E39 Fjordkryssing: Automatisering av data-pipeline for analyse av bølgedata

Bacheloroppgave 2019

Oppgave 62

Dataingeniør

NTNU, institutt for datateknologi og informatikk

Trym Vegard Gjelseth-Borgen

Magnus Conrad Hyll

Trondheim, mai 2019

Forord

Denne rapporten omhandler bacheloroppgaven “E39 Fjordkryssing: Automatisering av data-pipeline for analyse av bølgedata”. Formålet med rapporten er å gi innsikt i hvordan oppgaven har blitt løst, bakgrunnen for oppgaven, samt resultater av utviklingen og utviklingsprosessen knyttet til produktet som er laget i løpet av prosjektet.

Oppgaven ble valgt fordi den så omfattende, variert og interessant ut. Den ga oss også mulighet til å jobbe for en bedrift, noe som kom til å gi oss nyttig lærdom både i form av erfaring fra ansatte, men også med tanke på arbeidsprosess. I tillegg til dette var det motiverende at det ferdige produktet kom til å bli tatt i bruk i forbindelse med et stort og omfattende prosjekt.

I forbindelse med utviklingen av produktet og arbeidet med oppgaven vil vi takke følgende personer:

- Andreas Ravnestad fra Norconsult Informasjonssystemer avdeling Fundator, for å ha bidratt som Scrum Master, stilt til teknisk hjelp og veiledning gjennom prosjektet.
- Athul Sasikumar fra Norconsult avdeling Samferdsel, for å ha bidratt som Produkteier og stilt seg til rådighet under utviklingen av produktet.
- Atle Olsø fra NTNU, Institutt for datateknologi og informatikk, for hans rolle som veileder for prosjektet, og bidratt med god kunnskap rundt prosess og dokumentasjon.
- Øvrige på Norconsult Informasjonssystemer avd. Fundator for gode råd og tips innen systemutvikling og maskinlæring.

Dette dokumentet er utarbeidet i henhold til mal gitt av NTNU, Institutt for datateknologi og informatikk.

Trondheim, 20.05.2019

Oppgavetekst

Tittel:

E39 Fjordkryssing: Automatisering av data-pipeline for analyse av bølgedata.

Hensikt med oppgaven:

Automatisere og effektivisere en manuell pipeline for nedlasting, prosessering, analyse og presentasjon av bølgedata fra 8 målestasjoner langs E39.

Beskrivelse av oppgaven:

Norconsult Informasjonssystemer (NoIS) gjennomfører i dag analyser av bølgedata fra 8 målestasjoner langs E39 på Mørekysten. Dette gjøres på oppdrag fra Statens Vegvesen og i samarbeid med Fugro Oceanor. I dag lastes det ned filer i NetCDF-format to ganger i året som går gjennom en manuell prosess for kvalitetssjekk, separasjon av data, og analyse og plotting. Denne prosessen skal automatiseres og effektiviseres, og samtidig skal man utforske muligheter for å anvende maskinlæringsalgoritmer på datasettene.

Sammendrag

Bakgrunnen for dette systemutviklingsprosjektet er Statens Vegvesens prosjekt "Ferjefri E39". I forbindelse med dette har Norconsult, på oppdrag fra Statens Vegvesen og i samarbeid med Fugro Oceanor, blitt leid inn til å gjennomføre analyser av bølge- og vinddata fra 8 målestasjoner langs Europavei 39. Formålet med datainnsamlingen og analysen er å danne et kunnskapsgrunnlag for bølge- og vindforholdene i fjordene langs kysten.

Målet med denne oppgaven stilt av Norconsult er å automatisere og effektivisere en manuell pipeline for nedlasting, prosessering, analyse og presentasjon av bølge- og vinddata. Dette innebærer å hente filer fra en filtjener hvor Fugro Oceanor laster opp data fra de forskjellige bølge- og vindmålingsstasjonene. Filene er lagret på formatet NetCDF, og må derfor konverteres. Etter konvertering skal dataene analyseres, og deretter presenteres på en meningsfull måte som gir innblikk i bølge- og vindforholdene i fjordene hvor målestasjonene er utplassert. I tillegg til dette var det et ønske om å se på muligheten for å bruke maskinlæring til å imputere manglende data.

Systemet som er utviklet er en web-applikasjon hvor ansatte hos Norconsult kan logge seg inn og enkelt generere en rapport. Web-applikasjonen er skrevet i C# og er bygget på ASP.NET Core 2.2. Systemets tjener, database og fillagring blir alle kjørt i skyløsninger hos Microsoft Azure. Web-applikasjonen har et system mellomlagring av rådata og konverterte filer. Systemet holder også kontroll over ferdig genererte rapporter og tilhørende rådatafiler, og kan dermed varsle om en tidligere generert rapport er utdatert grunnet oppdatering i rådatafilene. Maskinlæringsteknikken kNN har blitt implementert for å imputere manglende måledata, og kan skrus av eller på om ønskelig.

Denne rapporten tar for seg arbeidet med utviklingen av systemet og prosessen rundt. Den gir innblikk i hvilke teknologier og prosesser som har blitt brukt, og begrunnelsen for valget av dem. Rapporten inneholder også vitenskapelige, ingeniørfaglige og administrative resultater for prosjektet, og diskusjon rundt disse

Innholdsfortegnelse

Figurliste

Tabelliste

1 Introduksjon og relevans

1.1 Bakgrunn for prosjektet

Bakgrunnen for dette systemutviklingsprosjektet er Statens Vegvesens prosjekt "Ferjefri E39". Ferjefri E39 er et prosjekt med mål om å bygge ut E39 mellom Kristiansand og Trondheim. Veien er 1100 km lang og går gjennom seks fylker. Å kjøre hele strekningen tar i dag 21 timer hvor trafikantene må benytte seg av syv forskjellige ferjesamband for å komme fram. Målet er å bygge ut E39 ved å bytte ut ferjene med bruer og undersjøiske tunneler, og kutte ned veien med om lag 50 kilometer. Disse utbedringene vil kutte reisetiden for bilister i to, og har en prislapp på om lag 340 milliarder kroner [7].

I forbindelse med dette vegprosjektet har Statens Vegvesen leid inn konsulenthjelp fra kystingeniører hos Norconsult, som skal utarbeide rapporter med data fra bølge- og vindmålinger i de aktuelle fjordene [16]. Disse rapportene skal hjelpe til med å ta avgjørelser om hvordan bruene og tunellene burde konstrueres. Nedlasting av data og utarbeiding av rapporter gjøres manuelt, og dagens løsning på oppgaven krever mye tid. På grunn av dette oppstod det et ønske om å automatisere og effektivisere prosessen for å redusere tidsbruk, og eliminere potensielle feilkilder knyttet til menneskelig interaksjon. Samtidig var det et ønske om å utforske muligheter for å anvende maskinlæringsalgoritmer på datasettene. Disse skulle brukes til å fylle hull i datasettene som potensielt kan oppstå ved feil på målestasjonene.

1.2 Akronymer og definisjoner

Attributt
I dette dokumentet definert som de ulike målinger pr. datapunkt generert av en målestasjon, som f.eks. bølgehøyde (målt i meter), vindstyrke (målt i m/s), vindretning (målt i grader), posisjon (angitt med koordinater) osv.

Blob
Binary Large Object. Betegnelse på en binær datatype ved persistering av data.

CRUD
Create, Read, Update og Delete er de fire grunnleggende funksjonene for persistert lagring.

CSV
Comma-separated values er et filformat som bruker komma (eller i vårt tilfelle, semikolon) for å separere verdier.

Datapunkt
I dette dokumentet definert som en samling av måledata gjort av en målestasjon i intervaller på 10 minutter. Hvert datapunkt består av flere attributter.

Datasett
I dette dokumentet definert som settet med datapunkter tilhørende én målestasjon, med data innenfor en gitt tidsperiode.

Jobb
I sammenheng med generering av rapporter defineres en jobb som én kjørende instans av en Pipeline, som genererer rapporter fra gitte målestasjoner for en gitt tidsperiode.

JSON
JavaScript Object Notation er en tekstbasert standard brukt for å formatere meldinger som brukes for datautveksling.

kNN
K-nærmeste naboer. Regresjons- og klassifiseringsalgoritme.

NetCDF-fil
Fil lagret med formatet NetCDF, brukt for lagring av matrise-orientert forskningsdata. NetCDF står for Network Common Data Form, og er en standard utviklet av Unidata [8].

ORM
Object-Relational Mapping. Teknikk for å oversette mellom objekter og data persistert i en relasjonsdatabase.

Pipeline
Sekvensen av stegene som behandler måledataene, fra innlesing av rådatafiler til ferdig genererte Excel-rapporter.

Proxy-tjener
En tjener som står mellom en klient og en annen tjener.

Thredds.met.no
Filserver hos Meteorologisk institutt hvor Fugro Oceanor publiserer rådata fra målestasjonene.

1.3 Problemstilling

Problemstillingen vi ønsker å besvare i dette bachelorprosjektet er følgende:

Hvordan kan en automatisk pipeline og maskinlæring forbedre og effektivisere prosesseringen og analysen av bølge- og vinddata?

Mer konkret ønsker vi å se på og besvare følgende forskningsspørsmål knyttet til problemstillingen:

> Kan man ved hjelp av maskinlæring forbedre rapportene ved å gjøre dem mer komplette?
>
> Er det mulig å redusere tidsbruken en kystingeniør bruker på å lage en rapport?
>
> Er det mulig å forbedre rapportene ved å eliminere feilkilder som følge av menneskelig interaksjon?

2 Teori

2.1 Utviklingsmetoder

2.1.1 SCRUM

Scrum er en arbeidsprosess laget for å håndtere komplekse systemutviklingsprosjekter. Scrum er en iterativ prosess som fokuserer på å få utviklet funksjonerende kode ved bestemte intervaller.

Personene som arbeider i et Scrum-prosjekt kalles Scrum-teamet. Scrum-teamet består av personen som ønsker produktet (Produkteier), en sjef for prosjektet (Scrum Master) og utviklerne. Det er vanlig at det bare er èn Produkteier. Produkteieren bestemmer hva som skal utvikles for hver iterasjon (Sprint), og evaluerer resultatet på slutten av iterasjonen. Personen

som er Scrum Master har ansvaret for å lede utviklingsteamet gjennom prosjektet. Scrum Master har gjerne fått sertifisering som Scrum Master, eller har tidligere erfaring med suksessfull gjennomføring av Scrum-prosjekter [1].

Et Scrum-prosjekt starter med at medlemmene av Scrum-teamet samles, og diskuterer det arbeidet som skal gjøres og hvordan de skal arbeide sammen. Det er viktig at visjonen for prosjektet kommer fram, samt hvilke akseptansekriterier som må oppfylles for at prosjektet skal kunne klassifiseres som vellykket. For å holde orden på kravene til produktet som skal opprettes en Produktbacklog. Produktbacklogen inneholder funksjonaliteten Produkteieren ønsker at produktet skal ha [1].

Utviklingen av systemet foregår i Sprinter. Sprinter er korte tidsperioder, som kan vare fra noen få dager, opp til 30 dager eller mer. Sprint Planning er et møte hvor Scrum-teamet planlegger neste Sprint. Her opprettes en Sprint Backlog. Den inneholder alle krav Scrum-teamet tenker at de klarer å få utviklet i løpet av den kommende sprinten. Utviklerne på Scrum-teamet har ansvaret for å dele kravene opp i mindre, overkommelige deler, og estimere hvor lang tid det tar å utvikle disse. Nøyaktigheten for estimeringen av tid avhenger av hvor erfarne utviklerne er, både i arbeid med Scrum, men også i teknologien de skal jobbe med [1].

Når Sprinten er over møtes Scrum-teamet og diskuterer hva som ble gjort ved forrige sprint, i et Sprint Review-møte. Review-møtet består av å se hva som har blitt gjort, hvor mye som ble gjort og hvor effektivt og nyttig arbeidet var. Om det skal utføres enda en Sprint etter slutten av forrige, utfører Scrum-teamet en Sprint Planning. Her går man gjennom planen for neste Sprint, og hvilken funksjonalitet som skal være med på Sprintens Backlog.

Scrum-teamet går gjennom stegene forklart over, og utfører dem til målene er oppnådd, eller tiden for prosjektet har rent ut [1].

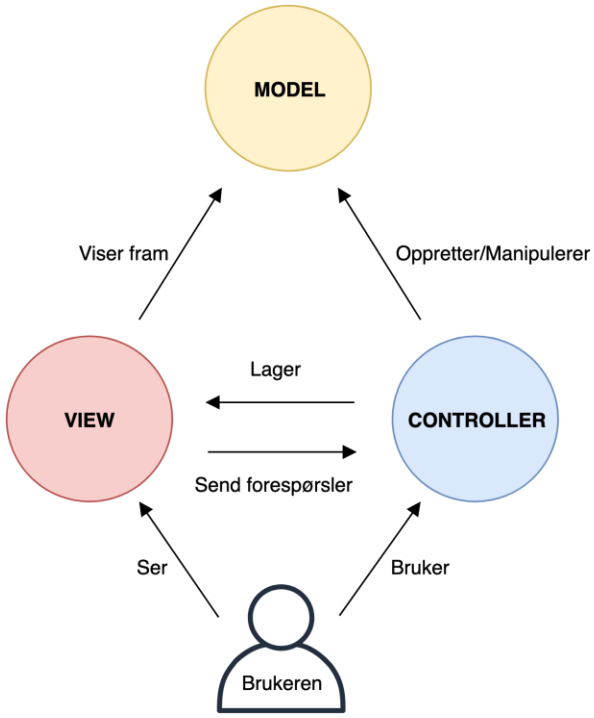
2.2 Programvarearkitektur og designmønstre

2.2.1 MVC (Model-View-Controller)

Model-View-Controller (MVC), er et arkitekturmønster brukt i systemutvikling av applikasjoner med grafisk brukergrensesnitt. Fordelen med MVC er at det hjelper til med å fordele ansvar

(separation of concerns) ved å dele koden opp i tre løst koblete deler. Disse delene er *Model*, *View* og *Controller*. *Model* er klasser som beskriver dataen som skal håndteres, samt regler for hvordan dataene kan bli endret og opprettet. *View* beskriver hvordan applikasjonens brukergrensesnitt skal bli vist fram.*Controller* er klassene som tar hånd om kommunikasjon fra brukeren, flyten i applikasjonen og applikasjonsspesifikk logikk [2].

Med denne tredelingen kan de forskjellige delene utvikles og endres separat fra hverandre. Oppdelingen av koden skaper litt økt kompleksitet, men de positive sidene ved arkitekturmønsteret regnes for å overgå de negative [2].



Figur 2.2.1-1 Figuren over viser i MVC [3, 4]

2.2.2 REST

Representational State Transfer (REST) er en arkitektonisk stil som definerer et sett med regler eller krav brukt for å lage web-tjenester. Web-tjenester som etterkommer disse kravene blir betegnet som RESTful. Det er seks forskjellige krav som må oppfylles, disse er listet under [5, 6].

Separasjon av klient og tjener. Ved å gjøre klient og tjener uavhengig av hverandre, sørger man for at de kan utvikles fritt.

Tilstandsløs. Kommunikasjon mellom klient og tjener skal være tilstandsløs. Dette betyr at hver forespørsel fra klienten til tjeneren må inneholde alt av nødvendig informasjon, og ikke være avhengig av informasjon som ligger på tjeneren. Alt av informasjon som er nødvendig å lagre for en sesjon, må derfor ligge lagret hos klienten.

Hurtiglager. Her kreves det at data i respons fra en tjener eksplisitt merkes som enten hurtiglagerbar, eller ikke hurtiglagerbar. På denne måten øker man effektiviteten til systemet ved at klienten kan gjenbruke data merket som hurtiglagerbar, når like forespørsler gjentas.

Uniformt grensesnitt. Dette lar klienten kommunisere med tjeneren gjennom ett enkelt språk, uavhengig av arkitekturen disse er bygget på. Grensesnittet bør tilby en standardisert måte å kommunisere på mellom klient og tjener. Dette kan gjøres med å bruke HTTP med URI-ressurser, CRUD (se kap. 1.2) og JSON (se kap. 1.2).

Lagdelt system. Mellom klienten som sender en forespørsel og tjeneren som responderer, kan det være flere andre tjenere. Disse tjenerne kan tilby sikkerhet i form av en proxy-tjener (se. kap 1.2), balansering av trafikk eller annen funksjonalitet. For at API-et skal være RESTful skal ikke klient eller tjener bli påvirket av de mellomliggende tjenerne, men kommunisere med hverandre som om de ikke eksisterer.

Code on demand. “Code on demand” er et det eneste valgfrie av de seks kravene. REST tillater utvidelse av klient-funksjonaliteten ved å laste ned og eksekvere kode i form av for eksempel skript.

2.2.3 Designmønsteret Observer

Designmønsteret Observer forsøker å oppnå løse koblinger i et system mellom komponenter som ikke er direkte relaterte, men likevel behøver å sende meldinger til hverandre eller utveksle informasjon. Det implementeres ved at såkalte observatører registrerer seg som lyttere til meldinger fra en tilbyder [10]. Hvilken type melding, og når slike meldinger vil sendes, avhenger helt av tilbyderen. Når tilbyderen ønsker å publisere en melding kaller den en

bestemt metode på hver lytter. På denne måten unngår man tette koblinger og direkte avhengigheter mellom komponentene.

2.2.4 Designmønsteret Singleton

Formålet med designmønsteret Singleton er å sørge for at det bare finnes én instans av en klasse [11]. Dette kan være hensiktsmessig for eksempel dersom man trenger global tilstand i et program, eller et sentralt punkt for tilgangsstyring til en bestemt tjeneste.

2.2.5 Object-Relational Mapping

Object-Relational Mapping (ORM) er en programmeringsteknikk brukt for å konvertere data fra en relasjonsdatabase til objekter i kode. Måten en relasjonsdatabase lagrer data på gjør at dataene ikke direkte kan konverteres fra tabellformatet de ligger lagret på, til objektmodellen kjent fra objektorientert programmering [18]. Ved å bruke en ORM får man et abstraksjonslag mellom relasjonsdatabasen og koden. ORM-en sørger for å gjøre om tabellene i relasjonsdatabasen til objekter, slik at logikken i koden slipper å forholde seg til relasjonsdatabasens tabellformat. ORM-en tilbyr wrapper-funksjoner rundt tradisjonelle SQL-spørringer som gjør det enkelt å aksessere objekter direkte fra databasen [19].

2.3 Rådata fra bølge- og vindmålinger

I dette prosjektet jobbet vi med måledata samlet inn fra bølge- og vindmålere i forskjellige fjorder langs Mørekysten. Disse målestasjonene lages og driftes av Fugro Oceanor, og rådataene som samles inn av stasjonene behandles og distribueres av Meteorologisk Institutt. Rådataene distribueres som NetCDF-filer, inndelt i stasjon og perioder på én måned. Datasettene består av bølge- og vinddata over tid fra de forskjellige målestasjonene, hvor et datapunkt består av gjennomsnittet av målingene gjort i løpet av et intervall på 10 minutter. De inneholder 22 attributter (attributt, se kap. 1.2) for blant annet retning og høyde på bølger og hastighet på vind. I tillegg finnes det en attributt med tidspunkt for målingen, samt attributter for posisjon med lengde- og breddegrad til målestasjonen.

2.4 Imputering og kNN-regresjon

Imputering er innen statistikk metoder for å estimere manglende data i et datasett [12]. Det finnes flere måter dette kan gjøres på, som alt fra å klone verdiene til et eksisterende datapunkt

og benytte disse for det manglende datapunktet, til å benytte maskinlæringsmetoder for å estimere de manglende verdiene ut i fra eksisterende attributter.

K-nærmeste naboer (kNN), fra engelsk k-nearest neighbors, er en enkel maskinlæringsalgoritme for regresjon og klassifisering av data. Algoritmen går i enkle trekk ut på å finne de *k* nærmeste naboene til et manglende datapunkt, og benytte disse til å klassifisere eller estimere verdier for et manglende datapunkt. Ved kNN-regresjon fylles attributtene til datapunktet inn med gjennomsnittet av hver attributt i nabo-datapunktene. En utvidelse av dette er å vektlegge naboene ut ifra avstanden deres til datapunktet, slik at nærmere naboer i større grad innvirker på det imputerte datapunktet [20].

kNN krever en distanse-metrikk som måler avstand mellom datapunkter, for å avgjøre hvilke andre datapunkter er de nærmeste naboene. Den Euklidske distansen er mye brukt som distanse-metrikk [14]. FORMEL?

kNN-algoritmen er såkalt instans-basert, som som betyr at den ikke behøver å trene opp en modell med bruk av treningsdata i forkant. Istedenfor benytter den dataene i det gjeldende datasettet samtidig som den klassifiserer eller imputerer en gitt input [14]. Dette gjør kNN fleksibel til endringer i attributter i datapunktene, og hvilke attributter som medregnes i distanse-metrikken [9].

Hyperparameteren *k* må velges på forhånd, ut i fra den verdien som gir minst avvik mellom imputerte verdier og faktiske verdier. Dette testes og avgjøres gjennom kryssvalidering [14].

3 Valg av teknologi og metode

3.1 Valg av utviklingsmetode

Utviklingsmetoden for prosjektet ble valgt på bakgrunn av omstendighetene rundt prosjektet, og tidligere erfaringer med systemutvikling. Sluttbrukeren vi jobbet med, som er kystingeniør i Norconsult, hadde ingen klare krav for systemet annet enn å få automatisert prosessen for nedlasting av data og utarbeiding av rapporter. Dette førte til at vi hadde behov for en smidig utviklingsmetode, ettersom kravene for produktet ville bli utformet underveis i prosessen. Oppgavestiller hadde ekspertise innenfor utviklingsmetodikken Scrum, og kunne derfor bidra

med hjelp om vi valgte den. Vi hadde også tidligere utviklet i Scrum, og valgte derfor å bruke Scrum som utviklingsmetode.

For at utviklingsmetoden skulle vært mer tilpasset prosjektet vi skulle jobbe med, ble noen deler av Scrum utelatt. Vi valgte å ekskludere tidsestimering for utvikling av funksjonalitet og "burn down chart", ettersom teknologien vi skulle bruke til utvikling var helt ny for oss, og ingen av gruppemedlemmene hadde grunnlag for å estimere tidsbruk for oppgavene.

User stories ble definert ganske sent i prosjektet, og står beskrevet i kravdokumentet (vedlegg B, kapittel 2). Disse ble brukt for å holde oversikt over kravene til produkteier, men ble ikke brukt aktivt i prosessen. I Sprint boardene brukte vi *Sprint tasks* som ikke var direkte knyttet mot en User story. Dette var fordi User storyene kom på plass så sent som de gjorde.

3.2 Valg av plattform

Det var ikke spesifisert eller ønsket en spesifikk plattform applikasjonen skulle kjøres på. Vi valgte derfor å utvikle systemet som en web-applikasjon, da dette hadde flere fordeler sammenlignet med en skrivebordsapplikasjon. Spesifikt valgte vi å utvikle web-applikasjonen i Microsofts ASP.NET Core-rammeverk.

Et viktig argument for å velge web-applikasjon var at det er enklere for brukeren å besøke en nettside, enn å måtte laste ned en programvare for å bruke systemet. Dette knyttet vi til problemstillingen og forskningsspørsmålet om å få ned tidsbruken for utarbeiding av rapporter. I tillegg ville det gjøre det enklere å distribuere eventuelle oppdateringer til applikasjonen, siden nyeste versjon kan publiseres direkte og være tilgjengelig for brukeren ved neste besøk på websiden, istedet for at brukeren må laste ned ny programvare.

Videre begrunner vi valget i muligheten for å la komponenter i systemet kjøre kontinuerlig eller periodisk i bakgrunnen. Dette gjøres mulig siden web-tjeneren vil være oppe og kjøre hele tiden, i motsetning til en arbeidsstasjon som bare kjører under bruk. Vi drar nytte av dette for blant annet å kjøre mellomlagringsfunksjonen (se kapittel 4.2.1.7), samt for å la generering av rapporter kjøres i bakgrunnen, uavhengig om brukeren har nettleseren åpen eller ikke.

