

# DATA3900 Bacheloroppgave

Hovedprosjektets tittel

System for behandling av kreftmeldinger

Skrevet av:

- Jørund Topp Løvlien, s341822
- Nikola Dordevic, s341839
- Ola Gynnild Berg, s341874
- Tom Henrik Melting Basmo, s340432
- Hajin Barzingi, s238727



Et samarbeid mellom Kreftregisteret og studenter ved OsloMet - storbyuniversitet



Institutt for Informasjonsteknologi

Postadresse: Postboks 4 St. Olavs plass, 0130 Oslo

Besøksadresse: Holbergs plass, Oslo

PROSJEKT NR.

2

TILGJENGELIGHET

Åpen

# BACHELORPROSJEKT

Telefon: 22 45 32 00

|   |                           |
|---|---------------------------|
| HOVEDPROSJEKTETS TITTEL   | DATO                      |
| System for behandling av kreftmeldinger.  | 25.05.2022                |
|   | ANTALL SIDER / BILAG      |
|   | 197 / 60                  |
| PROSJEKTDELTAKERE   | INTERN VEILEDER           |
| Jørund Topp Løvlien, s341822<br>Nikola Dordevic, s341839<br>Ola Gynnild Berg, s341874<br>Tom Henrik Melting Basmo, s340432<br>Hajin Barzingi, s238727 | George Anthony Giannoumis |

|                |                  |
|----------------|------------------|
| OPPDRAKGIVER   | KONTAKTPERSON    |
| Krefregisteret | Jan Franz Nygård |

|   |
|---|
| SAMMENDRAG  |
| Denne rapporten beskriver utviklingen av et system for behandling av kreftmeldinger med forklaringer av oppgavens bakgrunn, mål, prosess og sluttprodukt.   |
| Prosjektet ble utført som et Scrum-prosjekt med tett samarbeid med produkteier og brukerne. Hensikten med applikasjonen er å effektivisere bruken og forbedre brukeropplevelsen, samt å lage en modulær og skalerbar løsning. |
| 3 STIKKORD  |
| XML, Java, Surveyjs   |

# Forord

Sluttproduktet er **tilgjengelig** som en webapplikasjon på <https://demokrg.herokuapp.com/> (sist sjekket 24.05.2022). Det kan ta fra 30 til 60 sekunder før siden laster inn.

Denne rapporten er en del av bacheloroppgaven vår i ingeniørfag-data ved OsloMet – Storbyuniversitet. Rapporten inneholder bakgrunn for oppgaven, relevant teori og beskrivelse av produktet og prosessen som førte til sluttproduktet – Et nytt system for behandling av kreftmeldinger. Prosjektet har vært utfordrende, men givende. At systemet skal videreutvikles i sommer og bli integrert i Kreftregistrets fagsystemer, betyr at systemet blir et viktig ledd i kreftforskningen. Dette, i tillegg til den gode oppfølgingen fra våre veiledere Jan Franz Nygård og George Anthony Giannoumis, har holdt motivasjonen vår oppe og vi er stolte av sluttproduktet vi leverer.

I tillegg til en stor takk til våre veiledere, vil vi også takke:

- Andreas, Thomas og Gintaras fra Kreftregisteret som hver uke har vært med på statusmøter og tatt imot spørsmål fra oss på strak arm.
- Hedvig som har veiledet oss rundt det skriftlige i bacheloroppgaven.

Ved bruk av den opplastede webapplikasjonen, gjøres det oppmerksom på at brukere **ikke** kan fjerne meldinger. Vær derfor svært forsiktig med å laste opp persondata.

# Leserveiledning

Denne rapporten består av tre større uavhengige dokumenter: Oppgavens bakgrunn og mål, Prosessdokumentasjon og Produktdokumentasjon. Starten av rapporten inneholder en felles ordliste med typiske teknologiske ord og uttrykk med beskrivelse av hva de står for. I siste del av rapporten har vi inkludert en felles konklusjon basert på problemstillingen i «Oppgavens bakgrunn og mål», en refleksjon og videre anbefaling som samler sammen essensen i dokumentene. For å gi deg som leser en lettere leseopplevelse og mulighet til å følge flyten, fortsetter hver hoveddel med kontinuerlig nummerering fra forrige del. I tillegg er skjermbilder av koder og brukergrensesnitt lagt til for å gi en bedre forståelse av avsnittene, men det er også mulig å se på koden gjennom å følge lenken til vår kodebase:

<https://github.com/bachelor-krefregisteret/System-for-behandling-av-XML-meldinger>

God lesning.

