Brage Halse Snarud
Joakim B. W. Solheim
Hallvard Sælthun

Operasjonalisering av en maskinlæringsmodell for å detektere avvik og farer for grunnstøtinger i sanntid

Bacheloroppgave

Trondheim, 20. mai 2019

Institutt for Datateknologi og Informatikk (IDI)

Norges Teknisk- og Naturvitenskapelige Universitet (NTNU)

Veileder: Assisterende professor, Atle Olsø

NTNU

Forord

Denne rapporten omhandler prosjektet: “Operasjonalisering av en maskinlæringsmodell for å detektere avvik og fare for grunnstøtinger i sanntid”. Rapporten gir innsikt i hvilke krav som er satt til prosjektet og produktet som har blitt utviklet, samt hvordan disse er forsøkt løst.

Det var flere årsaker til at vi valgte denne oppgaven, men de viktigste grunnene var:

- En spennende problemstilling, hvor vi kunne få muligheten til å lære mye nytt om netteknologi og maskinlæring.
- Muligheten til å få førstehånds erfaring med hvordan det er å jobbe på en arbeidsplass og med ansatte i en bedrift.
- Muligheten til å utvikle noe som kan gjøre sjøtrafikk i Norge tryggere og redusere risikoen for helse- og miljømessige konsekvenser relatert til grunnstøtinger.

Utviklingen av produktet er basert på manifestet for smidig programvareutvikling, og er sterkt påvirket av utviklingsrammeverket Scrum. Norconsult Informasjonssystemer har fungert som produkteier, fulgt opp utviklingen tett, veiledet og hjulpet til med å legge gode forutsetninger for prosjektet.

Vi ønsker å takke følgende personer for deres hjelp og støtte under prosjektet:

- Andreas Ravnestad og Vigleik Lund fra Norconsult Informasjonssystemer, for teknisk og faglig veiledning under hele prosjektet.
- Jarle Hauge og Harald Åsheim fra Kystverket, for at de har hjulpet oss med å forstå produktet sin målgruppe bedre, og for at vi har fått tilgang til historiske AIS-data.
- Bernhard Engelschiøn, Henrik Kuosmanen og Magnus Torkelsen, studenter ved NTNU Ålesund, for at de delte sine resultater fra deres bacheloroppgave med oss.
- Øvrige ansatte ved Norconsult Informasjonssystemer, for gode råd, bistand og tilbakemeldinger under prosjektet.
- Atle Olsø fra Institutt for datateknologi og informatikk, ved NTNU, for hans rolle som prosjektets veileder.
- Roy Mikal Stokvik fra Kystverket, for hans innsikt og at han tok seg tid til å intervjues.

Dokumentet er utarbeidet etter mal fra institutt for datateknologi og informatikk, ved NTNU.

Trondheim, 20.05.2019

Brage Halse Snarud

Joakim B. W. Solheim

Hallvard Sælthun

Oppgavetekst

Tittel: Operasjonalisering av en maskinlæringsmodell for å detektere avvik og farer for grunnstøtinger i sanntid

Hensikt med oppgaven: Hensikten er å operasjonalisere en maskinlæringsmodell for å teste den under realistiske forhold, og for få svar på om den kan avverge grunnstøtinger. En vellykket operasjonalisering har som hensikt å være en komplementering eller et alternativ for digital overvåking av sjøtrafikk i Norge, ved å tilby tidsnok varsling og beslutningsstøtte til sjøtrafikksentral(VTS)-operatører. Produktet som utvikles vil basere seg på funksjonaliteten maskinlæringsmodellen tilbyr, og skal fungere som en prototype og brukes for å demonstrere hva maskinlæringsmodellen er i stand til å gjøre.

Sammendrag

20 prosent av alle maritime ulykker er laste- og cruiseskip som går på grunn. Kystverket ønsker å redusere antall grunnstøtinger fordi de utgjør en miljø-, og helsemessig risiko. I dag er det VTS-operatører som står for overvåking av sjøtrafikk i Norge. De følger med på trafikken, og ser stort sett etter avvik manuelt. Dette er en prosess som er ressurskrevende, og som ikke er god nok til at alle mulige grunnstøtingssituasjoner blir oppdaget og avverget. Det er flere som har foreslått løsninger på problemet, men det er per dags dato ingen som har laget noe som kan tas i bruk. Norconsult Informasjonssystemer (NoIS) utviklet en maskinlæringsmodell som baserer seg på tidligere seilingsdata. Den kan detektere avvik i sjøtrafikken, og varsle om mulige grunnstøtinger.

Vi har gjennom vårt arbeid laget en produkt som operasjonaliserer NoIS sin maskinlæringsmodell. Vi har brukt produktet til å teste maskinlæringsmodellen i et realistisk miljø, og simulert tidligere grunnstøtinger. For å teste maskinlæringsmodellen har vi lyttet på Kystverket sin åpne AIS-strøm. AIS-meldinger fra strømmen er prosessert av produktet i sanntid, og med forskjellige frekvenser. Simuleringer av tidligere grunnstøtinger er gjort ved å prosessere historiske AIS-data fra fartøy som har gått på grunn.

Våre resultater viser at maskinlæringsmodellen fungerer godt i et realistisk miljø, og håndterer alle meldingene fra Kystverket sin åpne AIS-strøm med god margin. Resultatene fra simuleringene viser at maskinlæringsmodellen klarer å oppdage og varsle om grunnstøtinger i ni av 13 simuleringer, og at varslingen om grunnstøtingen kom tidsnok til at grunnstøtingen kunne vært avverget.

Innholdsfortegnelse

Figur- og tabelliste

Akronymer og forkortelser

AIS: Automatic Identification System/ Automatisk Identifikasjonssystem
ETA: Estimated Time of Arrival / Estimert tid for ankomst
MADART: Machine learning Anomaly Detection with AIS in Real Time
NoIS: Norconsult Informasjonssystemer
SQL: Structured Query Language
SSNN: SafeSeaNet Norway
TLS: Transport Layer Security
VTS: Vessel Traffic Service/Stjøtrafikksentraltjenesten
TCP: Transmission Control Protocol

1. Introduksjon og relevans

1.1 Bakgrunn for prosjektet

Sjøtransport er viktig i Norge, og i en rapport publisert av Kystverket er det spådd at skipstrafikken vil øke med 37 prosent frem mot 2040 [1]. I samme rapport er det samtidig spådd at økningen i sjøtrafikk vil føre til at antall ulykker i norske farvann vil øke med 31 prosent, forutsatt at dagens sjøsikkerhetstiltak videreføres uten endringer [2].

Mellom 2000 og 2014 registrerte Kystverket totalt 2638 maritime ulykker, hvor 1156 av de var knyttet til grunnstøtinger. Av disse 1156 grunnstøtingene, var 552 laste- eller cruiseskip. Dette betyr at 45 prosent av alle grunnstøtinger, og 20 prosent av alle maritime ulykker er laste- eller cruiseskip som går på grunn [3]. Årsaken til de fleste grunnstøtingene som er kartlagt har vært menneskelig feil [1]. Kystverket sin beredskapsdirektør, Johan Marius Ly [4] [5], har ytret at antallet grunnstøtinger er for høyt: “De siste fire årene har gjennomsnittlig hatt 75,5 grunnstøtinger hvert år. Det er for høyt”, og i rapporten *Vurdering av forebyggende sjøsikkerhetstiltak* [6], står det: “Som alternativ eller komplettering kan risikoovervåkingen basere seg på erfarte seilingsmønster”.

Grunnstøtinger kan gi miljømessige konsekvenser, og setter besetningen om bord på fartøyet som grunnstøter i fare. Mellom 2000 og 2014 var det 16 personer som pådro seg fysiske skader som følge av grunnstøtinger som involverte laste- eller cruiseskip [3]. Det er som regel akutte utslipp som gir de største konsekvensene ved grunnstøtinger. Da fartøyet “Full City” gikk på grunn ved Langesund i 2009, førte grunnstøtingen til at olje begynte å lekke fra fartøyet [7]. Totalt lakk 292 tonn olje ut i norsk farvann. Utslippet resulterte til at en kystlinje på 128 km fra Stavern i Vestfold, til Lillesand i Aust-Agder ble forurenset. Det er estimert at oljen tok livet av 2000 sjøfugler. Oppryddingen krevde en samlet innsats fra Kystverket, Forsvaret og frivillige, og kostet omtrent 250 millioner kroner.

Digitalisering av sjøtrafikken åpner for nye måter å overvåke sjøtrafikken på. I dag er det pålagt at alle laste- og passasjerskip med bruttotonnasje 300 eller mer, skal sende AIS[1]-meldinger med jevne mellomrom. Dette betyr at alle store laste- og cruiseskip er koblet til det nasjonale AIS-nettet i Norge. Frekvensen på AIS-meldingene varierer fra hvert tredje minutt, til annethvert sekund for posisjonsoppdateringer og hvert sjette minutt for statisk seilasinformasjon [8]. Ved å sette sammen alle AIS-meldingene for et fartøy i kronologisk rekkefølge er det mulig å kartlegge hvor fartøyet har beveget seg. Det er sjøtrafikksentralene som har oppgaven med å overvåke fartøyene utenfor norskekysten, noe de gjør ved blant annet å ta imot AIS-meldinger og analysere disse. Ved mistanke om feil kurs, eller andre unormale forhold, skal de raskt ta kontakt med fartøyet. Det er omtrent 2300 fartøy langs Norges kystlinje til en hver tid. For å gjøre oppgaven til sjøtrafikksentralene mer oversiktlig, har hver sjøtrafikksentral et bestemt

[1] AIS: Automatisk Identifikasjonssystem. Enkle meldinger som fartøy sender over radiobølger. Inneholder blant annet identifikasjon, posisjon, retning og hastighet til avsender.

område de overvåker. Dette er med unntak av Vardø sjøtrafikksentral, som har ansvaret med å overvåke resten av kysten utenfor grunnlinjen[2]. De som overvåker sjøtrafikken kalles VTS-operatører. Brukermiljøet til VTS-operatører er beskrevet i vedlegg A, *Visjonsdokument*, kapittel 3.3.

I 2015 startet NoIS arbeidet med å finne en løsning for å redusere antall grunnstøtinger, og utviklet en maskinlæringsmodell som senere ble døpt: "Machine learning Anomaly Detection with AIS in Real Time" (MADART). Modellen bruker historiske AIS-data som grunnlag til å lage standardiserte seilingsruter og tilbyr metoder for å beregne avviket fra disse. NoIS har begrenset problemet til laste- og cruiseskip som navigerer langs norskekysten. Den 24. januar 2018, fikk NoIS første bekreftelse på at MADART kunne detektere avvik fra en standardisert rute og varsle om grunnstøtinger, da lasteskipet Havbris gikk på grunn ved Hustadvika [10]. Per 2019 finnes ingen andre løsninger for å detektere avvik i sjøtrafikk i Norge. MADART kan trygge sjøtrafikken rundt Norges kyst, og tilbyr et unikt overvåkningsverktøy av sjøtrafikken.

1.2 Problemstilling

Oppgavestiller ønsket å undersøke hvordan MADART ville håndtere et realistisk miljø. De var ikke sikre på om MADART ville varsle om fartøy som var på vei til å gå på grunn. Et annet usikkerhetsmoment var om den i det hele tatt var egnet til å håndtere så mye trafikk som det er utenfor Norges kyst, og om den hadde tid til å gjøre aktuelle beregninger mens meldingen fremdeles var relevant til situasjonen. I samarbeid med oppgavestiller formulerte vi problemstillingen som følger:

Kan en maskinlæringsmodell som skal predikere grunnstøtinger i sanntid operasjonaliseres, og gi rask nok beslutningsstøtte til en VTS-operatør, slik at grunnstøtinger kan avverges?

For å konkretisere problemstillingen og gjøre den enklere å drøfte, så har vi formulert noen forskningsspørsmål om de essensielle aspektene én og samme løsning må kunne tilby, for å svare på problemstillingen:

1. *Kan en løsning klare å oppdage og varsle om mulige grunnstøtingssituasjoner tidsnok til at aktører får tid til å handle?*
2. *Kan en løsning klare å prosessere alle relevante data fra Kystverket sin åpne AIS-strøm i sanntid?*
3. *Kan en løsning tilby nok beslutningsstøtte for en VTS-operatør ved mulige grunnstøtingssituasjoner?*

Her menes "grunnstøtingssituasjoner", som situasjoner hvor et fartøy ikke følger normale seilingsmønster, og hvor fartøyet må gjøre en endring i kurs og/eller hastighet for å unngå å gå på grunn.

[2] Grunnlinjen er en fiktiv linje som trekkes mellom alle kystens ytterpunkter, og nærliggende øyer [9].

Det å varsle “tidsnok”, betyr her å gi varsling tidsnok til at en VTS-operatør kan nå ut til fartøyet, og fartøyet rekker å utføre en handling for å avverge grunnstøtingen.

Her menes “alle relevante data”, som den dataen fra AIS-strømmen som hjelper oss med å klassifisere og filtrere fartøy.

1.3 Struktur

Rapporten er oppdelt i 8 hovedkapitler. Under er en kort beskrivelse av hvert hovedkapittel

Kapittel 1: Introduksjon og relevans, redegjør for prosjektets formål, problemstilling, forskningsspørsmål og struktur.

Kapittel 2: Teori, redegjør for relevant teori og bakgrunnsinformasjon.

Kapittel 3: Valg av teknologi og metode, tar for seg begrunnelser av valgt teknologi og metode.

Kapittel 4: Resultater, presenterer resultatene fra prosjektet.

Kapittel 5: Diskusjon, drøfter resultatene opp mot problemstilling, forskningsspørsmål og krav til produktet.

Kapittel 6: Konklusjon og videre arbeid, svarer på problemstillingen og forskningsspørsmålene fra kapittel 1.

Kapittel 7: Kilder, viser alle kilder som er brukt.

Kapittel 8: Vedlegg, viser alle vedlegg til rapporten.

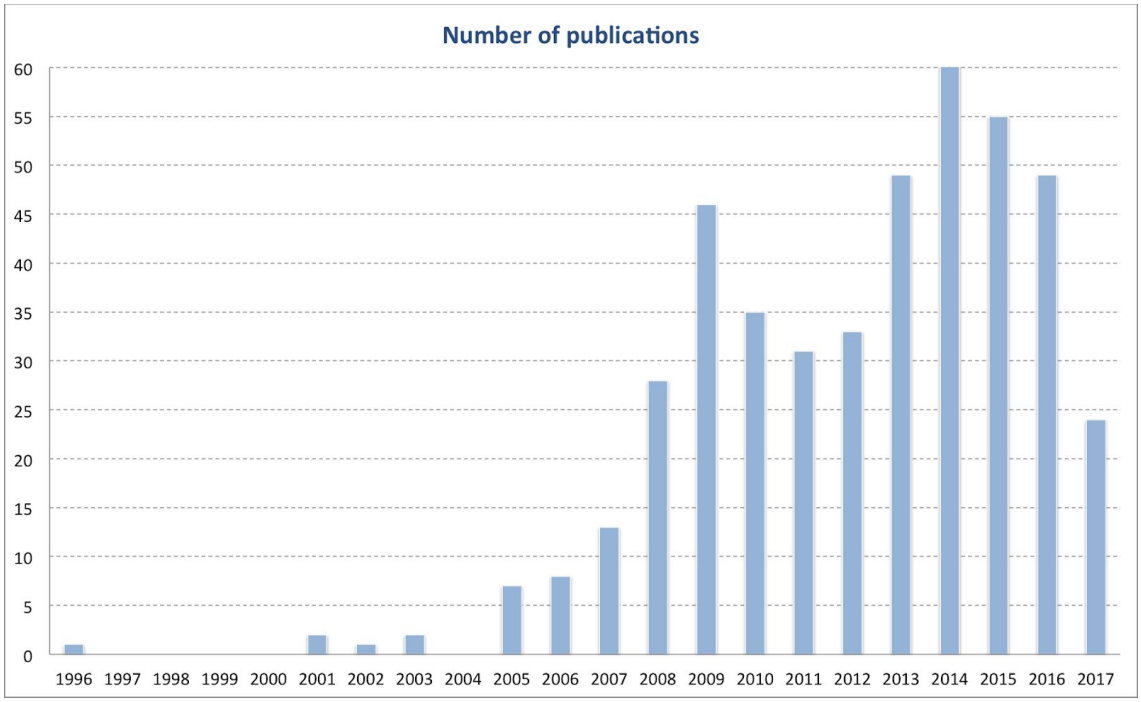
2. Teori

I dette kapittelet skal vi ta for oss hva som har blitt gjort av forskning på fagfeltet fra før, og forskjellige teorier og metoder som har vært essensielle for å løse problemstillingen og utvikle produktet.

2.1 Forskning på feltet

2.1.1 Avviksdeteksjon

Kjernen til problemet med å detektere grunnstøtinger, er å kunne detektere fartøy som avviker fra normale seilingsmønster. Problemet er komplekst, fordi fartøy av forskjellige typer navigerer og oppfører seg på forskjellige måter. I tillegg vil fartøy i trange farvann oppføre seg på en helt annen måte enn når det er utenfor grunnlinjen. Dette gjør det vanskelig å lage en generell løsning. Som figuren under viser, har det vært en enorm økning av vitenskapelige artikler som har undersøkt, og gjort forskning på problemet med å detektere avvik i sjøtrafikken. Det er mange foreslåtte løsninger og noen prototyper, men det har ikke vært et gjennombrudd ennå [11].



Figur 2.1-1: Publikasjoner funnet under søk som omhandler håndtering av maritim avviksdeteksjon og relaterte aspekter, per år [11].

2.1.2 Ekspertvurderinger

Bernhard Engelschiøn, Henrik Kuosmanen og Magnus Torkelsen, er tre studenter ved NTNU i Ålesund som har skrevet en bacheloroppgave i Nautikk. I bacheloroppgaven studerer de grunnstøtinger i Norge, og drøfter i hvilken grad disse kunne vært unngått dersom besetningen om bord hadde blitt gjort oppmerksomme på faren. De simulerte tiden mannskapet om bord på et fartøy trengte for å reagere på varsel om grunnstøting, og fant en maksimal reaksjonstid på tre minutter [12]. Videre analyserte de sjøtrafikken i området hvor fartøyet hadde gått på grunn for å finne punktet hvor det først hadde avvik fra normalt seilingsmønster. De brukte dette punktet for å finne tidspunktet hvor mannskapet om bord ville blitt varslet. Av de 47 grunnstøtingssituasjonene de studerte, mener de at 19 av disse kunne vært avverget [13]. De mente også at fartøyet Roslagen, som er 55 meter langt og 9 meter bredt, med en masse på 600 tonn og som hadde en hastighet på 9 knop, kunne rekke å gjøre en unnvikende manøver for å unngå en grunnstøting senest 400 meter, eller 90 sekunder før den ville gått på grunn.

