
Available at: https://developers.google.com/maps
[Accessed 15 01 2023].
Havforskningsinstituttet, 2023. *Havforskningsinstituttet Hjem.* [Online]
Available at: https://hi.no/hi
[Accessed 05 04 2023].
HiveMQ, 2023. *HiveMQ Github.* [Online]
Available at: https://github.com/hivemq/hivemq-mqtt-client
[Accessed 28 01 2023].

HVL, 2023. *HVL Hjem.* [Online]
Available at: https://www.hvl.no/
[Accessed 05 04 2023].
HVL, 2023. *HVL Retningslinjer for bachelor.* [Online]
Available at: https://www.hvl.no/globalassets/hvl-internett/dokument/forskrifter-reglar-retningsliner/retningslinjer-for-bacheloroppg-bokm.pdf
[Accessed 10 02 2023].
Jevons, C., 2019. *HiveMQ.* [Online]
Available at: https://www.hivemq.com/blog/mqtt-client-library-enyclopedia-hivemq-mqtt-client/
[Accessed 26 01 2023].
Kongsberg, 2023. *Kongsberg.* [Online]
Available at: https://www.kongsberg.com/#
[Accessed 05 05 2023].
Leaflet, 2023. *Leaflet.* [Online]
Available at: https://leafletjs.com/
[Accessed 01 03 2023].
Lucid, 2023. *Lucidchart.* [Online]
Available at: https://www.lucidchart.com/pages/
[Accessed 15 02 2023].
Metas, 2023. *Metas About.* [Online]
Available at: https://metas.no/about/
[Accessed 05 05 2023].
Microsoft, 2023. *Microsoft Power BI.* [Online]
Available at: https://powerbi.microsoft.com/en-us/
[Accessed 05 05 2023].
MongoDB, 2023. *MongoDB.* [Online]
Available at: https://www.mongodb.com/atlas/database
[Accessed 25 03 2023].
Mozilla, 2023. *Mozzila.* [Online]
Available at: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON
[Accessed 03 03 2023].
NORCE, 2023. *NORCE Hjem.* [Online]
Available at: https://www.norceresearch.no/
[Accessed 05 04 2023].
NORCE, 2023. *NORCERESEARCH.* [Online]
Available at: https://www.norceresearch.no/prosjekter/sfi-smart-ocean-fleksibel-og-kost-effektiv-monitorering-for-forvaltning-av-et-baerekraftig-og-produktivt-hav
[Accessed 02 02 2023].
NORCE, 2023. *SFI Smart Ocean Data Exploration Portal.* [Online]
Available at: https://smartocean.web.norce.cloud/#/view/default
[Accessed 05 02 2023].
OpenStreetMap, 2023. *OpenStreetMap.* [Online]
Available at: https://www.openstreetmap.org/about
[Accessed 27 02 2023].

Oracle, 2023. *DocumentBuilderFactory Documentation.* [Online]
Available at:
https://docs.oracle.com/javase/7/docs/api/javax/xml/parsers/DocumentBuilderFactory.html
[Accessed 01 03 2023].
Oracle, 2023. *JAXBContext Documentation.* [Online]
Available at: https://docs.oracle.com/javaee/7/api/javax/xml/bind/JAXBContext.html
[Accessed 10 02 2023].
Reach Subsea, 2023. *Reach Subsea Company.* [Online]
Available at: https://reachsubsea.no/company/
[Accessed 05 05 2023].
React, 2023. *React.* [Online]
Available at: https://react.dev/
[Accessed 01 02 2023].
Scrum, 2023. *Scrum.* [Online]
Available at: https://www.scrum.org/resources/what-scrum-module
[Accessed 20 01 2023].
SFI Smart Ocean, 2023. *Data exploration for SFI-Smart Ocean.* [Online]
Available at: https://sfismartocean.no/elementor-2042/
[Accessed 05 05 2023].
Smart Ocean, 2023. *Smart Ocean.* [Online]
Available at: https://sfismartocean.no/about/
[Accessed 24 04 2023].
Tampnet, 2023. *Tampnet About.* [Online]
Available at: https://www.tampnet.com/about
[Accessed 05 05 2023].
TSC Subsea, 2023. *TSC Subsea About.* [Online]
Available at: https://www.tscsubsea.com/about-us/
[Accessed 05 05 2023].
UiB, 2020. *UiB.* [Online]
Available at: https://www.uib.no/smartocean/134560/slik-vil-de-lage-et-smartere-hav
[Accessed 27 1 2023].
UiB, 2023. *UiB Hjem.* [Online]
Available at: https://www.uib.no/
[Accessed 04 04 2023].
W Sense, 2023. *W Sense.* [Online]
Available at: https://wsense.it/
[Accessed 26 04 2023].