# Ordliste

**API (Application programming interface):** Et programmerings-grensesnitt, som gjør det mulig å anvende deler fra en programvare i en annen programvare.

**Array:** En datatype som brukes til å definere variabler som er tabeller. Også kjent som matrise.

**Backend:** Også kalt tjener, tjener-del. Den delen av en applikasjon som håndterer data, og som brukeren ikke direkte kan aksessere.

**Bug:** Feil i programvaren.

**Branch:** En gren er en versjon av systemet.

**Controller:** En klasse i backend, ofte en del av «Model-View-Controller» designmønsteret, som kontrollerer hvordan data blir vist frem gjennom API-endepunkt.

**Commit:** Oppdatering av repositoriet med lokale endringer.

**Domene:** Et administrativt delområde i et datasystem i et datasystem eller datanettverk

**End-of-life:** Slutten i livssyklusen for et produkt.

**Frontend:** Også kalt klient-del. Den delen av en applikasjon brukeren direkte berører.

**GDPR:** Personvernforordningen, som er en lov EU har vedtatt som skal beskytte personvernopplysninger.

**Git:** Et versjonskontrollsysten.

**Interface:** Abstraksjon i Java. Tvinger relaterte klasser til å følge samme oppsett som interfacet.

**Java:** Et Objektorientert programmeringsspråk.

**JSON:** Dataformat basert på JavaScript. Står for JavaScript Object Notation.

**Klasse:** En måte å dele opp programkode på som samler lignende tilfeller av data.

**Kodebase:** Samlingen av all kode brukt i en applikasjon eller programvare.

**Kontroller:** En (input) kontroller er et grensesnittelement som brukeren anvender for å oppnå et mål.

**Legacy:** Utdatert.

**LTS:** Long-time-support. På norsk langtidsstøtte. Versjon som er anbefalt av utvikler.

**Melding:** Fellesbetegnelse på alle XML-filer som systemet behandler, det vil si kreftmeldingene som Kreftregisteret mottar, redigerer og lagrer.

**Modell (klasse):** En modell er en klasse som representerer data, i vår sammenheng representerer modellen ressursen *Kreftmeldingen*.

**Parse:** Her brukt som konvertering fra en datatype til en annen.

**POJO:** Forkortelse for «Plain old Java Object». Et vanlig Java-objekt, som ikke har spesielle begrensninger.

**Push:** Oppdatere lokale endringer til ekstern repositoriet (repo).

**Refaktorering:** Også kalt omstrukturering av kode, en måte å kontinuerlig eller trinnvis forbedre kvaliteten på kode.

**Rip-and-replace:** Lage noe som kan erstatte eksisterende system med minimal dødtid.

**Sentence case:** Stor forbokstav kun i første ordet i en setning. Uansett lengde på setning.

**XML-skjema:** Et XML-skjema eller en XSD-fil er en fil som beskriver strukturen til et XML-dokument.

# Innholdsliste

|  |           |
|--|-----------|
| Forord.....                              | 3         |
| Leserveiledning.....                     | 4         |
| Ordliste.....                            | 5         |
| <b>Oppgavens bakgrunn og mål .....</b>   | <b>13</b> |
| 1. Aktører .....                         | 14        |
| 1.1. Prosjektgruppen.....                | 14        |
| 1.2. Oppdragsgiver .....                 | 15        |
| 1.3. Veiledere .....                     | 16        |
| 1.3.1. Kreftregisteret.....              | 16        |
| 1.3.2. OsloMet – Storbyuniversitet ..... | 16        |
| 2. Dagens situasjon.....                 | 17        |
| 3. Problemstilling.....                  | 19        |
| 4. Mål og rammebetingelser .....         | 20        |
| 4.1. Funksjonelle krav.....              | 21        |
| 4.2. Ikke-funksjonelle krav.....         | 21        |
| 4.3. Organisatoriske krav.....           | 22        |
| 4.4. Eksterne krav.....                  | 22        |
| <b>Prosessdokumentasjon .....</b>        | <b>23</b> |
| 5. Metoder og verktøy.....               | 24        |
| 5.1. Smidig Metodikk.....                | 24        |