3.3 Valg av maskinlæringsalgoritme for imputering

I dette prosjektet jobbet vi med datasett som fra tid til annen mangler datapunkter. Det var ønsket å se på muligheten til å imputere disse manglende dataene ved hjelp av maskinlæring, for å prøve å gjøre de genererte rapportene mer komplette. Likevel var dette bare en ønsket tilleggsfunksjon, og kundens viktigste behov var å få automatisert rapportgenereringen. Ut i fra dette vurderte vi at omfanget av denne delen av oppgaven måtte reduseres henholdsvis, for å forsikre nok tid til implementering av hovedfunksjonalitetene. Vi valgte å benytte kNN-regresjon for å imputere manglende datapunkter da dette er en rimelig enkel algoritme å implementere, og har blitt benyttet til imputering i tidligere studier [9, 21].

3.4 Valg av arkitektur og designmønstre

3.4.1 MVC

Vi har valgt å bruke en Model-View-Controller-arkitektur for web-applikasjonsdelen av systemet. En av grunnene til dette var at det er enkelt å sette opp med rammeverket ASP.NET Core MVC gjennom Visual Studio 2017, og sparte oss dermed tid.

En annen grunn til at vi har valgt å bruke MVC er at vi kan dra nytte av dens oppdelte arkitektur, slik at teamet kan jobbe med deler av web-applikasjonen i parallell. Slik unngår vi unødvendig ventetid innad i teamet, og kan dermed utvikle koden mer effektivt.

3.4.2 REST

REST ble valgt som arkitekturmønster fordi det var kjent teknologi vi allerede hadde god kunnskap om. I tillegg er det enkelt å sette opp REST-grensesnitt gjennom Controllers i ASP.NET Core MVC, noe som underbygget valget av dette arkitekturmønsteret.

3.4.3 Observer-pattern

Web-applikasjonen består av forskjellige komponenter med forskjellig ansvar. En del av applikasjonen tar seg av kjøring av jobber (jobb, se kap. 1.2), mens en annen sender meldinger og status fra jobber til klientens nettleser for å holde brukeren oppdatert. Vi valgte å la disse komponentene utveksle meldinger gjennom å bruke designmønsteret Observer. Dette var

gunstig for å oppnå en løs kobling mellom disse komponentene, noe som er en fordel da komponentene har forskjellige oppgaver og ansvar.

3.4.4 Singleton-pattern

I web-applikasjonen er det nødvendig å ha en samlet oversikt over alle kjørende jobber (jobb, se kap. 1.2), i form av en sentral klasse som håndterer opprettelse av og har referanser til kjørende jobber. Det er derfor viktig at alle som ønsker tilgang til denne klassen, har en referanse til en og samme instans. Vi valgte derfor å la denne klassen følge designmønsteret Singleton, for å sikre at det bare eksisterer én instans av denne klassen. Dermed vil alle deler av applikasjonen med behov for tjenester fra denne klassen alltid referere til samme instans.

Mer spesifikt oppnår vi egenskapene til en Singleton gjennom ASP.NET Core sitt Dependency Injection-system, hvor en kan registrere en tjeneste-klasse med Singleton-livstid [15].

3.4.5 Oppdeling i klassebibliotek og web-grensesnitt

I starten av prosessen var det ikke bestemt hvilken type applikasjon og plattform vi skulle utvikle for. Uavhengig av dette ville applikasjonen ha samme kjernefunksjonalitet, som tar for seg ned- og innlasting av rådata, konvertering, imputering og skriving til rapportmaler. Vi kom derfor frem til at det var fornuftig å lage disse grunnleggende funksjonene i form av et uavhengig klassebibliotek. Dette ville la oss utvikle hovedfunksjonaliteten til systemet uavhengig av brukergrensesnitt og plattform. Når vi senere skulle bestemme oss for plattform, kunne vi utvikle dette som et grensesnitt mot klassebiblioteket, uten å måtte gjøre endringer i grunnfunksjonene.

3.4.6 ORM

Systemet bruker en SQL-basert relasjonsdatabase for å persistere informasjon om genererte rapporter, rådatafiler, og relasjonene mellom dem. Tradisjonelt skriver man SQL-spørresetninger for å hente ut og lagre informasjon i databasen, men å benytte SQL-spørringer direkte i programkoden kan medføre noen ulemper. En risiko det kan medføre er å potensielt åpne for SQL-injiseringer. En annen ulempe er at programkoden blir tett knyttet til det spesifikke databasesystemet og SQL-dialekt. For å unngå dette har vi valgt å benytte en ORM.

3.5 Valg av teknologi

3.5.1 Entity Framework Core

Vi valgte spesifikt å benytte ORM-en Entity Framework Core. Alle kall mot databasen gjøres som metodekall mot EF Core, som gir oss dataene automatisk representert som C#-objekter. Dette hjelper i tillegg på å gjøre koden mer lettlest. Videre har EF Core en egenskap som kalles Migrations, hvor rammeverket sørger for å oppdatere databasestrukturen ut i fra eventuelle endringer i modellene som representerer de persisterte dataene. Dette passet godt til utviklingsprosessen vår, da vi regnet med at det kunne bli endringer i datamodellene underveis.

3.5.2 SignalR

Vi valgte å benytte programvarebiblioteket SignalR. Dette rammeverket lar webtjeneren sende meldinger til klienten i sanntid, i motsetning til at brukeren måtte laste inn nettsiden på nytt for hver oppdatering. Vi benytter dette for å fortløpende holde brukeren oppdatert på statusen til generering av rapporter, uten at brukeren må oppdatere nettsiden. Dette knytter vi til kravet om tilbakemelding under generering av rapporter (vedlegg B, tabell 2.2.2-B).

3.5.3 Python

Rådataene som systemet benytter lagres og distribueres i NetCDF-format, men støtte for dette formatet er begrenset. Python-skriptspråket har et rikt pakkebibliotek med god støtte for mange teknologier, hvorav det finnes støtte for lesing av NetCDF-filer. På grunn av begrenset støtte for dette formatet i .NET-rammeverket som systemet benytter, valgte vi å benytte et Python-skript for å konvertere rådataene fra NetCDF-formatet til det mer utbredte CSV-formatet. Dermed kunne systemet lese inn rådataene fra CSV og jobbe ut i fra det. Å lese inn NetCDF-filene direkte fra .NET-applikasjonen ble satt på vent, med håp om at vi ville få på plass dette senere.

3.6 Valg av databasesystemer

Når det kom til lagring av filer og data i systemet valgte vi å dele dette opp i to. Metadata om filene og knytningene mellom de forskjellige fil-entitetene ligger lagret separat fra de faktiske filene. Siden størrelsene på rapportene, NetCDF-filene og CSV-filene kunne bli ganske store, kom vi fram til at det ikke ville være hensiktsmessig å lagre dem i en relasjonsdatabase,

ettersom slike databaser ikke er beregnet for store filer. Filene ligger derfor lagret i Microsoft Azure Blob Storage (se blob, kap. 1.2), og metadata i Microsoft Azure SQL Database.

Ved å lagre filene i blob-lagring på Microsoft Azure slipper vi å tenke på problemer knyttet til samtidig skriving og lesing. Dette er problemer som kunne oppstått hvis filene hadde ligget lagret lokalt på tjeneren. Dette valget er knyttet opp mot kravet om støtte for flere samtidige brukere (vedlegg A, kapittel 6.3).

3.7 Valg av skytjenester

Tjeneren, SQL-databasen og blob-lagringen ligger alle i Microsofts skytjeneste Azure. Ved å bruke skytjenester for disse løsningene får vi en plattform hvor systemet kan skaleres etter behov. Ekstra minne eller prosessorkraft til tjeneren, eller størrelse på SQL-databasen kan enkelt allokeres.

3.8 Valg av arbeids- og rollefordeling

Ettersom at gruppen bare besto av to personer fant vi ut at det ikke var hensiktsmessig å fordele noen faste arbeidsoppgaver før prosjektets start. Ettersom oppgaver dukket opp gjennom prosjektet har den personen med mest kunnskap eller tid tatt hånd om disse.

4 Resultater

4.1 Vitenskapelige resultater

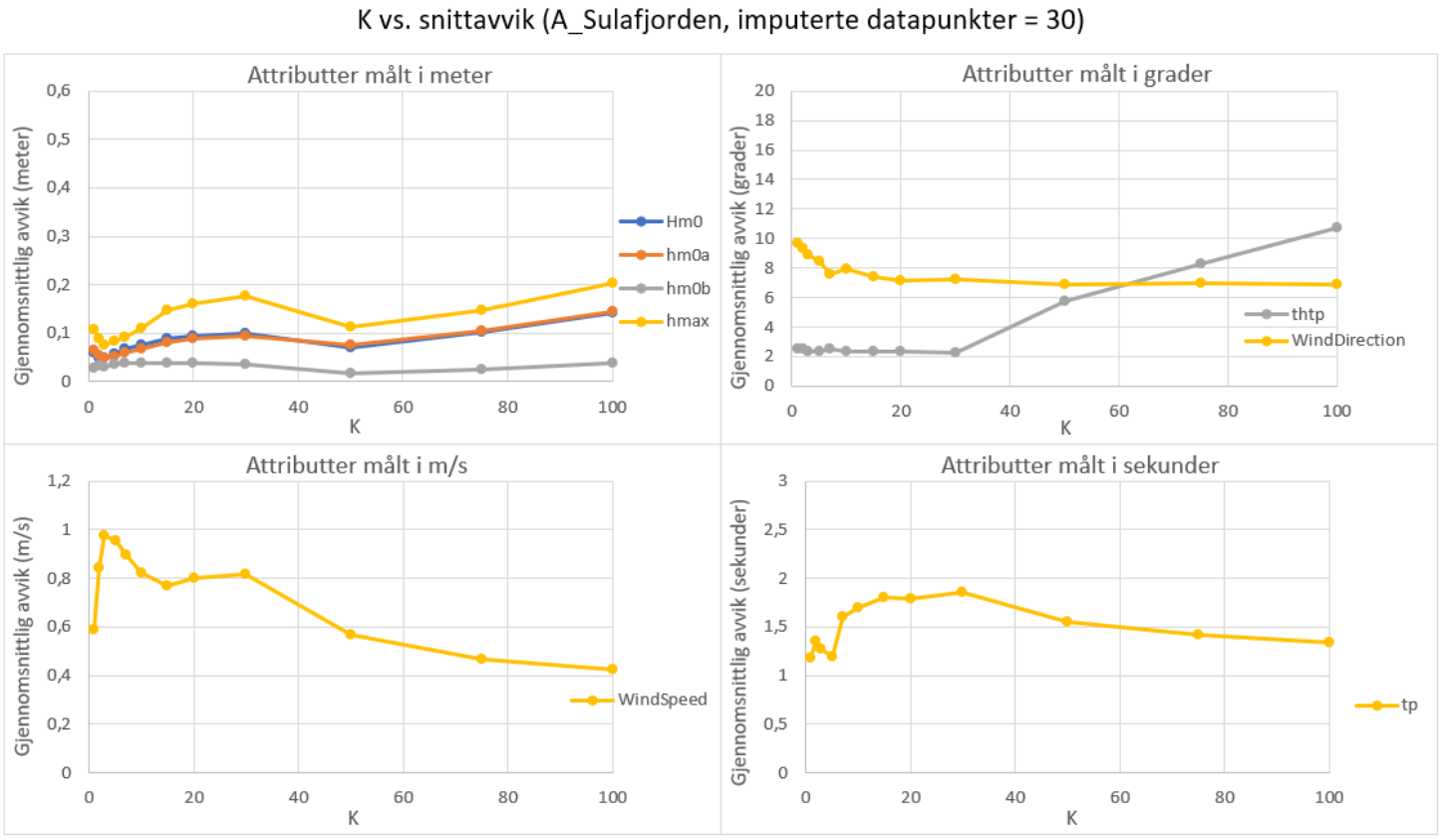
I dette delkapittelet beskrives de vitenskapelige resultatene funnet i forbindelse med problemstillingen og dens forskningsspørsmål stilt i kapittel 1.4.

4.1.1 Forbedre rapporter ved hjelp av maskinlæring

For å finne ut om maskinlæringsalgoritmen kan gjøre rapportene mer komplette har vi validert hvor godt den estimerer manglende data ved imputering. Her har vi gjort flere tester, hvor vi har fjernet et varierende antall påfølgende datapunkter fra flere datasett med eksisterende data. Dette gjorde vi for å simulere forskjellige størrelser av hull i datasettene, da dette gjenspeiler det som kan oppstå i de virkelige datasettene. I tillegg har vi testet med forskjellige verdier for *k*, for å finne en optimal verdi som gir mest nøyaktige estimater. I hver test har vi latt kNN-regresjonen imputere de fjernede datapunktene. Dermed har vi regnet ut gjennomsnittlig

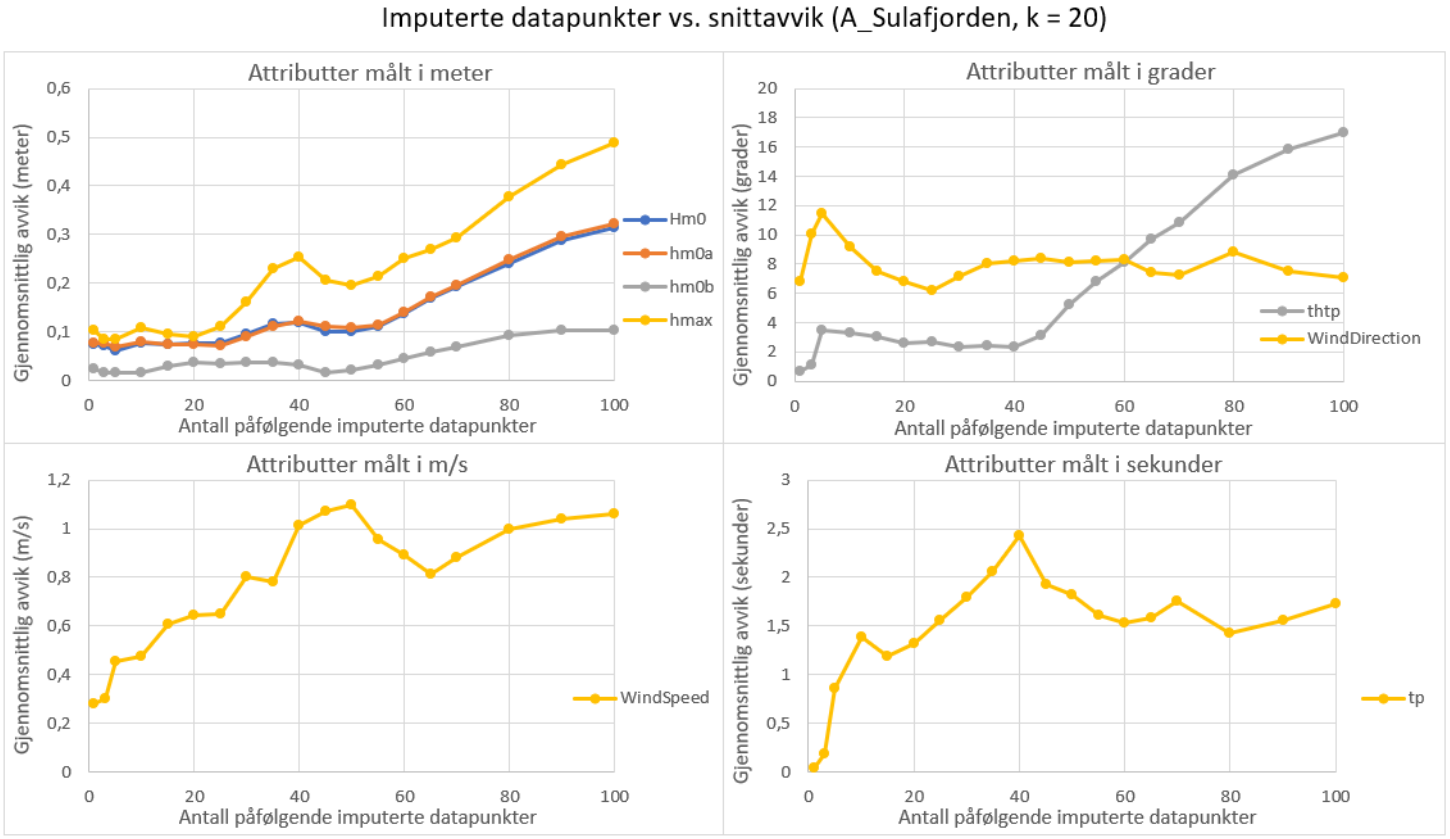
avvik mot de opprinnelige datapunktene for å se når kNN gir mest nøyaktige estimater. Datasettene som er benyttet i testene er fra januar 2018, og grafene viser resultater etter imputering av manglende datapunkter for målestasjon A_Sulafjorden.

Følgende grafer viser gjennomsnittlig avvik mellom imputerte og faktiske verdier med varierende *k*, når det mangler 30 påfølgende datapunkter. Attributtene som er plottet er de samme som benyttes i de genererte rapportene, og er gruppert i grafer etter måleenhet.



Figur 4.1.1-1 Grafer K vs. snittavvik

Ut i fra resultatene kan vi se at gjennomsnittlig avvik for de forskjellige atributtene varierer ulikt for lik k.



Figur 4.1.1-2 Grafer imputerte datapunkter vs. snittavvik

Følgende grafer viser gjennomsnittlig avvik mellom imputerte og faktiske verdier med varierende antall imputerte datapunkter, med *k* = 20. Attributtene er også her gruppert etter måleenhet.

Ut i fra resultatene over ser vi at gjennomsnittlig avvik for de forskjellige attributtene, har en tendens til å stige når antall påfølgende imputerte datapunkter stiger.

4.1.2 Redusere tidsbruk for å lage en rapport

Et av spørsmålene knyttet til problemstillingen var "*Kan vi redusere tidsbruken som en kystingeniør bruker på å lage en rapport?*". Etter en samtale med kystingeniøren som lager analyserapportene manuelt, finner vi ut at å lage én rapport for én målestasjon for en periode på 6 måneder, tar mellom 1 og 2 timer. Arbeidet med å lage en rapport innebærer å overføre rådata fra en Excel-fil til fire ulike maler. Etter at det er gjort, må malene tilpasses dataene som blir brukt.

Vi har automatisert hele denne prosessen. Tabell 4.1.2-A viser en sammenligning mellom tiden brukt for å lage rapporter manuelt og tiden systemet bruker på å lage rapporter. Her har vi satt anslått tid for å lage én rapport til å være 60 minutter (lavt estimat).

Tabell 4.1.2-A Tabell som viser anslått tid for å lage rapport manuelt mot tiden systemet bruker.

Antall rapporter	Anslått tid for å lage rapporten(e) manuelt	Tiden systemet bruker på å lage rapporten(e)	Differanse i tid
1	60 minutter	7 minutter	53 minutter
3	180 minutter	10 minutter	170 minutter
7	420 minutter	16 minutter	404 minutter

Som vi kan se ut i fra tabellen har tidene for å generere rapporter blitt redusert. Mens tidsbruken for å lage en rapport manuelt øker proporsjonalt med antall rapporter, ser vi at tiden systemet bruker ikke har den samme proporsjonale økningen. Gjennomsnittstiden per rapport synker fra 7 minutter når det genereres en rapport, til ca. 2 minutter og 20 sekunder når det genereres 7 rapporter.

4.1.3 4.1.3 Forbedre rapporter ved å eliminere potensielle menneskelige feil

Problemstillingen i kapittel 1.3 spør "*Hvordan kan en automatisk pipeline forbedre og effektivisere prosesseringen og analysen av bølge- og vinddata?*". I forbindelse med delen av problemstillingen som omhandler forbedring, har vi stilt spørsmålet "*Er det mulig å forbedre rapportene ved å eliminere feilkilder som følge av menneskelig interaksjon?*".

Mellomlagringsfunksjonen beskrevet i kapittel 4.2.1.7 henter og mellomlagrer de nyeste rådatafilene fra thredds.met.no. På den måten sørger systemet for at det alltid er de nyeste og sist oppdaterte filene som blir brukt når det genereres en rapport. Den manuelle prosessen for å lage en rapport innebar å få tilsendt rådata direkte fra Fugro Oceanor. Disse filene er ikke alltid helt oppdatert, og kan mangle data. I en test sammenlignet kystingeniøren filene tilsendt fra Fugro Oceanor med filene lagret i mellomlagringen. Resultatet viste at de mellomlagrede filene manglet litt over 50 datapunkter, mens filene fra Fugro Oceanor manglet i overkant av 350 datapunkter. Mellomlagringsfunksjonen holder også orden på tidligere genererte rapporter

ved å gi dem en status (se kapittel 4.2.1.7). Hvis en rapport får status "Outdated" vil kystingeniøren nå vite at den tidligere genererte rapporten ikke lenger samsvarer med rådataene. På den måten gir systemet mulighet til å forbedre tidligere genererte rapporter.

Når rapportene blir laget manuelt, må data fra en tilsendt rådatafil kopieres over i en mal. For at malen skal være brukbar må den tilpasses datamengden manuelt. Malen inneholder funksjonalitet som legger til tomme rader hvor det er mangler i datasettene. Når dette er gjort kan dataene kopieres fra den første malen, over i de resterende tre malene. Prosessen med å lage rapportene manuelt inneholder mange steg hvor det er muligheter for feil som følge av menneskelig interaksjon. Systemet som har blitt laget eliminerer alle disse potensielle feilene, ved å fjerne mennesket fullstendig fra prosessen. Dataene lastes inn i malene og tilpasses datamengden automatisk.

4.2 Ingeniørfaglige resultater

Målet med prosjektet var å utvikle en automatisk pipeline for å generere rapporter for bølge- og vinddata. Systemet skulle også ved ønske kunne benytte seg av maskinlæring for å imputere manglende data. Både visjonsdokumentet og kravspesifikasjonen inneholder krav for produktet som skulle utvikles. Visjonsdokumentet ble laget i en tidlig fase av utvikling og inneholder de aller viktigste kravene for systemet. Kravspesifikasjonen inneholder flere nye og mer detaljerte krav, og har blitt oppdatert gjennom prosessen.

Under beskrives status for målene beskrevet i visjonsdokumentet og kravspesifikasjonen ved leveringstidspunkt for prosjektet.

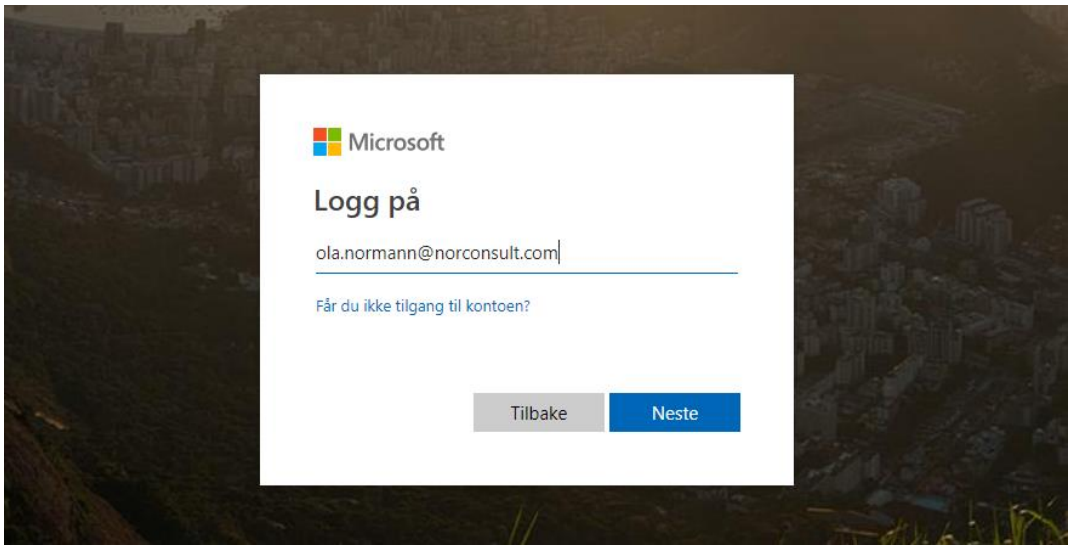
4.2.1 Funksjonelle krav

I dette kapittelet vil måloppnåelsen for kravene spesifisert gjennom User stories i kravdokumentet (vedlegg B, kap. 2) gjennomgås. User stories i kravspesifikasjonen dekker de funksjonelle kravene beskrevet i visjonsdokumentet (vedlegg A, kap. 5).