Vi intervjuet Roy Mikal Stokvik, trafikkleder ved Vardø sjøtrafikksentral. Ifølge Stokvik må en VTS-operatør minimum se informasjon om fartøyets posisjon og hastighet i et detaljert sjøkart, for å kunne vurdere situasjonen til et fartøy [14]. Et tema som ble tatt opp under intervjuet, var at avvik fra vanlig sjøtrafikk ikke er uvanlig, og skyldes som oftest at fartøyet er på vei til spesielle lokasjoner, eller gir plass til møtende trafikk. Om fartøyet er på vei til en spesiell lokasjon, ville fartøyet ha rapportert dette til SSNN. Med informasjon om fartøyets destinasjon er det enklere å se om fartøyet har feilnavigert. I intervjuet ønsket han ikke å gi et estimat på reaksjonstiden til mannskap om bord fartøy, og understreker at reaksjonstiden varierer mye. Hvor mye tid og rom et fartøy trenger for å utføre en unnvikende manøver, ville Stokvik heller ikke gi et generelt estimat på, fordi dette avhenger av for mange faktorer. Fartøyets masse, lengde og dypgang[3] er faktorer som påvirker manøvrerbarheten til et fartøy. Bemanningen om bord, og området fartøyet er i, er noen av mange faktorer som påvirker mannskapets reaksjonstid. Dersom et fartøy har los[4] om bord, eller er i en spesiell VTS-sone, så kan man forvente en kortere responstid fra fartøyet. Store bølger vil også påvirke fartøyets handlingsrom, fordi undervannsskjær blir grunnere.

2.2 Utviklingsrammeverk

2.2.1 Smidig metodikk

Smidig metodikk er en arbeidsmåte for prosjektprosesser utarbeidet for at utviklerteamet skal kunne være fleksible til endringer i løpet av en prosess. Det er vanlig at prototyper utvikles tidlig for å kunne revurdere krav fortløpende. Smidig metodikk anvender en inkrementell

[3] Dypgang er distansen mellom skrogets bunn og vannlinjen.
[4] Person som har til yrke å lede fartøy gjennom vanskelig farvann

fremgangsmåte. Man går syklisk gjennom de forskjellige utviklingsfasene: definering av krav, utvikling og evaluering. Dette gir fleksibilitet i dynamiske utviklingsprosjekt. Manifestet for smidig utvikling legger grunnlaget for essensielle verdier i en smidig prosess: individer og samspill, fungerende kode, kundesamarbeid og handling ved endringer [15].

2.2.2 Scrum

Scrum er et rammeverk utviklet for å administrere prosjekter gjennom en smidig prosess.

Scrum-rammeverket definerer tre hovedroller: utviklerteamet, produkteier og Scrum Master. Utviklerteamet skal være selvstendig, og styre den interne prosessen. Produkteieren skal representere interessentene i prosjektet, og har det overordnede ansvaret for å prioritere oppgaver til utviklerteamet. En Scrum Master skal støtte Scrum-prosessen ved å fungere som en fasilitator og har en mer administrativ rolle mellom utviklerteamet og produkteier [16].

Scrum definerer tre artefakter: Product Backlog, Sprint Backlog og inkrement [16]. Product Backlog er en dynamisk men overordnet liste fra produkteier som definerer arbeidsoppgaver som skal gjennomføres. For å definere hva som skal gjøres i en isolert sprint, så flyttes elementer fra Product Backlog inn i en egen liste, kalt en Sprint Backlog. Ved slutten av hver sprint lages et inkrement. Et inkrement er summen av alle elementene fra Sprint Backlog som er fullført i løpet av sprinten.

For å skape rutine og for å redusere behovet for møter og koordineringer, bruker Scrum fem hendelser: sprint, Sprint Planning, Daily Scrum, Sprint Review og sprint retrospektiv. Alle hendelsene er bestemt før sprinten starter, og etter sprinten har startet er det ikke mulig å endre disse. En sprint er et bestemt tidsintervall hvor et brukbart inkrement utvikles og ferdigstilles. Når en sprint er ferdig starter den neste umiddelbart etterpå. Før hver sprint gjennomfører hele utviklerteamet en Sprint Planning med produkteier. Sprint Planning har til hensikt å bestemme hva som skal gjøres denne sprinten. For hver arbeidsdag utfører utviklerteamet en Daily Scrum, hvor hvert medlem av utviklerteamet forteller hva de har gjort, hva de skal gjøre denne arbeidsdagen og eventuelt deler utfordringer de har. Daily Scrum har til hensikt å koordinere utviklerteamet, og legge en plan for arbeidsdagen. Ved slutten av sprinten avholdes det et Sprint Review, hvor utviklerteamet presenterer hva de har gjort denne sprinten. Produkteier og interessenter får inspisere og evaluere arbeidet, samt gjøre endringer i Product Backlog, kalt Product Backlog Refining [16]. Hensikten med Sprint Review er å presentere inkrementet som er utviklet og få tilbakemeldinger på denne. Sprinten avsluttes med en sprint retrospektiv, hvor utviklerteamet reflekterer tilbake på prosessen under sprinten som var, og diskuterer hva som kan gjøres for å forbedre prosessen til neste sprint.

Utviklerteamet benytter seg som regel av et Scrum Board, for å visualisere og organisere artefaktene, samt for å koordinere arbeid.

Scrum har en rekke fordeler i en systemutviklingssammenheng sammenlignet med mer tradisjonelle utviklingsrammeverk, som Fossefallmetoden. Blant annet er Scrum responsiv til endringer i krav til produktet, sammenlignet med Fossefallmetoden [17].

2.3 Prosjektstruktur og kommunikasjon

2.3.1 Modulbasert arkitektur

Modulbasert arkitektur går ut på å dele større, komplekse systemer inn i mindre moduler. Modulene skal være minst mulig avhengige av hverandre, og hver modul skal ha et klart ansvarsområde [18]. Dette fører til et mer oversiktlig system med redusert kompleksitet. Moduler lar oss gjenbruke kode og funksjonalitet. Eventuelle feil blir isolert uten at de forplanter seg gjennom hele systemet.

2.3.2 Object-relational mapping

Object-relational mapping (ORM) er en teknikk som brukes for å omforme objekter i et programmeringsspråk og entiteter lagret i en relasjonsdatabase [19]. Entitetene i databasen blir kartlagt til de ulike objektene, dette kan bli gjort manuelt eller med Scaffolding. Scaffolding er å generere klasser i et programmeringsspråk fra eksisterende databasetabeller og entiteter. Ved å heller utføre operasjoner og spørringer mot objekter slipper man å forholde seg til syntaksen til databasen. ORM fungerer som et konverteringslag mellom to helt forskjellige lagringsformater og syntakser.

2.3.3 Ekstern AIS-strøm

Kystverket har en åpen AIS-strøm. Denne gir tilgang til AIS-data fra fartøy som er lengre enn 45 meter og innenfor et dekningsområde på 12 nautiske mil fra norskekysten [20]. Meldingene kan leses fra nettjeneren i tekstformat, men de er kodet etter National Marine Electronics Association sin standard.

2.3.4 WebSocket

WebSocket er en kommunikasjonsprotokoll som tillater full dupleks kommunikasjon over internett, altså kommunikasjon i begge retninger. WebSocket oppretter en vedvarende Transmission Control Protocol (TCP) forbindelse [21]. Etter at en forbindelse er opprettet kan tjeneren å sende data til klienten uten å måtte be om en forespørsel først.

2.4 Programmeringsteknikker og designmønstre

2.4.1 Singleton

Singleton-mønsteret går ut på at vi har kun ett objekt av en klasse eller entitet [22]. Dette objektet skal ha et overordnet mål og leve like lenge som programmet kjører. Objektet skal kunne nå ut til, eller nås av mer kortlevde deler av programmet.

2.4.2 Inversion of Control

Inversion of Control er et prinsipp som går ut på å invertere kontroller-klasser i et objektorientert design, slik at man oppnår løse koblinger mellom disse [23].

2.4.3 Dependency Injection

Dependency Injection er en programmeringsteknikk hvor klasser og objekter som har avhengigheter får disse injisert [24]. Dette fører til løsere koblinger mellom objekter som avhenger av hverandre, som gjør det enklere å teste, refaktorere og vedlikeholde. Dependency Injection er nært tilknyttet Inversion of Control.

2.4.4 Repository

Repository er et designmønster som har som formål å abstrahere data [25]. Repositories innkapsler logikk og oppkobling som går opp mot datakilder som for eksempel databaser. Dette lar oss skille datakildene fra resten av systemet.

2.4.5 Parallellprosessering

Parallellprosessering er en benevning på teknikker for å uttrykke potensiell parallellisering, og for å løse synkroniserings- og kommunikasjonsproblemer som resulterer av parallellisering [26]. En teknikk for å oppnå parallellprosessering er å få prosessen til å lage tråder som jobber parallelt på samme maskin.

2.4.6 Separation of Concerns

Separation of Concerns er et designmønster og har som formål å fordele ansvaret til et program i mindre deler [27]. Delene skal ha hvert sitt ansvarsområde å bekymre seg om, og være godt definerte. Separation of Concerns gjør det enklere å ha oversikt, gjøre endringer, vedlikeholde og teste koden.

2.4.7 Kontinuerlig integrasjon og utrulling

Kontinuerlig integrasjon er en teknikk som går ut på å koble et sentralisert prosjekt opp mot en isolert maskin, som vil bygge og teste prosjektet når det lagres endringer [28]. Dersom bygging eller tester feiler kan medlemmer automatisk varsles. Dersom både bygging og testing gjennomføres uten feil, er det mulig å automatisk rulle ut programvaren til en skybasert tjeneste.

2.5 Maskinlæring

2.5.1 Formål med maskinlæring

Maskinlæring er en teknikk hvor programvare bruker algoritmer til å ta beslutninger basert på tidligere innhentet data. Den formelle definisjonen til maskinlæring er: “Et studieområde som gir datamaskinen evnen til å lære uten å eksplisitt bli programmert” [29].

Maskinlæring har mange praktiske bruksområder, blant annet mønstergjenkjenning. Dette gir et program muligheten til å analysere og gjenkjenne komplekse mønster på egenhånd. Dette er funksjonalitet som er egnet til å løse problemet med å gjenkjenne et fartøys seilingsmønstre.

2.5.2 Algoritmer og teknikker

2.5.2.1 Datasett

For å kunne utvikle en god maskinlæringsmodell er det viktig å ha et datasett med tilstrekkelig kvalitet og kvantum som kan brukes for å trene opp modellen. Da er det viktig å hente ut relevante data fra et større sett. Treningsdata må kobles opp mot en hensikt du vil at modellen skal ha. Data er ofte full av støy og feilkilder som vi ikke vil påvirke algoritmen med, derfor er datavasking viktig. Datavasking er fjerning av data og variabler som ikke er relevante til programmets hensikt [30].

2.5.2.2 K-Nærmeste Naboer

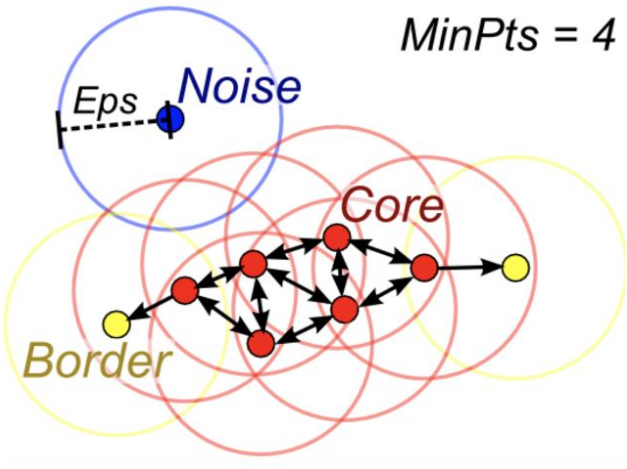
K-Nærmeste Naboer (KNN) er en algoritme som grupperer datapunkter på et n-dimensjonalt plan [31]. For å klassifisere et nytt punkt regner man ut avstand mellom det gitte punktet og de tidligere datapunktene. Punktet klassifiseres ut fra de k nærmeste datapunktene. For å beregne avstand kan man bruke forskjellige typer metoder, den mest vanlige er euklidsk avstand.

2.5.2.3 Hierarchical Density-Based Spatial Clustering of Applications with Noise

For å gi en maskin mulighet til å analysere store datasett er det viktig å sammenligne og gruppere deler av dataen for å finne meningsfull struktur. Når man snakker om dette, brukes ofte det engelske begrepet “clustering”, eller gruppering. En grupperingsalgoritme er en algoritme som prøver å gruppere datapunkter i et datasett automatisk i egne grupper ut ifra

fellestrekk. Density-Based Spatial Clustering of Applications with Noise (DBSCAN) er en mye brukt grupperingsalgoritme.

DBSCAN velger ut tilfeldig punkter som startverdi, og for et startpunkt greiner den seg utover i dimensjonen og sjekker om de nærmeste punktene er innenfor en viss avstand. De som er nærme nok får samme gruppering. Figuren under visualiserer ni punkter, der åtte blir klassifisert, mens ett punkt er utenfor rekkevidde.

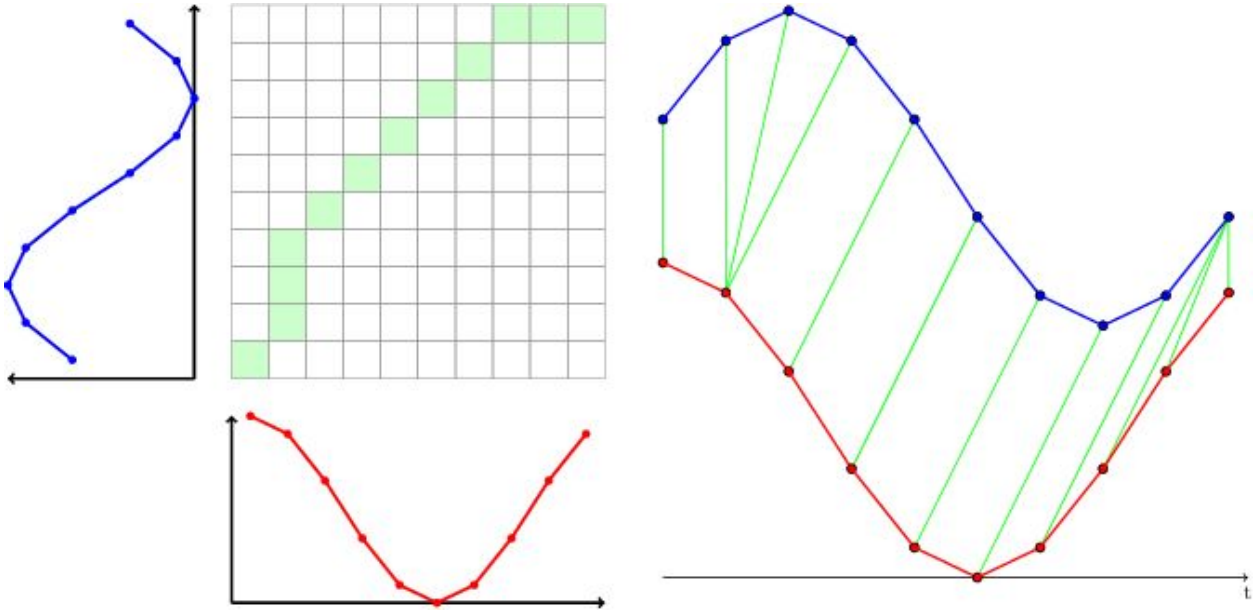


Figur 2.5-1: Visualisering av åtte punkter klassifisert av DBSCAN [32].

Hierarchical DBSCAN (HDBSCAN) er en hierarkisk utvidelse av DBSCAN, dette gjør den mer robust mot varierende tetthet i dataen enn DBSCAN [33].

2.5.2.4 Dynamic Time Warping

Dynamic Time Warping (DTW) er en algoritme for å finne beste match mellom punktene i to tidsserier [34]. DTW går ut på å minimere den akkumulerte avstanden mellom to tidsserier. Dette betyr at DTW finner matchen mellom punktene i tidsseriene, som gjør at avstanden mellom tidsseriene blir minst. Å sammenligne punktene på tvers av tidsaksen gjør at DTW fungerer bra på tidsserier som er forskjøvet i tid i forhold til hverandre.



Figur 2.5-2: Visualisering av DTW mellom to tidsserier, med matrise og linjer for å vise hvilke punkter som blir matchet.

Måten DTW sammenligner to tidslinjer, *t1* og *t2* på er ved å først lage en matrise *D* med form *L1 x L2,* der L1 og L2 står for antall punkter hver tidslinje henholdsvis har. Matrisen fyller vi opp med den individuelle avstanden mellom punktene. Deretter summerer vi elementene i D. Vi begynner i element $D_{1,1}$ og jobber oss ut. For hvert element $D_{i,j}$ plusser vi på den minste verdien av de tre mest nærliggende forrige naboene; $D_{i-1,j}$, $D_{i-1,j-1}$, eller $D_{i,j-1}$. Etter dette trinnet kan vi finne beste vei. Vi begynner i $D_{L1,L2}$ og jobber oss mot $D_{1,1}$ ved å alltid velge det naboelementet med lavest verdi. Dette gir oss en vei som tilegner punkter fra tidsseriene opp mot hverandre. Veien vi velger er den veien med minst akkumulert distanse mellom *t1* og *t2*. Dette er en allsidig metode for å identifisere tidsserier som har likheter, samtidig som vi også kan lage kombinasjoner av to tidsserier.

2.5.2.5 Gradient Boosting - CatBoost

Gradient Boosting er et begrep innenfor maskinlæring som går ut på å kombinere og legge sammen enkle beslutningstrær som individuelt har lav kompleksitet, men høy variasjon. Når man kombinerer de får man et resultat som vil prøve å redusere feilene fra de individuelle trærne [35]. Man går iterativt gjennom problemet og kombinerer flere og flere enkle beslutningstrær for å få en kompleks løsning som egentlig består av mange enkle løsninger [36].

CatBoost er en variasjon av Gradient Boosting som har flere sterke sider. CatBoost produserer bra resultat, på en effektiv måte, man slipper mye preprosessering og man kan bruke GPU for å trene opp modellen [37].

2.6 MADART

MADART er en maskinlæringsmodell utviklet av NoIS. MADART tilbyr klassifisering av seilaser, avviksdeteksjon fra klassifiseringer, prediksjon av grunnstøting og anslått ankomsttid for seilaser.

2.6.1 Formålet med MADART

Som nevnt i kapittel 1.1 og i vedlegg A, *Visjonsdokument*, så forekommer det et betraktelig antall grunnstøtinger rundt Norges kyst, selv om alle kystpunkter og skjær er kartlagt. Det har alltid vært interesse for å oppdage og hindre grunnstøtinger, men en stor utfordring var mengden støy. Det finnes mange fartøy rundt Norges kyst til hver enkelt tid, og mange vil bevege seg i farlig farvann uten at det er en reell risiko.

NoIS har betraktelig erfaring innenfor maskinlæring, og de har en teori om at det kunne være mulig å gruppere seilas. Disse grupperingene vil kunne brukes som utgangspunkt for å klassifisere et fartøy sitt seilingsmønster, samt beregne fartøyets avvik fra disse.

2.6.2 Datagrunnlag

NoIS fikk tilgang til all AIS-historikk utenfor norskekysten i perioden 2014 - 2017, noe som tilsvarte omtrent 3 TB med data. AIS-dataen var veldig kaotisk, og preget av støy og feil. For å kunne filtrere ut støy og gi dataen mer struktur og hensikt, ble dataen delt inn i komplette seilaser som de kartla opp mot rapporterte seilaser fra SafeSeaNet Norway (SSNN), som vist i figuren under.