|          |   |    |
|----------|---|----|
| 5.1.1.   | Scrum .....                                 | 25 |
| 5.1.2.   | Scrum master .....                          | 26 |
| 5.1.3.   | Minimum viable product.....                 | 26 |
| 5.2.     | Parprogrammering .....                      | 27 |
| 5.3.     | Prosjekttavle.....                          | 29 |
| 5.4.     | Brukerhistorier .....                       | 30 |
| 5.5.     | Bakgrunnslitteratur .....                   | 31 |
| 5.5.1.   | Brukersentrert design og teori .....        | 31 |
| 5.5.1.1. | Forstå og spesifisere brukskonteksten ..... | 32 |
| 5.5.1.2. | Spesifiser krav/behov .....                 | 33 |
| 5.5.1.3. | Designforslag .....                         | 34 |
| 5.5.1.4. | Evaluere design .....                       | 40 |
| 5.5.2.   | Backend – teori .....                       | 41 |
| 5.5.2.1. | Designmønstre .....                         | 41 |
| 5.5.2.2. | Programmeringsprinsipper og teknikker ..... | 42 |
| 5.5.2.3. | Testing .....                               | 43 |
| 6.       | Planleggingsfasen .....                     | 45 |
| 6.1.     | Teknologier.....                            | 45 |
| 6.1.1.   | Frontend.....                               | 45 |
| 6.1.1.1. | SurveyJs og React .....                     | 45 |
| 6.1.1.2. | Node.js .....                               | 46 |
| 6.1.1.3. | Reactstrap og Axios .....                   | 46 |
| 6.1.1.4. | Figma .....                                 | 46 |

|                                      |           |
|--------------------------------------|-----------|
| 6.1.2. Backend.....                  | 46        |
| 6.1.2.1. Java.....                   | 46        |
| 6.1.2.2. Spring Boot.....            | 47        |
| 6.1.2.3. JAXB.....                   | 47        |
| 6.1.3. Prosjektstyringsverktøy ..... | 47        |
| 6.1.4. Andre verktøy.....            | 48        |
| 6.2. Fremdriftsplan .....            | 49        |
| 6.3. Milepæl .....                   | 50        |
| 7. Utviklingsfasen .....             | 52        |
| 7.1. Sprint 1.....                   | 52        |
| 7.1.1. Oppgaver.....                 | 52        |
| 7.1.2. Arbeidet .....                | 53        |
| 7.2. Sprint 2.....                   | 54        |
| 7.2.1. Oppgaver.....                 | 54        |
| 7.2.2. Arbeidet .....                | 54        |
| 7.3. Sprint 3.....                   | 54        |
| 7.3.1. Oppgaver.....                 | 54        |
| 7.3.2. Arbeidet .....                | 55        |
| 8. Avslutningsfasen.....             | 56        |
| <b>Produktdokumentasjon .....</b>    | <b>57</b> |
| 9. Design.....                       | 58        |
| 9.1. Presentasjon av design .....    | 58        |
| 9.1.1. Personas.....                 | 59        |

|  |    |
|--|----|
| 9.1.2. Designforslag.....                                    | 62 |
| 9.1.3. SurveyJs og brukervennlighet.....                     | 65 |
| 9.1.3.1. Tilbakemeldinger.....                               | 65 |
| 9.1.3.2. Validering .....                                    | 66 |
| 9.1.3.3. Søkbare nedtrekkmenyer .....                        | 66 |
| 9.1.3.4. Modal for bekreftelse/tilbakemelding.....           | 67 |
| 9.1.3.5. Trykkbare beskrivelser/ledetekster .....            | 68 |
| 9.2. Brukertesting.....                                      | 69 |
| 9.2.1. Første brukertesting.....                             | 69 |
| 9.2.2. Andre brukertesting .....                             | 71 |
| 9.3. Universell utforming.....                               | 73 |
| 9.3.1. Tilgjengelighet for brukere med synsutfordringer..... | 73 |
| 9.3.2. Tastaturnavigasjon .....                              | 74 |
| 9.3.3. Tilgjengelig modalvindu .....                         | 74 |
| 9.4. Endelig design.....                                     | 75 |
| 10. Frontend .....   | 76 |
| 10.1. Komponenter og funksjoner .....                        | 76 |
| 10.1.1. GetMeldinger .....                                   | 77 |
| 10.1.2. Dashboard .....                                      | 79 |
| 10.1.2.1. Utforming av dashboard.....                        | 79 |
| 10.1.2.2. MeldingList .....                                  | 80 |
| 10.1.3. FormLogic .....                                      | 81 |
| 10.1.3.1. Sidebar .....                                      | 83 |