4.2.1.1 Tilgang til systemet

Brukere kan logge seg inn og ut av web-applikasjonen ved å bruke sin egen Norconsult e-postadresse. Når man går inn på nettsiden vil man bli videresendt til Microsoft sin Account

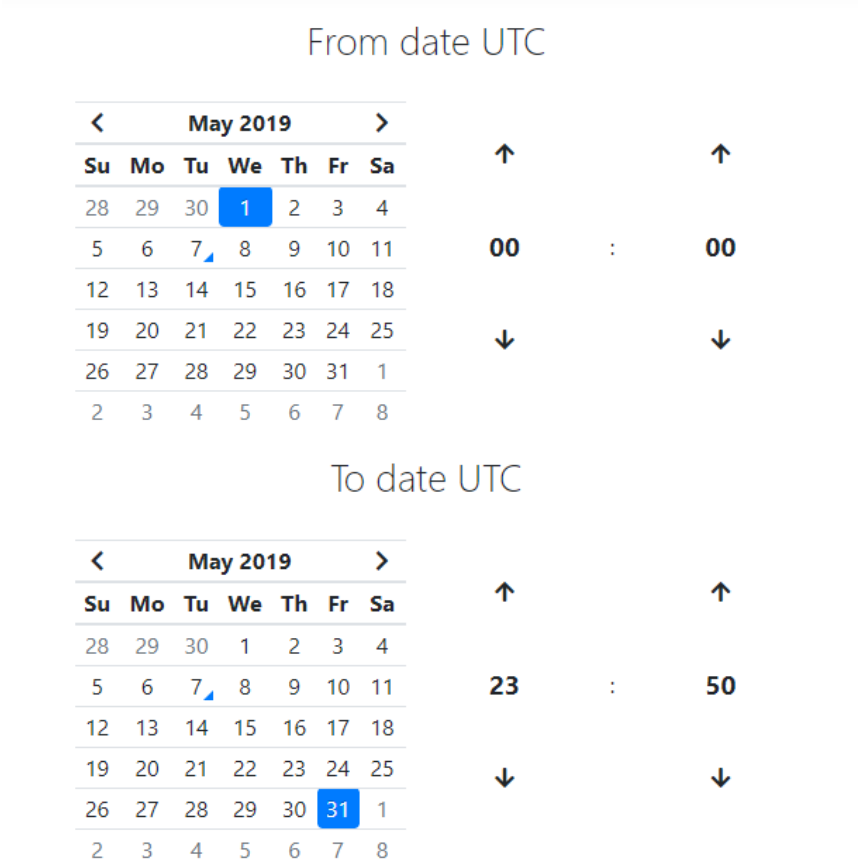
Login-side. Her skriver brukeren inn sin egen Norconsult e-postadresse og tilhørende passord, og vil deretter blir videresendt til web-applikasjonen. Hvis brukeren allerede er logget inn vil man hoppe over dette steget, og bli ført direkte til web-applikasjonen.



Figur 4.2.1-1 Innloggingsskjerm for systemet.

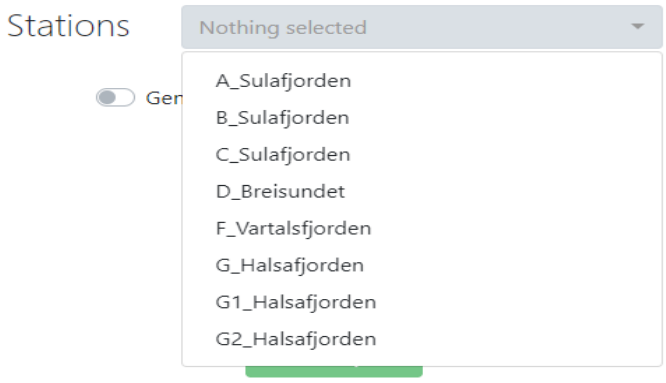
4.2.1.2 Oppsett av rapporter

Brukeren kan velge start- og slutt-tidspunkt for rapporten gjennom to dato- og tidspunkt-velgere på “Create report”-siden. Tiden de velger er på tidsformatet UTC, og kan spesifiseres ned til 10-minutters intervaller.



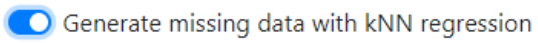
Figur 4.2.1-2 Dato- og tidvelgere.

Ved hjelp av en flervalgs nedtrekksliste er det mulig å velge hvilke stasjoner det er ønskelig å generere rapporter for.



Figur 4.2.1-3 Nedtrekksliste for valg av stasjon.

En bryter under nedtrekkslisten gjør det mulig å spesifisere om det ønskes å bruke kNN for å imputere manglende data eller ikke.



Figur 4.2.1-4 Bryter for å slå av og på kNN-regresjon.

Imputerte verdier vil i rapporten være markert med gul farge.

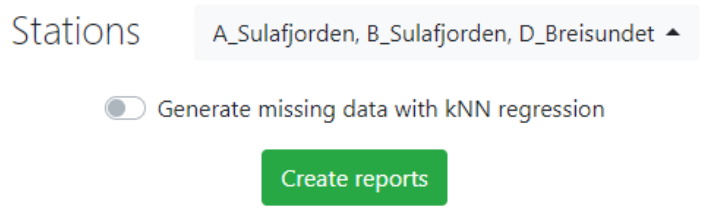
2019-04-04 15:40	0,5127	0,41016	0,30762	0,80566	62,42704391	6,04288101	293,2031	291,0938	298,125	25,3125
2019-04-04 15:50	0,5127	0,41016	0,30762	0,78125	62,42702484	6,04276657	292,5	290,3906	296,0156	18,9844
2019-04-04 16:00	0,4834	0,36621	0,32227	0,78125	62,42702484	6,04276657	292,5	291,7969	293,2031	21,0938
2019-04-04 16:10	0,497622	0,387063	0,314357	0,775192	62,42700418	6,042814253	292,6026	290,84738	295,49219	22,34094
2019-04-04 16:20	0,505056	0,393441	0,314308	0,80354	62,42698351	6,042861937	293,5317	291,55991	296,65763	20,93339
2019-04-04 16:30	0,502874	0,387828	0,316551	0,79822	62,42696285	6,04290962	294,8752	292,76581	297,56764	19,65329
2019-04-04 16:40	0,517156	0,398692	0,323063	0,798233	62,42694219	6,042957303	296,1379	293,89098	298,67526	19,06089
2019-04-04 16:50	0,521465	0,403955	0,321198	0,812612	62,42692152	6,043004987	296,806	294,15774	299,88531	18,63038
2019-04-04 17:00	0,5127	0,39551	0,30762	0,85449	62,42690086	6,04305267	298,125	294,6094	302,3438	14,7656
2019-04-04 17:10	0,49805	0,36621	0,33691	0,75684	62,42690086	6,04305267	297,4219	295,3125	298,125	20,3906

Figur 4.2.1-5 Utdrag fra rapport med imputerte verdier.

4.2.1.3 Generering av rapporter

Ved å velge flere stasjoner i nedtrekkslisten og trykke på “Create reports”-knappen vil det genereres fire rapporter per stasjon. Disse vil være i henhold til malene vi har fått av kystingeniørene.

Klassebiblioteket E39_Lib tar hånd om sammenslåing av datasett fra forskjellige rådatafiler. Den sørger for at rapportene inneholder en rad med data for hvert tiende minutt, mellom valgt start- og sluttdato. Datapunkter som mangler i rådatafilene vil bli lagt inn i rapporten med bare dato og tids-kolonnen utfylt.

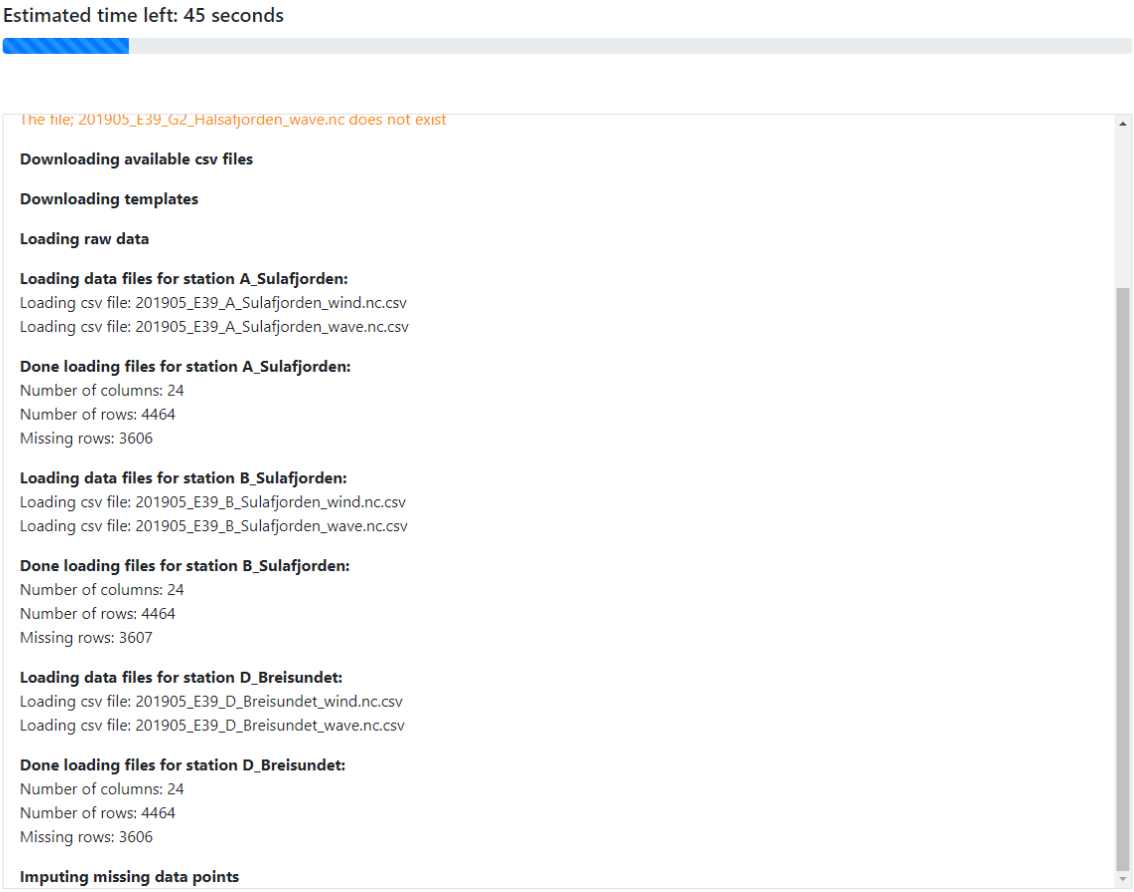


Figur 4.2.1-6 Knapp for å generere rapporter, samt nedtrekksliste med flere valgte stasjoner.

Mens rapporten genereres vil brukeren få tilbakemelding om hva som skjer i en boks til høyre i skjermbildet. Infoboksen vil inneholde nyttig informasjon om status for generering. Nærmere bestemt vil den inneholde informasjon om:

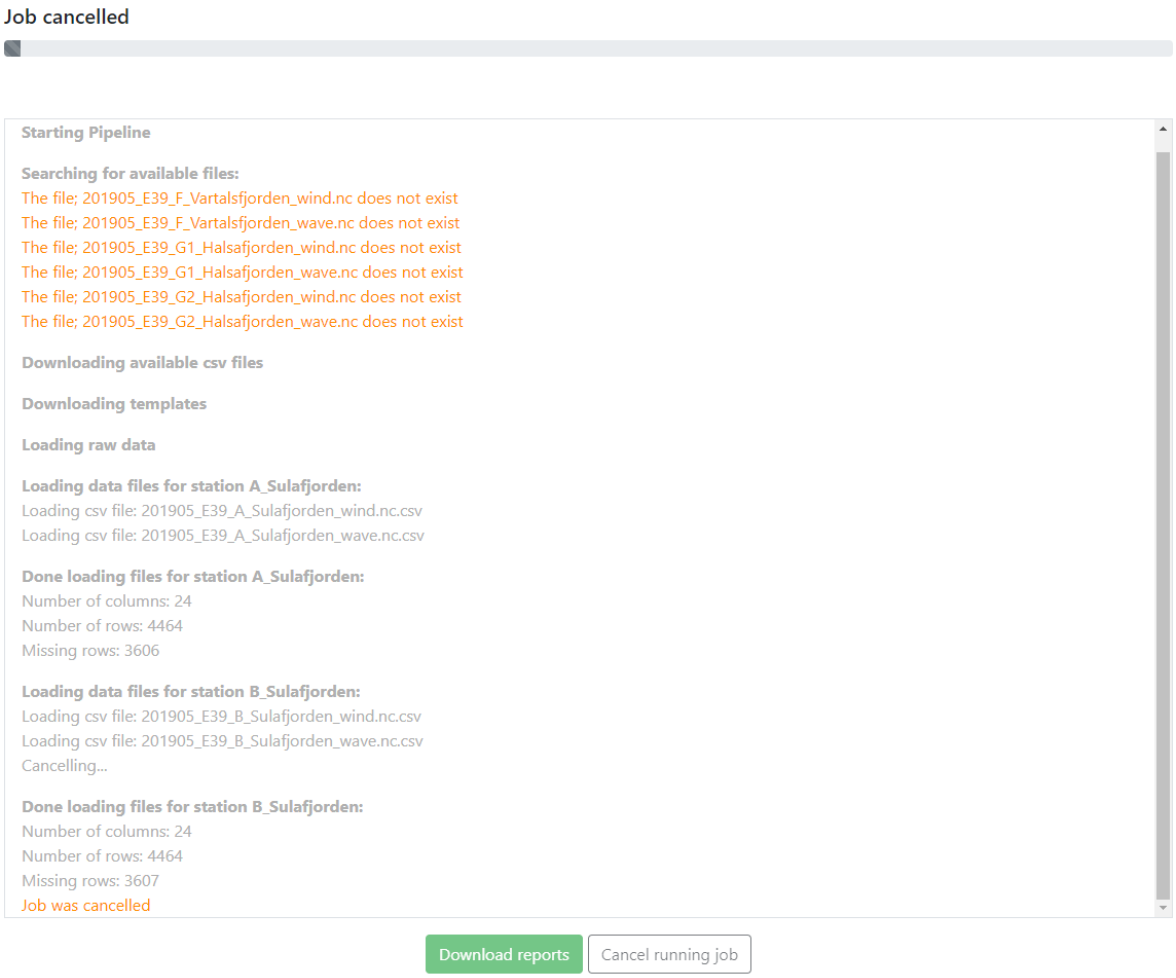
- Progresjon for genereringen
- Hvilke rådatafiler det ikke var mulig å få tak i
- Hvilke konverterte CSV-filer som brukes for hver stasjon
- Antall rader, manglende rader og kolonner for hver rapport
- Informasjon om en rapport ikke kunne bli generert, grunnet manglende rådatafiler
- Status etter ferdig generering
- Benyttelse av manuelt opplastede filer

Over infoboksen står det et estimat for hvor lang tid som gjenstår av genereringen.



Figur 4.2.1-7 Infoboks med estimert tid og progresjonsbar over.

Hvis man ønsker å avbryte en kjørende jobb kan man trykke på “Cancel running job”-knappen. Jobben vil da avslutte seg selv så snart som mulig.



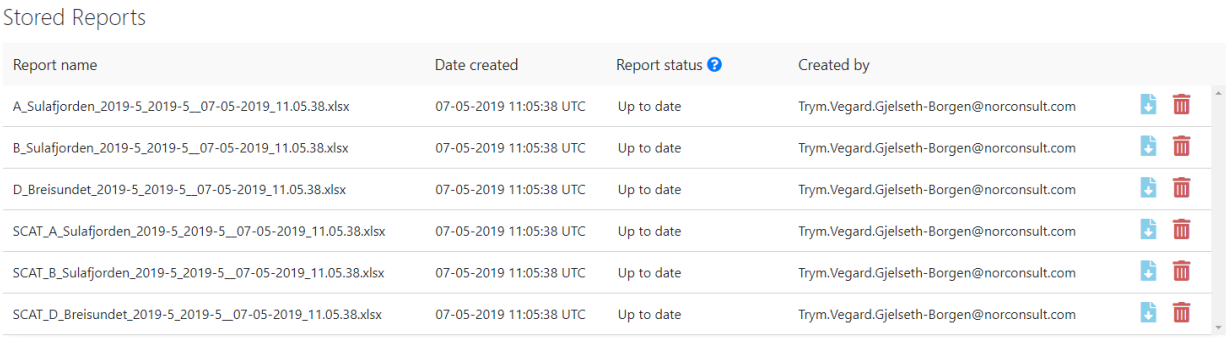
Figur 4.2.1-8 Infoboks etter "Cancel running job"-knapper har blitt trykt, og jobben har blitt kansellert.

4.2.1.4 Nedlasting av rapporter

Når genereringen av rapporter er ferdig, vil "Download reports"-knappen (Figur 4.2.1-8) bli aktiv. Ved å trykke på den vil alle rapportene som ble generert bli lastet ned i en zippet mappe. Mappen vil også inneholde en txt-fil med teksten som ble skrevet ut i infoboksen.

4.2.1.5 Tidligere genererte rapporter

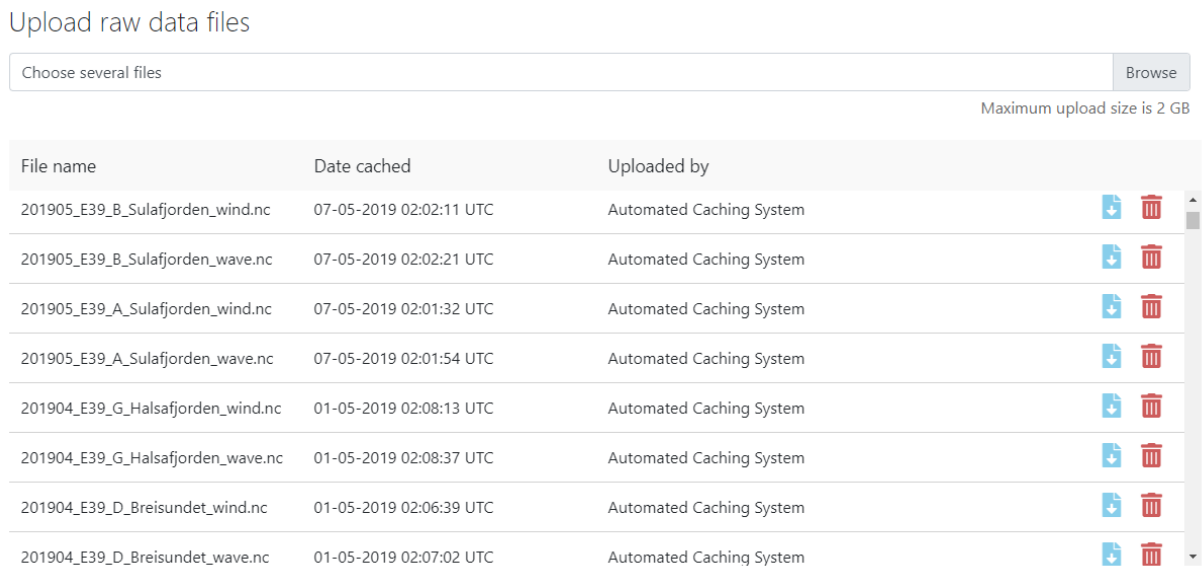
På siden "Stored reports" kan man laste ned og slette tidligere genererte rapporter. Her vises alle rapportene generert av alle brukere av systemet med rapportnavn, dato generert, status og e-postadressen til den som genererte den.



Figur 4.2.1-9 Tabell på siden "Stored reports" som viser alle lagrede rapporter.

4.2.1.6 Opplasting av NetCDF-filer manuelt

NetCDF-filer kan lastes opp manuelt på siden "Upload raw data files". Her vises alle NetCDF-filene som ligger lagret i systemet. Man kan slette og laste ned alle filene, samt se informasjon om når filene ble lastet opp til systemet og hvem som lastet dem opp.



Figur 4.2.1-10 Felt for opplasting av filer, samt tabell som viser alle lagrede NetCDF-filer.

4.2.1.7 Mellomlagring av NetCDF-filer

Systemet mellomlagrer NetCDF-filer fra thredds.met.no, som er Meteorologisk Institutt sin filtjener for rådatafilene. En konvertert versjon av NetCDF-filen ligger også lagret, slik at generering av rapporter skal gå raskere. På den måten kan brukerne av systemet lage rapporter selv om filserveren filene originalt ligger på er nede. Hvis systemet finner en nyere versjon av en

NetCDF-fil, vil den hente den og erstatte den gamle filen med den nye. Når en fil blir oppdatert i mellomlagringen, vil alle rapporter som ble generert av den utdaterte filen bli markert med status “Outdated”. Filer som ikke er utdatert har status “Up to date”, mens filer hvor det ikke er mulig å avgjøre om den er utdatert eller ikke har status “Unknown”. Et slikt tilfelle er når en rådatafil som ble brukt til å lage rapporten har blitt slettet fra mellomlagringen, etter at rapporten har blitt generert.

4.2.2 Ikke-funksjonelle egenskaper og andre krav

I dette delkapittelet beskrives måloppnåelse i forbindelse med målsettingene i forhold til de ikke-funksjonelle egenskapene til systemet, beskrevet i visjonsdokumentet (vedlegg A, kapittel 6).

4.2.2.1 Brukbarhet

Det grafiske brukergrensesnittet har blitt utviklet slik at det kan brukes uten opplæring. Det består av veldig få komponenter, og har beskrivelser og informasjon der det er behov for forklaring. Web-applikasjonen har også en egen informasjonsside som forklarer hvordan systemet fungerer. Siden inneholder forklaringer på funksjoner og bruk som ikke passet inn andre steder. Web-applikasjonen har blitt tatt i bruk og testet av sluttbruker, uten hjelp fra manual eller andre personer.

Systemet er laget for å tilpasse seg skjermstørrelser for bærbare datamaskiner og pc-skjermer.

4.2.2.2 Pålitelighet

Det var ikke satt opp krav for oppetid til systemet. Systemet har robust kode som skal takle feil som kan oppstå under vanlig kjøring, og holde web-applikasjonen gående dersom feil oppstår. Systemet har ikke datatap av vind- og bølgedata under generering av rapporter.

4.2.2.3 Ytelse

Systemet støtter samtidig bruk av opp til flere brukere. Det var ikke satt krav til kjøretid for generering av rapporter.

4.2.2.4 Støtte

Systemet er bygget modulært. Det består av et klassebibliotek og en web-applikasjon. Klassebiblioteket kan generere en rapport på egenhånd, og er ikke avhengig av andre deler av prosjektet. Web-applikasjonen bruker dette klassebiblioteket for å lage rapporter. Koden er godt dokumentert både gjennom systemdokumentasjonen og kommentarer i koden.

4.2.3 Implementasjon av maskinlæringsteknikken kNN

Maskinlæringsteknikken kNN har blitt implementert i koden for å imputere manglende data. Den bruker en kombinasjon av fysisk avstand og avstand i tid mellom datapunkter for å avgjøre hvilke datapunkter som skal brukes ved imputering.