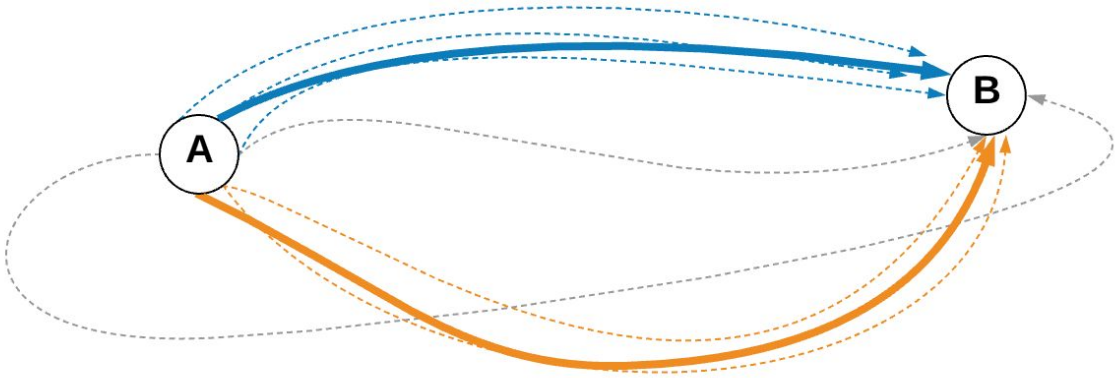
			Departure		Arrival	
Created	Ship Name	Call Sign	Location	ETD	Location	ETA
22.03.2017 00:21	AZURA	ZCEE2	Stavanger	11.09.2017 16:15	Flam	12.09.2017 07:15
14.07.2017 10:42	KARIN LEHMANN	V2BC9	Mantyluoto	28.08.2017 22:30	Mo i Rana	03.09.2017 00:40
14.07.2017 10:44	KARIN LEHMANN	V2BC9	Mo i Rana	08.09.2017 18:40	Skuru (Pohjankuru)	11.09.2017 10:30
27.07.2017 10:37	ANANGEL FORTUNE	SXVD	Narvik	29.07.2017 10:35	El Dekheila	30.08.2017 04:00
04.08.2017 07:01	ARCTIC AURORA	9HA3589	Melkoya	08.08.2017 04:00	Jordan	07.09.2017 21:31
11.08.2017 09:37	SAMJOHN SOLIDARITY	V7ZX5	Narvik	13.08.2017 02:00	Al Jubayl Industrial City	02.09.2017 12:00
11.08.2017 10:51	ARMIA KRAJOWA	D5JD7	Heroya	19.08.2017 15:30	Brazil	29.08.2017 00:02
12.08.2017 19:14	EEMS SKY	PHJF	Sunndalsora	16.08.2017 15:50	Gaeta	02.09.2017 22:00
14.08.2017 11:07	BW HAWK	9V2920	Hammerfest	18.08.2017 07:00	Houston	08.09.2017 21:00
15.08.2017 18:15	ENERGY PROGRESS	2ATN7	Mongstad	16.08.2017 12:00	New York	29.08.2017 23:15

mmsi	date_time_utc	lon	lat	sog	cog
1	2016-01-02 21:40	10.3887	59.2667	0.1	178.7
4747	2016-01-02 00:04	10.4155	59.8328	0	100.4
4747	2016-01-02 00:07	10.4155	59.8328	0	100.4
4747	2016-01-02 00:10	10.4155	59.8328	0	100.4
4747	2016-01-02 00:13	10.4155	59.8328	0	100.4
4747	2016-01-02 00:16	10.4156	59.8328	0	100.4
4747	2016-01-02 00:19	10.4156	59.8329	0	100.4
4747	2016-01-02 00:22	10.4155	59.8328	0	100.4
4747	2016-01-02 00:25	10.4155	59.8328	0	95.6
4747	2016-01-02 00:28	10.4155	59.8328	0	93.2
4747	2016-01-02 00:31	10.4155	59.8328	0	93.2
4747	2016-01-02 00:34	10.4155	59.8328	0	93.2
4747	2016-01-02 00:37	10.4155	59.8328	0	93.2
4747	2016-01-02 00:40	10.4155	59.8328	0	98.4
4747	2016-01-02 00:46	10.4155	59.8328	0	96.1
4747	2016-01-02 00:49	10.4155	59.8328	0	96.1
4747	2016-01-02 00:52	10.4155	59.8328	0	96.1
4747	2016-01-02 00:55	10.4155	59.8328	0	96.1
4747	2016-01-02 00:58	10.4155	59.8328	0	96.1

Figur 2.6-1: Gruppering av AIS-meldinger til seilas.

Dette førte til en samling av diverse antall seilaser koblet opp mot alle kombinasjoner av avgangs- og ankomsthavner i individuelle filer. Hver fil inneholdt da alle seilasene gjennomført mellom to havner. Dette la datagrunnlaget for det MADART skulle trenes opp mot.

2.6.3 Prosessering - modelltrening

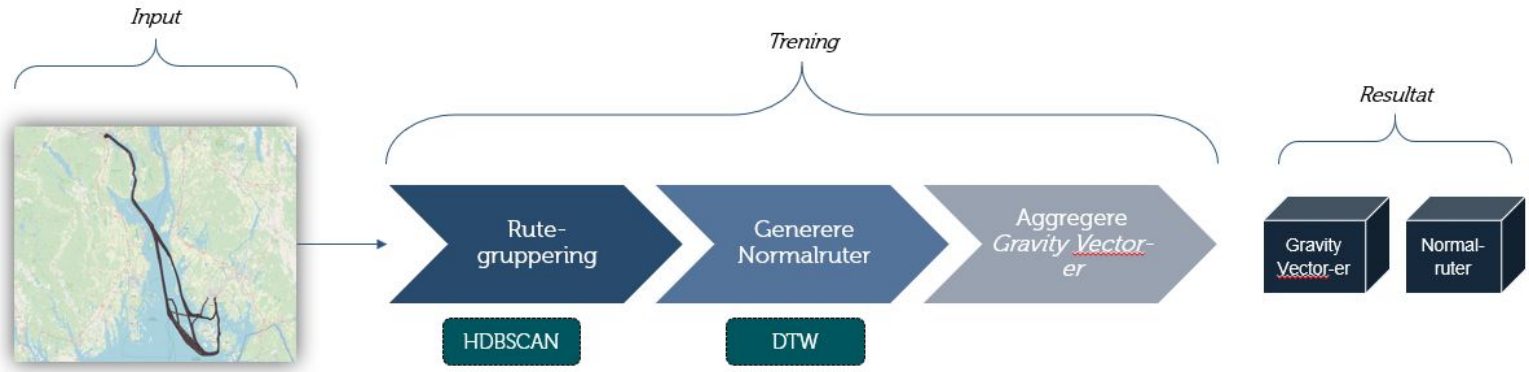


Figur 2.6-2: Generering av normalruter fra havn A til havn B.

Figuren over illustrerer konstruering av normalruter basert på tidligere seilas. Hver fil med seilaser mellom to havner ble behandlet ved å først utføre HDBSCAN på seilasene. De tidligere seilasene er visualisert som stiplede linjer. Seilaser med fellestrekk grupperes til samme gruppe

(farge) av HDBSCAN, mens seilaser som passer dårlig blir deklarert som støy (grå farge). Deretter genereres en normalrute fra hver gruppe ved hjelp av DTW (tykk, sammenhengende linje).

For hver normalrute genereres et sett av Gravity Vector-er. Gravity Vector-er er en aggregert oppsummering av seilasene som er koblet opp mot en normalrute. Disse inneholder informasjon om blant annet gjennomsnittsfart, standard- og maksimalt avvik i avstand, gjennomsnittstid til havn og lignende. Figuren under illustrerer alle trinnene i treningsprosessen til MADART.



Figur 2.6-3: Treningsprosessen til MADART - trinnvis.

2.6.4 Klassifisering og predikering

Etter at treningen er gjennomført; ruter og vektorer er ferdig generert, kan man bruke de til å sammenligne nye seilas. Først brukes KNN for å finne de nærmeste normalrutene, der man bruker DTW-vekten som rangering. Dette er en nokså naiv filtrering, og brukes som et grovfilter. Deretter anvendes en CatBoost-rangering som bruker flere variabler (antall seilaser i normalruten, ekstra vekt på siste posisjon og lignende) som en finere filtrering og rangering.

Det å sammenligne og tilegne en pågående seilas til et fartøy mot en normalrute er det vi definerer som klassifisering av fartøyet. Etter at et fartøy har blitt klassifisert kan man beregne avvik på nye posisjoner, da sammenligner man verdiene for fart, kurs og posisjon opp mot nærmeste Gravity Vector til den aktuelle normalruten.

MADART har også mulighet for å sjekke om et fartøy er på vei mot grunne. Dette er en nokså enkel beregning uavhengig av klassifisering. Metoden tar kun for seg hastighet og posisjon til fartøyet. Metoden henter kystkontur fra en ekstern kilde, og ser om det er et kystpunkt eller skjær foran fartøyet.

3. Valg av teknologi og metode

3.1 Valg av back-end rammeverk

3.1.1 Azure

Azure er en plattform som inneholder en gruppe av ulike skytjenester laget av Microsoft. Azure fokuserer på tjenester som tilbyr kontinuerlig integrasjon, utrulling og drifting av applikasjoner i skyen [38]. Det er mulig å sette Azure opp slik at den vil bygge en git-branch. Hver gang git-branch-en oppdateres vil løsningen bli bygget, testet og rullet ut. I Azure betaler man for prosesseringskraften og lagringsplassen man bruker.

Tidlig i utviklingen av produktet var det et ønske om å ha kontinuerlig integrasjon og utrulling av produktet. Etter kort tid, innså produkteier at det var utfordrende å ha kontinuerlig integrasjon og utrulling, fordi MADART ville bruke så mye prosesseringskraft, at det ikke ville være økonomisk hensiktsmessig å ha den i skyen. Sammen med produkteier skisserte vi noen alternativer for å jobbe oss rundt problemet, men vi konkluderte med at dette ikke var nødvendig egenskap for å svare på problemstillingen. Kontinuerlig integrasjon ble derfor fjernet fra visjonsdokumentet.

3.1.2 ASP.NET Core

ASP.NET Core er et open source rammeverk som bygger på .NET Core for utvikling av nettapplikasjoner. Rammeverket .NET Core er en gratis utviklingsplattform utgitt av Microsoft [39]. Plattformen støtter de største operativsystemene, Windows, macOS og Linux, og blir brukt til å lage ulike typer applikasjoner. NuGet er pakkebehandleren for .NET rammeverket, og gjør det mulig å laste opp eller å bruke eksterne pakker som andre utviklere har laget, på en oversiktlig måte.

ASP.NET Core ble valgt som rammeverk på grunn av støtten til de ulike plattformene. Oppgavestiller mente at dette var en fordel, fordi det var uvisst hvilket miljø produktet skulle kjøre i tiden fremover. ASP.NET Core har også god integrering mot Azure, som vi tenkte å bruke for å rulle ut applikasjonen. Det er lett å bruke eksterne biblioteker ved hjelp av NuGet i .NET Core. Vi mente dette var en fordel ettersom vi visste vi måtte bruke eksterne biblioteker i prosjektet.

ASP.NET Core er lagt til rette for enkel og effektiv implementasjon av Inversion of Control og Dependency Injection med egne støtteklasser og metoder. Dette ville gi mer eksplisitt kontroll over levetiden til objekter, som var et stort pluss da vi tidlig visste at Singleton-mønsteret kom til å være relevant for lytting til den eksterne AIS-strømmen. I tillegg tilbyr ASP.NET Core støtte for oppretting og håndtering av tråder. Dette var et viktig element for oss, da parallellisering var sentralt for sanntidsaspektet i vår problemstilling.

3.1.3 SignalR

Vi valgte å bruke SignalR for kommunikasjon mellom tjener og klient. ASP.NET Core sitt SignalR bibliotek tilbyr funksjonalitet for å forenkle kommunikasjon mellom distribuerte komponenter. Et hovedfokus for SignalR er sanntidskommunikasjon mellom en tjener og flere klienter. SignalR implementerer blant annet WebSocket-protokollen for å garantere rask og trygg kommunikasjon, men vil falle tilbake til eldre protokoller om det er nødvendig [40]. SignalR tillater 5000 TCP-tilkoblinger per CPU som standard [41], men denne grensen kan utvides. Vi valgte å bruke SignalR for å løse problemet med å oppdatere flere klienter samtidig, slik at det er mulig å distribuere informasjon om fartøy fortløpende.

3.1.4 Entity Framework Core

Entity Framework Core (EF Core) er en teknologi for å aksessere data, laget for applikasjoner basert på .NET Core rammeverket [42]. EF Core er et ORM-rammeverk som er laget med hensikt for å skalere godt og for å støtte flere typer relasjonsdatabaser.

Entity Framework Core ble valgt som ORM fordi det er et kjent og godt dokumentert ORM-rammeverk for .NET Core applikasjoner, som også har pakker som støtter geografiske data. I tillegg støtter EF Core Scaffolding for å opprette klasser, som i sammenheng med å anvende Repositories-mønsteret lar oss holde datatilgangen adskilt fra resten av logikken. Dette gjorde det lettere å ha en dynamisk databasestruktur gjennom utviklingsprosessen.

3.2 Valg av database

PostgreSQL er et objekt-relasjonsdatabasesystem som bruker og utvider Structured Query Language (SQL) [43].

Produktet som skulle utvikles har behov for å lagre mye geografisk data, og det var derfor viktig for ytelsen til programmet at databasen kunne håndtere dette på en effektiv måte. Ved oppstart av utviklingen anbefalte oppgavestiller PostgreSQL som database, fordi de hadde erfart at PostgreSQL var et bedre databasesystem enn andre relasjonsdatabaser for å håndtere geografiske data.

3.3 Front-end

3.3.1 Leaflet

Leaflet er et open source javascript-bibliotek som blir brukt til å implementere kart i nettjenester [44]. Leaflet gjør det lett for utviklere å fremstille dynamiske kart, samt legge til ulike markører og figurer.

Under oppstart av utviklingen anbefalte oppgavestiller Leaflet for å integrere kart i klienten. Bakgrunnen for anbefalingen var god erfaring med biblioteket fra før, funksjonalitet som gjør det enkelt å komme i gang og muligheten til å bruke egendefinerte markører.

3.3.2 Bootstrap

Bootstrap er et populært rammeverk for utforming av nettapplikasjoner. Med Bootstrap er det lettvint å ta i bruk ferdigdefinerte komponenter i HTML og CSS, og definere dynamisk utforming av disse [45].

Bootstrap er et rammeverk vi hadde god erfaring med fra før av, og som vi mente hadde tilstrekkelig med funksjonalitet for prosjektets omfang.

3.4 Arkitektur

ASP.NET Core har god støtte for å bruke Model-View-Controller-strukturen som er gunstig å bruke om programmet har mange nettsider som klienten skal navigere seg gjennom. Vårt produkt var tenkt som en enkeltside-applikasjon, der forbindelser med klienter ble opprettet over WebSocket-protokollen. Dette gjorde det lite hensiktsmessig å bruke MVC-strukturen. Vi anvendte derfor en mer generell struktur som fokuserer på modulbasert arkitektur. Produktet ble delt opp i mindre moduler som hadde hvert sitt hovedansvar. For eksempel hadde en modul ansvar for database-kommunikasjon, mens en annen hadde ansvar for kommunikasjon mot MADART som lå på en ekstern tjener. Slik ville vi oppfylle prinsippet om Separation of Concerns.

3.5 Prosess

3.5.1 Scrum

Før vi startet å utvikle ble det fort klart at produkteier ikke hadde noen klare krav til produktet, men hadde i stedet overordnede visjoner og intensjoner om hva produktet skulle være. Som beskrevet i kapittel 2.2.2, så er Scrum godt egnet i situasjoner hvor man forventer at endringer vil dukke opp underveis i utviklingen. I kombinasjon med at produkteier ønsket en smidig metodikk for utviklingen, og at vi følte oss kompetente i Scrum, så ble Scrum et naturlig valg som utviklingsrammeverk. For å holde oversikt over Sprint Backlog og Product Backlog brukte vi Trello[5], som Scrum Board.

Vi har valgt å følge Scrum-metodikken med unntak av rollen Scrum Master. Denne avgjørelsen er tatt på bakgrunn av at teamet og prosjektet sin størrelse var så liten, at vi ikke så det som

[5] Trello er en nettjeneste, som brukes til visualisering og planlegging.

nødvendig å ha en Scrum Master til å organisere og administrere. Oppgaver som normalt faller på en Scrum Master, ble fordelt på innad i utviklerteamet.

3.5.2 Rollefordeling

På grunn av størrelsen på utviklerteamet, så var vi enige om at vi skulle ha en flat struktur. Vi valgte heller ikke å fordele produktet inn i ansvarsområder, men hadde heller som målsetning at alle skulle jobbe litt på alt. Tanken bak dette var at dersom alle kunne litt om alt, var det mulig å kraftsamle seg rundt et problem, og bytte arbeidsoppgaver om man skulle gå lei.

4. Resultater

4.1 Vitenskapelige resultater

Her vil vi vise resultater som tar for seg:

- Simuleringer fra tidligere grunnstøtingsulykker.
- Stresstester av produktet.
- Informasjon produktet tilbyr for å gi beslutningsstøtte.

4.1.1 Oppdage mulige grunnstøtinger tidsnok

For å teste om produktet kan brukes som et verktøy for å avverge grunnstøtinger, så var det nødvendig å simulere tidligere grunnstøtinger.

Dataen brukt i simuleringene er historiske AIS-data levert av Kystverket. For å produsere resultatene, har vi brukt en modifisert versjon av produktet, hvor inputen er en Comma-Separated Values[6] (csv)-fil. Csv-filen har et tidsstempel som sier når meldingen ble sendt. Antallet sendte meldinger varierer for hver csv-fil, og baserer seg på AIS-meldingene fartøyet sendte.

En simulering ble gjort ved å kjøre den nødvendige mengden AIS-meldinger fra csv-filen gjennom produktet, for å klassifisere fartøyet i forkant av grunnstøtingen.

Figuren under viser resultatene av de simulerte grunnstøtingene. Dersom en celle ikke har fått fylt inn en verdi betyr det at begivenheten ikke forekom. I cellene hvor det står “ikke aktuelt”, har produktet sagt fra om grunnstøtingsvarsling på tidligere punkter før fartøyet gikk på grunn. Estimert tid og faktisk tid har derfor ikke noen sammenheng.

[6] Comma-Separated Values, er en måte å formatere en fil på.

Navn på Fartøy	Tidspunkt for grunnstøting	Tidspunkt for første varsel om grovt avvik	Første varsel om grunnstøting	Estimert tid til grunnstøting	Faktisk tid til grunnstøting
Tonny (1)	13:58:30	13:32:31	13:40:11	387 sekunder	*Ikke aktuelt*
Scan Master	23:36:14	23:31:25	23:31:25	260 sekunder	289 sekunder
Selvaagsund	01:48:06	01:38:18	01:41:58	389 sekunder	368 sekunder
Bjugnholm	04:04:40	01:01:04	02:53:20	312 sekunder	*Ikke aktuelt*
Optimar	04:18:30	04:06:13	04:12:32	366 sekunder	358 sekunder
Tonny (2)	11:47:01	11:38:30	11:39:59	420 sekunder	422 sekunder
Vitin	02:12:40	01:51:00	02:05:50	309 sekunder	410 sekunder
Samba	18:46:25	18:35:38	18:40:58	309 sekunder	327 sekunder
Akvaprins	02:23:54	02:05:34	02:19:16	190 sekunder	278 sekunder
Frydholm	02:30:15	02:26:16	02:26:16	235 sekunder	239 sekunder
Wilson Lista	17.24.00	13.30:32	*varsler ikke*		
Ronia Diamond	03:37:04	03:21:34	*varsler ikke*		
Havbris	05:30:12	*ikke grovt avvik*			

Figur 4.1-1: Tabell med resultat fra simuleringer.