9 VEDLEGG

Github linker:
smartoceanplatform (github.com)
588406/SmartOceanDashboardBachelor (github.com)
smartoceanplatform/pd1-sampledata (github.com)

9.1 Fremdriftsplan

Milepæl 1 - Oppnådd 19.februar
En publish simulator som kobler seg til et HiveMQ-cluster og publiserer en XML-fil med sensordata.

Milepæl 2 - Oppnådd 21.februar
En Middleware modul som kobler til og abonnerer til et emne som til et HiveMQ-cluster.

Milepæl 3 - Oppnådd 22.februar
Et Java-program som filtrerer ut sensordata fra XML-filene og oppretter Java-objekter.

Milepæl 4 - Oppnådd 5.mars
Første utkast av nettside med et interaktivt kart med en sidemeny for fremtidig sensordata.

Milepæl 5 - Oppnådd 5.mars
Opprettet en database til å lagre sensordata.

Milepæl 6 - Oppnådd 25.mars
En modul som kobler seg til og sender sensordata til databasen.

Milepæl 7 - Oppnådd 20.april
Dashbord som mottar sensordata og viser dem.

Smart Ocean Data Visualisation Dashboard Fremdriftsplan

Project Start Date: 1/2/2023 (Monday) Display Week: 1

Week 1 Week 2 Week 3 Week 4 Week 5 Week 6 Week 7 Week 8

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 2

M T W Th F Sa Su M T W Th F Sa Su M T W Th F Sa Su M T W Th F Sa Su M T W Th F Sa Su M T W Th F Sa Su M T W Th F Sa Su M T W Th F S

WBS	Task	Lead	Start	End	Days	Hours	% Done	Work Days
1	**Oppstart**		Mon 1/09/23	Sun 1/29/23	21		100%	15
1.1	Planlegging	Pål	Mon 1/09/23	Mon 1/09/23	1	2	100%	1
1.2	OA-4 Oppstartsmøte	Pål	Thu 1/12/23	Thu 1/12/23	1	2	100%	1
1.3	OA-5 Statusmøte	Eirik	Thu 1/19/23	Thu 1/19/23	1	4	100%	1
1.4	OA-6 Tittel,håndbok,visjon	Eirik	Mon 1/23/23	Sun 1/29/23	7	8	100%	5
2	**Modellering**		Mon 1/30/23	Tue 2/07/23	9		100%	7
2.1	Modellering	Pål	Mon 1/30/23	Tue 1/31/23	2	2	100%	2
2.2	OA-7 Statusmøte	Pål	Wed 2/01/23	Wed 2/01/23	1	2	100%	1
2.3	Modellering Front end	Eirik	Thu 2/02/23	Sun 2/05/23	4	4	100%	2
2.4	Modellering Back end	Pål	Mon 2/06/23	Thu 2/09/23	4	2	100%	4
3	**Utvikling**		Mon 2/13/23	Wed 2/22/23	10		100%	8
3.1	HiveMQ publish simulator	Pål	Mon 2/13/23	Sun 2/19/23	7	8	100%	5
3.3	Milepæl 1	Pål	Sun 2/19/23	Sun 2/19/23	0		100%	0
3.4	HiveMQ subscribe simulator	Pål	Sun 2/19/23	Tue 2/21/23	3	4	100%	2
3.5	Milepæl 2	Pål	Tue 2/21/23	Tue 2/21/23	0		100%	0
3.6	SensorData XML-Parser	Pål	Tue 2/21/23	Wed 2/22/23	2	8	100%	2
3.7	Milepæl 3	Pål	02/22/2023	Wed 2/22/23	0		100%	1
3.8	Første utkast til webside	Eirik	02/13/2023	Wed 2/22/23	10	18	100%	8
3.9	Milepæl 4	Eirik	02/22/2023	Wed 2/22/23	0		100%	1