Når et datapunkt mangler forsvinner også informasjonen om dens lokasjon. Uten lokasjonen til datapunktet kunne vi ikke kalkulere fysisk avstand, og dermed heller ikke benytte kNN. For å løse dette problemet valgte vi å lineærinterpolere fra koordinatverdiene (lengde- og breddegraden) til sist kjente datapunkt frem til neste kjente datapunkt, og benytte disse verdiene som posisjon til de manglende datapunktene i mellom. Målestasjonene er forankret i havbunnen [17], og forflytter seg minimalt innenfor en veldig liten radius mellom hver måling. Vi vurderer derfor at lineærinterpolering vil gi et godt nok estimat.

For å regne om fra koordinatverdier til fysisk avstand brukte vi Haversine-formelen. Den brukes til å finne avstanden mellom to punkter på en kule, og kan brukes til å finne distansen mellom to posisjoner på jordoverflaten gitt deres lengde- og breddegrader.

Avstanden *s* i metere gitt ved:

$$s = 2r * arcsin\left(\sqrt{sin^2\left(\frac{b_1 - b_2}{2}\right) + cos(b_1)cos(b_2)sin^2\left(\frac{l_1 - l_2}{2}\right)}\right)$$

hvor b_1 og b_2 er breddegrad for henholdsvis første og andre punkt, l_1 og l_2 er lengdegradene deres, og r er jordas radius [13]. Videre benytter vi den Euklidske distansen for å beregne avstand mellom datapunktene:

$$d = \sqrt{s^2 + t^2}$$

hvor d er målet på avstand mellom datapunktene, s er den romlige avstanden gitt av Haversine-formelen, og t er tid-avstanden mellom målingene. Tid-avstanden t er differansen i sekunder mellom målingene delt på 6, slik at to målinger gjort ti minutter fra hverandre vil få t = 100. Denne skaleringen gjør vi for å unngå at datapunkter får for stor avstand fra hverandre for raskt, kun basert på differansen i tid. Derimot kan denne skaleringen justeres for å finne en optimal balansegang mellom tid og romlig avstand.

4.3 Administrative resultater

4.3.1 Fremdriftsplan

Fremdriftsplanen for prosjektet er beskrevet i form av et Gantt-diagram (vedlegg D, kap. 2, figur 2-1 og tabell 2-A). Tabellen under viser timer ført på de forskjellige aktivitetene.

Tabell 4.3.1-A Oversikt over faktisk antall timer brukt og anslått timeforbruk.

Aktivitet	Anslått start og slutt	Faktisk start og slutt	Anslått timeforbruk	Faktisk timeforbruk
Bachelorprosjekt	7. jan. - 20. mai.	7. jan. - 20. mai	1000	1065
Møter	7. jan. - 20. mai.	7. jan. - 13. mai	50	45
Prosjektoppsett	14. jan. - 18. feb.	21. jan - 3. mai	20	43
Obligatoriske forelesninger	21. jan. - 11. feb.	21. jan - 1. feb.	20	20
Research og selvlæring	21. jan. - 20. apr.		70	
Programmering	28. jan. - 6. mai	28. jan - 3. mai	400	575
Testing	25. feb. - 6. mai		120	
Dokumentasjon	14. jan - 13. mai	18. feb - 20. mai	120	74,5

Hovedrapport	8. apr - 20. mai	28. apr - 20. mai	200	240,5

Timer for punktene *Research og selvlæring* og *Testing* har gjennom prosjektet blitt ført på punkter i timelisten som omhandler programmering. I tillegg til aktivitetene oppført i tabellen og Gantt-diagrammet, har vi også ført 67 timer på posten *Div. admin* (diverse administrasjon) i timelistene (vedlegg D, kapittel 2).

Den anslåtte timebruken i fremdriftsplanen for de forskjellige aktivitetene er ikke så langt unna den faktiske timebruken, med noen unntak. For *Prosjektoppsett* ser vi at vi har brukt omlag dobbelt så lang tid som først antatt. Hvis vi legger sammen timene fra *Reasearch og selvlæring*, *Programmering* og *Testing* i *Anslått timeforbruk* ender vi opp på et timeantall som er veldig nært de anslåtte 575 timene for *Programmering* i *Faktisk timeforbruk*. Vi ser også at vi har brukt litt mindre tid på *Dokumentasjon* enn antatt, mens *Hovedrapport* har fått en del ekstra timer.

Anslått start- og sluttdatoer stemmer ganske godt med faktiske start- og sluttdatoer for de fleste aktivitetene. Aktivitetene “Prosjektoppsett” og “Dokumentasjon” er de to som ligger lengst unna de anslåtte datoene.

4.3.2 Timeforbruk

Gjennom prosjektet har timer blitt dokumentert i form av timelister i prosjekthåndboka (vedlegg D, kap. 4). Oppføringene er fordelt på forskjellige aktiviteter som var relevant for prosjektet vi gjennomførte. Vi endte til slutt opp med å bruke 1065 timer til sammen på prosjektet.

4.3.3 Utviklingsprosess

Utviklingsmetoden Scrum ble valgt for dette prosjektet. Dokumentasjon for sprintene finnes i prosjekthåndboka (vedlegg D, kap. 5). User stories ble brukt for å spesifisere kravene i kravdokumentet (vedlegg B, kap. 2).

5 Diskusjon

5.1 Vitenskapelige resultater

5.1.1 Forbedre rapporter ved hjelp av maskinlæring

I forsøket på å finne en optimal verdi for *k* ser vi av grafen "*K vs. snittavvik*" at forskjellige attributter i snitt gir lavere avvik mellom imputerte og faktiske verdier ved forskjellige verdier for *k*. Eksempelvis får vi lavere avvik for attributten *WindSpeed* ved høy verdi for *k*, men lavt avvik for attributtene *Hm0*, *hm0a*, *hm0b* og *hmax* ved lav verdi for *k*. Dette forteller oss at det er vanskelig å fastsette en enkelt verdi for *k* som gir best mulige estimater for alle typer attributter. Vi ser derfor at det kunne vært fornuftig å la algoritmen benytte en egen verdi for *k* per attributt.

Videre ser vi av grafen "*Imputerte datapunkter vs. snittavvik*" at avviket mellom imputerte og faktiske verdier avhenger sterkt av antallet datapunkter som må imputeres. I tillegg ser det ut til at kNN-algoritmen klarer å imputere enkelte attributter mer nøyaktig enn andre. Eksempelvis ser vi at avviket for attributtene *Hm0*, *hm0a*, *hm0b* og *hmax* øker fra ca. 0,1 til ca. 0,5 meter i løpet av testene vi har kjørt, mens avviket for attributten *thtp* øker fra ca. 1 grad til ca. 17 grader.

Mye av grunnen til at kNN-regresjonen gir unøyaktige estimater er trolig at algoritmen har for lite data å basere seg på. Det er vanskelig å estimere værdata basert på bare tid og sted, da været kan slå om brått, selv på samme sted innen kort tid. I tillegg vil værforholdene variere stort mellom de forskjellige målestasjonene, på grunn av forskjellig topologi, vannstrømmer og andre fysiske forhold.

Ut i fra disse resultatene vurderer vi at vår implementasjon av kNN-regresjon for imputering egner seg greit når det er små hull i datasettene, men dårligere ved større mangler. I virkelige scenarioer er det både små og store perioder med manglende data, og de imputerte datapunktene vil da kunne inneholde unøyaktige verdier som potensielt ikke kan brukes. Vi vil derfor påstå at systemet vårt i sin nåværende tilstand i liten grad klarer å forbedre rapportene ved å imputere manglende data.

5.1.2 Redusere tidsbruk for å lage en rapport

I kapittel 4.1.2 kan vi ut i fra resultatene se at tidsbruken i forbindelse med å lage en rapport har blitt redusert. Tiden vi brukte for å sammenligne systemets tid med var en anslått tid gitt av en kystingeniør som lager rapportene. Ideelt sett skulle vi målt tiden en kystingeniøren faktisk bruker, og deretter sammenlignet med tiden systemet bruker. Dette lot seg ikke gjøre ettersom ingen rapporter skulle lages i perioden vi jobbet med prosjektet.

Derimot fikk vi kystingeniøren vi har vært i kontakt med til å prøve en ferdig versjon av systemet. Han kunne bekrefte at tidsbruken var kuttet betraktelig.

Uavhengig av resultatene kan vi si at vi har fått kuttet tiden en kystingeniør bruker på å lage en rapport, ettersom hele prosessen har blitt automatisert. Systemet loggfører hele rapportgenereringen, noe som gjør at det ikke er behov for å følge med under kjøring. Tiden en kystingeniør bruker på å lage en rapport, vil da være lik tiden det tar å logge inn på web-applikasjonen, velge stasjoner, start- og slutttdato, og trykke på "Create reports"-knappen.

5.1.3 Forbedre rapporter ved å eliminere potensielle menneskelige feil

For å svare på spørsmålet "*Er det mulig å forbedre rapportene ved å eliminere feilkilder som følge av menneskelig interaksjon?*", må den manuelle prosessen sammenlignes med den nye automatiserte prosessen. Dette blir gjort i kapittel 4.1.3 hvor resultatene viser at der det før var rom for feil som følge av menneskelig interaksjon, er det nå en automatisert prosess.

I resultatene blir blant annet forbedringene som følge av mellomlagringsfunksjonen presentert. Her ser vi at systemet potensielt kan være med på å forbedre rapportene på to ulike måter. Den ene forbedringen er når mellomlagringen har mer oppdaterte filer enn de som kystingeniørene får tilsendt fra Fugro Oceanor. Den andre er når systemet varsler om at en tidligere generert rapport er utdatert. Selv om begge disse funksjonene potensielt sett kan være med på å forbedre rapportene, kan det diskuteres hvorvidt feilkildene som funksjonene eliminerer kommer som følge av menneskelig interaksjon.

Vi vet ikke årsaken til hvorfor rådatafilene som kystingeniørene får tilsendt fra Fugro Oceanor er ufullstendige, og kan derfor ikke konkludere på grunnlag av dette.

Når det kommer til de utdaterte rapportene er disse mer en konsekvens av den kontinuerlige oppdateringen av rådatafilene, enn en konsekvens av menneskelig interaksjon.

Kapittel 4.1.3 inneholder også resultatene for forbedringen av rapportene i forbindelse med den automatiserte prosessen for innlasting av data i malene, samt tilpassing av malene etter innlasting. Her er det ingen tvil om at potensielle feil i den manuelle prosessen kan oppstå som følge av menneskelig interaksjon. Resultatene viser at systemet har potensialet til å eliminere mulige feilkilder. Likevel kan vi ikke svare på om systemet vil forbedre rapportene, ettersom vi ikke har noen konkrete kilder å vise til hvor systemet har gjort en bedre jobb en kystingeniørene.

5.2 Ingeniørfaglige resultater

Systemet samsvarer med de funksjonelle og ikke-funksjonelle kravene beskrevet i visjonsdokumentet (vedlegg A, kap. 5 - 6). De funksjonelle kravene beskrevet som User stories i kravspesifikasjonen (vedlegg B, kap. 2) har også blitt oppfylt.

5.2.1 Funksjonelle krav

5.2.1.1 Tilgang til systemet

Kravene om tilgang til systemet ble oppfylt gjennom å bruke Norconsults egen Azure Active Directory for autentisering ved innlogging. Med denne løsningen kan ansatte hos Norconsult, som har en Norconsult e-postadresse logge seg både inn og ut av web-applikasjonen.

Fordelene med å gjøre det på denne måten er at tilgangen til web-applikasjonen enkelt kan både utvides og innskrenkes. Om Norconsult ønsker at bare noen få navngitte brukere skal få tilgang, kan dette enkelt endres gjennom Azure-portalen til Microsoft. Løsningen sørger også for automatisk innlogging til web-applikasjonen om du har vært logget inn tidligere. En annen fordel er at systemet selv ikke trenger å håndtere sensitive data for innlogging og brukerautentisering.

5.2.1.2 Oppsett av rapporter

De funksjonelle kravene for oppsett av rapporter har blitt oppfylt. To dato- og tidvelgere gjør at en kan velge start- og sluttidspunkt for rapportene. Her er også akseptansekravene (vedlegg B, tabell 2.2-B) for 10-minutters intervaller og UTC tidspunkt oppfylt.

Valg av stasjoner skjer gjennom en flervalgs nedtrekksliste. Stasjonene inneholdt i listen blir satt i en konfigurasjonsfil i systemet. Her kunne det vært mulig å implementert funksjonalitet slik at brukerne kunne endret listen over stasjoner gjennom brukergrensesnittet.

Kravene som omhandlet maskinlæring, i form av muligheten til å skru imputering av eller på før generering, samt markering av imputerte verdier i rapportene, har blitt oppfylt.

5.2.1.3 Generering av rapporter

De funksjonelle kravene for generering av rapporter var å kunne velge en eller flere stasjoner, og dermed generere et valgfritt antall rapporter, avbryte genereringen under kjøring, og å få tilbakemeldinger om status for kjøringen. Disse kravene har blitt oppfylt.

I tillegg til akseptansekravene (vedlegg B, tabell 2.2.2-C) for tilbakemeldinger under kjøring, har det blitt lagt til ekstra tilbakemeldinger som kan være nyttig for brukeren. Det vil også bli gitt et tidsestimat for kjøretiden. Denne blir kalkulert ut i fra kjøretid for andre genererte rapporter.

I kapittel 4.2.1.3 blir det forklart hvordan systemet tar hånd om sammenslåing av data fra forskjellige rådatafiler. Måten det har blitt implementert på sørger for at alle datapunktene som skal være med i en rapport er med, uavhengig av mangler i rådatafilene. Den eliminerer også alle potensielle feil som kunne oppstå under den manuelle prosessen, der kystingeniørene klippet og limet data fra de forskjellige konvertere rådatafiler, inn i rapporten.

5.2.1.4 Nedlasting av rapporter

User storyen for nedlasting av rapporter hadde to akseptansekrav (vedlegg B, tabell 2.2.2-F). Disse var at rapportene skulle være tilgjengelig for nedlasting i zippet mappe etter generering, og at mappen skulle inneholde tilbakemeldingene gitt til brukeren under kjøring. Disse kravene har blitt oppfylt.

Løsningen tillater ikke at andre brukere enn den som genererte rapporten får tilgang til zip-mappen med alle rapportene og tilbakemeldingene i txt-filen. Funksjonalitet for å få til dette ble ikke implementert fordi det ikke ble sett på som nødvendig.

5.2.1.5 Tidligere genererte rapporter

Kravene for nedlasting og sletting av tidligere genererte rapporter har blitt oppfylt. Løsningen viser i tillegg til dette relevant informasjon om en lagrede rapporter (se kap. 4.2.1.5).

En svakhet med løsningen er at tabellen med tidligere genererte rapporten kan bli uoversiktlig når antallet blitt stort. En tabell med mulighet for sortering på ulike kolonner eller et søkefelt kunne løst dette problemet. Denne funksjonaliteten ble ikke implementert grunnet tidsbegrensninger.

5.2.1.6 Opplasting av NetCDF-filer manuelt

Kravene for manuell opplasting av NetCDF-filer har blitt nådd. Filer kan lastes opp, lastes ned og slettes fra systemet. Alt dette kan gjøres på siden “Upload raw data files”, som også inneholder en oversikt over alle opplastede NetCDF-filer. For hver NetCDF-filer er det også informasjon om når de ble lastet opp, og hvem som lastet dem opp.

For at en opplastet fil skal bli tatt i bruk under kjøring kreves det at den følger et bestemt format. Et problem med løsningen er at brukeren ikke får tilbakemelding om filen som blir lastet opp er på feil format. Brukeren har selv mulighet til å verifisere om en manuelt opplastet fil blir tatt i bruk under generering av en rapport. Informasjon om dette står beskrevet på infosiden, og vi valgte derfor å ikke bruke tid på å implementere en sjekk for dette.

5.2.1.7 Mellomlagring av NetCDF-filer manuelt

Akseptansekravene for User storyen for mellomlagring av NetCDF-filer (vedlegg B, 2.2.5-B) har blitt nådd. Systemet kjører en periodisk oppgave på et tidspunkt satt i konfigurasjonsfilen til systemet. Den sjekker når alle filene på fil-serveren til thredds.met.no ble endret og sammenligner med “DateModified” i databasetabellen “RawDataFiles” (vedlegg C, Figur 5.5-1).

En svakhet ved systemet er at manuelt opplastede filer aldri blir oppdatert. Disse ligger lagret med flagget “ManuallyUploaded”. For at en slik fil skal bli automatisk oppdatert, må den først

slettes av en bruker. En løsning på dette kunne være å ha to separate lagringsplasser for manuelt opplastede filer, og automatisk opplastede filer.

5.2.2 Ikke-funksjonelle egenskaper og andre krav

5.2.2.1 Brukbarhet

For det ikke-funksjonelle kravet om brukbarhet var det satt krav om at det skulle være enkelt å bruke, og at det ikke bør være behov for opplæring av systemet. Det har ikke blitt gjennomført brukertester av systemet. Likevel har en av brukerne fått testet systemet på egenhånd, uten noen form for opplæring eller forklaring på bruk. Vi vil derfor påstå at systemet er så enkelt at det kan tas i bruk uten foreliggende kunnskap, og at kravet derfor er nådd.

Systemet oppfyller også kravet om å være tilpasset ulike skjermstørrelser for bærbare datamaskiner og pc-skjermer. Systemet fungerer ikke på små enheter som mobiltelefon og har ikke blitt testet for bruk på nettbrett, da dette ikke var et krav.

5.2.2.2 Pålitelighet

Det var ikke satt opp krav for oppetid for systemet. Systemet skal likevel være oppe så lenge tjeneren hos Microsoft Azure kjører.

Systemet består av robust kode som skal takle normale feil under kjøring. En generering som feiler under kjøring vil ikke stoppe serveren, men vil si i fra til brukeren hva som eventuelt gikk galt. I tillegg til dette er det satt opp systemlogging i backend til en loggfil for debugging ved feil eller lignende som utviklere vil ha nytte av. Vi vil derfor anse det ikke-funksjonelle kravet om at "*Systemet vil bestå av robust kode som takler eventuelle feil, og sørger for at systemet fortsetter å kjøre.*" (vedlegg A, kap. 6.2) er nådd.

Kravet for dataintegritet for vind- og bølgedata har blitt nådd. Systemet utfører ingen kalkulasjoner på dataene i seg selv, og overfører dem tapsfritt fra NetCDF-format til Excel-rapportene.

5.2.2.3 Ytelse

Kravet om ytelse i form av støtte for flere samtidige brukere har blitt nådd. Jobber (jobb, se kap. 1.2) for generering av rapporter kjøres i individuelle Tasks i .NET [22], som tillater at flere rapporter kan genereres samtidig.

5.2.2.4 Støtte

Det første ikke-funksjonelle kravet til støtte er at backend for systemet skal bestå av et frittstående klassebibliotek. Dette har vi løst med å lage klassebiblioteket E39_Lib. E39_Lib kan kjøres frittstående fra kommandolinjen og har ingen avhengigheter i web-applikasjonen.

Det andre kravet er at "*Koden skal være så godt dokumentert at den blir enkel å vedlikeholde og utvide om ønskelig.*". Koden i seg selv har kommentarer som forklarer alle offentlige metoder og klasser. Det er også kommentert forklaring i koden hvor vi følte det var nødvendig. Systemspesifikasjonen (vedlegg C) inneholder også den nødvendige informasjonen for at en utenforstående utvikler enkelt skal kunne sette seg inn i systemet. Vi vil derfor anse dette kravet som nådd.

5.2.3 Implementasjon av maskinlæringsteknikken kNN

Maskinæringsteknikken kNN har blitt implementert og fungerer etter intensjonen. Det var ikke satt noen krav til hvilken maskinlæringsteknikk som skulle tas i bruk. Det var heller ikke satt noen krav til resultatene for imputeringen.

Tidlig under utforskningen av mulighetene for maskinlæring så vi på rammeverket Accord.NET, som er et maskinlæringsbibliotek for .NET. Denne hadde støtte for kNN-klassifikasjon, men ikke regresjon. Ved videre søk fant vi ingen ferdige biblioteker med støtte for kNN-regresjon som kunne brukes på våre datasett, og vi implementerte derfor algoritmen selv.

Vi ser ut i fra de vitenskapelige resultatene i kapittel 4.1.1 at endringer i implementasjonen av kNN kunne ført til bedre resultater. Dette er diskutert nærmere i kapittel 5.1.1.

5.2.4 Styrker med systemet

Et av systemets styrker er at det mellomlagrer NetCDF-filer fra filtjeneren thredds.met.no, slik at rådataene til enhver tid er tilgjengelige for bruk. Filene på thredds.met.no forsvinner stadig fra filtjeneren grunnet oppdateringer av filene. Ved å ha en oppdatert komplett database med alle NetCDF-filene sørger man for at det er mulig å lage rapporter når man ønsker.

Systemet er rullet ut på Microsofts skytjenester i Azure. Dette gjør at ressurser systemet har bruk for enkelt kan tildeles ved behov. Dette gjelder både prosessorkraft og minnekapasitet på tjenersiden, samt hastighet og lagringsplass for databaseløsningene.

5.2.5 Svakheter med systemet

For å laste ned rådatafiler fra filtjeneren bruker systemet URL-en til filen som skal lastes ned for å avgjøre hvilken stasjon og tidsperiode filen inneholder data for. Ved store endringer i URL-formatet vil ikke programmet lengre klare å finne fram til filene den leter etter, og koden vil derfor trenge tilpasninger.

En annen svakhet med systemet er at innstillingene for bruk ikke er konfigurerbar gjennom brukergrensesnittet. Hvis man for eksempel ønsker å legge til en ekstra målestasjon i nedtrekkslisten for valg av målestasjoner, må dette gjøres gjennom konfigurasjonsfilen til systemet. En løsning på dette hadde vært å implementert en side for instillinger.

Systemet bruker fire standardiserte maler når det skal genereres rapporter. Disse er ikke tilpasset og optimalisert for systemets bruk, og gjør at kjøretiden for noen deler av programmet er lengre enn den potensielt trenger å være.

En ulempe med at systemet er en web-applikasjon er at den, avhengig av tjeneren den kjører på, potensielt har dårligere hardware-ressurser enn en lokal arbeidsstasjon kan ha. Dette kan føre til at kjøretiden for å generere en rapport er noe lengre enn det ville vært ved kjøring på en arbeidsstasjon med mer minne og CPU-kraft.

5.3 Administrative resultater

5.3.1 Fremdriftsplan

Fremdriftsplanen for prosjektet ble satt opp i en tidlig fase og har avvik sammenlignet med prosessen som ble utført. Avvikene fra fremdriftsplanen tror vi kan ligge i at vi har brukt en smidig utviklingsmetodikk, framfor en sekvensiell. Gantt-diagrammet er laget for en sekvensiell utviklingsmetodikk, og vil i motsetning til en smidig metodikk som Scrum, ha fastsatte tidspunkter for når deler av prosessen skal være ferdig, og en ny skal begynne. Prosessen Scrum, derimot, tillater iterativ utvikling og forbedringer gjennom prosessens levetid, og vil forandre seg kontinuerlig basert på framgang og behov.

5.3.2 Timeforbruk

Det totale timeforbruket stemmer godt overens med det anslåtte timeforbruket på til sammen 1000 timer.

5.3.3 Utviklingsprosess

Scrums iterative prosess passet utmerket til prosjektet. Ved prosjektets start hadde ikke Produkteieren noen klar tanke om hva systemet skulle inneholde, annet enn kjernefunksjonaliteten beskrevet i visjonsdokumentet (vedlegg A, kapittel 5). Ny funksjonalitet ble etterspurt etterhvert som kjernefunksjonaliteten kom på plass, og vi var hele tiden nødt til å tilpasse arbeidet og Sprintene.