Vi ønsker å utdype og gi mer innsikt om de fire simuleringene, med kortest, eller manglende "Faktisk tid til grunnstøting". Fartøyet "Frydholm" var den simuleringen hvor MADART varsler om fare for grunnstøting senest, og går på et skjær 239 sekunder senere. "Wilson Lista" og "Ronia Diamond" var begge fartøy som gikk på grunn på undervannsskjær. For begge tilfellene identifiserte MADART at fartøyene hadde grove avvik fra sine normalruter, men varslet ikke om fartøyene var i fare for å gå på grunn. Fartøyet "Havbris" var det eneste fartøyet som gikk på grunn uten at MADART registrerte stort nok avvik til at produktet begynte å lete etter farer for grunnstøtinger. For å se resultatene fra alle simuleringene, samt mer detaljer om simuleringene og figurer som viser hva brukeren ser, så finnes dette i vedlegg F, *Simuleringer av utvalgte grunnstøtinger*.

4.1.2 Evne til å håndtere meldinger i sanntid

En av hovedutfordringene med problemstillingen var at løsningen skulle holde tritt med Kystverket sin åpne AIS-strøm. Etter å ha observert antall meldinger Kystverket sin strøm sender fant vi at dette som oftest lå på ca. 1500 meldinger i minuttet, og det høyeste gjennomsnittet vi målte var på 1900 meldinger i minuttet.

For å teste produktets kapabilitet gjennomførte vi stresstester. Produktet prosesserte AIS-data fra fil, og vi justerte hyppigheten av innkommende meldinger til vi fant et punkt der produktet ble hengende etter. Tjeneren MADART kjører på er en maskin som har 24 kjerner til rådighet.

Test nr.	Hyppighet på linjer, antall meldinger per minutt	Kjøretid i minutter	Status etter kjøretid i minutter
1	ca. 1850	70	128300/130265 meldinger prosessert, (98,49%)
2	ca. 3800	30	111000/112573 meldinger prosessert, 98,6%
3	ca. 5600	30	164900/168735 meldinger prosessert (97,73%)
4	ca. 7500	20	149100/ 152511 meldinger prosessert (97,76%)
5	ca. 10 000	60	634600/649285 meldinger prosessert (97,74%)
6	ca. 30 000	45	1281900/1408078 meldinger prosessert (91,04%)

Figur 4.1-2: Tabell med oversikt over stresstester gjort mot produktet.

Merknader til testen: Produktet vil ikke nå 100 prosent prosessert. Dette er på grunn av at meldinger som har kommet inn og blitt telt ikke har blitt prosessert ferdig i det målingen blir gjort.

Merknad til test 5: Her når vi grensen på hva vår klient takler. Visse fartøy får ikke gamle varsler fjernet, og det samler seg opp sektorer uten fartøy. Det ble forkastet 5400 meldinger på grunn av thread overflow[7] Dette tilsvarer rundt 1% av innkommende meldinger.

Merknad til test 6: Vi må forkaste opp mot 9% av meldingene på grunn av thread overflow. Klienten sliter med å prosessere alle meldingene som sendes fra tjeneren. MADART rekker ikke alltid å prosessere en melding før neste melding kommer inn.

En annen faktor i forskningsspørsmålet er hvor lang tid det tar fra vi mottar en melding, til den er ferdig prosessert og kringkastes til klientene. Meldingene som kommer til å ta lengst tid er de som går igjennom avviksdeteksjon → reklassifisering → ny avviksdeteksjon → sjekk for fare for grunnstøting. Vi tok tiden fra tjeneren mottok meldingen, til meldingen var sjekket for grunnstøting, og målte gjennomsnittstiden etter å ha kjørt programmet i forskjellige tidsintervaller.

Test nr.	Varighet i minutter	Antall meldinger nådd sjekk for fare for grunnstøting	Gjennomsnitt tid brukt på prosessering
1	3381	317 610	3,016 sekunder
2	104	4 350	1,34 sekunder
3	647	33 000	1,341 sekunder
Sum	4132	354960	2,969 sekunder

Figur 4.1-3: Tabell med oversikt over prosesseringstid brukt på de mest krevende meldingene systemet må håndtere.

4.1.3 Beslutningstøtte

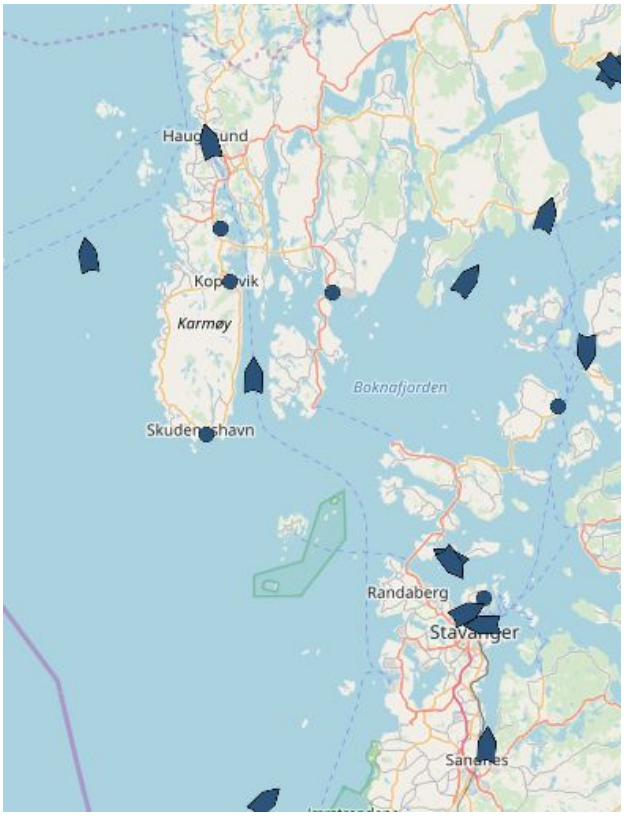
Ut ifra kapittel 2.1.2, *Ekspertvurderinger,* har vi delt opp nødvendig beslutningsstøtte i fire deler.

- Posisjon og retning for fartøy
- Informasjon om fartøy
- Varsel for grunnstøting med informasjon
- Sjøkart

[7] thread overflow: vårt program krever flere tråder enn det tjeneren som MADART kjører på kan håndtere

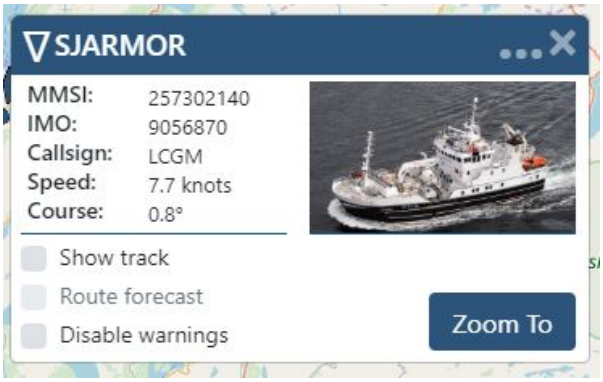
4.1.3.1 Posisjon og retning for fartøy

Som vist i figuren til høyre, viser produktet posisjon og retning til fartøy fra en AIS-strøm i sanntid. Posisjonen til alle fartøy vises ved at de plottes i et kart. Hvordan fartøyet plottes avhenger av om fartøyet er i bevegelse eller ikke. Om et fartøy er i bevegelse vil det vises som et ikon som kan minne om et skip sett ovenfra. Om fartøyet står i ro vises det som et punkt. Retningen til fartøyet vises ved at ikonet roteres tilsvarende rotasjonen som er oppgitt i AIS-meldingen. For en bruker vil et fartøy som har en rotasjon lik 0 grader vises på kartet som at baugen til fartøyet peker rett opp i kartet (mot nord), og en rotasjon lik 90 grader viser at baugen peker rett til høyre i kartet (mot øst). Rotasjonen og posisjonen oppdateres når programmet får en ny AIS-melding fra samme fartøy.



Figur 4.1-4: Skjermbilde som viser hvordan fartøyets posisjon og retning vises.

4.1.3.2 Informasjon om fartøy

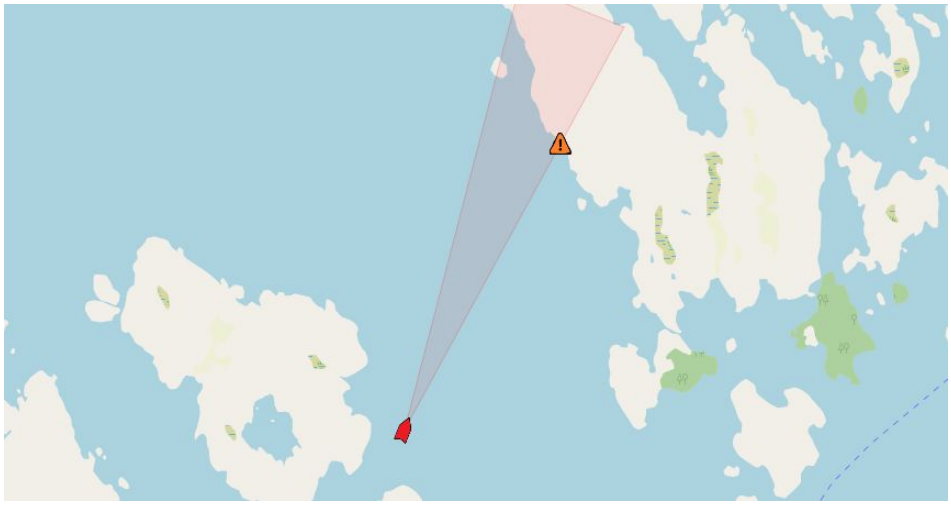


Figur 4.1-5: Skjermbilde som viser hvordan fartøysinformasjon vises.

Detaljert informasjon om fartøyet vises ved å trykke på ikonet for det aktuelle fartøyet. Da vil en modal som viser informasjon om fartøyet åpnes. Som vist i figuren til venstre, viser modalen navn, MMSI, IMO, kjenningssignal, hastighet, retning og et bilde av fartøyet. Hastighet og retning oppdateres når programmet får en ny AIS-melding fra fartøyet.

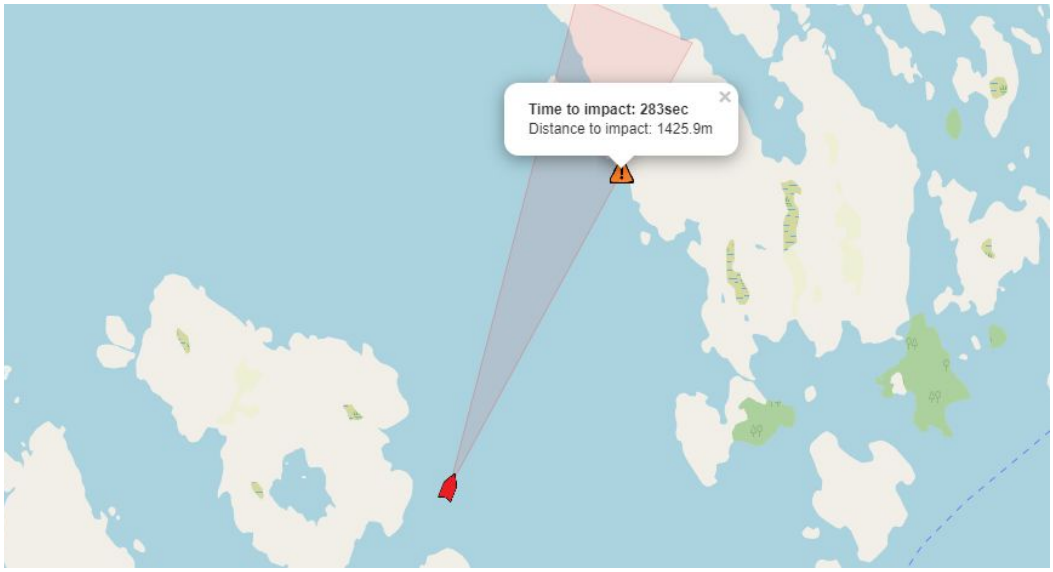
4.1.3.3 Varsel om grunnstøting med informasjon

Som vist i figuren under, varsler produktet om at fartøy er i fare for å gå på grunn ved å tegne en sektor foran fartøyet og en oransje varseltrekant på nærmeste punkt på land eller skjær.



Figur 4.1-6: Skjermbilde som viser et fartøy som er i fare for å gå på grunn.

I tillegg til å varsle kan produktet presentere tilleggsinformasjon om varselet. Ved å holde musepekeren over varseltrekanten blir man presentert for tid og distanse til grunnstøting, slik som vist i figuren under.

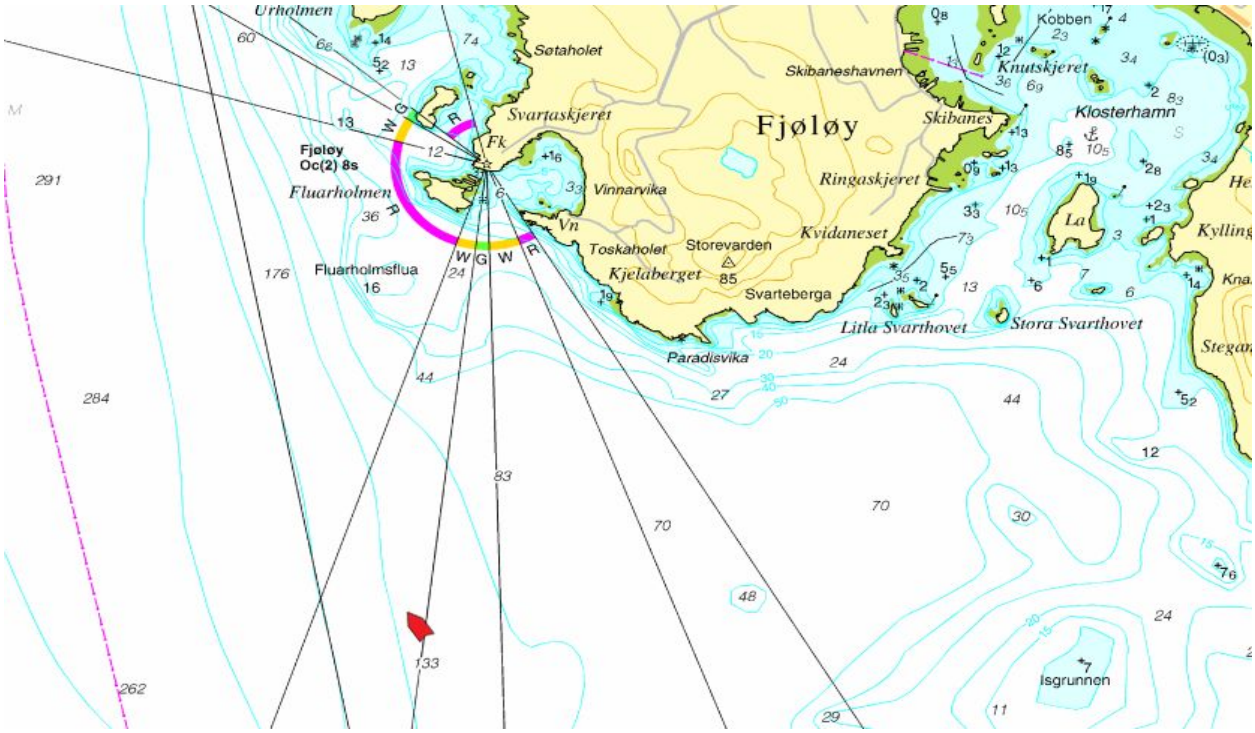


Figur 4.1-7: Skjermbilde som viser et fartøy som er i fare for å gå på grunn, og tilleggsinformasjon om advarselen.

Størrelsen på sektoren er gitt av en funksjon av hastigheten. Et fartøy med høy hastighet vil bruke lenger tid på å manøvrere, så sektoren vil bli lengre og smalere enn ved lavere hastighet.

4.1.3.4 Sjøkart

Som vist i figuren under, har produktet mulighet for visning i et sjøkart. Sjøkartet tilbyr mer informasjon om Norges kyst.



Figur 4.1-8: Skjermbilde som viser et utklipp av produktets sjøkart.

4.2 Ingeniørfaglige resultater

De ingeniørfaglige resultatene er koblet opp mot krav fra oppgavestiller, og er hentet fra vedlegg A, *Visjonsdokument*, kapittel 5 og 6. I dette kapittelet beskrives måloppnåelsen for disse kravene.

4.2.1 Funksjonelle egenskaper

4.2.1.1 Kartvisning

For beskrivelse av hvordan posisjon og retning for fartøy på er implementert, se kapittel 4.1.3.1, *Posisjon og retning for fartøy*.

4.2.1.2 Fartøysdetaljer

For beskrivelse av hvordan fartøysdetaljer er implementert, se kapittel 4.1.3.2, *Informasjon om fartøy.*

4.2.1.3 Fartøyshistorikk

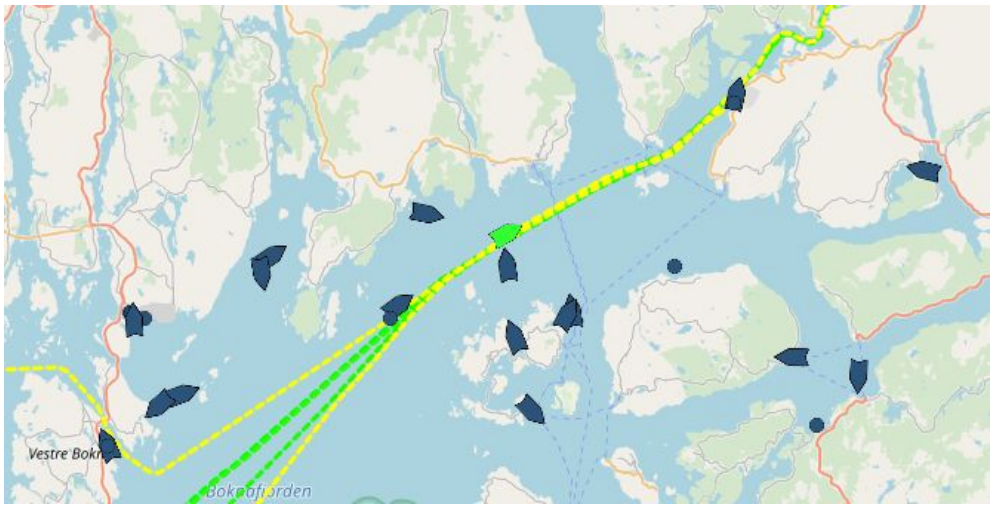
Informasjon om hvor fartøyet har vært vises ved å huke av for “Show track” i modalen for fartøyet. Som vist i figuren under, vises historikken til fartøyet som en heltrukken linje mellom alle tidligere rapporterte posisjoner i databasen til fartøyet. Historikken vil oppdateres når produktet får en ny AIS-melding fra fartøyet.

Figur 4.2-1: Skjermbilde som viser hvordan historikken til et fartøy vises.