WBS	Task	Lead	Start	End	Days	Hours	% Done	Work Days
4	**Rapport**		Wed 2/22/23	Sun 2/26/23	5		100%	3
4.1	Midtveisrapport iterasjon 1	E&P	Wed 2/22/23	Sun 2/26/23	5	12	100%	3
5	**Utvikling**		Tue 2/28/23	Fri 3/03/23	4		100%	4
5.1	Middelware	Pål	Tue 2/28/23	Fri 3/03/23	4		100%	4
5.2	Milepæl 5		Sun 3/05/23	Sun 3/05/23	1		100%	0
6	**Rapport**		Fri 3/03/23	Thu 3/09/23	10		100%	5
6.1	Midtveisrapport iterasjon 2	E&P	Fri 3/03/23	Thu 3/09/23	7		100%	5
6.2	Midtveispresentasjon	E&P	Mon 3/13/23	Wed 3/15/23	3		100%	3
7	**Utvikling**		Mon 3/20/23	Sun 4/09/23	21		100%	15
7.1	Database	Pål	Mon 3/20/23	Sun 3/26/23	7		100%	5
7.2	Milepæl 6		Sat 3/25/23	Sat 3/25/23	1		100%	0
7.3	Middleware	Pål	Mon 3/27/23	Sun 4/09/23	14		100%	10
7.4	Front end	Eirik	Mon 3/20/23	Sun 4/09/23	21		100%	15
7.5	Milepæl 7		Thu 4/20/23	Thu 4/20/23	1		100%	1
7.6	Testing		Mon 4/17/23	Sun 4/23/23	7		100%	5

WBS	Task	Lead	Start	End	Days	Hours	% Done	Work Days
7	**Utvikling**		Mon 3/20/23	Sun 4/09/23	21		100%	15
7.1	Database	Pål	Mon 3/20/23	Sun 3/26/23	7		100%	5
7.2	Milepæl 6		Sat 3/25/23	Sat 3/25/23	1		100%	0
7.3	Middleware	Pål	Mon 3/27/23	Sun 4/09/23	14		100%	10
7.4	Front end	Eirik	Mon 3/20/23	Sun 4/09/23	21		100%	15
7.5	Milepæl 7		Thu 4/20/23	Thu 4/20/23	1		100%	1
7.6	Testing		Mon 4/17/23	Sun 4/23/23	7		100%	5
8	**Rapport**		Mon 4/10/23	Mon 5/22/23	43		100%	31
8.1	OA-12 Brukerhåndbok iter.3		Mon 4/10/23	Tue 4/11/23	2		100%	2
8.2	Sluttrapport iter. 1		Tue 4/11/23	Mon 4/24/23	14		100%	10
8.3	Sluttrapport iter. 2		Mon 4/24/23	Sat 5/06/23	13		100%	10
8.4	Sluttrapport siste		Sat 5/06/23	Mon 5/22/23	17		100%	11

Tabell 1 - Fremdriftsplan

9.2 Risikoanalyse

	Hendelse/Risiko	Årsak	Sannsynlighet	Konsekvens	Risikoprodukt	Tiltak
1	Feil i sensordata.	Sensor rapporterer feil data.	Middels (3)	Middels (3)	9	Ha en prosedyre klar ved feil data.
2	Mangel på sensordata.	Sensor rapporterer ikke data.	Høy (4)	Middels (3)	12	Ha en prosedyre for hva vi skal gjøre hvis det mangler data.
3	Problem med software rammeverk.	Rammeverk virker ikke som forventet.	Middels (3)	Høy (4)	12	Ha alternative rammeverk som vi kan bytte til.
4	Problem med utviklingsmiljø.	Utviklingsmiljø virker ikke som ønsket.	Middels (3)	Middels (3)	9	Ha alternative utviklingsmiljø vi kan bytte til.
5	Problem med hardware.	Hardware med uforventet oppførsel.	Middels (3)	Middels (3)	9	Benytte github og annen skylagring for å forhindre tap av data ved alvorlige hardware problemer.
6	Misforståelser i forhold til kravspesifikasjon.	Ikke god nok kommunikasjon mellom gruppe og veileder.	Middels (3)	Høy (4)	12	Tett dialog med veileder.
7	Gruppemedlem blir langtidssyk.	Sykdom.	Lav (2)	Høy (4)	8	Ha god oversikt over hva som må gjøres og tidsfrister

Tabell 2 - Risikoanalyse

Sannsynlighet	Svært Høy (5)					
	Høy (4)					
	Middels (3)					
	Lav (2)					
	Svært Lav (1)					
	Konsekvens					

Tabell 3 – Risikoanalyse