Sprint Review- og Sprint Planning-møtene ble holdt samtidig. På møtene fikk vi gode tilbakemeldinger fra både Scrum Master og Produkteieren, og proriterte sammen hva som burde gjøres i neste sprint. Et tett samarbeid med Produkteieren gjorde det lett å utvikle funksjonalitet etter brukerens behov.

Sprint Boards (vedlegg D, kap. 5) holdt oversikt over ønsket funksjonalitet for produktet i Produktbacklog. Scrum Master og Produkteier var med og bestemte hva som skulle føres over på Sprint Backlog for hver sprint. Dette gjorde at vi til enhver tid hadde kontroll over hvilken funksjonalitet som var prioritert høyest, og dermed hva vi burde jobbe mest med.

User storyene har hjulpet oss med å holde oversikt over hvilken funksjonalitet Produkteieren ønsket i systemet, og hvilke krav som måtte oppfylles i forbindelse med funksjonaliteten.

5.3.4 Profesjonsetiske problemstillinger

5.3.4.1 Hemmeligstemplede rapporter

Rapportene som ble laget manuelt i forkant av prosjektet, og som nå blir generert av systemet, inneholder hemmelighetsstemplet informasjon. Det har derfor vært viktig at vi som dataingeniør har sørget for begrenset tilgang til systemet, og rapportene som har blitt og vil bli generert.

5.3.4.2 Dataintegritet

Rapportene som blir produsert skal brukes i forbindelse med utbygging av veier, broer og tunneler langs Europavei 39. Dataene fra rapportene skal være med på danne et kunnskapsgrunnlag for vind- og bølgeforhold i fjordene. Som dataingeniør i dette prosjektet har det vært viktig å være klar over dette, og sørge for å ivareta dataintegriteten.

5.3.5 Gruppearbeid

Kommunikasjonen og arbeidet i gruppe har fungert godt gjennom hele prosjektet. Ingen av gruppemedlemmene har vært borte uten begrunnelse, og forskjellene i arbeidstimer har vært små.

Vi har brukt Trello for å organisere Sprintene som skulle utføres, og hvilke oppgaver som skulle gjøres for hvert Sprint. Trello har hjulpet oss med å holde orden på hvem som gjør hva, og status for utvikling på de forskjellige oppgavene

6 Konklusjon og videre arbeid

6.1 Konklusjon

Rapporten ser på problemstillingen "Hvordan kan en automatisk pipeline og maskinlæring forbedre og effektivisere prosesseringen og analysen av bølge- og vinddata?", og forsøker å svare på denne ved å se på følgende forskningsspørsmål:

1. Er det mulig å redusere tidsbruken en kystingeniør bruker på å lage en rapport?
2. Er det mulig å forbedre rapportene ved å eliminere feilkilder som følge av menneskelig interaksjon?
3. Kan man ved hjelp av maskinlæring forbedre rapportene ved å gjøre dem mer komplette?

Resultatene i kapittel 4.1.2 viser at vi har klart å redusere tidsbruken en kystingeniør bruker på å lage en rapport. Det kommer også fram i diskusjonen i kapittel 5.1.2 at automatiseringen av systemet i seg selv gjør at tidsbruken går ned. Med grunnlag i dette kan vi derfor besvare forskningsspørsmål 1 med at det er mulig å redusere tidsbruken en kystingeniør bruker på å lage en rapport.

Forskningsspørsmål 2 blir diskutert i kapittel 5.1.3. Her kommer vi fram til, med grunnlag i resultatene fra kapittel 4.1.3, at systemet har potensialet til å eliminere feilkilder som følge av menneskelig interaksjon. Likevel har vi ikke grunnlag til å svare verken ja eller nei på forskningsspørsmålet, ettersom at vi ikke kan vise til forbedringer i en rapport grunnet eliminasjon av feilkilder som følge av menneskelig interaksjon.

For å gjøre rapportene mer komplette ved tilfeller av manglende data, så vi på maskinlæringsteknikken kNN. Ut i fra resultatene i kapittel 4.1.1 ser vi at nøyaktigheten til verdiene som blir imputert avhenger sterkt av mengden manglende data, og hvilken attributt verdiene tilhører. Diskusjonen i kapittel 5.1.1 konkluderer med at systemet i liten grad klarer å forbedre rapportene ved hjelp av maskinlæring.

Ut i fra forskningsspørsmål 1 kan vi konkludere med at man kan effektivisere analysen av bølge- og vinddata ved å lage en automatisk pipeline. Derimot kommer vi ikke fram til noen konkret

løsning på hvordan man eventuelt kan forbedre rapportene, verken ved hjelp av maskinlæring eller eliminasjon av feilkilder som følge av menneskelig interaksjon. Likevel kan vi fastslå at selv om systemet ikke forbedrer rapportene i seg selv, har det potensialet til å hjelpe kystingeniørene i arbeidet med rapportene.

De ingeniørfaglige resultatene og diskusjonen rundt disse i henholdsvis kapittel 4.2 og 5.2 viser at vi har klart å utvikle et system som er i henhold til kravene spesifisert både i visjonsdokumentet (vedlegg A, kapittel 5 - 6) og kravspesifikasjonen (vedlegg B, kapittel 2). Med grunnlag i dette kan vi konkludere med at kravene og spesifikasjonene som ble satt av både oppgavestiller (Norconsult) og kunde (kystingeniør hos Norconsult), har blitt nådd.

6.2 Videre arbeid

Gjennom prosjektet har vi nådd alle mål for beskrevet i visjonsdokumentet og kravspesifikasjonen. Likevel er det deler av systemet som kunne vært forbedret eller utviklet på en annen måte.

For lesing av NetCDF-filer hadde det vært raskere og enklere å slippe å bruke et Python-skript som mellomledd. Under utviklingen forsøkte vi å bruke et bibliotek kalt SDSLite for innlesing av NetCDF-filene direkte i C#. Dette fungerte ikke grunnet en feil i biblioteket, og med lite tid igjen av prosjektet fikk vi ikke sett på muligheten til å reparere feilen. Hvis vi hadde fått til å implementere innlesing direkte i C#, hadde vi fått ned kjøretiden for generering av rapporter, samt sluppet å lagre ferdig konvertere CSV-filer. I tillegg hadde det redusert kompleksiteten til applikasjonen, da vi hadde unngått avhengigheten dens til Python-programvaren.

Når det kommer til maskinlæringsdelen av systemet hadde det vært interessant å utforske flere måter å imputere manglende bølge- og vinddata på. Ved videreutvikling av kNN-algoritmen kunne man sett på muligheten for å ha egne verdier for *k* per attributt i datapunktene.

Slik systemet fungerer nå lager det rapporter basert på standardiserte Excel-maler. En videreutvikling av dette kunne vært å presentere dataene i web-applikasjonen direkte. For å gjøre dette kan rådataene lagres direkte i en database, framfor å lagre hele NetCDF-filen. På

denne måten kunne man valgt stasjoner og tidsperioder for rapporter dynamisk i brukergrensesnittet, og eventuelt fått en rapport skrevet ut basert på dette.

For tabellene som viser fram lagrede rådatafiler og rapporter ville videre arbeid innebært å implementere funksjonalitet for sortering og søk, slik at det ville bli lettere å finne fram i tabellen.

7 Referanser

[1] K. Schwaber og J. V. Sutherland, "The Scrum Guide," i *Software in 30 Days: how Agile managers beat the odds, delight their customers, and leave competitors in the dust*. Hoboken, N.J.: John Wiley & Sons, 2012. [Online]. Hentet fra: https://ebookcentral.proquest.com/lib/ntnu/detail.action?docID=821878

[2] J. Galloway, B. Wilson, K. Allen, og D. Matson, "Getting Started," i *Professional ASP.NET MVC 5*. Hoboken: Wiley, 2014. [Online]: Hentet fra: https://ebookcentral.proquest.com/lib/ntnu/detail.action?docID=1744259

[3] TutorialsTeacher (n.d.). *MVC Architecture*. [Bilde] Hentet fra: https://www.tutorialsteacher.com/Content/images/mvc/mvc-architecture.png. Lastet ned 01.05.2019.

[4] D., Miessler. (2017). *MVC Architecture*. [Bilde] Hentet fra: https://danielmiessler.com/images/MVC1.png. Lastet ned 01.05.2019.

[5] MuleSoft, *What is a REST API?*. Hentet fra:

https://www.mulesoft.com/resources/api/what-is-rest-api-design. Lastet ned 02.05.2019.

[6] University of California, Irvine, *Representational State Transfer (REST)*.

https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, funnet (05.02.2019)

R. T. Fielding, "Representational State Transfer (REST)," i *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000. [Online]: Hentet fra: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[7] Statens Vegvesen, *Ferjefri E39*. Hentet fra: https://www.vegvesen.no/vegprosjekter/ferjefriE39. Lastet ned: 10.05.2019.

[8] Unidata, *What is NetCDF?*. Hentet fra: https://www.unidata.ucar.edu/software/netcdf/docs/. Lastet ned: 14.05.2019.

[9] E. A. P. A. Batista, G. og Monard, M. A Study of K-Nearest Neighbour as an Imputation Method. [Online] conteudo.icmc.usp.br. Hentet fra: http://conteudo.icmc.usp.br/pessoas/gbatista/files/his2002.pdf. Lastet ned 14.05.2019.

[10] Microsoft, *Observer Design Pattern*, 2017. Hentet fra: https://docs.microsoft.com/en-us/dotnet/standard/events/observer-design-pattern. Lastet ned 15.05.2019.

[11] OODesign.com, *Singelton Pattern*. Hentet fra: https://www.oodesign.com/singleton-pattern.html. Lastet ned: 15.05.2019.

[12] J. Bjørnstad, "Utvalgsundersøkelse," i *Store norske leksikon*, 2017. [Online]. Hentet fra: https://snl.no/utvalgsunders%C3%B8kelse. Lastet ned: 16.05.2019.

[13] H. Hafting, *Øving 13 Algoritmer og datastrukturer.* TISIP. Hentet fra: http://www.iie.ntnu.no/fag/_alg/Astjerne/opg13.pdf. Lastet ned: 16.05.2019.

[14] K. Zakka, *A Complete Guide to K-Nearest-Neighbors with Applications in Python and R*, 2016. Hentet fra: https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/. Lastet ned 17.05.2019.

[15] Microsoft, *Dependency injection in ASP.NET Core*, 2019. Hentet fra: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection. Lastet ned: 18.05.2019.

[16] Statens Vegvesen, Har inngått Norges største kontrakt for miljømålinger, 2016. Hentet fra:

https://www.vegvesen.no/Europaveg/e39sulafjorden/nyhetsarkiv/har-inng%C3%A5tt-norges-st%C3%B8rste-kontrakt-for-milj%C3%B8m%C3%A5linger. Lastet ned 18.05.2019

[17] Vikebladet, *Torsdag kjem første målestasjon*, 2016. Hentet fra: https://www.vikebladet.no/nyhende/2016/10/08/Torsdag-kjem-f%C3%B8rste-m%C3%A5lestasjon-13615428.ece. Lastet ned: 18.05.2019.

[18] C. Xia, G. Yu, og M. Tang, "Efficient Implement of ORM (Object/Relational Mapping) Use in J2EE Framework: Hibernate," presentert på International Conference on Computational Intelligence and Software Engineering, Wuhan, China, Desember 2009, [Online]. Hentet fra: https://ieeexplore.ieee.org/document/5365905

[19] UIO, *Object Relational Mapping (OMR) and Hibernate*. Hentet fra: https://www.uio.no/studier/emner/matnat/ifi/INF5750/h13/lecture-presentations/inf5750---lecture-2.-c---hibernate-intro.pdf. Lastet ned: 19.05.2019.

[20] S. A. Dudani, "The Distance-Weighted k-Nearest-Neighbor Rule". IEEE Transactions on Systems, Man, and Cybernetics SMC-6.4, s. 325-27, 1976. [Online]. Hentet fra: https://ieeexplore.ieee.org/document/5408784

[21] J. Huang et al.,"Cross-validation based K nearest neighbor imputation for software quality datasets: An empirical study," The Journal of Systems & Software, Vol.132, s. 226-252, Okt 2017. Hentet fra: https://www.sciencedirect.com/science/article/pii/S0164121217301516

[22] Microsoft, *Task-based asynchronous programming*, 2017. Hentet fra: https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/task-based-asynchronous-programming. Lastet ned: 20.05.2019.

8 Vedlegg

A. Visjonsdokument
B. Kravdokument
C. Systemdokumentasjon
D. Prosjekthåndbok

Visjonsdokument

Vedlegg A

Bacheloroppgave 2019

Oppgave 62

Dataingeniør

NTNU, institutt for datateknologi og informatikk

Trym Vegard Gjelseth-Borgen

Magnus Conrad Hyll

Trondheim, mai 2019

Innholdsfortegnelse

1 Innledning

Dette dokumentet beskriver problemområdet, produktet, interessentene og brukerne, omgivelsene og rammene for prosjektet.

Hensikten med prosjektet er å automatisere en data-pipeline benyttet i forbindelse med analyse av bølge- og værdata. Ved å implementere en data-pipeline vil man redusere tidsbruken og feilkildene knyttet til det manuelle arbeidet rundt disse analysene. Bakgrunnen for analysen av disse dataene er Statens Vegvesens visjon om å bygge en ferjefri E39. I den forbindelse skal man se på muligheten for å bygge broer eller undervannstunneler, ved å utrede grunnforholdene til fjordene som skal gjøres ferjefrie og vurdere ut ifra dette.

Dagens løsning på problemet er en manuell prosess for kvalitetssjekk, separasjon, analyse og plotting av disse dataene. Omfanget av dette prosjektet er å automatisere hele prosessen, fra nedlasting av rå måledata, til generering av rapporter. Dette involverer også å predikere målinger ved hjelp av maskinlæring hvor det er mangler i rådataene.

2 Sammendrag problem og produkt

2.1 Problemsammendrag

Tabell 2.1-A Problemsammendrag

Problem med	Dagens system består av store mengder manuelt arbeid. Dataene som skal brukes ligger åpent på thredds.met.no sine nettsider. Hver bølge- og vindmålingsstasjon har sine egne datafiler, som igjen er delt inn på månedsbasis. Filer kan bare lastes ned enkeltvis. Formatet på filene er NetCDF. Dette gjør at de må konverteres før de kan brukes. Når filene har blitt konverter blir de kopiert og limt inn i Excel-maler. Det finnes to forskjellige maler som inneholder forskjellig dataprosessering og framvisning. Når dataene har blitt lagt inn og malene har blitt tilpasset, brukes de til å lage rapporter som presenteres for Statens Vegvesen. Fra målestasjonene samles det inn data hvert 10. minutt. Iblant hender det at hull i tidsserien oppstår. Disse kan være små hull på bare noen timer, eller store hull som strekker seg over flere dager. Det finnes ingen måter å tette disse hullene på, noe som gir mangler i rapportene som lages basert på de innsamlede dataene.
Berører	Den manuelle innsamlingen og prosesseringen av dataen berører de kystingeniørene som jobber med dette. Hullene i dataene påvirker mottakeren, Statens Vegvesen.
Som et resultat av dette	Som et resultat av dette brukes det unødvendig mye tid på innsamling av data. Manuelt arbeid på store datasett gir også rom for at det gjøres menneskelige feil. Hullene i dataene og eventuelle menneskelige feil, resulterer også i ufullstendige rapporter.

En vellykket løsning vil	En vellykket løsning vil generere rapportene automatisk, og dermed eliminere nesten all tidsbruk i forhold til disse. Den vil også eliminere alle menneskelige feil. Den vil også gi predikere dataene hvor det er hull i tidsserien fra målestasjonene. Dette vil føre til mer fullstendige rapporter.

2.2 Produktsammendrag

Tabell 2.2-A Produktsammendrag

For	Norconsult, Statens Vegvesen
Som	Har behov for å få automatisert prosessen i forbindelse med innsamling og prosessering av bølgedata
Produktet navngitt	Ferjefri E39
Som	Automatiserer nedlasting, konvertering, prosessering og predikering av manglende data
I motsetning til	Dagens system med manuelt arbeid, som medfører flere tidkrevende steg
Har vårt produkt	Nesten ingen manuelle oppgaver, ingen tidkrevende steg og ingen hull i datasettene.

3 Overordnet beskrivelse av interessenter og brukere

3.1 Oppsummering av interessenter

Tabell 3.1-A Oppsummering av interessenter.

Navn	Beskrivelse	Rolle under utviklingen
Norconsult Informasjonssystemer AS, avdeling Fundator	Oppgavestiller.	Stiller med teknisk rådgivning og veiledning.
Statens Vegvesen	Kunden i prosjektet. Benytter rapportene som de får av kystingeniørene.	Ingen rolle under utviklingen.
Fugro Oceanor og Meteorologisk Institutt	Samler inn og publiserer rådata.	Ingen rolle under utviklingen.
Kystingeniører ved Norconsult AS	Brukerne av systemet. Konsulent for Statens Vegvesen.	Stiller krav til systemet. Bidrar med fagekspertise innen bølge- og værdata.
Atle Olsø	Veileder for oppgaven.	Stiller med rådgivning i forbindelse med dokumentasjon.

3.2 Oppsummering av brukere

Tabell 3.2-A Oppsummering av brukere

Navn	Beskrivelse	Rolle under utvikling	Representert ved
Kystingeniør	Kystingeniørene som jobber med dataene som samles inn, og genererer rapporter som igjen sendes inn til Statens Vegvesen.	Stiller krav til systemet. Bidrar med fagekspertise innen bølge- og værdata.	Konsulenter hos Norconsult AS: Athul Sasikumar, Onno Musch

3.3 Brukermiljøet

Systemet skal inngå som et verktøy i arbeidet til kystingeniørene som utarbeider rapportene for Statens Vegvesen. Det er en liten og ganske snever brukergruppe som skal benytte systemet, som betyr at systemet i stor grad kan skreddersys deres behov og arbeidsprosess. Brukergruppen sitter til vanlig lokalisert i kontorer i samme bygning som oppgavestiller.

3.4 Sammendrag av brukernes behov

Tabell 3.4-A Sammendrag av brukernes behov.

Behov	**Prioritet**	**Påvirker**	**Dagens løsning**	**Foreslått løsning**
Laste ned vind- og bølgedata.	Middels	Kystingeniør	Manuelt trykke på link.	Automatisk nedlasting basert på stasjon og tidsintervaller
Konvertere NetCDF-data til lesbart format.	Høy	Kystingeniør	Konvertere hver datafil i MatLab.	Automatisk konvertering etter nedlasting.
Overføre data til excel-maler.	Høy	Kystingeniør	Kopiere og lime inn.	Automatisk innskriving av data i excel-maler etter konvertering.
Predikere manglende data.	Middels	Kystingeniør Statens Vegvesen	Ingen løsning.	Bruke maskinlæringsteknikker for å predikere passende data basert på nærliggende vind- og bølgemålingsstasjoner, eventuelt andre værstasjoner i nærheten.
Mellomlagre rådata	Middels	Kystingeniør	Brukere kan laste ned filer manuelt	Et system som laster ned rådatafiler fra thredds.met.no og holder dem oppdatert.

3.5 Alternativer til vårt produkt

Det finnes ingen alternativer til vårt produkt, annet enn det manuelle arbeidet som allerede utføres.

4 Produktoversikt

4.1 Produktets rolle i brukermiljøet

Når det er ønskelig å generere en rapport kan en kystingeniør logge inn på “Ferjefri E39”, og velge hvilken stasjon og tidsrom det ønskes å genereres en rapport for.

4.2 Forutsetninger og avhengigheter

Ettersom at systemet er en webapplikasjon, forutsettes det at brukeren har internett-tilkobling mens de bruker av systemet, samt en nettleser for å kjøre applikasjonen i.

På den plattformen systemet skal kjøre vil det være nødvendig å installere programvarene og rammeverkene som systemet avhenger av. Systemet vil i tillegg være avhengig av internett-tilkobling. For at rådatafilene skal bli kontinuerlig oppdatert, er det nødvendig at web-tjeneren med rådatafiler hos thredds.met.no er tilgjengelig. Utenom dette vil ikke systemet være avhengig av eksterne arbeidsprosesser, annen maskinvare, integrasjon mot andre systemer eller andre forhold.

5 Produktets funksjonelle egenskaper

5.1 Innlogging

1. Logge inn
2. Logge ut

5.2 Nedlasting og prosessering

1. Nedlastning av rådata fra web-tjener gitt tidsintervall
2. Hente ut rådata fra nedlastet NetCDF-fil
3. Slå sammen rådata fra stasjon fra flere perioder
4. Predikere data for manglende målepunkter

5.3 Rapporter

1. Generere rapport med valgt stasjon og tidsintervall
2. Få tilbakemelding om data ikke er tilgjengelig for nedlasting
3. Få tilbakemelding om antall manglende datapunkter i datasettene
4. Få visualisert hvor manglende data oppsto

6 Ikke-funksjonelle egenskaper og andre krav

De ikke-funksjonelle kravene er spesifisert i forhold til FURPS-modellen [1].

6.1 Brukbarhet

Brukerne av systemet er normalt datakyndige mennesker, og det kreves derfor et brukergrensesnitt som er enkelt og effektivt å bruke.

1. Systemet vil være en web-app og vil derfor ha et grafisk brukergrensesnitt. Dette grensesnittet burde være selvforklarende og enkelt å bruke. Det skal ikke være behov for opplæring i bruk av web-applikasjonen. All informasjon om bruk av systemet og systemet i sin helhet bør bli kommunisert gjennom brukergrensesnittet.
2. Systemet vil tilpasse seg normale skjermstørrelser som man finner på monitorer og bærbare datamaskiner, men vil ikke være utviklet for mobiltelefoner og nettbrett.

6.2 Pålitelighet

Oppgaven systemet skal utføre er ikke en kritisk oppgave, noe som gjør at det ikke stilles strenge krav til pålitelighet.

1. Det er ikke satt krav til oppetid på systemet.
2. Systemet vil bestå av robust kode som takler eventuelle feil, og sørger for at systemet fortsetter å kjøre.
3. Systemet vil ikke ha datatap i forbindelse med vind- og bølgedata under kjøring.

6.3 Ytelse

Ettersom at systemet skal brukes av noen få personer, noen få ganger i året stilles det ikke strenge krav til ytelse.

1. Systemet tillater flere samtidige brukere.
2. Det stilles ikke krav til tidsbruk for generering av rapporter.

6.4 Støtte

Systemet skal være bygget modulært, noe som gjør det enkelt å vedlikeholde og utvide om ønskelig.

1. Backend skal bestå av et frittstående klassebibliotek.
2. Koden skal være så godt dokumentert at den blir enkel å vedlikeholde og utvide om ønskelig.

7 Kilder

[1] University of New Hampshire, *CS 619 Introduction to OO Design and Development*, 2013. Hentet fra: http://www.cs.unh.edu/~cs619/slides/usecase.pdf. Lastet ned: 04.04.2019.

Kravdokument

Vedlegg B

Bacheloroppgave 2019

Oppgave 62

Dataingeniør

NTNU, institutt for datateknologi og informatikk

Trym Vegard Gjelseth-Borgen

Magnus Conrad Hyll

Trondheim, mai 2019

Innholdsfortegnelse

1 Introduksjon

Dette dokumentet er skrevet i forbindelse med utviklingen av en automatisk pipeline for nedlasting og prosessering av bølgedata for Norconsult. Her beskrives funksjonelle krav til løsningen som skal utvikles. Systemet består av en backend som laster ned og mellomlagrer bølgedata. Deretter blir dataen prosessert, og det blir generert Excel-rapporter som viser fram dataene. Med i systemet er også en frontend, som skal hjelpe brukerne av programmet med å lett generere en slik Excel-rapport. En bruker skal kunne logge inn, velge tidsrom for hvilken rapport som skal lages, og deretter laste ned en slik rapport rett fra nettleseren.

Dokumentet inneholder User stories for å beskrive all brukerfunksjonalitet i systemet, domenemodell over systemet, og wireframes for det grafiske brukergrensesnittet.