4.2.1.4 Fartøysklassifisering

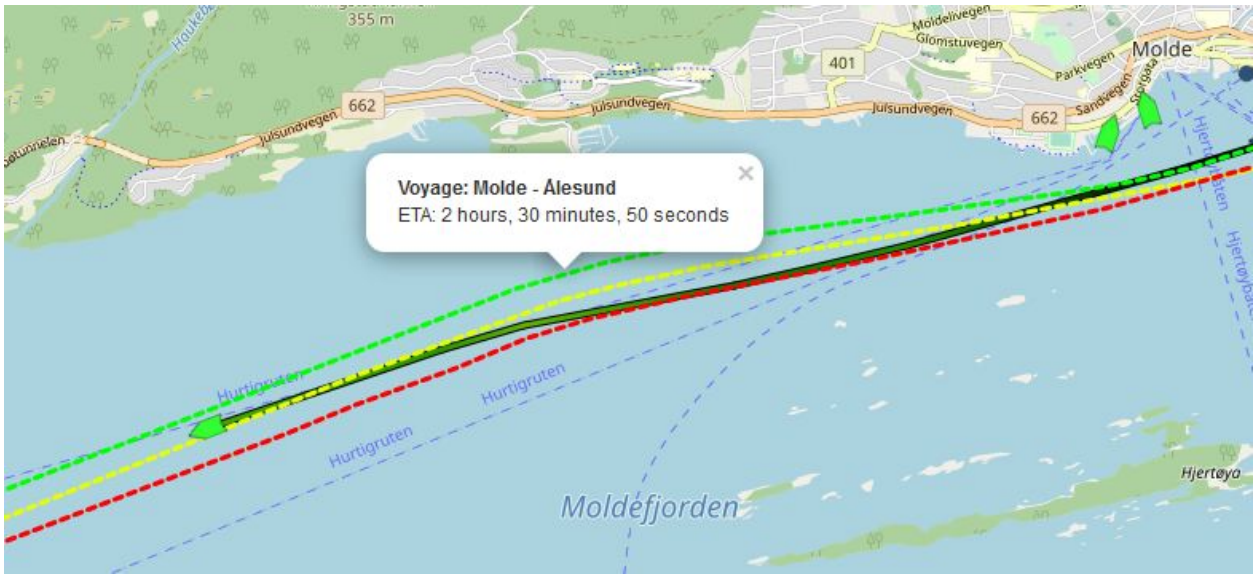
Normalrutene et fartøy er klassifisert til, vises ved å huke av for “Route forecast” i modalen til fartøyet. Dette er bare mulig om fartøyet har blitt klassifisert til én eller flere normalruter. Som vist i figuren under, vises normalrutene til fartøyet som stiplede linjer i kartet. Fargen på den stiplede linjen indikerer hvor godt klassifiseringen passer til fartøyet, hvor grønt er bra og rødt er dårlig. Dersom et fartøy ikke er klassifisert, har fartøysikonet en mørkeblå farge.

Normalrutene oppdaterer seg også automatisk når produktet får inn nye AIS-meldinger fra det aktuelle fartøyet.



Figur 4.2-2: Skjermbilde som viser normalruten til et fartøy.

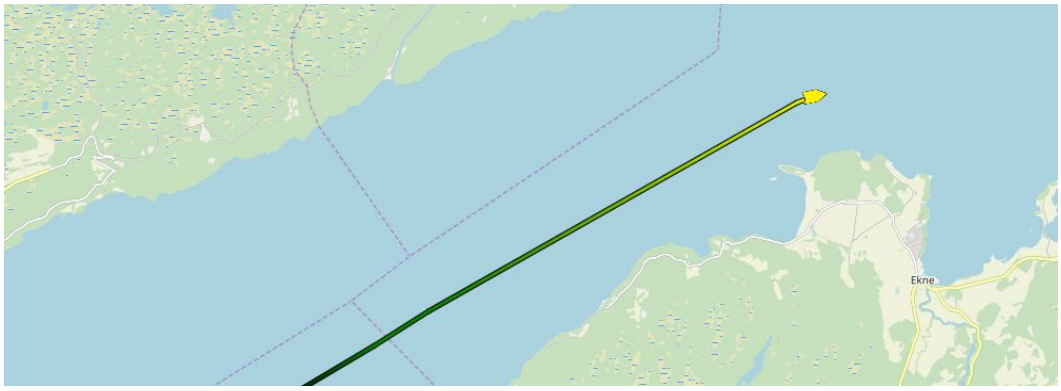
Alle normalrutene går fra en havn til en annen. Som figuren under viser, så er disse havnene, samt den estimerte tiden til ankomst (ETA), vist når brukeren holder musepekeren over normalruten.



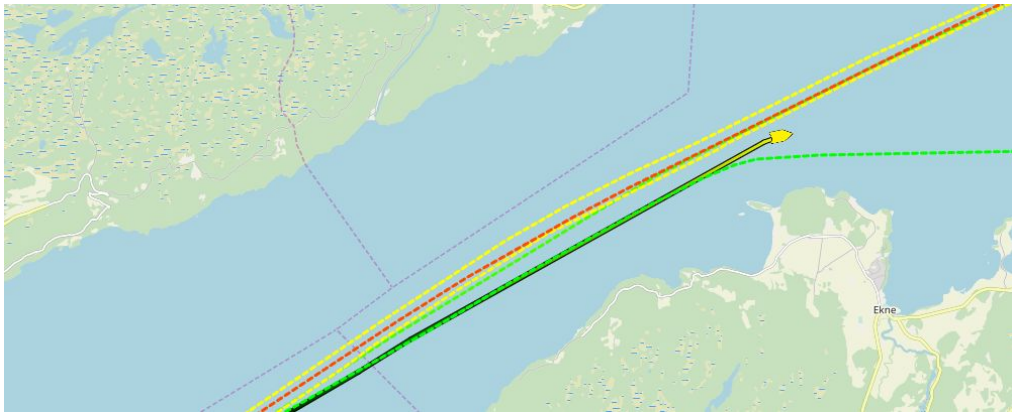
Figur 4.2-3: Skjermbilde som viser informasjon om seilasen til et fartøy.

4.2.1.5 Avviksdeteksjon

Produktet oppdager avvik fra normalruter automatisk, og opplyser om dette. Brukeren kan ved å se på fargen til fartøyet, tolke avviket til den klassifiserte normalruten med best rangering. Grønn betyr at det ikke er noe, eller svært lite avvik fra normalruten, gult betyr at det er et moderat avvik fra normalruten og rødt betyr at det er betydelig avvik fra normalruten. Tidligere avvik vises også i historikken. Fargen i historikken gjenspeiler hva avviket til fartøyet var på det punktet.



Figur 4.2-4: Skjermbilde som viser et fartøy som har et avvik fra normalruten, og dens historikk.



Figur 4.2-5: Skjermbilde som viser et fartøy som har et avvik fra normalruten, dens historikk og klassifiseringer.

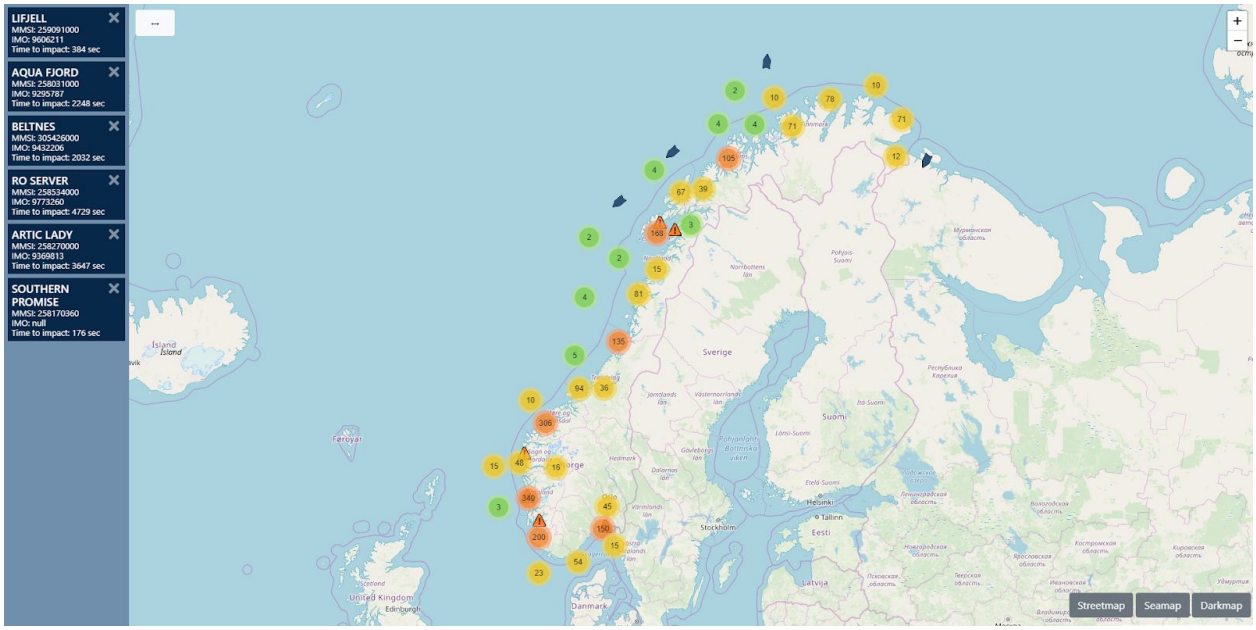
Figur 4.2-5 viser det samme skjermbildet som i figur 4.2-4, men med fartøyets klassifiseringer i tillegg. Her ser vi at fartøyet tidligere har fulgt normalruten, men at den nå har et avvik. Dette stemmer godt overens med historikken som viser at fartøyet tidligere hadde lite til ingen avvik.

4.2.1.6 Grunnstøting

For beskrivelse om hvordan visualisering av varsling av grunnstøtinger er implementert, se kapittel 4.1.3.3, *Varsel om grunnstøting med informasjon.*

4.2.1.7 Oversiktsbilde

Produktet har et oversiktsbilde som viser fartøy rundt norskekysten med alle varsler om farer for grunnstøtinger. Ved å zoome ut får man et oversiktsbilde som viser fartøy som ligger utenfor norskekysten. Disse grupperes og vises som klynger med et tall, hvor tallet representerer antall fartøy i det området. Varseltrekanter representerer steder hvor det er et varsel om grunnstøting. I en uttrekkbar liste til venstre i bildet vises alle varslinger om grunnstøtinger. Oversiktsbildet vises av figuren under.



Figur 4.2-6: Skjermbilde som viser et oversiktsbilde av produktet.

Listen oppdateres automatisk når det kommer nye varslinger, hvor den nyeste kommer øverst. Fartøy som ikke lengre er i fare blir fjernet fra listen.

4.2.2 Ikke-funksjonelle egenskaper

Resultatene er koblet opp mot krav til ikke-funksjonelle egenskaper fra vedlegg A, *Visjonsdokument,* kapittel 6.

4.2.2.1 Funksjonell egnethet

Oppgavestiller har stilt krav til grad av funksjonell kompletthet. Resultatene av dette kravet er beskrevet i vedlegg E, *Systemtest*.

Det er krav til grad av funksjonell nøyaktighet, som er løst ved at produktet fremstiller informasjon korrekt og at den oppdateres i sanntid. Input blir kontrollert slik at korrupt data og feilaktig data blir forkastet. Hver modul håndterer feilmeldinger og unntak selv. Dette fører til at produktet ikke krasjer under kjøring. Det eneste unntaket til dette er om tjeneren mister internettilgang. Da vil tjeneren kaste et unntak som stopper tjeneren.

Resultatene som viser hvordan vi har løst kravet om grad av hensiktsmessig funksjonalitet, er beskrevet i kapittel 4.2.1, *Funksjonelle egenskaper*.

4.2.2.2 Pålitelighet

Oppgavestiller har stilt krav til produktets grad av pålitelighet og nøyaktighet. Vi har løst dette kravet gjennom enhets- og stresstester. Enhetstester kontrollerer at programmet kan håndtere all inngående data fra AIS-strømmen. Stresstester viser at produktet kan håndtere frekvensen

av inngående data fra en AIS-strøm. Resultatene fra stresstester er beskrevet i kapittel 4.1.3.2, *Informasjon om fartøy*.

Det er også stilt krav til gjenoppretting, som er løst ved at historikken til alle fartøy er lagret i en database.

4.2.2.3 Effektiv ytelse

Oppgavestiller har satt krav til produktets grad av prosesseringstid. Resultatene fra dette kravet er beskrevet under kapittel 4.1.2, *Evne til å håndtere meldinger i sanntid*.

Oppgavestiller sitt krav til minnebruk er testet og minnebruken ligger gjennomsnittlig på 500 MB, under realistiske forhold.

Det er også stilt krav til produktets kapasitet, som vi har løst gjennom å bruke SignalR. Produktet er testet med 10 samtidige tilkoblinger uten noen registrerte problem eller svekkelse i ytelse.

4.2.2.4 Brukbarhet

Oppgavestiller har stilt krav til produktets grad av gjenkjennelig hensiktsmessighet. Resultatene fra dette kravet er beskrevet under kapittel 4.1.3.2, *Informasjon om fartøy*, og kapittel 4.2.1.7, *Oversiktsbilde*.

Det er krav til produktets grad av lærbarhet. Resultatene fra dette er beskrevet i kapittel 4.2.1, *Funksjonelle egenskaper*.

Kravet som er stilt til produktets grad av tilgjengelighet, er løst ved at all funksjonalitet som produktet tilbyr, kan benyttes ved å bruke musen alene.

Oppgavestiller har stilt krav til grad av beskyttelse mot brukerfeil. Dette løses i vårt produkt ved at det ikke er mulig for bruker å gjøre input, utenom hva som er definert mellom klienten og tjeneren, gjennom vanlig bruk. Grensesnittet tilbyr ingen operasjoner, eller kombinasjon av operasjoner som kan føre til en brukerfeil.

Kravet til grad av UI-estetikk er løst ved å tilby et grafisk grensesnitt som tilbyr gode kontraster og intuitiv fargebruk på fartøy, normalruter, historikk og modaler.

4.2.2.5 Sikkerhet

Oppgavestiller har stilt krav til produktets grad av integritet. Dette kravet er løst gjennom at produktet kun tilbyr tjenester til klienter over HTTPS[8]. Alle forsøk på å få en forbindelse over HTTP[9], nektes av tjeneren.

4.2.2.6 Vedlikehold

Oppgavestiller har stilt krav til produktets grad av modularitet. Kravet er løst ved at kildekoden til produktet følger prinsippet om Separation of Concerns gjennom å anvende en modulbasert arkitektur. Hvert delprosjekt i produktet fungerer som en egen modul som tar for seg et aspekt ved produktet.

Kravet som er stilt til grad av modifiserbarhet er løst ved at kildekode til produktet er gjort tilgjengelig på GitHub. Alle med tilgang til git-repoet kan klone prosjektet og gjøre lokale endringer.

Oppgavestiller sitt krav til testbarhet er løst ved at produktet anvender testrammeverket NUnit. NUnit tilbyr omfattende dokumentasjon for hvordan lage og kjøre tester. Produktet tilbyr enhetstester for klassen som skal ta imot AIS-meldinger, klassen som kontrollerer om AIS-meldinger er gyldige og klassen som holder orden på informasjon om alle fartøy som sender AIS-meldinger. Det er i tillegg integrasjonstester mot MADART.

4.2.2.7 Mobilitet

Produkteier har stilt krav til produktets grad at tilpasningsevne. Dette kravet er løst ved at produktet er bygget i ASP.NET Core, som lar tjeneren kjøre på operativsystemene Linux, Microsoft og macOS. Klienten kjøres gjennom en nettleser. Nettleserne Google Chrome v74, Firefox v66, og Microsoft Edge v44 er testet og fungerer. Internet Explorer er også testet, men fungerer ikke.

Produkteier sitt krav til installerbarhet er løst ved å tilby en installerings- og avinstalleringsveiledning. Veiledningene er tilgjengelige i vedlegg C, *Systemdokumentasjon*, kapittel 9*, Installasjon og kjøring*, og kapittel 10, *Avinstallering*. Veiledningene er også tilgjengelige gjennom produktet sin Readme, i git-repoet.

[8] Hypertext Transfer Protocol Secure, er en utvidelse av HTTP, og som bruker TLS eller SSL for å kryptere kommunikasjonen over nettet.
[9] Hypertext Transfer Protocol.

4.3 Administrative resultater

4.3.1 Scrum

Vi valgte å følge Scrum som utviklingsprosess med unntaket som er beskrevet i kapittel 3.5.1. Under første sprint hadde vi ikke Daily Scrum som et rituale. Ved etterfølgende sprint retrospektiv ble vi enige om å innføre ritualet. Vi hadde faste sprinter med Sprint Planning, Daily Scrum, Sprint Review med produkteier, og sprint retrospektiv på slutten av nesten hver sprint. Med unntak av Daily Scrum, er alle Scrum-hendelser dokumentert i vedlegg D, *Prosjekthåndbok*, kapittel 2.

Andreas Ravnestad, som var NoIS sin representant, fungerte som produkteier gjennom hele utviklingen. Innledningsvis var Joakim Solheim Scrum Master, men under første sprint retrospektiv ble vi enige om å gå vekk fra å ha en Scrum Master. Et annet avvik fra tradisjonell Scrum-gjennomføring var at vår Sprint Backlog var mer dynamisk enn den ofte vil være. Det viste seg å være vanskelig å gjette på antall arbeidsoppgaver vi klarte å gjennomføre i løpet av en sprint, så det var ikke unormalt at vi overførte arbeidsoppgaver fra Product Backlog til Sprint Backlog i løpet av en sprint. Et annet problem vi opplevde underveis var at vår Product Backlog måtte vedlikeholdes mye, og Produkt Backlog Refining ble til slutt en integrert del av vår Daily Scrum. Det var satt av en uke til hver sprint, men det var ikke alltid vi hadde tid til rådighet for å jobbe med utviklingen hele uken. Varigheten på sprintene varierte dermed etter tiden til rådighet. Totalt hadde vi åtte sprinter. Antall planlagte sprinter var syv.

4.3.2 Veiledningsmøter

Gjennom prosessen har vi hatt flere møter med veileder angående problemer og spørsmål vi har hatt rundt rapporten. Alle møter med referat er dokumentert i vedlegg D, *Prosjekthåndbok*, kapittel 3. Møtene med veileder ble gjort på Kalvskinnet Campus, NTNU. Det ble holdt et møte med veileder hver tredje uke.

4.3.3 Måloppnåelse av fremdriftsplan

Fremdriftsplanen av prosjektet er laget som et Gannt-diagram i vedlegg D, *Prosjekthåndbok*, kapittel 1. Fremdriftsplanen viser målsetninger til fremgangen i prosjektet som milepæler, med et tidspunkt for når målet skulle være nådd. Et av målene var koblet opp mot problemstillingen, tre var milepæler for produktet, og to var koblet opp mot rapport og fremføring. Med unntak av målet om når problemstillingen skulle være ferdigstilt, som ikke ble nådd før i uke 12, ble resten av målene nådd innen tidsfristen som var satt.

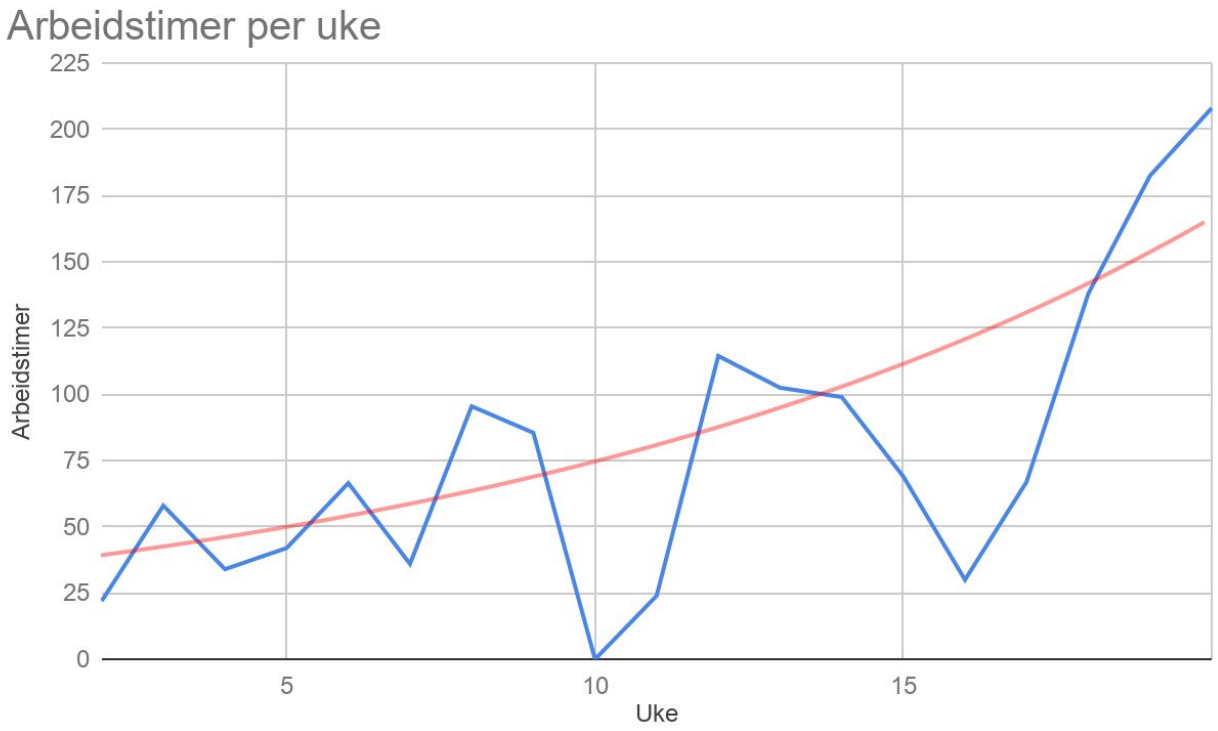
Vi planla totalt syv sprinter, men sprint seks ble delt opp i to sprinter, slik at det til sammen ble åtte sprinter. Antallet sprinter var planlagt med oppgavestiller, som ønsket kortere sprinter innledningsvis i prosjektet, og lengre sprinter senere i prosjektet. Da vi kom til Sprint Planning

for sprint seks, ønsket produkteier å redusere lengden til én uke, for å kunne gi hyppigere tilbakemeldinger på det ferdige produktet.

4.3.4 Timebruk

Timebruken er fordelt på kategoriene: oppstart/planlegging, dokumentering, administrativt, utvikling, forskning og hovedrapport. I fremdriftsplanen er det planlagt hvor mye tid som skulle brukes på hver av aktivitetene, og i hvilket tidsrom disse skulle jobbes med. For hver uke har vi summert antall arbeidstimer som er brukt på hver av aktivitetene, og disse er igjen summert slik at vi får det totale antallet arbeidstimer som er brukt på hver av aktivitetene. For fullstendig oversikt over timer brukt, se vedlegg D, *Prosjekthåndbok*, kapittel 4.

Antallet timer vi har jobbet per uke, fra uke 2, frem til uke 20, er vist av figuren under:



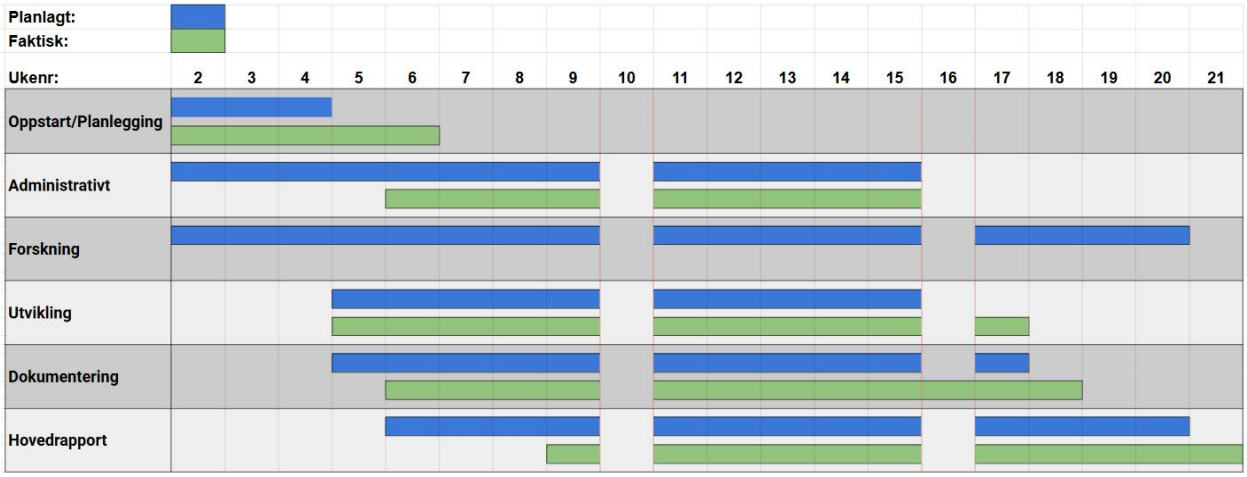
Figur 4.3-1: Antall timer jobbet i uken, per uke.

Det planlagte timeforbruket og faktisk antall timer brukt er gitt av tabellen under.

Aktivitet	Planlagte timer	Faktisk timer	Differanse
Oppstart/Planlegging	85,0	152,0	67,0
Dokumentering	150,0	143,5	−6,5
Administrativt	50,0	69,0	19,0
Utvikling	600,0	545,5	−54,5
Forskning	45,0	0,0	−45,0
Hovedrapport	600,0	609,5	9,5
Sum	**1530,0**	**1519,5**	**−10,5**

Figur 4.3-2: Oversikt over planlagte og faktisk brukte timer for ulike aktiviteter.

Figuren under viser når det var planlagt å jobbe med hver av hovedaktivitetene (hentet fra fremdriftsplanen i vedlegg D, *Prosjekthåndbok*, kapittel 1), opp mot når vi faktisk jobbet med aktiviteten.



Figur 4.3-3: Oversikt over planlagt og faktisk ukefordeling av hovedaktiviteter.

5. Diskusjon

5.1 Vitenskapelige resultater

I dette kapittelet vil vi drøfte problemstillingen og forskningsspørsmålene definert i kapittel 1.2 sammen med resultatene som ble fremstilt i kapittel 4.

5.1.1 Problemstilling

Problemstillingen i sin helhet kan betraktes som definert i kapittel 1.2, eller som en kombinasjon av de tre forskningsspørsmålene som blir utredet under samme kapittel. Dette kapittelet tar for seg de mest sentrale utfordringene i vår prosess. Systemet vi skulle utvikle hadde som oppgave å integrere mange eksterne komponenter sammen til et fungerende produkt. For å ha mulighet til å teste problemstillingen og nå kravene som er stilt til produktet, var det viktig for både oppgavestiller og oss at koden holdt en god standard. Designmønstre som preget planleggingen av produktet var Separation of Concerns, Inversion of Control og parallellprosessering. Disse gjorde det enklere å holde produktet modifiserbart, oversiktlig og effektivt.

Applikasjonsrammeverket ASP.NET Core leverte god støtte for modulbasert arkitektur og Dependency Injection, som var essensielt for å ha en god arkitektur. Vi har tilstrebet å opprettholde god kodestandard og å følge designmønstrene som er nevnt tidligere. En merkbar mangel, er at vi ikke klarte å injisere Repositories som ønsket, på grunn av problemer med levetid. Dette påvirket ikke resultatene, men medførte at vi måtte gå noe bort fra de verdiene vi ville opprettholde under hele prosjektstrukturen.

5.1.2 Forskningsspørsmål

5.1.2.1 Kan en løsning klare å oppdage og varsle om mulige grunnstøtingssituasjoner tidsnok til at aktører får tid til å handle?

Vi vil gjøre noen antakelser for å drøfte hvorvidt de simulerte grunnstøtingene kunne vært avverget. Vi antar at en VTS-operatør følger med og reagerer på varsler om grunnstøtinger nesten umiddelbart. Videre antar vi at VTS-operatøren nesten umiddelbart identifiserer faren, og tar kontakt med fartøyet som er i fare. Vi antar også fartøyet ikke har noen tekniske problemer, samt at mannskapet på fartøyet sover eller opptatt med andre ting, og har derfor en responstid på tre minutter. Videre antar vi at fartøyet trenger å starte en kursendring senest 90 sekunder før fartøyet ville gått på grunn, for at det skal være mulig å avverge grunnstøtingen. Med disse antakelsene mener vi at et fartøy vil være i stand til å unngå en grunnstøting dersom det blir varslet om fare 270 sekunder før fartøyet ville ha gått på grunn. Vi mener derfor at for alle grunnstøtingssituasjoner i våre simuleringer hvor produktet varsler om grunnstøting mer enn

270 sekunder før fartøyet går på grunn, kunne med svært høy sannsynlighet vært avverget. Dette utgjør ni av de 13 simuleringene vi gjennomførte. Videre skal vi ta for oss de fire simuleringene hvor første varsling kom senere enn 270 sekunder før fartøyet gikk på grunn.

Grunnstøtingen til fartøyet “Frydholm”, kunne muligens vært avverget. Selv om første varsling om grunnstøting kommer for sent til at vi kan med høy sannsynlighet anta at grunnstøtingen kunne vært avverget, så har fremdeles mannskapet 149 sekunder til å reagere. Vi mener det er sannsynlig at mannskapet om bord på fartøyet kunne reagert i tide til å avverge eller minimere skadeomfanget av grunnstøtingen, dersom en VTS-operatør hadde gjort dem oppmerksomme på faren.

Grunnstøtingen til fartøyet “Wilson Lista”, kunne med moderat sannsynlighet vært avverget. Produktet detekterer grovt avvik nesten fire timer før fartøyet går på grunn. Det er ikke usannsynlig at en VTS-operatør hadde lagt merke til fartøyet sitt grove avvik fra dens normalrute, og tatt kontakt med fartøyet lenge før den gikk på grunn. Videre kunne VTS-operatøren holdt et ekstra godt øye med fartøyet.

Grunnstøtingen til fartøyet “Ronia Diamond”, kunne med lav sannsynlighet vært avverget. Fordi produktet ikke varsler om fare for grunnstøting, så ville det kreve at en VTS-operatør følger med på fartøyet i tidsrommet før ulykken finner sted, dersom faren skulle vært identifisert i tide.

Grunnstøtingen til fartøyet “Havbris”, kunne mest sannsynlig ikke ha vært avverget. Produktet detekterer kun at fartøyet har et moderat avvik fra sin normalrute, og sjekker derfor ikke om fartøyet er i fare for å gå på grunn. Produktet er til liten nytte for en VTS-operatør i denne situasjonen, og det ville kreve at en VTS-operatør følger nøye med på fartøyet i tidsrommet, om faren skulle vært identifisert.

Det er svakheter i våre antakelser. Estimatet for tiden et fartøy trenger for å handle, er veldig generelt og tar ikke hensyn til fartøyets individuelle faktorer som er nevnt i kapittel 2.1.2, *Ekspertvurderinger*.

5.1.2.2 Kan en løsning klare å prosessere alle relevante data fra Kystverket sin åpne AIS-strøm i sanntid?

For å kunne svare på forskningsspørsmålet deler vi problemet opp i to testbare egenskaper: om produktet klarer å håndtere frekvensen av meldinger fra Kystverket sin åpne AIS-strøm, og om vi kan garantere at produktet sender ut informasjon om meldinger i sanntid.

Resultatene i 4.1.2 viser hvordan produktet håndterer store mengder data på kort tid. Vi ville stressteste produktet for å garantere at produktet klarer håndtere alle meldinger fra Kystverket sin åpne AIS-strøm. Vi satte et krav til at produktet måtte kunne håndtere minst 3000 meldinger i minuttet, for å ha en god margin på høyeste målte frekvens fra Kystverket sin AIS-strøm.

Testene viste at produktet klarer å bearbeide omtrent 10 000 meldinger i minuttet. Dette er mer enn fem ganger så mye som vi har målt fra Kystverket sin AIS-strøm, og over tre ganger mer enn kravet vi definerte.

For å kunne ta en avgjørelse på om produktet klarer å håndtere en strøm i sanntid, må vi først definere “sanntid”. Det er viktig at informasjon kommer frem til brukeren mens den fortsatt er relevant, men ettersom vi snakker om varsling opp til flere minutter i forkant av en grunnstøting, er det ikke nødvendig at informasjonen vises umiddelbart etter produktet mottar meldingen. Vi definerer her sanntid som en prosesseringstid på under 10 sekunder. Denne grensen samsvarer med det ikke-funksjonelle kravet satt i vedlegg A, *Visjonsdokument,* kapittel 6.

Vi målte tidsbruken på de mest tidkrevende meldingene produktet ville få, og fant en gjennomsnittstid på rett under 3 sekunder. Dette er godt innenfor vår definisjon av sanntid.

Testene vi har gjennomført viser at produktet klarer å håndtere frekvensen av AIS-meldinger fra en åpen AIS-strøm, og at tiden produktet bruker på å prosessere meldingen er innenfor de fastsatte kravene. En viktig variabel for disse testene er spesifikasjonene til maskinene som tjeneren og MADART kjører på. Vi velger å ikke gå i dybden på dette, da problemstillingen setter ikke setter krav til spesifikasjoner, og resultatene skal kun underbygge muligheten.

5.1.2.3 Kan en løsning tilby nok beslutningsstøtte for en VTS-operatør ved mulige grunnstøtingssituasjoner?

Sammen med oppgavestiller ble det utarbeidet krav for å konkretisere hva en VTS-operatør trenger for å analysere en situasjon og ta beslutninger. Disse er beskrevet i vedlegg A, *Visjonsdokument*, kapittel 3.4. Resultatene fra kapittel 4.1.3 og 4.2 oppfyller kravene som er satt.

En VTS-operatør kan ta avgjørelser kun ved å se posisjon og retning til et fartøy i et sjøkart. Basert på ekspertvurderingene gitt av Roy Stokvik og observasjoner under intervjuet, ser vi at dette er tilstrekkelig for VTS-operatør til å ta en beslutning. Produktet vårt tilbyr dette, og mer.

Om vårt produkt var integrert med SSNN, ville dette gi mer beslutningsstøtte til VTS-operatøren i flere tilfeller.

5.2 Ingeniørfaglige resultater

5.2.1 Funksjonelle egenskaper

Produktet samsvarer svært godt med de planlagte funksjonelle egenskapene som er beskrevet i vedlegg A, *Visjonsdokument*, kapittel 5, og kravene som er bestemt i vedlegg B, *Kravdokumentasjon*.

5.2.1.1 Posisjon og retning for fartøy

Produktet skal tilby et realistisk bilde av skipstrafikken utenfor norskekysten, og det er svært viktig at produktet kan vise frem hvor fartøy er lokalisert og hvilken retning det har, på en intuitiv og oversiktlig måte.

I visjonsdokumentet står det at produktet skal: “Vise et fartøys posisjon og retning fra en AIS-strøm, i sanntid”. Som resultatene i kapittel 4.1.3.1 viser, ser vi at kravet er oppfylt ved at fartøyet er plottet i et kart, som umiddelbart gir VTS-operatører en forståelse av posisjonen og retningen til fartøyet.

5.2.1.2 Informasjon om fartøy

Som beskrevet i kapittel 5.1.2.3 er informasjon om fartøy tett knyttet opp mot VTS-operatørens beslutningsstøtte.

I visjonsdokumentet står det at produktet skal: “Vise detaljert informasjon om et fartøy”. Resultatene som er beskrevet i kapittel 4.1.3.2 dekker kravet gitt av visjonsdokumentet.

5.2.1.3 Fartøyshistorikk

For en VTS-operatør så er historikken til et fartøy verdifull informasjon. Den gjør det mulig å forstå hvor et fartøy har gått, slik at det er enklere å forstå hvor fartøyet skal.

I visjonsdokumentet står det at produktet skal: “Vise informasjon om hvor et fartøy har vært”. Som resultatene beskrevet i kapittel 4.2.1.3, så er kravet nådd ved at historikken vises som en heltrukken linje.

5.2.1.4 Fartøysklassifisering

Å kunne klassifisere fartøy gjør det mulig å detektere om et fartøy har feilnavigert eller er på vei ut av kurs. Ved å kunne se normalruter for fartøy, kan en VTS-operatør raskt identifisere om et fartøy er ute av kurs, uten å ha kjennskap til farvannet fra før.

I visjonsdokumentet står det at produktet skal: “Vise hvilken klassifisering et fartøy har”. Som resultatene i kapittel 4.2.1.4 viser, så er kravet nådd ved at produktet viser normalrutene et fartøy er klassifisert til, som stiplede linjer.

5.2.1.5 Avviksdeteksjon

For VTS-operatører som skal overvåke nesten hele Norges kystlinje med omtrent 2300 fartøy til enhver tid, så vil det være en stor avlastning med et program eller en tjeneste som kan automatisk detektere avvik.

I visjonsdokumentet står det at produktet skal: “Detektere og opplyse om at et fartøy avviker fra sin normalrute”. Som resultatene i kapittel 4.2.1.5 viser, så er kravet nådd ved at produktet fargelegger ikonet til fartøyet etter grad av avvik, som er tilstrekkelig for at brukeren oppdager fartøyet, og kan undersøke det nærmere.

5.2.1.6 Varsel om grunnstøting

Grunnstøtinger er farlige, og kan gi store økonomiske, helsemessige og miljømessige konsekvenser. For en VTS-operatør kunne det vært et viktig hjelpemiddel å ha et program eller en tjeneste som kan detektere og varsle om grunnstøtinger tidsnok.

I visjonsdokumentet står det at produktet skal: “Detektere og varsle om et fartøy er i fare for å gå på grunn”. Som resultatet i kapittel 4.1.3.3 viser, så er dette kravet oppfylt ved at produktet tegner en sektor og en varseltrekant til nærmeste punkt til land eller skjær. Brukeren varsles ved at en advarsel vises i varsellisten til produktet, slik at brukeren blir gjort oppmerksom på varselet, uavhengig av skjermbilde.

5.2.1.7 Oversiktsbilde

For en VTS-operatør er overvåking et sentralt begrep. Et program eller tjeneste som gir VTS-operatøren en komplett, korrekt og oppdatert oversikt over fartøy rundt norskekysten, med advarsler om avvik fra normale seilingsmønstre og grunnstøtinger, kunne vært et viktig hjelpemiddel for VTS-operatører. Dette gjelder spesielt VTS-operatører på Vardø sjøtrafikksentral.

I visjonsdokumentet står det at produktet skal: “Vise et oversiktsbilde over alle fartøy rundt norskekysten, med alle varsler og potensielle farer”. Som resultatene fra kapittel 4.2.1.7 viser, så er dette kravet om et oversiktsbilde oppfylt ved at produktet viser alle fartøyene i kartet. Kravet om å vise alle varslinger er oppnådd ved at produktet har en varselliste til venstre for kartet, som viser alle varsler.