2 User stories

2.1 Generelt

2.1.1 Tilgang til systemet

Tabell 2.1.1-A User story for tilgang til systemet.

Som	bruker
Ønsker jeg	å logge inn
Slik at	jeg får tilgang til systemet

2.2 Rapporter

2.2.1 Oppsett

Tabell 2.2.1-A User story for å velge start- og sluttdato og tid for rapport.

Som	bruker
Ønsker jeg	å kunne velge start- og sluttdato og tid for rapporten
Slik at	rapportene går mellom ønskede perioder

Tabell 2.2.1-B Akseptansekriterier for User story i tabell 2.2.1-A.

Det skal være mulig å velge tidspunkt på 10-minutters intervaller.
Klokkeslettene skal oppgis på tidsstandarden UTC.

Tabell 2.2.1-C User story for å velge stasjoner det skal lages rapport for.

Som	bruker
Ønsker jeg	å kunne velge mellom en og flere stasjoner
Slik at	rapportene lages kun for ønskede stasjoner

Tabell 2.2.1-D User story for å bruke maskinlæring for å imputere manglende data.

Som	bruker
Ønsker jeg	å kunne velge om det skal brukes maskinlæring for å imputere manglende data eller ikke
Slik at	rapportene blir så komplette som mulig

Tabell 2.2.1-E Akseptansekriterier for User story i tabell 2.2.1-D.

Imputerte verdier skal markeres i de ferdige rapportene.

2.2.2 Generering

Tabell 2.2.2-A User story for å generere rapporter.

Som	bruker
Ønsker jeg	å generere en eller flere rapporter
Slik at	slik at jeg kan analysere dataen som har blitt samlet inn.

Tabell 2.2.2-B User story for tilbakemelding under generering.

Som	bruker
Ønsker jeg	å kunne følge med på tilbakemeldinger om eventuelle feil eller mangler, mens rapportene genereres
Slik at	jeg kan ha kontroll over prosessen som utføres.

Tabell 2.2.2-C Akseptansekriterier for User story i tabell 2.2.2-B.

Det skal gis tilbakemelding om hvilke rådatafiler som ikke eksisterer.
Det skal gis tilbakemelding om antall rader, manglende rader og kolonner for en ferdig rapport.
Det skal gis tilbakemelding om en rapport ikke kunne bli generert, fordi alle dens rådatafiler manglet.
Det skal gis tilbakemelding om at en manuelt opplastet rådatafil blir tatt i bruk.

Tabell 2.2.2-D User story for avbryting av generering av rapporter.

Som	bruker
Ønsker jeg	å kunne avbryte generering av rapport
Slik at	jeg slipper å vente til den er ferdig.

Tabell 2.2.2-E User story for nedlasting av rapporter.

Som	bruker
Ønsker jeg	å kunne laste ned genererte rapporter
Slik at	jeg kan bruke dem lokalt.

Tabell 2.2.2-F Akseptansekriterier for user story i tabell 2.2.2-E.

Rapportene generert i en sesjon skal være tilgjengelig for nedlasting i en zippet mappe.
Mappen som blir lastet ned skal inneholde informasjonen brukeren fikk under generering som en txt-fil.

2.2.3 Tidligere genererte rapporter

Tabell 2.2.3-A User story for nedlasting av tidligere genererte rapporter.

Som	bruker
Ønsker jeg	å kunne laste ned tidligere genererte rapporter
Slik at	jeg kan bruke dem lokalt.

Tabell 2.2.3-B User story for sletting av tidligere genererte rapporter.

Som	bruker
Ønsker jeg	å kunne slette tidligere genererte rapporter
Slik at	de ikke tar plass eller vises på brukergrensesnittet.

2.2.4 Opplasting av NetCDF-filer manuelt

Tabell 2.2.4-A User story for opplasting av NetCDF-filer manuelt.

Som	bruker
Ønsker jeg	å kunne laste opp NetCDF-filer manuelt
Slik at	rapportene kan ta nytte av filer som ikke ligger lagret i database eller hos thredds.met.no

Tabell 2.2.4-B User story for oversikt av lagrede NetCDF-filer.

Som	bruker
Ønsker jeg	å kunne følge med på en oversikt over lagrede NetCDF-filer
Slik at	jeg vet når de sist ble oppdatert og hvem som lastet dem opp

Tabell 2.2.4-C User story for nedlasting av NetCDF-filer.

Som	bruker
Ønsker jeg	å kunne laste ned lagrede NetCDF-filer
Slik at	jeg kan kontrollere dem

Tabell 2.2.4-D User story for sletting av NetCDF-filer.

Som	bruker
Ønsker jeg	å kunne slette lagrede NetCDF-filer
Slik at	uønskede filer ikke er i systemet

2.2.5 Mellomlagring av NetCDF-filer

Tabell 2.2.5-A User story for mellomlagring av filer.

Som	bruker
Ønsker jeg	NetCDF-filer fra thredds.met.no lagres i systemet
Slik at	jeg alltid har tilgang til dem og kan lage rapporter

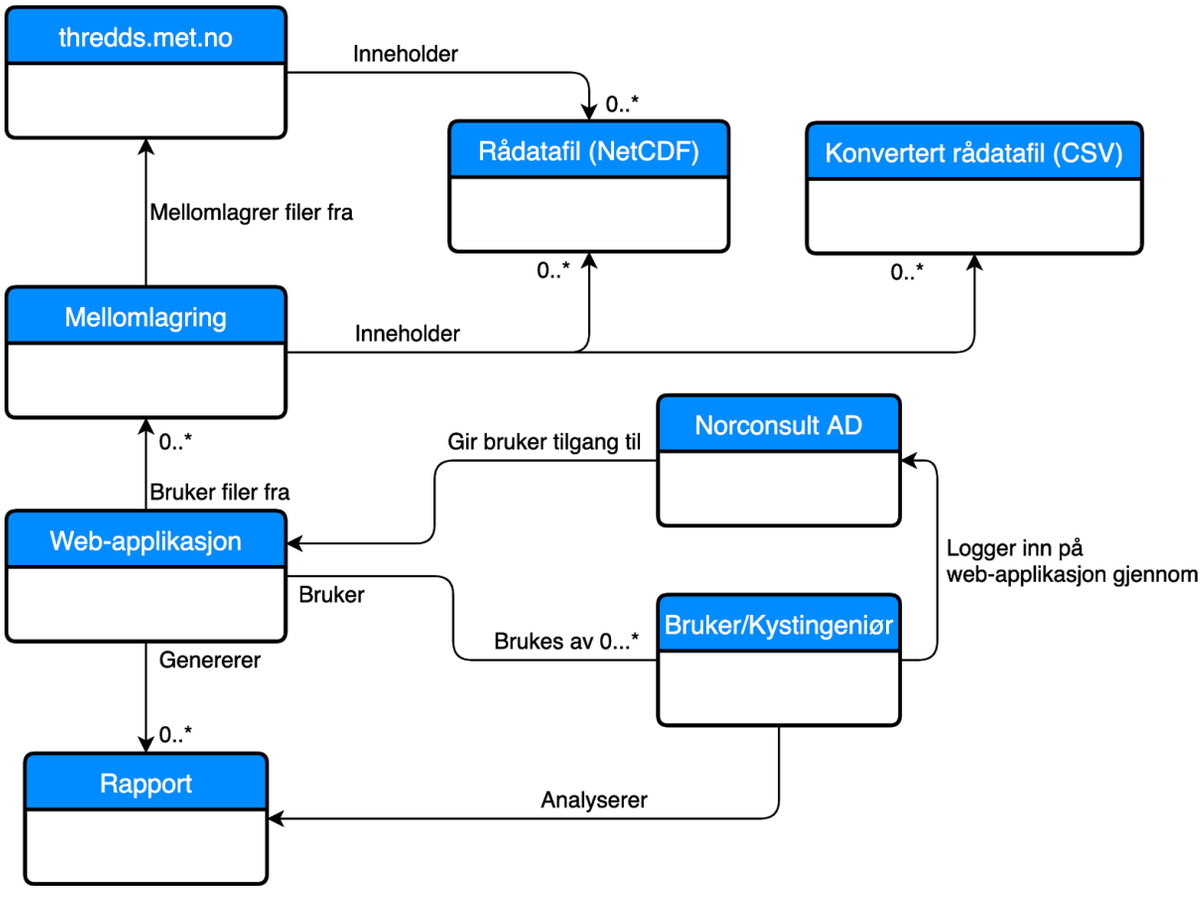
Tabell 2.2.5-B Akseptansekriterier for User story i tabell 2.2.5-A.

Hvis filene på thredds.met.no oppdateres, skal mellomlagringen også oppdateres.
Hvis en NetCDF-fil oppdateres skal alle tilhørende rapporter markeres som utdatert.

3 Problemdomener

3.1 Domenemodell

I figur 3.1-1 ser vi hovedkomponentene til systemet. Diagrammet viser hvordan *Bruker/Kystingeniør* bruker *Norconsult AD* for å få tilgang til *Web-applikasjon*. *Web-applikasjon* bruker filer fra *Mellomlagring*, som igjen henter filer fra filserveren *thredds.met.no*. Filserveren *thredds.met.no* og *Mellomlagring* innheolder *Rådatafil*, mens bare *Mellomlagring* inneholder *Konvertert rådatafil*. *Web-applikasjon* genererer *Rapport*, som analyseres av *Bruker/Kystingeniør*.

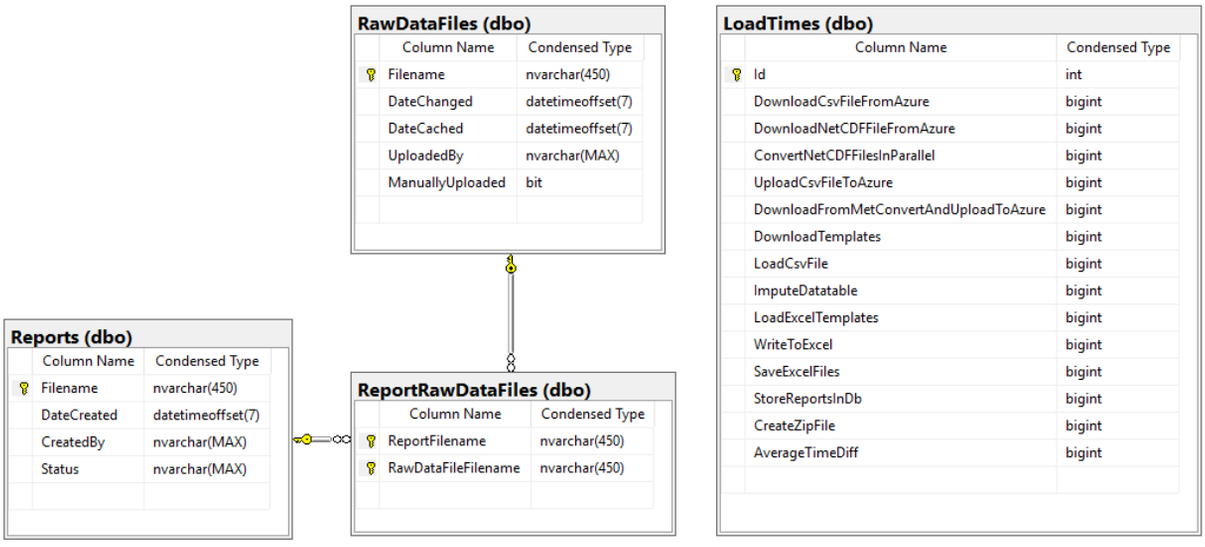


Figur 2.2.5-1 Domenemodell for systemet.

4 Prototyper

4.1 Databasemodell

Databasemodellen viser fire tabeller, hvorav tre er koblet sammen. Tabellene *Reports* og *RawDataFiles* har et mange-til-mange-forhold, og er koblet sammen gjennom knytningstabellen *ReportRawDataFiles*. Tabellen *LoadTimes* inneholder gjennomsnittstider for kjøring av de forskjellige delene av systemet, og brukes til å beregne kjøretid. *LoadTimes* inneholder til enhver tid bare en rad.

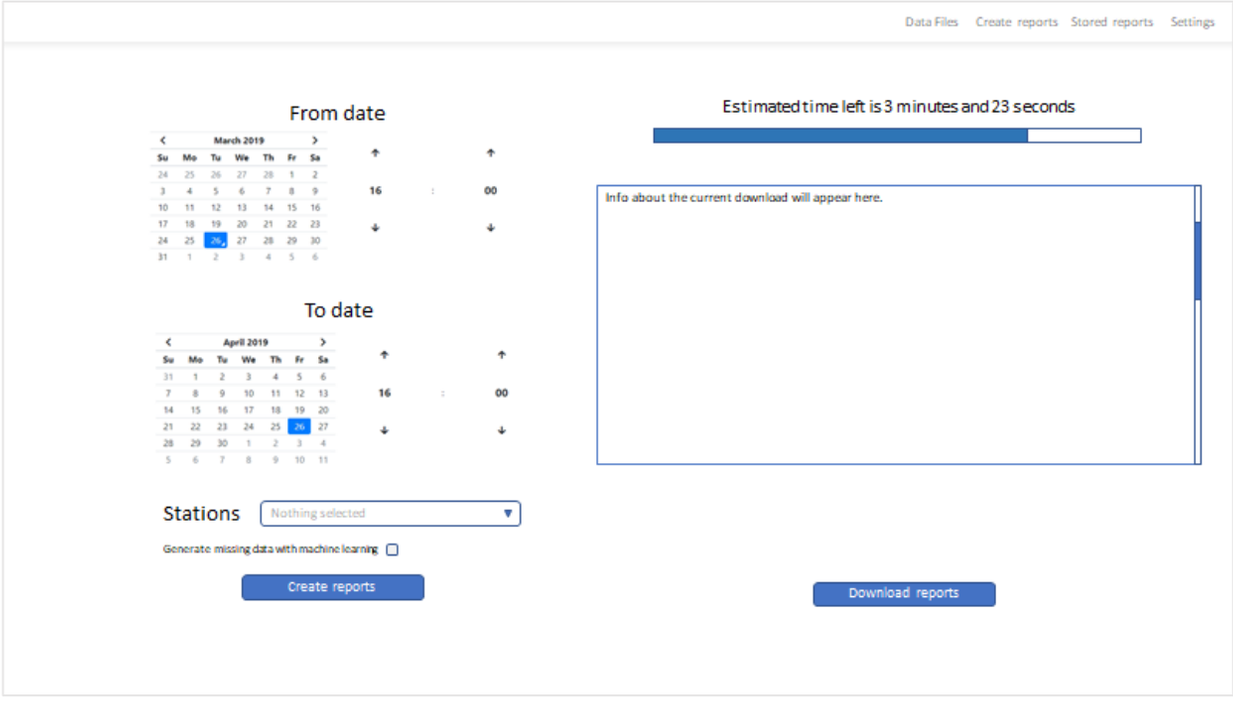


Figur 4.1-1 Databasemodell for databasen som systemet bruker.

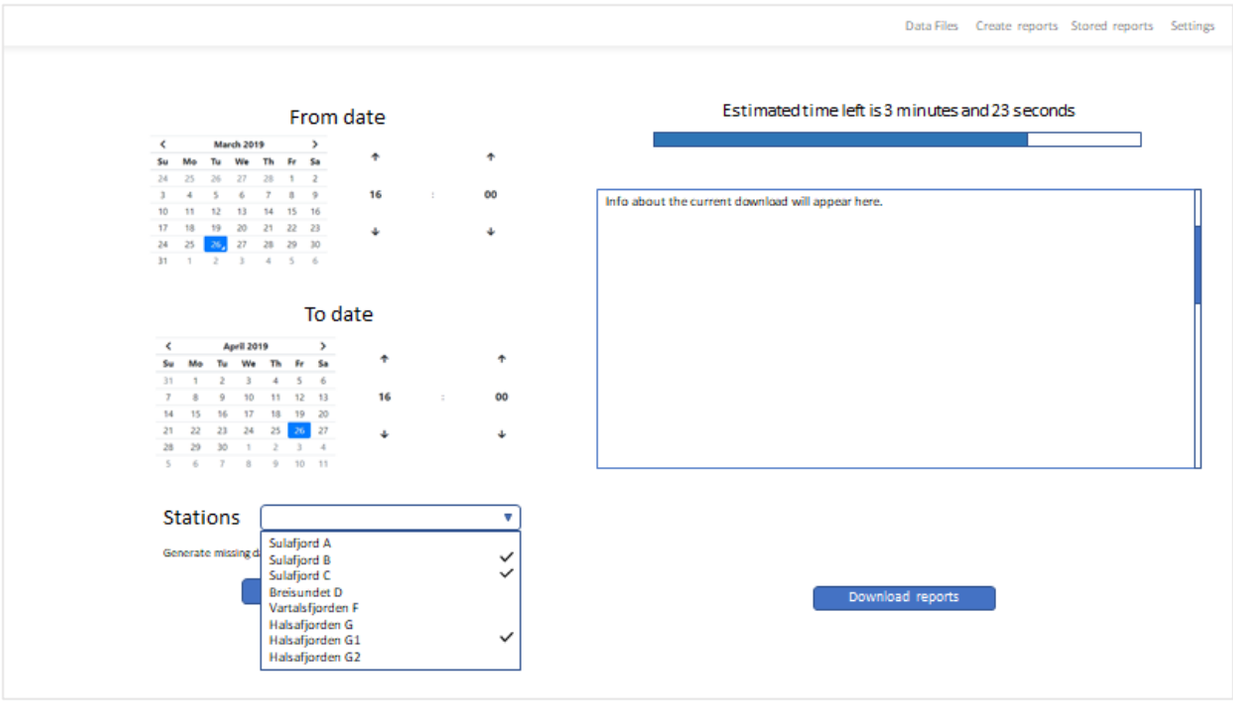
4.2 Wireframes

4.2.1 Create reports

Hovedsiden for å lage rapporter. Her velger man dato for når en rapport skal starte og når en rapport skal slutte. Man velger også hvilke stasjoner som det skal lages rapport for. Om man ønsker at det skal brukes maskinlæring for å estimere manglende data, krysser man av for det i avkrysningsboksen rett over "Create reports"-knappen. Til høyre i skjermbildet ser man en estimasjon av gjenstående tidsbruk for generering av rapporter, samt en logg-boks som viser hva som foregår i prosessen og om det er noen feil. I logg-boksen vil det også komme informasjon om totalt antall rader, antall manglende rader og antall kolonner med data per rapport.



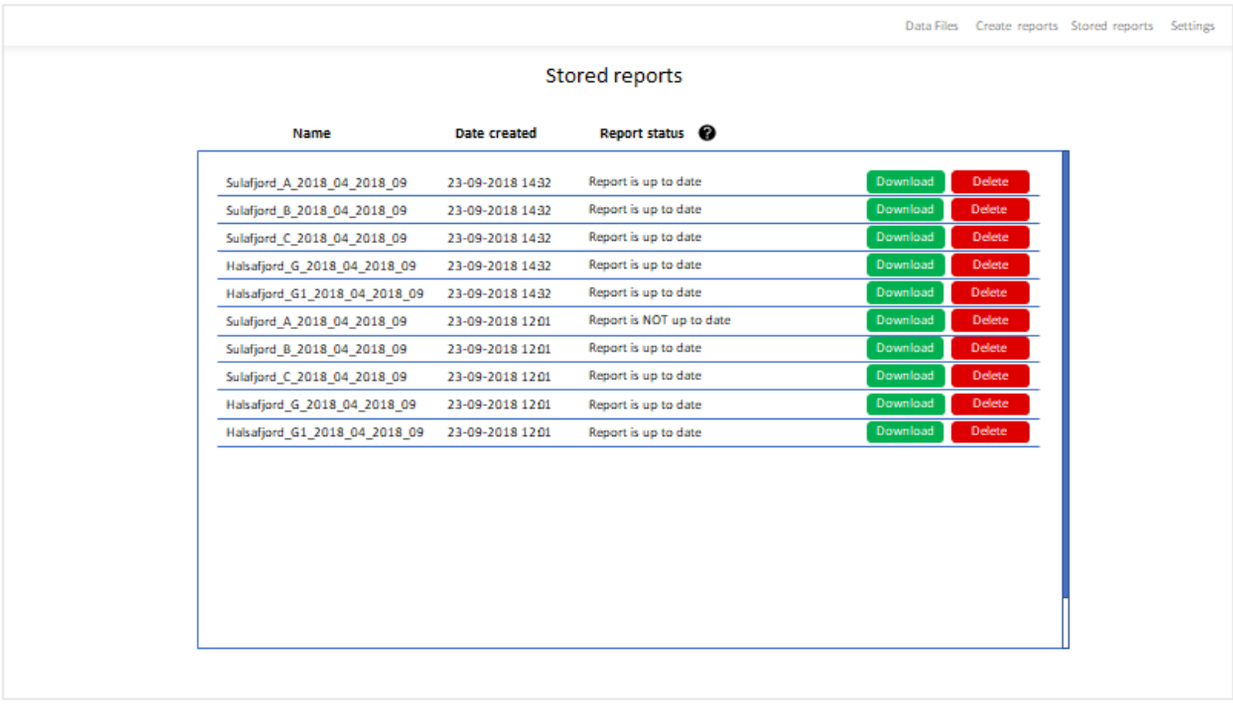
Figur 4.2.1-1 Hovedside for å lage rapporter.



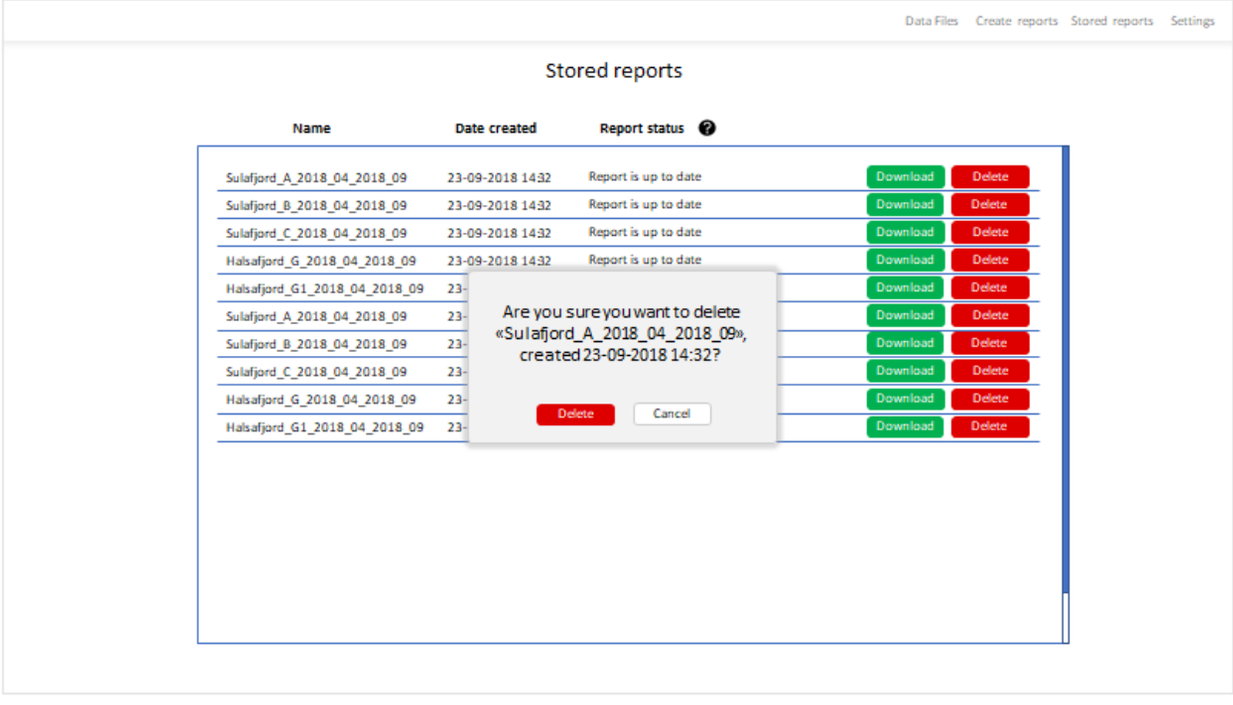
Figur 4.2.1-2 Hovedside for å lage rapporter med stasjonslisten ekspandert.

4.2.2 Stored reports

Denne siden inneholder en oversikt over rapporter som tidligere har blitt laget, og som nå ligger lagret i en database. Oversikten viser navnet på filen, datoen rapporten ble lagd og en status for rapporten. Statusen forteller om rådatafilene som ble brukt til å generere rapporten har blitt oppdatert i etterkant av at rapporten ble laget. Informasjon om dette får man ved å holde musen over spørsmålstegnet ved siden av “Report status”. Man kan velge og enten laste ned en tidligere generert rapport, eller slette en tidligere generert rapport. Hvis man forsøker å slette, vil man måtte bekrefte dette med enda et trykk (Figur 4.2.3-2) før slettingen blir utført.



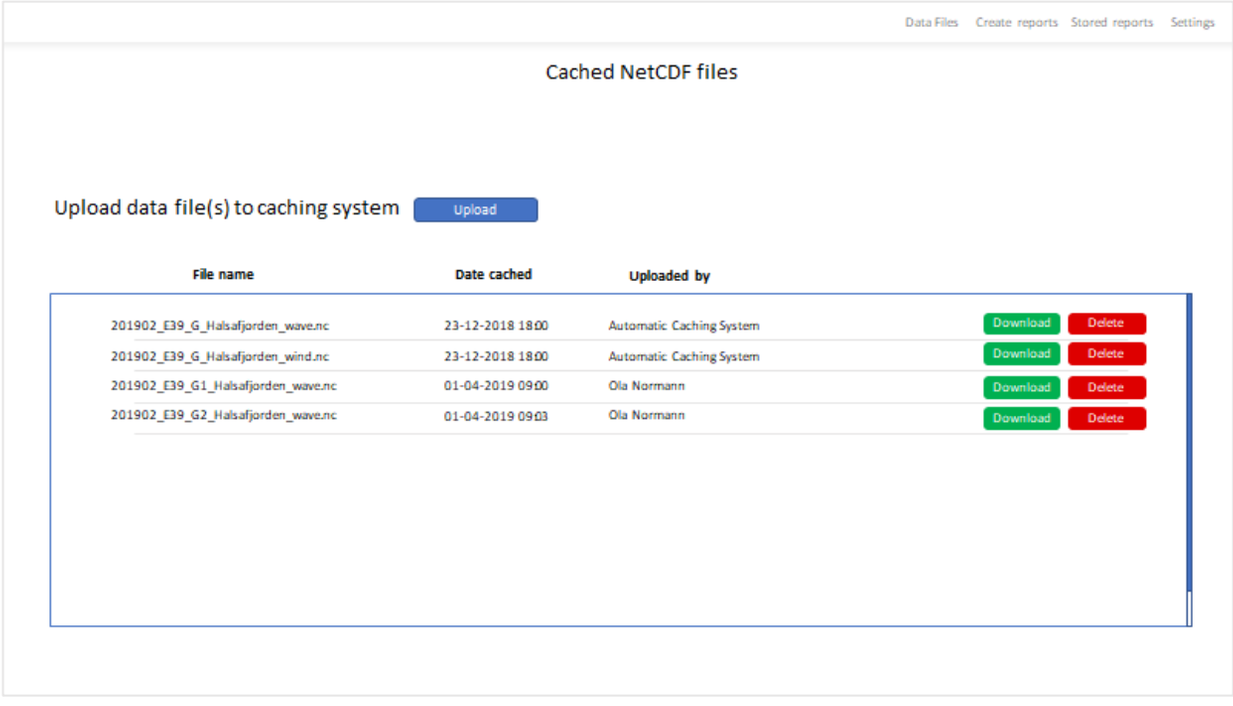
Figur 4.2.2-1 Side med oversikter over rapport som tidligere har blitt lagd, og som ligger lagret i database.



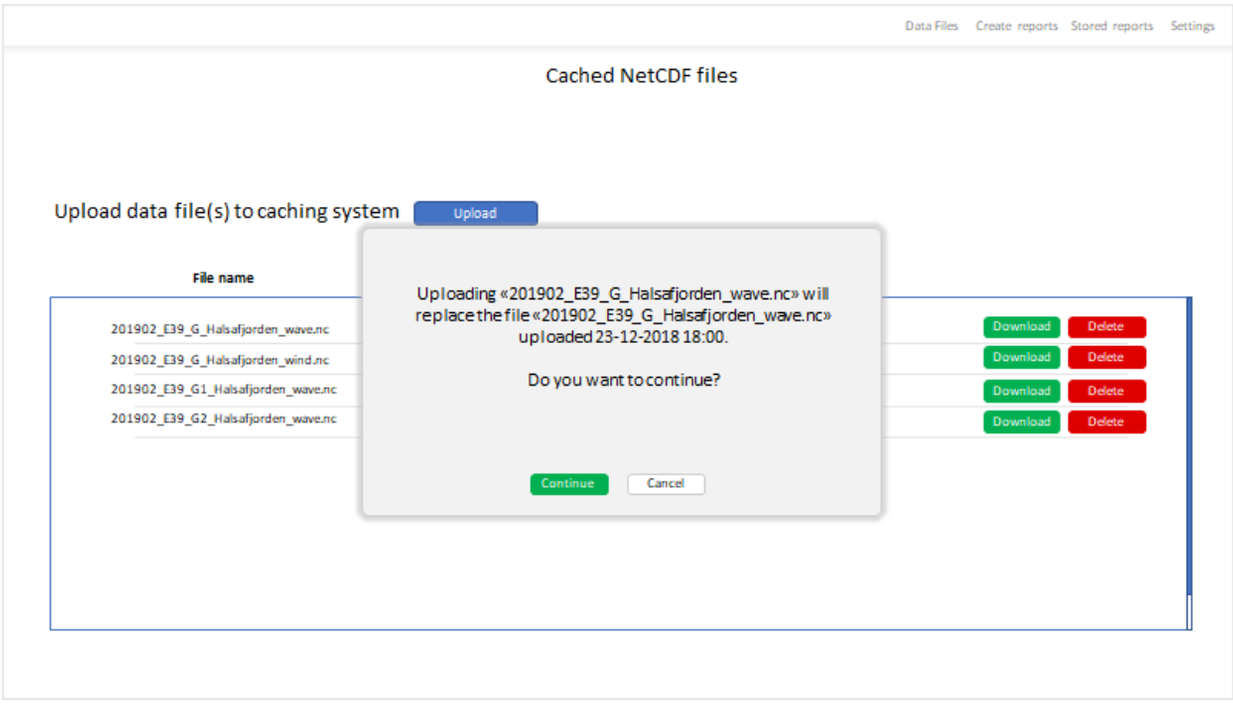
Figur 4.2.2-2 Tekstboks med info når man trykker "Delete" på en rapport.

4.2.3 Data files

En side hvor man manuelt kan lasta opp NetCDF-filer til cachingsystemet for rådatafiler. I tillegg er det info om hvilke filer som allerede ligger cachet, når de ble cachet og hvem eller hva som cachet filene. Dette kan enten være en bruker av systemet, eller det automatiske cachingsystemet. Man har mulighet til å slette filer i cachingsystemet, samt å laste ned filer derfra. Hvis man prøver å laste opp en fil som allerede eksisterer i systemet må man bekrefte dette med et ekstra trykk (Figur 4.2.4-2), før filen blir overskrevet.



Figur 4.2.3-1 Side med mulighet for å laste opp egne NetCDF-filer til mellomlagringssystemet.



Figur 4.2.3-2 Boks for å bekrefte at brukeren ønsker å overskrive lagret fil.

Systemdokumentasjon

Vedlegg C

Bacheloroppgave 2019

Oppgave 62

Dataingeniør

NTNU, institutt for datateknologi og informatikk

Trym Vegard Gjelseth-Borgen

Magnus Conrad Hyll

Trondheim, mai 2019

Innholdsfortegnelse

Tabell- og figurliste

Tabeller

Figurer

1 Introduksjon

Dette dokumentet er skrevet i forbindelse med prosjektet; E39 Fjordkryssing: Automatisering av data-pipeline for analyse av bølgedata. Hensikten med dokumentet er å beskrive systemet som har blitt laget, sånn at det blir enklere å vedlikeholde og eventuelt videreutvikle. Systemdokumentasjonen inneholder beskrivelser av systemets arkitektur, prosjektarkitektur, databasemodell, servertjenester (REST og SignalR), sikkerhet, installasjon og oppsett, dokumentasjon av kildekode og kontinuerlig integrasjon og testing.

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

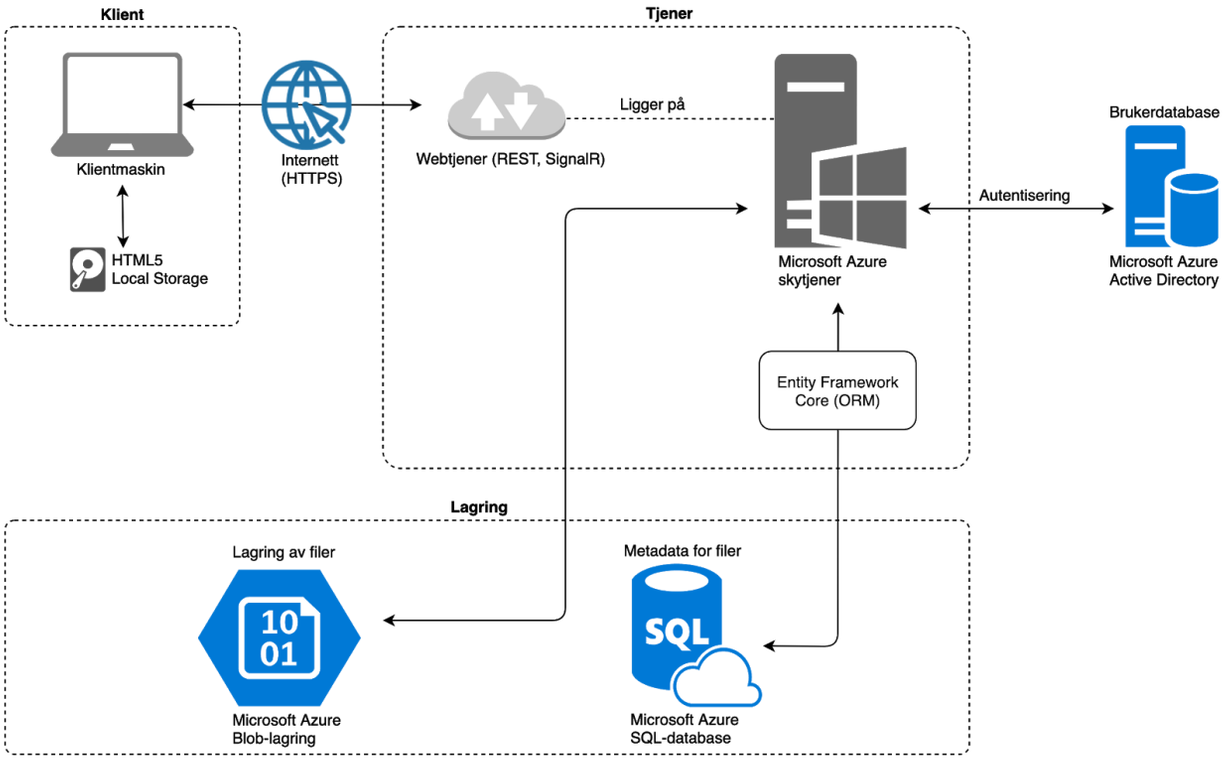
2 Arkitektur

Systemet er delt inn i to hovedkomponenter; et klassebibliotek som prosesserer data og skriver rapporter, og en web-applikasjon som fungerer som et brukergrensesnitt mot klassebiblioteket. Klassebiblioteket inneholder også et enkelt kommandolinjegrensesnitt, slik at en kan laste ned data og generere rapporter uten å sette opp hele web-grensesnittet.

Systemet har en klient-tjener-arkitektur, med en todelt database for lagring. Klient-koden kjører hos brukeren av systemet, som tar nytte av Web Storage for å lagre relevant data for en sesjon i nettleseren. HTTPS-trafikk fra klient blir mottatt hos tjeneren gjennom et REST-api. Tjeneren er en Microsoft Azure skytjener. Lagring av data er delt mellom Microsoft Azure Blob-lagring og Microsoft Azure SQL-database. I Blob-lagringen ligger filene som brukes og produseres av programmet. I SQL-databasen ligger metadata om filene, relasjoner mellom produserte rapporter og rådatafiler, og data for beregning av kjøretid for programmet.

2.1 Arkitekturskisse

Skissen viser de viktigste komponentene for E39 Wave Master. Figuren består av klient, tjener og database/lagring og viser hvordan de kommuniserer med hverandre.

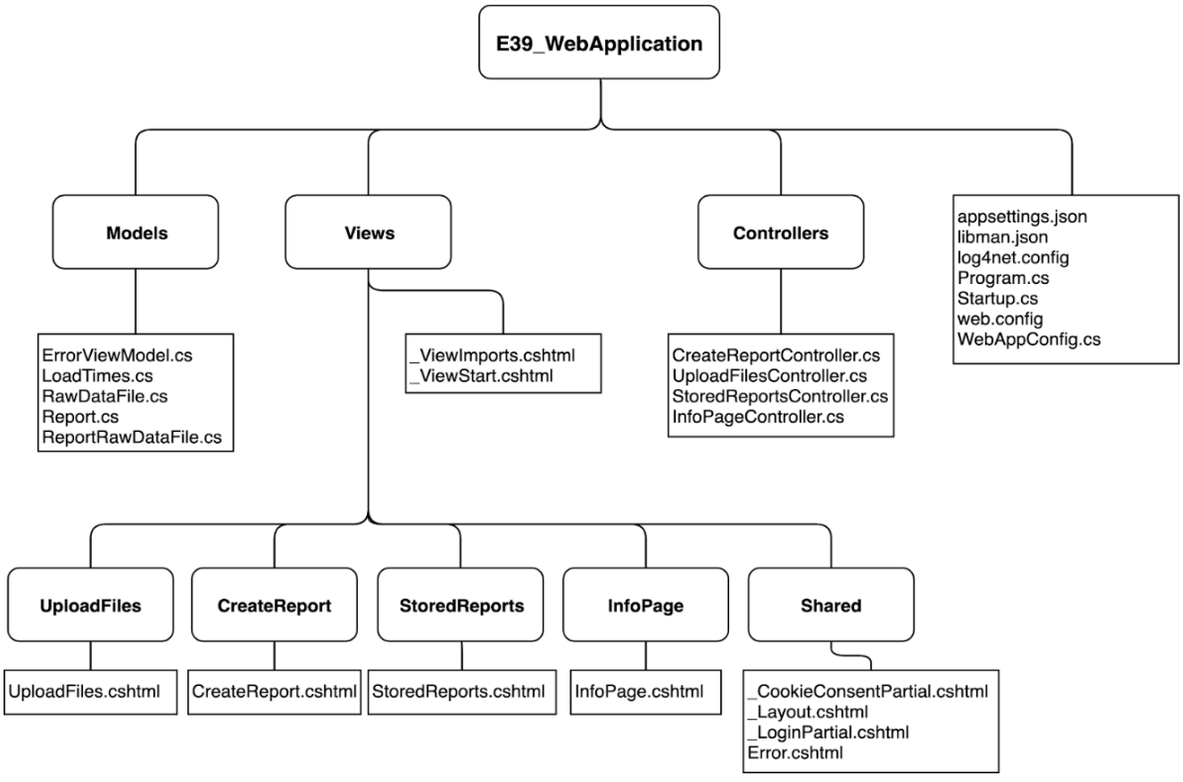


Figur 2.1-1 Arkitekturskisse for systemet.

3 Prosjektstruktur

3.1 MVC

Webapplikasjonen er lagd i henhold til et MVC designmønster, med ASP.NET Core MVC.



Figur 3.1-1 Visualisering av mappestruktur for MVC-delen av web-applikasjonen.

3.1.1 Beskrivelse av mappestruktur

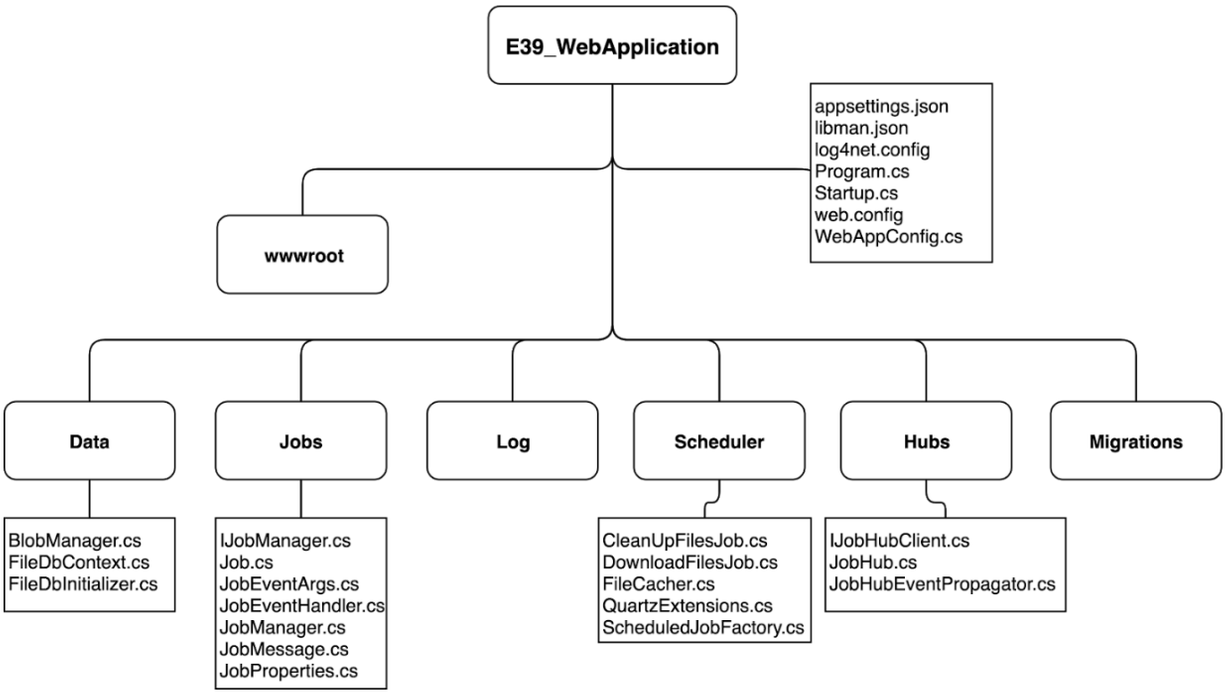
Tabellen inneholder en beskrivelse av mappe og filstruktur for filer og mapper inneholdt i MVC. Avrundede bokser er mapper og firkantede bokser inneholder filene som er i den overordnede mappen.

Tabell 3.1.1-A Beskrivelse av mappestruktur for MVC-delen av web-applikasjonen.

Mappenavn	**Beskrivelse**
Models	Objekter som blir sendt i forbindelsene klient-tjener og tjener-database.
View	Beskrivelsen av hvordan det grafiske brukergrensesnittet på klientsiden skal se ut.
Controllers	Klassene som håndterer forespørsler via HTTPS fra en klients nettleser.
Views/UploadFiles	Beskrivelsen av GUI-et for siden UploadFiles
Views/CreateReport	Beskrivelsen av GUI-et for siden CreateReport
Views/StoredReports	Beskrivelsen av GUI-et for siden StoredReports
Views/InfoPage	Beskrivelsen av GUI-et for siden InfoPage
Views/Shared	Beskrivelsen av GUI-elementer som er felles for alle Views.

3.2 Andre filer i E39_WebApplication

Filer som ikke direkte omhandler MVC, men som fortsatt tilhører webapplikasjonen.



Figur 3.2-1 Visualisering av mappestruktur for deler av web-applikasjonen som ikke direkte omhandler MVC.

3.2.1 Beskrivelse av mappestruktur

Tabellen inneholder en beskrivelse av mappe og filstruktur for filer og mapper utenfor MVC i E39_WebApplication. Avrundede bokser er mapper og firkantede bokser inneholder filene som er i den overordnede mappen.

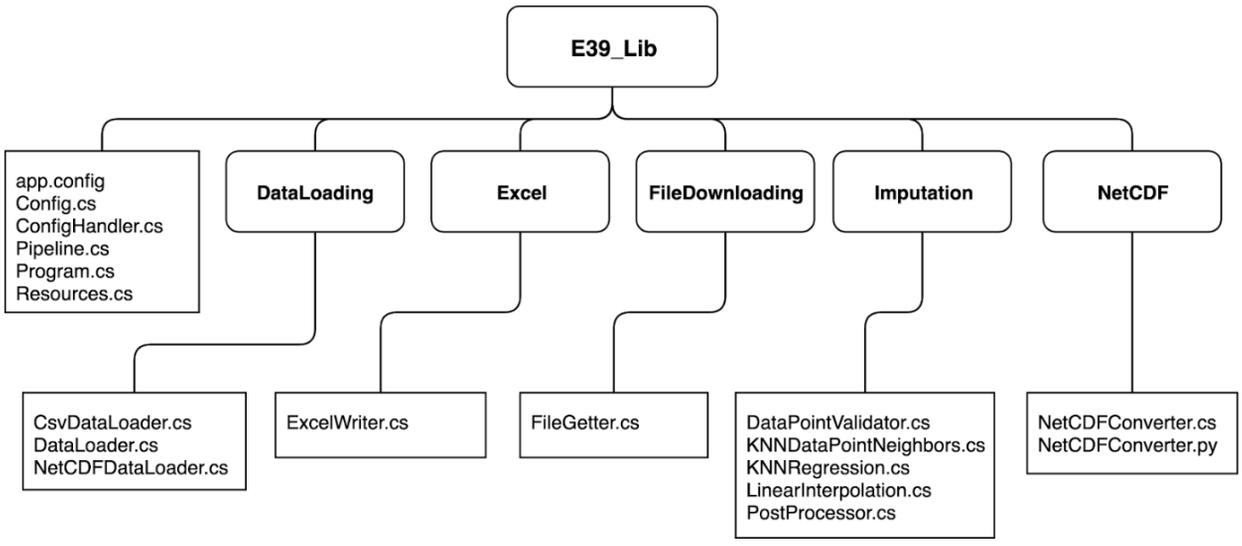
Tabell 3.2.1-A Beskrivelse av mappestruktur for deler av web-applikasjonen som ikke direkte omhandler MVC.

Mappenavn	**Beskrivelse**
wwwroot	Alle statiske filter, som inkluderer CSS, javascript, bilder, favicon, samt javascript- og CSS-biblioteker.
Data	Klasser som har ansvaret for å laste opp og ned filer fra Blob-lagringen, og databasekontekst for å hente og lagre data mot SQL-databasen gjennom EF Core.
Jobs	Klasser som tar seg av kjøring av jobber (jobb, se Hovedrapport kap. 1.2) for generering av rapporter, opprettelse av jobber, og hendelser knyttet til endringer i status for jobber.
Log	Systemlogger som produseres under kjøring.
Scheduler	Klasser som tar hånd om planlagte periodiske oppgaver, samt implementasjonene av oppgavene som skal kjøres periodisk.

Hubs	Klasser som definerer tjenerens SignalR-metoder, samt klasse som knytter publisering av jobb-meldinger til SignalR-metoder hos klienter.
Migrations	Sjekkpunkter ved oppdatering av databas strukturen. Inneholder automatisk genererte filer av Entity Framework Core Migrations.

3.3 E39_Lib

E39_Lib er et klassebibliotek uavhengig av web-applikasjonen. Det inneholder klasser som kan laste ned og konvertere data, utføre kNN-regresjon, laste inn data fra CSV-filer og lage fullstendige Excel-rapporter.