5.2.2 Ikke-funksjonelle egenskaper

5.2.2.1 Funksjonell egnethet

I visjonsdokumentet er det krav om at produktet skal ha en grad av funksjonell egnethet. Produktet skal dekke alle krav med høy prioritet, vise korrekt informasjon, være et hjelpemiddel for målgruppen samt håndtere feilmeldinger og korrupt data.

Produktet viser oppdatert informasjon, ved å alltid vise informasjonen fra siste AIS-melding fra fartøyet. Produktet håndterer korrupt data fra AIS-strømmen og unntak som kan oppstå under kjøring håndteres. Unntaket til dette er om det oppstår problemer hvor det ikke lenger vil være hensiktsmessig å kjøre programmet, som om tjeneren mister internettilgang. Produktet viser et oversiktsbilde over fartøy utenfor norskekysten og varsler brukeren om fartøy som avviker fra

normale seilingsmønstre og er i fare for å gå på grunn. Dette er et verktøy som kan være til støtte for en VTS-operatør.

5.2.2.2 Pålitelighet

I visjonsdokumentet er det stilt krav til produktets grad av pålitelighet. Produktet skal kunne håndtere inngående data, være tilgjengelig døgnet rundt og ikke måtte trenge å stoppes under normale forhold. Det skal også være mulig å gjenopprette tidligere AIS-meldinger.

Fra kapittel 4.2.2.2, så ser vi at kravene til pålitelighet er nådd gjennom enhets- og stresstester som verifiserer at inngående data håndteres. Stresstester verifiserer at produktet kan håndtere nok meldinger over tid til å holde tritt med AIS-strømmen. I tillegg har produktet kjørt sammenhengende over en lengre periode uten problemer. Historikken til fartøy for de siste 24 timer ligger tilgjengelig i databasen. Denne påliteligheten gjør at produktet kan stå og kjøre på en sjøtrafikksentral døgnet rundt, uten behov for vedlikehold, som er viktig for at produktet skal være til verdi for brukerne av produktet.

5.2.2.3 Effektiv ytelse

I visjonsdokumentet er det stilt krav til produktets effektive ytelse: Prosessering av AIS-meldinger skal ikke ta lengre enn 10 sekunder, tjeneren skal ikke ha minnebruk som overstiger 1 GB, og produktet må kunne tilby tjenesten til flere klienter.

Fra kapittel 4.2.2.3, så ser vi at kravene er testet: Prosesseringstiden for AIS-meldinger som krever mest prosessering av MADART er målt til gjennomsnittlig 2,969 sekunder, som er godt under kravet. Produktet er testet over tid, og minnebruken holder seg på rundt 500 MB, som er godt under kravet på 1 GB. Produktet er testet med ti samtidige tilkoblinger til tjeneren uten at det er observert problemer.

Det er en svakhet at testene kun viser gjennomsnittstid, og ikke har informasjon om hva som er øverste tak for prosesseringstid for AIS-meldinger og minnebruk. På grunn av dette kan vi ikke garantere at minnebruken ikke overstiger 1 GB, eller at alle meldinger ble behandlet på under 10 sekunder. Til tross for dette er gjennomsnittsverdiene så lave at det er svært usannsynlig at grenseverdiene overstiges.

5.2.2.4 Brukbarhet

I visjonsdokumentet er det satt krav til grad av brukbarhet. Produktet skal være gjenkjennelig, og minne om et program for overvåking av sjøtrafikk. Det skal ikke være nødvendig med opplæring for å bruke produktet, og det skal være enkelt å utnytte all funksjonalitet. Det må være utformet på en intuitiv måte, slik at få brukerfeil oppstår. Brukergrensesnittet skal ha elementer som er lett å skille fra hverandre.

Fra kapittel 4.2.2.4, så ser vi at de fleste kravene er delvis eller helt oppnådd. På grunn av måten fartøyene er vist i kartet og fartøysikonene er utformet, minner produktet om andre

programmer som overvåker trafikken til sjøs. Produktet tilbyr begrenset og enkel funksjonalitet som brukeren med høy sannsynlighet allerede kjenner. Å zoome inn og ut med å bruke musehjulet, og å trykke på fartøyer for å få opp informasjon om fartøyet, kombinert med ikoner som har en intuitiv betydning gir produktet et intuitivt brukergrensesnitt. Produktet er enkelt å bruke, gjenkjenne, forstå og lære.

Det er en svakhet at mange av kravene ikke er mulige eller er svært vanskelige å måle grad av oppnåelse for. Det som for noen er intuitivt og enkelt å forstå, kan for andre oppleves utfordrende.

5.2.2.5 Sikkerhet

I visjonsdokumentet er det stilt krav til at produktet skal ha en grad av integritet mellom klient og tjener. Integritet betyr at mottakeren av en melding kan være sikker på at den mottatte meldingen er den samme meldingen som ble sendt.

Fra kapittel 4.2.2.5, så ser vi at dette målet er nådd ved at det kun er mulig å opprette en HTTPS-forbindelse mellom tjener og klient, som sikrer full integritet.

5.2.2.6 Vedlikehold

I visjonsdokumentet stilles det krav til grad av vedlikeholdbarhet til produktet: Produktet skal bestå av utbyttbare moduler, være enkel å modifisere og enkelt å lage tester samt ha enhets- og integrasjonstester mellom tjeneren og MADART.

I kapittel 4.2.2.6 så ser vi at disse kravene er oppnådd. Produktet er delt opp i delprosjekter, som enkelt kan byttes ut. Endringer i programmet gjøres enkelt ved at kildekoden lastes ned fra GitHub, og endres lokalt. Produktet har enhetstester og integrasjonstester mellom tjeneren og MADART.

5.2.2.7 Mobilitet

I visjonsdokumentet er det satt krav til grad av mobilitet: Tjeneren må kunne kjøre på en Microsoft- eller Linuxserver, klienten må kunne kjøre på alle nettlesere som har støtte for HTML5 og produktet må kunne avinstalleres.

Fra kapittel 4.2.2.7, så ser vi at de fleste kravene er oppnådd. Tjeneren til produktet kjører på ASP.NET Core, som er kompatibel med operativsystemene Microsoft, Linux og macOS. Produktet kan kjøres på velkjente nettlesere som Edge, Firefox og Chrome. Produktet kan ikke kjøres i Internet Explorer, da nettleseren ikke støtter asynkrone funksjoner i JavaScript. Kravet om at alle nettlesere som støtter HTML5 skal kunne kjøre produktet er et dårlig formulert krav, da ingen nettlesere støtter HTML5 fullt. I vedlegg C, *Systemdokumentasjon,* kapittel 10, er det en veiledning for å avinstallere produktet.

5.2.3 Systemtest

Vi lagde en systemtest for å verifisere om alle kravene fra oppgavestiller som ble formulert i vedlegg B, *Kravdokument*, kapittel 2, var godkjent. Systemtesten ble gjennomført av oppgavestiller for å være en bekreftelse på at produktet oppfyller kravene som var satt. Systemtesten ble godkjent uten bemerkninger, og oppgavestiller stilte seg fornøyd til produktet. Dette er en bekreftelse på at produktet oppfyller de funksjonelle kravene stilt av oppgavestiller.

5.2.4 Diskusjon om sluttproduktet

5.2.4.1 Svakheter med produktet

Produktet kan ikke automatisk justere eller begrense antall AIS-meldinger som skal prosesseres av tjeneren. Et resultat av dette er at produktet er helt avhengig av at AIS-strømmen har en frekvens som er lavere enn det stresstestene viser som knekkpunkt for ytelsen til produktet. For at produktet ikke skal kreve for mye prosesseringskraft for å kjøre effektivt, så kan en løsning til dette problemet være et filter som kan begrense antallet forespørsler som blir sendt til MADART.

Produktet klarer ikke å varsle om undervannsskjær på en pålitelig måte. Resultatene fra simuleringene av tidligere grunnstøtinger viser at to fartøy går på undervannsskjær, uten at produktet varsler om dette. Dette er en klar svakhet som svekker produktets evne til å varsle, dermed også tilliten en bruker har til systemet. For at produktet skal varsle om grunnstøtinger på skjær på en mer pålitelig måte, så må kilden MADART bruker for å hente kystkontur byttes ut med en som har mer omfattende dekning.

Det er ikke sikkert at produktet varsler om grunnstøtinger tidsnok. Som nevnt i kapittel 4.1.3.3, så er størrelsen på sektoren produktet ser etter grunnstøtinger i, en funksjon av hastigheten til fartøyet. Dette fanger ikke alle de andre faktorene som bestemmer hvor lang tid et fartøy trenger for å avverge en grunnstøting. Dette gir en usikkerhet om produktet er i stand til å varsle om grunnstøtinger tidsnok. En mer kompleks funksjon som tar hensyn til flere faktorer burde derfor utvikles.

Produktet produserer falske varsler på grunnstøtinger. En del av varslene produktet viser, er knyttet til fartøy som ikke burde vært sendt til MADART for klassifisering. Tjeneren har et filter som skal sikre at slike fartøy ikke blir klassifisert, men databasen filteret gjør oppslag i, er mangelfull. I tillegg sender mange fartøy informasjon som er feil eller misvisende når det gjelder skipstype. Dette gjør at mange fartøy kommer seg igjennom filteret, blir klassifisert og produserer falske varsler til brukeren. For å hindre feilklassifisering så avhenger produktet av at informasjonen i databasen er oppdatert. En mulig løsning er at brukere kan legge inn og oppdatere fartøysdata i databasen gjennom klienten.

Logikken til klienten er uoversiktlig. Ettersom vi tenkte at produktet skulle fungere som en prototype, brukte vi ikke noe rammeverk for å strukturere koden som vi hadde på klientsiden. Her burde det brukes et rammeverk.

Det har ikke vært noen VTS-operatører med i prosessen for utformingen av produktet. Det kan derfor hende at det er flere funksjonaliteter som burde blitt implementert i produktet, og ulike grafiske utforminger som hadde gjort produktet bedre for bruker-miljøet.

Det er ikke laget enhetstester og integrasjonstester opp mot databasen, eller integrasjonstester på kommunikasjonen mellom tjener og klient. Dette er elementer som har blitt nedprioritert på grunn av begrenset tid.

5.2.4.2 Styrker med produktet

På enkelte punkter presterer vårt produkt bedre enn kravene som er formulert i vedlegg A, *Visjonsdokument*. I tillegg til å bare tilby varsling om grunnstøting, så tilbyr vårt produkt estimert tid og avstand til grunnstøting. Vårt produkt varsler ikke bare om avvik fra normalruter, men gir også indikasjon på hvor stort dette avviket er. Dette gjør det mulig å justere hvor følsomt produktet skal være til avvik.

Produktet bruker en asynkron utførelse på en god måte. Gjennomsnittlig prosesseringstid for de mest tidkrevende meldingene er godt under kravet for sanntid. Produktet kan håndtere en høyere hyppighet på meldinger enn kravet. Vi mener at det var riktig valg å behandle alle meldinger som en egen tråd, og dette er implementert på en måte som ikke går utover ytelsen til tjeneren.

Produktet tilbyr en detaljert visning av historikken til fartøy. Avviket fartøyet har hatt, vises i historikken. Dette er med på å øke beslutningsstøtten VTS-operatøren får gjennom produktet.

Produktet har et filter på tjeneren, som gjør at mange uinteressante fartøy ikke blir klassifisert av MADART. Dette gjør at MADART kun skal klassifisere fartøy som er interessante, og brukeren skal ikke bli brydd med varsler fra fartøy som ikke er av typen laste- eller cruiseskip. Alle fartøy blir likevel vist i kartet på vanlig måte, som resulterer i at brukeren fremdeles kan se alle fartøy.

Produktet er bygget på teknologi som gjør det enkelt å rulle ut produktet som en skyløsning. Alle rammeverk på tjeneren er kompatible med Azure, som gjør det mulig å knytte en av prosjektets git-branch-er til Azure, slik at prosjektet enkelt får kontinuerlig bygging, testing og utrulling.

5.2.4.3 Refleksjon om valg av teknologier

Det har vært flere valg som har vært kritiske for kvaliteten av produktet. Vi er veldig fornøyde med at vi valgte rammeverket ASP.NET Core med C# som programmeringsspråk, SignalR for å kommunisere med klienter og EF Core som ORM mot databasen. Med ASP.Net Core var det enkelt å sette opp prosjektet og komme i gang med utvikling. C# gjør det enkelt å lage og håndtere tråder. Syntaksen til C# ligner mye på Java, som gjorde at den var enkel å lære.

SignalR gjorde det enkelt å etablere en full dupleks tilkobling mellom tjener og klient. Med EF Core var det enkelt å implementere ORM-teknologi. Dette gjorde det enkelt å koble tjeneren mot databasen, og vi sparte mye tid takket være Scaffolding. Ved å bruke disse teknologiene fikk vi mye av funksjonaliteten vi trengte. Vi sparte også mye tid da alle var .NET Core teknologier som kunne lett integreres med hverandre.

Vi er også fornøyde med valget om å bruke Leaflet for å vise et interaktivt kart. Biblioteket gjorde at vi sparte mye tid. Leaflet var enkelt å lære og hadde nok funksjonalitet til å gjøre alt vi hadde behov for.

Vi har hatt få problemer med PostgreSQL som database. PostgreSQL gjorde det enkelt å behandle geografisk data, og syntaksen var en utvidelse av SQL, som vi allerede hadde erfaring med.

Vi er fornøyd med valgene vi har tatt. Selv om vi har brukt tid på å sette oss inn i teknologiene, tror vi likevel at vi har spart tid ved å bruke dem. Dette gjorde at vi kunne fokusere mer på utvikling.

5.2.4.4 Samfunnsnytte

I Norge er det et behov for nye og bedre teknologier for å overvåke sjøtrafikken. Vårt produkt kan være et viktig hjelpemiddel for VTS-operatører for å detektere mulige grunnstøtinger for laste- og cruiseskip. Om sjøtrafikksentralene stoler på produktet kan det effektivisere måten de jobber på.

Grunnstøtinger kan gi store miljømessige og økonomiske konsekvenser, og vi så hvor ille det kan gå da fartøyet “Full City” gikk på grunn. Vårt produkt kan avverge grunnstøtinger, og dermed redusere antall akutte utslipp i norske farvann. Slike utslipp har konsekvenser for miljøet i de berørte områdene. Oppryddingen av slike utslipp kan både være kostbare og kreve en enorm innsats.

Besetningen om bord på fartøy som går på grunn er ofte utsatt, og i fare for å påføres skader. Vårt produkt kan ved å avverge grunnstøtinger, være med på å øke tryggheten om bord på laste- og cruiseskip.

5.3 Administrative resultater

5.3.1 Scrum

Vi har i utviklingen av produktet fulgt utviklingsmetoden Scrum, men med noen unntak. Et stort unntak fra Scrum var at vi ikke hadde en Scrum Master. Dette var lite problematisk, fordi vi var et så lite utviklerteam, og alle involverte i utviklingen hadde god kjennskap til Scrum fra før. Dette skapte få situasjoner hvor det var nødvendig med koordinering utover Daily Scrum, eller et behov for å undervise oss eller produkteier om Scrum-prosessen. Administrativt arbeid ble

fordelt på oss alle. Produkteier var tilgjengelig for å legge til rette for utvikling, herunder innføring i teknologi og hjelp til å komme i kontakt med nøkkelpersoner. Vi opplevde aldri behov for en Scrum Master gjennom prosjektet.

Alle ritualer for Scrum ble fulgt, men med noen variasjoner. Sprintene våre hadde varierende lengder fra to til ti dager. Dette gjorde det vanskelig å finne ut av hvor produktive vi var, men det var ikke en hindring for utviklingen. Under Sprint Planning ble det vist hensyn til lengden på påfølgende sprint.

Det var også relativt kort lengde på hver sprint, sammenlignet med mer tradisjonell Scrum. Dette førte til hyppige leveranser og tett kontakt med produkteier gjennom hele prosjektet. Dette viste seg å være positivt for prosessen, fordi vi hadde behov for å gjøre mange avklaringer underveis. Til tross for korte sprinter, klarte vi alltid å presentere et inkrement til Sprint Review, og få gode tilbakemeldinger på disse. Det er vanlig at Product Backlog Refining finner sted under Sprint Review, men under vår prosess ble dette en del av Daily Scrum. Dette kom som et behov under prosjektets gang, og ble tatt opp under sprint retrospektiv til 4. sprint. Vi opplevde at kvaliteten på Product Backlog var dårlig, og det var enten vanskelig å forstå intensjonen til en arbeidsoppgave, eller at en oppgave var for omfattende, og måtte deles opp i mindre oppgaver. Det var også tilfeller vi hadde beregnet for få oppgaver til en sprint, og vi flyttet arbeidsoppgaver fra Product Backlog over til Sprint Backlog. Dette førte til at vi fikk en mye mer smidig utvikling siden vi raskt kunne respondere til endringer under en sprint. Vi var alltid produktive og hadde funksjonalitet under utvikling, og gikk aldri tom for oppgaver. Produkteier var fremdeles eier av Product Backlog, og all Product Backlog Refining ble avklart med produkteier. Etter at Produkt Backlog Refining ble en del av Daily Scrum, opplevde vi at Product Backlog fungerte mye bedre som verktøy igjen, og at det var lettere å identifisere nødvendige avklaringer som hindret videre utvikling.

Resten av ritualene og artefaktene ble fulgt i henhold til Scrum. Sprint Planning hjalp oss med å holde oversikt over hva som skulle bli gjort for hver sprint, og hvordan vi skulle prioritere de ulike oppgavene. Sprintene hjalp oss med å sette faste mål og førte til at vi alltid hadde et inkrement klart til hver Sprint Review. Det var også veldig fint å få tilbakemelding fra produkteier på hva vi hadde gjort. Da fikk vi avklart om vi hadde implementert funksjonalitet på samme måte som produkteier hadde forestilt seg, eller om vi hadde gjort noe produkteier mente burde gjøres annerledes. Sprint retrospektiv på slutten av sprinter førte til at vi kom fram til mange gode forslag til hvordan prosessen kunne bli bedre. Etter vi begynte med Daily Scrum, hjalp det oss med å koordinere og planlegge arbeidsdagen, og ble en viktig markering for starten av en arbeidsdag.

Vi er fornøyde med at vi valgte en agil utviklingsmetode for dette prosjektet. Det passet oss godt å ha en iterativ utviklingsprosess, spesielt fordi det ble endringer av produkteiers visjon underveis i utviklingen. De unntakene vi valgte å gjøre fra Scrum var med på å bedre og tilpasse prosessen vår.

5.3.2 Veiledningsmøter

Veiledningsmøtene ble gjennomført regelmessig, og de opplevdes som et nyttig verktøy for å gjøre avklaringer og vise frem hva vi hadde gjort. Møtene var også en god arena for å diskutere problemer vi hadde og fremarbeide løsninger. Vi opplevde det som mer nyttig å møtes fysisk for å gjøre avklaringer, enn å gjøre disse over e-post. Vi mener at hyppigere møter med veileder kunne ført til færre e-poster, bedre avklaringer og kortere møter.

5.3.3 Måloppnåelse av fremdriftsplan

Under prosjektet nådde vi ikke milepælen om å ha en ferdigstilt problemstilling i uke 6, slik det var planlagt i fremdriftsplanen. Dette førte ikke til noen store problemer da vi ikke startet å virkelig skrive på hovedrapporten før i uke 9. Erfaringene vi gjorde i mellomtiden hjalp oss med å revidere og spisse problemstillingen slik at det ble gjort endelig i uke 12.

Antall sprinter ble heller ikke som planlagt. Vi opplevde ikke noen problemer ved å ha én sprint mer enn planlagt, men heller at det var positivt å ha to sprinter hvor vi fokuserte på å ferdigstille produktet. Å ha flere sprinter opplevdes kun som positivt.

Utover avvikene fra fremdriftsplanen som er nevnt over, har vi fulgt den som planlagt. Gjennom prosjektet har vi brukt fremdriftsplanen og timelistene for å kartlegge fremgangen, og planlegge utviklingen videre. Vi opplevde blant annet at fremdriftsplanen hjalp oss med å planlegge de større fasene i prosjektet, som oppstart, utvikling og rapportskriving. I tillegg forenklet det planleggingen av antall sprinter, og lengden på hver sprint. Det lot oss også planlegge rundt andre aktiviteter, som obligatorisk undervisning, eksamenslesing og feriedager.

5.3.4 Timebruk

Det er flere avvik fra antall planlagte timer på enkelte aktiviteter, og hva som faktisk var antall brukte timer. Det var i utgangspunktet planlagt med 45 timer forskning, men vi førte ingen timer på denne hovedaktiviteten. Vi har gjort forskning, men forskningen har som oftest vært i forbindelse med en annen hovedaktivitet. Ettersom vi har kun ført én aktivitet for hver dag, så kommer forskningen ikke frem av timelisten vår.

Det var også et avvik knyttet til oppstart/planlegging. Denne aktiviteten varte i to uker lengre og krevde 67 flere timer enn planlagt. Dette hadde likevel ingen store konsekvenser. Mye av oppstartsaktivitetene var å sette seg inn i teknologi og metoder, samt starte utviklingen av produktet. Det var derfor mye overlapp mellom aktivitetene oppstart/planlegging og utvikling, hvor utvikling hadde et underskudd på 55 timer mot hva vi hadde planlagt.

Av resultatene fra timebruken ser man at utviklingen holdt på i to uker lengre enn planlagt, men at antallet timer var mer eller mindre som forventet. Om man justerer for utviklingen som ble gjort under oppstart av prosjektet, ser vi lignende resultater for antall timer brukt på

dokumentering og varigheten på aktiviteten. Dette kan forklares ved at vi ikke brukte like mange timer i starten av prosjektet, så perioden måtte forlenges.

Vi synes vi har gjort gode estimater på timebruken, om vi justerer for overlappet mellom aktiviteter. Et unntak til dette var at vi undervurderte antall timer som burde legges inn tidlig i prosjektet, så vi måtte jobbe noe mer mot slutten. Vi har brukt timelistene som et verktøy for å disponere tid samt starte og avslutte aktiviteter i tide.

5.3.5 Gruppearbeid

Gruppearbeidet har fungert godt gjennom prosessen. Det ble bestemt å bruke en flat struktur innad i gruppen, noe som vi holdt frem med til prosjektets slutt. Selv om vi ikke hadde klare arbeidsfordelinger, så var det ikke slik at alle gjorde like mye på alle områder. Eksempler på dette var at Joakim Solheim tok på seg hovedansvaret for det administrative, Hallvard Sælthun var den som hadde best oversikt og endret mest i databasen og Brage Snarud tok for seg det meste av utformingen på klientsiden. Dette førte til at arbeidet ble effektivt ettersom alle ble bedre og bedre på de oppgavene de ofte jobbet med. Ingen følte at det var områder de ikke fikk lov, eller hjelp til å jobbe med. Når flere begynte å jobbe på samme arbeidsoppgave opplevde vi få problemer, og det tok kort tid å få en innføring i arbeidsoppgaven. Selv om det i begynnelsen tok litt tid med opplæring, ble dette en styrke. Alle kunne litt om alle arbeidsoppgaver og kunne hjelpe til der det var behov.

Det har vært lite fravær hos medlemmene i gruppen. Dersom det skulle skje at noen ikke kunne jobbe like mye en uke, jobbet de mer senere slik at alle jobbet omtrent like mye.

Prosjektverktøyet Trello hjalp oss å holde god oversikt over hva alle i gruppen jobbet med til enhver tid. Trello var også til stor nytte for å holde orden på Product Backlog, Sprint Backlog og inkrementer. I Trello var det lett å legge til, fjerne og endre arbeidsoppgaver.

Gruppen har hatt en veldig smidig arbeidsprosess og har hatt klar fremgang for hver sprint. Produkteier har gitt uttrykk for at han har vært fornøyd med hva som ble gjort for hver sprint, og at vi alltid lå i rute med systemutviklingen av prosjektet.

6. Konklusjon og videre arbeid

6.1 Konklusjon

I denne rapporten har vi hatt problemstillingen og forskningsspørsmålene fra kapittel 1.2 som utgangspunkt. Følgende er en liste av forskningsspørsmålene, og en konklusjon til hver, utledet fra kapittel 5.1.2:

1. *Kan en løsning klare å oppdage og varsle om mulige grunnstøtingssituasjoner tidsnok til at aktører får tid til å handle?*
    - Simuleringene av tidligere grunnstøtinger ga resultater som viser at det er mulig å varsle om fare for grunnstøting tidsnok. Til tross for at vår beregning av "tidsnok" har svakheter, er marginene så store, at vi med stor sikkerhet kan si at det er mulig å varsle om en mulig grunnstøtingssituasjon tidsnok.

2. *Kan en løsning klare å prosessere alle relevante data fra Kystverket sin åpne AIS-strøm i sanntid?*
    - Alle tester og observeringer lar oss konkludere med at det er mulig å prosessere alle relevante data fra Kystverket sin åpne AIS-strøm i sanntid.

3. *Kan en løsning tilby nok beslutningsstøtte for en VTS-operatør ved mulige grunnstøtingssituasjoner?*
    - Resultatene fremlagt er avledet til å være tilstrekkelig beslutningsstøtte for en VTS-operatør.

Problemstillingen er formulert som følger: "*Kan en maskinlæringsmodell som skal predikere grunnstøtinger i sanntid operasjonaliseres, og gi rask nok beslutningsstøtte til en VTS-operatør, slik at grunnstøtinger kan avverges?*". Forskningsspørsmålene er utledet fra problemstillingen, og alle spørsmålene er bekreftet. Sammen med resultatene som er fremlagt kan vi med dette konkludere at produktet vi har utviklet er en løsning på det problemstillingen etterspør.

NoIS har satt krav til produktet som skulle utvikles. Alle kravene, med unntak av kravet til grad av brukbarhet og mobilitet, er oppfylt. Årsaken til at disse kravene ikke er oppfylt, er at målene i seg selv var for vanskelige å måle. Vi konkluderer derfor med at produktet er utformet etter oppgavestiller sine ønsker og krav.

Vårt produkt er ikke en ferdigstilt løsning på alle utfordringene til Kystverket, men produktet kan være en puslespillbrikke i deres visjon om å gjøre sjøtrafikken tryggere og redusere antall akutte utslipp rundt Norges kyst.

6.2 Videre arbeid

Det er per i dag ingen planer om å ferdigstille produktet, men det er planer om å ta i bruk produktet som en prototype, og videreutvikle produktet for å teste og utvikle mer funksjonalitet til MADART. Under følger en liste av funksjoner som kan være en del av videreutviklingen av produktet:

1. Integrasjon mot flere tredjeparts-verktøy som SSNN.
2. Utføre brukertester mot målgruppen.
3. Å rulle ut produktet til en skytjeneste som Microsoft Azure.
4. Teste ulike terskelverdier for avvik.

Nøyaktigheten og korrektheten til MADART er en sentral del av funksjonaliteten i produktet. Ved å integrere mot tredjeparts-verktøy som SSNN, så kan MADART få mer informasjon om seilasen, og dermed forbedre resultatene.

Dersom produktet skal ferdigstilles, burde det testes av brukerne som er målgruppen. Tilbakemeldingene vil kunne gi et godt bilde på hvordan produktet kan, eller burde videreutvikles.

For å gjøre produktet mer tilgjengelig for testing og som verktøy, kan produktet rulles ut og gjøres tilgjengelig som en skytjeneste. På den måten blir produktet enklere å ta i bruk, teste, skalere og administrere.

For å optimalisere produktets evne til å varsle burde det gjøres ytterligere tester og simuleringer av grunnstøtinger, hvor produktets følsomhet for avvik justeres. Slik kan man finne den beste terskelverdien.

Videre arbeid som ikke omhandler ny funksjonalitet eller integrasjoner mot andre systemer, burde være å gjennomgå svakheter med systemet, og implementere de anbefalte tiltakene som er beskrevet i 5.2.4, *Svakheter med produktet*.

Dersom noen vil utvikle et tilsvarende produkt, ville vi anbefalt å håndtere hver melding i en tråd, bruke en full dupleks datakommunikasjon mellom klient og tjener og ha en database som effektivt kan håndtere geografiske data.

7. Kilder

1. Johnsrud HJ, Relling T, Merwe F van de, Jonsson H, Lasselle S, Abusdal H. Forebyggende Sjøsikkerhets i norske farvann: I dag og frem mot 2040 [Internett]. DNV GL; 2016 mar. Report No.: 09/2015. Tilgjengelig på: https://www.kystverket.no/globalassets/nyheter/2015/november/prosjektsyntese-sjosikkerhetsanalysen_oppdatert.pdf

2. Peker ut videre kurs for arbeid med sjøsikkerhet [Internett]. Kystverket. 2015 [sitert 14. mai 2019]. Tilgjengelig på: https://www.kystverket.no/Nyheter/2015/November/Peker-ut-videre-kurs-for-arbeid-med-sjosikkerhet/

3. Kystinfo [Internett]. [sitert 9. mai 2019]. Tilgjengelig på: https://kart.kystverket.no/

4. Kjæraas AE. Avverget forurensning i 2016 [Internett]. Kystverket. 2017 [sitert 18. mars 2019]. Tilgjengelig på: https://kystverket.no/Nyheter/2017/januar/akutt-forurensning-2016/

5. Kjæraas AE. Bekymret over antall grunnstøtinger langs kysten [Internett]. Kystverket. 2018 [sitert 18. mars 2019]. Tilgjengelig på: https://www.kystverket.no/Nyheter/2018/januar/hendelsesrapport-2017/

6. Johnsrud HJ, Relling T, van de Merwe F. Vurdering av forebyggende sjøsikkerhetstiltak [Internett]. DNV GL; 2015 mai [sitert 18. mars 2019] s. 53. (Sjøsikkerhetsanalysen 2014). Report No.: 1908Z31-6 . Tilgjengelig på: https://www.kystverket.no/contentassets/f056df3c875140aa98ef49a25cc082c6/7_tiltaksanalyse_effektanalyse-vurdering-av-forebyggende-tiltak.pdf

7. Full City [Internett]. Kystverket. 2011 [sitert 9. mai 2019]. Tilgjengelig på: https://www.kystverket.no/Beredskap/aksjoner/Arkiv-over-aksjoner/Full-City/

8. Kleppe B. AIS om bord i skip [Internett]. Kystverket. 2015 [sitert 14. mai 2019]. Tilgjengelig på: https://www.kystverket.no/Maritime-tjenester/Meldings--og-informasjonstjenester/AIS/AIS-om-bord-i-skip/

9. Wikimedia-prosjekta BT. Grunnlinje (havrett) – Wikipedia [Internett]. Wikimedia Foundation, Inc. 2005 [sitert 16. mai 2019]. Tilgjengelig på: https://no.wikipedia.org/wiki/Grunnlinje_(havrett)

10. Eidhammer T. Bulkbåt på 74 meter grunnstøtte ved Hustadvika [Internett]. smp.no. 2018 [sitert 9. mai 2019]. Tilgjengelig på: https://www.smp.no/nyheter/2018/01/24/Bulkb%C3%A5t-p%C3%A5-74-meter-grunnst%C3%B8tte-ved-Hustadvika-15957706.ece

11. Riveiro M, Pallotta G, Vespe M. Maritime anomaly detection: A review. Wiley Interdiscip Rev Data Min Knowl Discov. 25. mai 2018;8(4):34.

12. Engelschiøn B, Kuosmanen H, Torkelsen M. Simulatorøvelse.pdf.

13. Solheim J. Email thread: Joakim Solheim - Bernhard Engelschiøn. 2019.

14. Stokvik RM. Intervju med Roy Stokvik, Trafikkleder.pdf. 2019.

15. Beck K, Beedle M, van Bennekum A, Cockburn A, Cunningham W, Martin F, mfl. Manifestet for smidig programvareutvikling [Internett]. Agile Manifesto. 2001 [sitert 18. mars 2019]. Tilgjengelig på: https://agilemanifesto.org/iso/no/manifesto.html

16. What is Scrum? [Internett]. Scrum.org. [sitert 5. mai 2019]. Tilgjengelig på: https://www.scrum.org/resources/what-is-scrum

17. Mahalakshmi M, Sundararajan DRM. Traditional SDLC Vs Scrum Methodology – A Comparative Study. International Journal of Emerging Technology and Advanced Engineering [Internett]. juni 2013 [sitert 10. mai 2019];3(6). Tilgjengelig på: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.413.2992&rep=rep1&type=pdf

18. Component Architectures [Internett]. http://www.michael-richardson.com. [sitert 25. mars 2019]. Tilgjengelig på: http://www.michael-richardson.com/processes/rup_classic/core.base_rup/guidances/supportingmaterials/use_component_architectures_CBC2F6B5.html

19. Ambler SW. Mapping Objects to Relational Databases: O/R Mapping In Detail [Internett]. Agile Data. [sitert 2. april 2019]. Tilgjengelig på: http://www.agiledata.org/essays/mappingObjects.html

20. Åsheim H. Brukertilgang til AIS Norge [Internett]. Kystverket. 2016 [sitert 1. mai 2019]. Tilgjengelig på: https://www.kystverket.no/Maritime-tjenester/Meldings--og-informasjonstjenester/AIS/Brukartilgang-til-AIS-Norge/

21. Fette I. The WebSocket Protocol. desember 2011 [sitert 15. mai 2019]; Tilgjengelig på: https://tools.ietf.org/html/rfc6455

22. Chitrapu S. Design Patterns - Singleton Pattern (4 Examples). Windows Dev Center [Internett]. 30. april 2013 [sitert 10. mai 2019]; Tilgjengelig på: https://code.msdn.microsoft.com/windowsapps/Singleton-Pattern-24a1da7f

23. Inversion of Control [Internett]. TutorialsTeacher. [sitert 10. mai 2019]. Tilgjengelig på: https://www.tutorialsteacher.com/ioc/inversion-of-control

24. Dependency Injection [Internett]. DevIQ. [sitert 10. mai 2019]. Tilgjengelig på: https://deviq.com/dependency-injection/

25. CESARDELATORRE. Designing the infrastructure persistence layer [Internett]. [sitert 10. mai 2019]. Tilgjengelig på: https://docs.microsoft.com/en-us/dotnet/standard/microservices-architecture/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design

26. Skaali TB. Concurrency and concurrent systems [Internett]. Lecture #2; 2011 [sitert 10. mai 2019]; University of Oslo. Tilgjengelig på: https://www.uio.no/studier/emner/matnat/fys/FYS4220/h11/undervisningsmateriale/forelesninger-rt/2011-2_Concurrent_systems.pdf

27. Separation of Concerns [Internett]. DevIQ. [sitert 10. mai 2019]. Tilgjengelig på: https://deviq.com/separation-of-concerns/

28. ThoughtWorks. Continuous integration [Internett]. ThoughtWorks. [sitert 9. mai 2019]. Tilgjengelig på: https://www.thoughtworks.com/continuous-integration

29. Ng A. Machine Learning | Lecture 1 - The Motivation & Applications of Machine Learning [Internett]. Stanford Engineering Everywhere; [sitert 10. mai 2019]. Tilgjengelig på: https://see.stanford.edu/Course/CS229/47

30. Datasets and Machine Learning [Internett]. Skymind. [sitert 24. april 2019]. Tilgjengelig på: http://skymind.ai/wiki/datasets-ml

31. Cover TM, Hart PE. Nearest Neighbor Pattern Classification [Internett]. Ben-Gurion University of the Negev; 2018. Tilgjengelig på: https://www.cs.bgu.ac.il/~adsmb182/wiki.files/borak-lecture%20notes.pdf

32. Lutins E. DBSCAN: What is it? When to Use it? How to use it [Internett]. Medium. Medium; 2017 [sitert 14. mai 2019]. Tilgjengelig på: https://medium.com/@elutins/dbscan-what-is-it-when-to-use-it-how-to-use-it-8bd506293818

33. How HDBSCAN Works — hdbscan 0.8.1 documentation [Internett]. [sitert 10. mai 2019]. Tilgjengelig på: https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html

34. Salvador S, Chan P. FastDTW: Toward Accurate Dynamic Time Warping in Linear Time and Space. [sitert 10. mai 2019]; Tilgjengelig på: https://pdfs.semanticscholar.org/05a2/0cde15e172fc82f32774dd0cf4fe5827cad2.pdf

35. Brownlee J. A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning [Internett]. Machine Learning Mastery. 2016 [sitert 6. mai 2019]. Tilgjengelig på: https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/

36. A Kaggle Master Explains Gradient Boosting [Internett]. No Free Hunch. 2017 [sitert 14. mai 2019]. Tilgjengelig på: http://blog.kaggle.com/2017/01/23/a-kaggle-master-explains-gradient-boosting/

37. Dorogush AV, Ershov V, Gulin A. CatBoost: gradient boosting with categorical features support [Internett]. 2018. Tilgjengelig på: https://arxiv.org/pdf/1810.11363.pdf

38. Bli kjent med Azure | Microsoft Azure [Internett]. [sitert 19. mai 2019]. Tilgjengelig på: https://azure.microsoft.com/nb-no/overview/

39. Rick-Anderson. ASP.NET Documentation [Internett]. [sitert 19. mai 2019]. Tilgjengelig på:

https://docs.microsoft.com/en-us/aspnet/

40. Real-time ASP.NET with SignalR [Internett]. Microsoft. [sitert 25. mars 2019]. Tilgjengelig på: https://dotnet.microsoft.com/apps/aspnet/real-time

41. SignalR/SignalR [Internett]. GitHub. [sitert 2. mai 2019]. Tilgjengelig på: https://github.com/SignalR/SignalR

42. rowanmiller. Overview - EF Core [Internett]. [sitert 19. mai 2019]. Tilgjengelig på: https://docs.microsoft.com/en-us/ef/core/

43. PostgreSQL: About [Internett]. [sitert 19. mai 2019]. Tilgjengelig på: https://www.postgresql.org/about/

44. Leaflet — an open-source JavaScript library for interactive maps [Internett]. [sitert 19. mai 2019]. Tilgjengelig på: https://leafletjs.com/index.html

45. Otto M, Thornton J, Bootstrap contributors. Introduction [Internett]. [sitert 19. mai 2019]. Tilgjengelig på: https://getbootstrap.com/docs/4.3/getting-started/introduction/

8. Vedlegg

Vedlegg A: Visjonsdokument

Vedlegg B: Kravdokumentasjon

Vedlegg C: Systemdokumentasjon

Vedlegg D: Prosjekthåndbok

Vedlegg E: Systemtest

Vedlegg F: Simuleringer av utvalgte grunnstøtinger