Figur 3.3-1 Visualisering av mappestrukturen for E39_Lib.

3.3.1 Beskrivelse av mappestruktur

Tabellen inneholder en beskrivelse av mappe og filstruktur for E39_Lib. Avrundede bokser er mapper og firkantede bokser inneholder filene som er i den overordnede mappen.

Tabell 3.3.1-A Beskrivelse av mappestrukturen for E39_Lib.

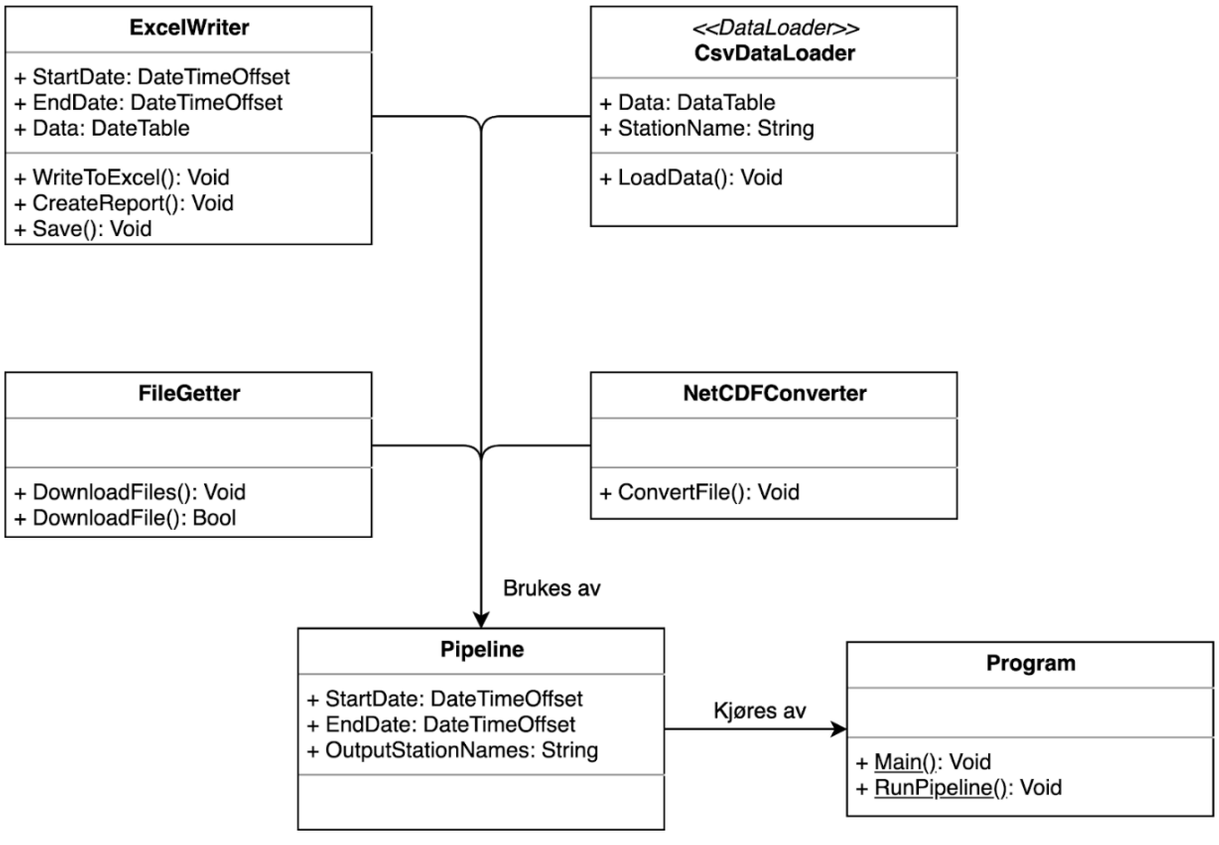
Mappenavn	**Beskrivelse**
DataLoading	Klasser som tar seg av innlasting av data til DataTables fra CSV- og NetCDF-filer
Excel	Klasse for behandling av Excel-filer.
FileDownloading	Klasse som laster ned filer gitt en fil-tjener.
Imputation	Klasser for å utføre KNN-regresjon på datasett der det mangler datapunkter.

NetCDF	Klasse som kaller et Python-script som utfører konvertering av NetCDF-filer til CSV-format.

4 Klassediagram

4.1 Klassediagram for E39_Lib

Klassediagrammet i figur 4.1-1 er et forenklet klassediagram for klassebiblioteket E39_Lib. Diagrammet viser hvordan de fire klassene *ExcelWriter*, *CsvDataLoader*, *FileGetter* og *NetCDFConverter* brukes individuelt av klassen *Pipeline*. Klassen *Program* kjører klassen *Pipeline* og gir den alle nødvendige parametere.

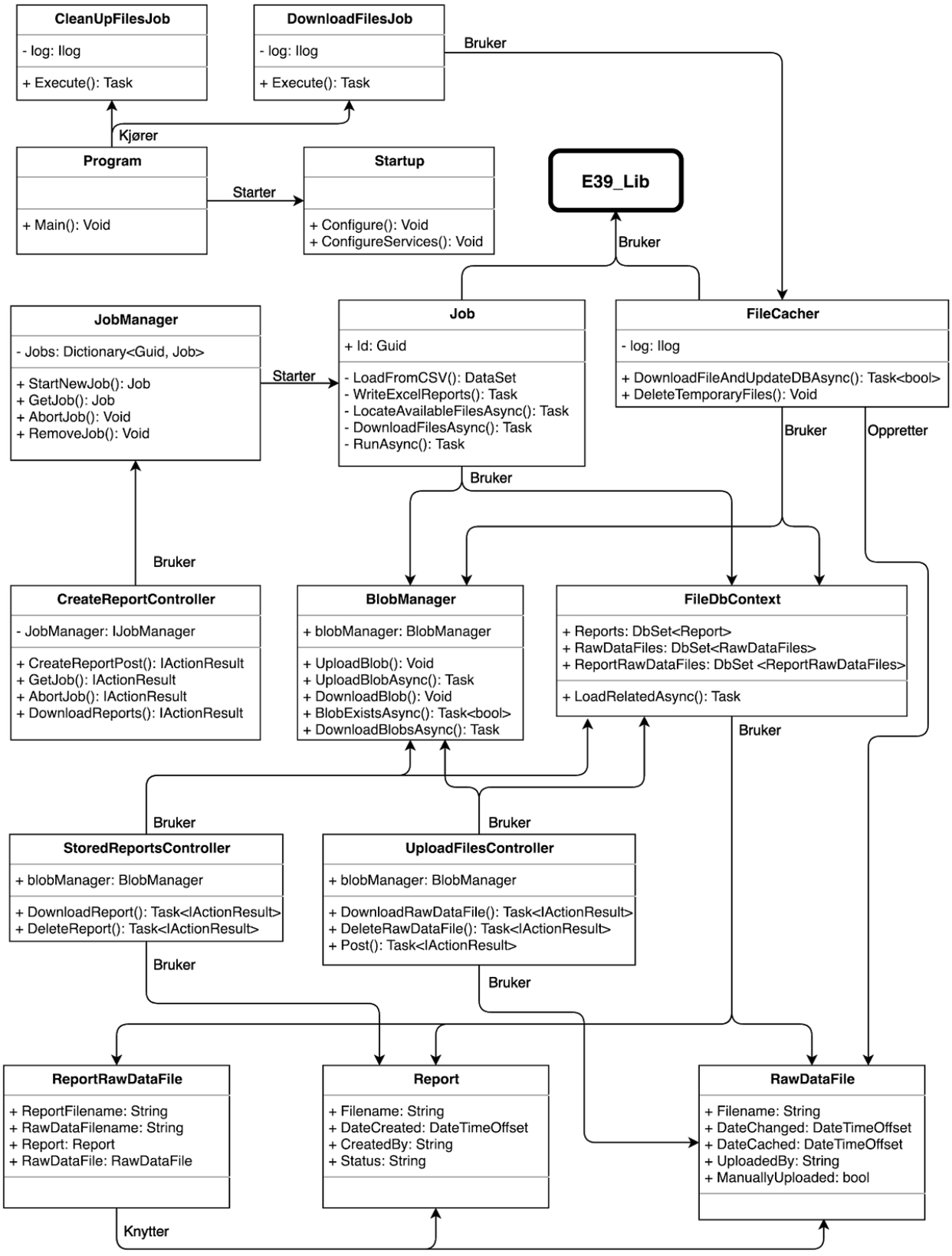


Figur 4.1-1 Forenklet klassediagram for klassebiblioteket E39_Lib.

4.2 Klassediagram for web-applikasjonen

Diagrammet i figur 4.2-1 er et forenklet klassediagram. Klassen *Program* er hovedklassen og inneholder funksjonen *Main()*. Den setter i gang klassen *Startup*, *CleanUpFilesJob* og *DownloadFilesJob*. *Startup* har ansvaret for å sette i gang web-applikasjonen. *CleanUpFilesJob* og *DownloadFilesJob* er begge periodiske oppgaver som kjøres på bestemte

intervaller. *Job*-klassen inneholder hovedloopen for generering av rapporter, og blir satt i gang av klassen *JobManager*. Klassene *Job* og *FileCacher* tar direkte nytte av klassebiblioteket *E39_Lib*.

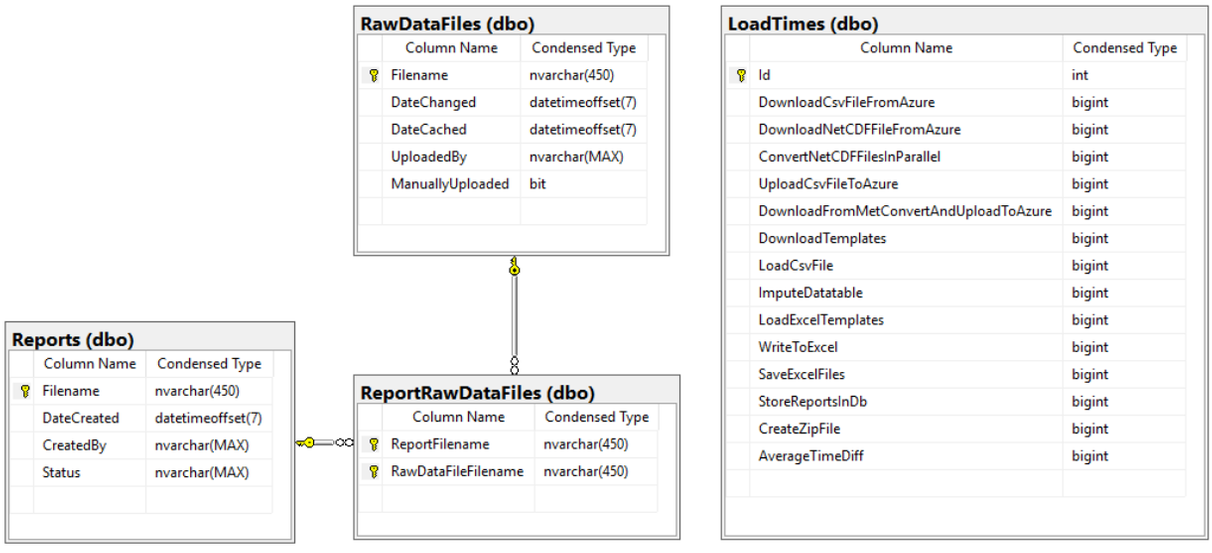


Figur 4.2-1 Forenklet klassediagram for web-applikasjondelen av systemet.

5 Databasemodell

5.1 Modell for SQL-database

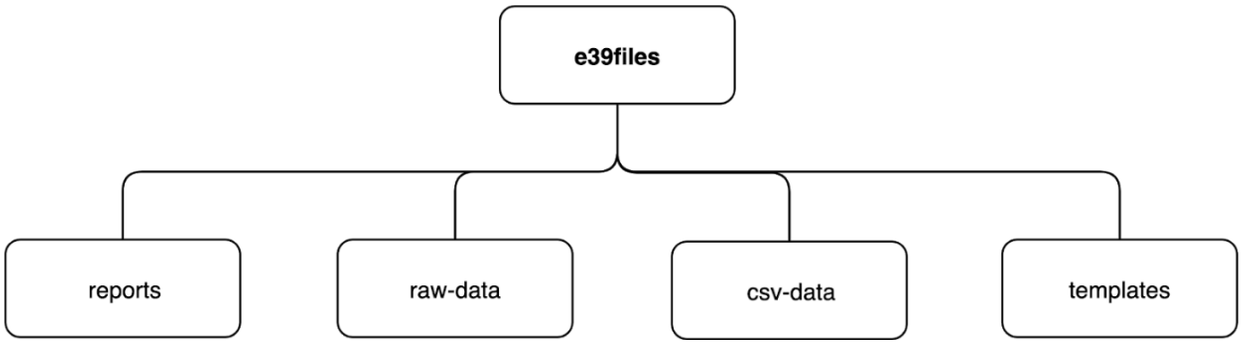
Databasen er en Microsoft Azure SQL Database som består av fire tabeller, hvorav tre er koblet sammen. RawDataFiles inneholder metadata om rådatafiler, Reports inneholder metadata om rapportene og ReportRawDataFiles er en mange-til-mange-tabell som kobler sammen RawDataFiles og Reports. LoadTimes er en tabell som inneholder informasjon om hvor lang kjøretid hver del av programmet bruker.



Figur 5.1-1 Databasemodell for systemet.

5.2 Modell for Blob-lagring

Filene ligger lagret i en Microsoft Azure Blob Storage.



Figur 5.2-1 Visualisering av strukturen for blob-lagringen.

5.2.1 Beskrivelse av mappestruktur

Tabellen inneholder en beskrivelse av mappestruktur for Blob-lagringen.

Tabell 5.2.1-A Beskrivelse av strukturen for blob-lagringen.

Mappenavn	**Beskrivelse**
reports	Mappe for genererte rapporter.
rawdata	Mappe for NetCDF-filer.
csvfiles	Mappe for CSV-filer.
templates	Mappe som inneholder malene brukt for å lage rapporter.

6 Server-tjenester

Serveren bruker er REST-API for å ta imot HTTPS-forespørsler fra klient, og sende svar tilbake. I tillegg er det definert SignalR-metoder både hos klienten og tjeneren, som benyttes til å sende oppdateringer på statusen til en kjørende jobb i sanntid.

6.1 REST-API

6.1.1 CreateReport

Tabell 6.1.1-A REST-endepunkter for "/CreateReport/" og "/".

Endepunkt	**Metode**	**Beskrivelse**
/station-names	GET	Returnerer alle stasjonsnavnene som en liste.
/jobs	POST	Tar inn et objekt av typen JobProperties, og starter å lage en ny rapport. GUID blir sendt tilbake til klient.
/jobs/{jobId}	GET	Returnerer statusen til jobben med ID {jobId} hvis den eksisterer.
/jobs/{jobId}/abort	POST	Avslutter jobben med ID {jobId} hvis den eksisterer.
/jobs/{jobId}/reports	GET	Returnerer rapporter fullført av jobben med ID {jobID}, hvis den eksisterer.

6.1.2 StoredReports

Tabell 6.1.2-A REST-endepunkter for "/StoredReports/".

Endepunkt	**Metode**	**Beskrivelse**

/{reportFilename}	GET	Omdirigerer klient til en URL hvor nedlasting fra Blob-lagring av en rapport med filnavn {reportFilename} starter automatisk, hvis filen eksisterer.
/{reportFilename}	DELETE	Sletter en rapport fra Blob-lagring med filnavn {reportFilename}, hvis filen eksisterer.

6.1.3 RawDataFiles

Tabell 6.1.3-A REST-endepunkter for "/RawDataFiles/".

Endepunkt	Metode	Beskrivelse
/	POST	Tar en list av IFormFile-objekter. Laster deretter disse filene opp i Blob-storage, sletter tilhørende CSV-filer, og oppdaterer databasen med fil-metadata.
/{filename}	GET	Omdirigerer klient til en URL hvor nedlasting fra Blob-lagring av en rådatafil med filnavn {filename} starter automatisk, hvis filen eksisterer.
/{filename}	DELETE	Sletter en rådatafil og den tilhørende CSV-filen fra Blob-lagring med filnavn {filename}, hvis filen eksisterer.

6.2 SignalR-metoder

6.2.1 Klientmetoder

Følgende metoder er definert for klientene, og kalles på i C#-koden hos tjeneren. I alle tilfeller kaller tjeneren klientmetodene på alle klienter som er tilkoblet SignalR-huben til enhver tid, uavhengig av hvilken jobb en klient lytter til. Klientene sjekker derfor om ID-en til jobben som sendte meldingen er lik jobb-ID-en som klienten har lagret i LocalStorage, hvilket betyr at klienten "lytter" til jobben.

Tabell 6.2.1-A Klientmetoder for SignalR.

Metodenavn	Beskrivelse
JobMessageAdded	Legger til en melding i logg-boksen i GUIet. Kalles når en jobb legger til en ny melding.
JobStateChanged	Oppdaterer GUIet tilsvarende den nye tilstanden til jobben. Kalles av tjeneren når en jobb endrer tilstand.
JobTimeFinishedEstimated	Oppdaterer nedtellingen for gjenstående tid for jobben i GUIet. Kalles av tjeneren når en jobb oppdaterer forventet tidspunkt ferdig.

JobProgressChanged	Oppdaterer indikatoren for framgang i GUIet. Kalles av tjeneren når en jobb endrer verdien for framgang.
ReceiveAllJobInfo	Oppdaterer GUIet tilsvarende jobb-tilstanden, meldingene, jobbens framgang og estimert tid ferdig. Kalles av tjeneren etter en klient har gjort et kall på tjenermetoden GetAllJobInfo.

6.2.2 Tjenermetoder

Følgende metoder er definert for tjeneren, og kalles på i Javascript-koden hos klienter.

Tabell 6.2.2-A Tjenermetoder for SignalR.

Metodenavn	**Beskrivelse**
GetAllJobInfo	Kalles av klienter når nettsiden lastes inn, dersom klienten lytter til en jobb. Kalles også av klienter etter en jobb startes. Forespør at tjeneren gjør et kall tilbake på klientene på klientmetoden ReceiveAllJobInfo med all relevant informasjon om en jobb.

7 Sikkerhet

7.1 HTTPS

Web-applikasjonen påtvinger at all kommunikasjon foregår over HTTPS.

7.2 Autentisering og identitet

Systemet autentiserer brukerne mot en Azure Active Directory. Denne løsningen benytter identitetsrammeverket OpenID Connect, som er basert på OAuth 2.0 [2]. Systemet lagrer derfor ingen sensitive brukerdata eller passord for autentisering.

7.3 ORM som beskyttelse mot SQL-injiseringer

Systemet benytter en SQL-database for metadata om rapporter, rådatafiler og knytningene mellom dem. Applikasjonen bruker ORM-en (Object-Relational Mapper) Entity Framework Core for å hente fra og lagre data til denne databasen. Alle kall mot databasen gjøres som metodekall mot EF Core, som gir oss dataene automatisk representert som C#-objekter. På denne måten slipper man å gjøre databasespørringer i form av SQL-spørresetninger, slik at man unngår faren for SQL-injiseringer.

7.4 Connection Strings

Tilkoblingsstrenger (Connection Strings) for oppkobling til både SQL-database og Blob-lagring i Microsoft Azure ligger trygt lagret i en konfigurasjon som er separert fra kildekoden.

8 Installasjon og kjøring

8.1 Avhengigheter

Systemet avhenger av .NET Core 2.2 SDK for å kunne kompileres og kjøres. I tillegg avhenger løsningen av Python 3.6 for å kjøre konverteringsskriptet for NetCDF-filer. Begge disse avhengighetene må installeres før man går videre i denne installasjonsguiden. Ved installering av Python inkluderes pakkehåndteringsverktøyet pip.

.NET Core 2.2 SDK lastes ned og installeres fra: https://dotnet.microsoft.com/download
Python 3.6 lastes ned og installeres fra: https://www.python.org/downloads/

Merk at systemet er testet og utviklet med de versjonene av .NET Core og Python som er nevnt over, men det utelukkes ikke at systemet er kompatibelt med andre og nyere versjoner.

8.2 Installasjon av klassebiblioteket og kommandolinjeapplikasjon

Åpne et kommandolinjeskall. Naviger til mappen hvor kildekoden til systemet befinner seg, og deretter til mappen E39_Lib.

Vi må først installere de nødvendige Python-pakkene for konvertering av NetCDF-filer. Dette gjøres med følgende kommando:

```
pip install -r NetCDF/requirements.txt
```

Deretter må vi laste ned de nødvendige .NET-pakkene fra NuGet. Dette gjøres med kommandoen:

```
dotnet restore
```

Klassebiblioteket er nå klar for å kompileres med kommandoen:

```
dotnet build
```

Det ferdigbygde biblioteket ligger nå i `bin/Debug/netcoreapp2.2/`. I denne mappen finner du E39_Lib.dll, som er den kompilerte programvaren, E39_Lib.dll.config som er konfigurasjonsfila for biblioteket, og NetCDFConverter.py som er Python-skriptet for konvertering av NetCDF-filer.

Klassebiblioteket inkluderer et enkelt kommandolinjegrensesnitt for å kjøre E39-pipelinen (pipeline, se Hovedrapport kap. 1.2), slik at en kan generere rapporter direkte uten web-grensesnittet. For å kjøre dette navigerer du til mappen med den kompilerte fila E39_Lib.dll og kjører kommandoen:

```
dotnet E39_Lib.dll
```

8.3 Installasjon og kjøring av web-applikasjonen

Før du gjennomfører følgende steg for bygging av web-applikasjonen forutsettes det at du har bygget klassebiblioteket ved stegene beskrevet i kap. 8.1.

Først må du konfigurere tilkoblingsstrenger for SQL-databasen og Azure Blob Storage. Disse setter du i filen appsettings.json, ved å legge til attributten "ConnectionStrings" som igjen inneholder attributtene "FileDb" og "e39files_AzureStorageConnectionString", for henholdsvis SQL-databasen og Azure Blob Storage.

For å konfigurere autentisering mot Azure AD følger du dokumentasjonen til Microsoft: https://docs.microsoft.com/en-us/azure/active-directory/develop/quickstart-v2-aspnet-core-webapp

Nå kan du bygge web-applikasjonen ved åpne et kommandolinjeskall, navigere til mappen hvor kildekoden til systemet befinner seg, og deretter til mappen E39_WebApplication.

Laste ned de nødvendge .NET-pakkene fra NuGet med kommandoen:

```
dotnet restore
```

Web-applikasjonen bygges og kjøres dermed med kommandoen:

```
dotnet run
```

Dersom du ønsker å konfigurere automatisk utrulling mot Azure App Service følger du dokumentasjonen til Microsoft:
https://docs.microsoft.com/en-us/azure/app-service/deploy-continuous-deployment

9 Dokumentasjon av kildekode

De offentlige metodene og variablene sitt API er dokumentert som XML-dokumentasjonskommentarer i kildekoden, både for klassebiblioteket og for web-applikasjonen. For å generere HTML-sider med API-dokumentasjon kan man eksempelvis benytte verktøyet Doxygen, som finnes på http://www.doxygen.nl/.

10 Kontinuerlig integrasjon og testing

Det er ikke satt opp kontinuerlig integrasjon med automatisk testing, men det er satt opp kontinuerlig utrulling mot Microsoft Azure. Når man pusher endringer til git-grenen "deployment_test" i repositoriet på GitHub, blir systemet bygget i Azure, og den nye versjonen blir automatisk rullet ut til App Servicen.

11 Referanser

[1] S. Smith et. al., Overview of ASP.NET Core MVC, 2018. Hentet fra: https://docs.microsoft.com/nb-no/aspnet/core/mvc/overview?view=aspnetcore-2.2. Lastet ned: 30.04.2019

[2] OpenID Connect. Hentet fra: https://openid.net/connect/. Lastet ned: 20.05.2019