|  |     |
|--|-----|
| 10.1.3.2. Footer.....                    | 87  |
| 10.1.3.3. ConfirmationModal .....        | 88  |
| 10.2. Skjemaenes oppbygning .....        | 90  |
| 10.2.1. Siders oppbygning .....          | 91  |
| 10.2.2. Elementers oppbygning.....       | 92  |
| 10.2.2.1. Inputfelt.....                 | 92  |
| 10.2.2.2. Nedtrekkmenyer .....           | 94  |
| 10.2.2.3. Radiokapper .....              | 96  |
| 10.2.2.4. Avkryssingsbokser .....        | 97  |
| 10.2.3. Skjemalogikk.....                | 98  |
| 10.3. Datahåndtering i skjema.....       | 99  |
| 10.3.1. Innlasting av data .....         | 99  |
| 10.3.2. Innsetting av data.....          | 99  |
| 10.3.3. Endring og lagring av data ..... | 102 |
| 10.4. Kommunikasjon med ELVIS.....       | 103 |
| 10.4.1. Hente verdier fra ELVIS.....     | 103 |
| 10.4.2. Validere verdier mot ELVIS ..... | 104 |
| 10.4.3. Problematikk ved ELVIS .....     | 104 |
| 11. Backend .....                        | 106 |
| 11.1. Controller.....                    | 106 |
| 11.2. MeldingManager .....               | 110 |
| 11.3. FileManager .....                  | 114 |
| 11.4. Validering.....                    | 116 |

|  |            |
|--|------------|
| 11.5.    Testing .....                   | 119        |
| 11.5.1.    Enhetstesting .....           | 119        |
| 11.5.2.    Andre testmetoder .....       | 120        |
| 11.6.    CI/CD.....                      | 122        |
| <br><b>Konklusjon .....</b>              | <b>127</b> |
| <br><b>Refleksjon .....</b>              | <b>128</b> |
| <br><b>Videre anbefalinger.....</b>      | <b>130</b> |
| <br><b>Kilder .....</b>                  | <b>132</b> |
| <br>Vedlegg A – Prosjektskisse .....     | 138        |
| Vedlegg B – Prosjektkontrakt .....       | 140        |
| Vedlegg C – Forprosjecktrapport .....    | 144        |
| Vedlegg D – Fremdriftsplaner.....        | 155        |
| Vedlegg E – Kanban tavler .....          | 157        |
| Vedlegg F – Beskrivelser av Sprint ..... | 159        |
| Vedlegg G – Brukertest 1 - Skjema .....  | 172        |
| Vedlegg H – Brukertest 1 Svar .....      | 178        |
| Vedlegg I – PP Brukertest/demo.....      | 179        |
| Vedlegg J – Brukertest 2 - Skjema.....   | 184        |
| Vedlegg K – Brukertest 2 - Svar .....    | 186        |
| Vedlegg L – Endelig design .....         | 187        |

# Oppgavens bakgrunn og mål

Dette kapittelet presenterer aktørene som har påvirket oppgaven, målet med oppgaven og hva som utløste behovet for et nytt kreftmeldingssystem hos oppdragsgiveren Kreftregisteret.

Videre vil vi også presentere hvilke rammer vi hadde å forholde oss til og kravene som ble satt til systemet.

## 1. Aktører

En aktør er en person, en gruppe eller institusjon som spiller en aktiv rolle på et bestemt område (Det Norske Akademi for Språk og Litteratur, u.å.). I denne prosjektoppgaven er systemet vi skal utvikle det bestemte området. Aktørene har påvirkningskraft på systemet og utfallet av hvordan den vil bli utviklet som kan påvirke det endelige resultatet. Ved å presentere aktørene i denne delen, vil man få et visst innblikk i aktørenes formål med systemet og hvordan det kan ha påvirket resultatet.

### 1.1. Prosjektgruppen

Vi er en gruppe på fem dataingeniør-studenter fra OsloMet – storbyuniversitet. Vi møttes i starten av studiet i faget Webutvikling og inkluderende design og samarbeidet om vår første prosjektoppgave her. Samarbeidet fungerte godt og vi har siden samarbeidet i flere andre fag og prosjekter. Gruppen består av følgende medlemmer:

**Jørund Topp Løvljen:** Jørund har alltid likt å drive på med PC-er og teknologi. Da han hadde IT på videregående skole, så syntes han dette var det som var mest interessant å drive med. Han liker også å lære og jobbe med matematikk og naturvitenskap, og hvordan verden er bygd opp. Det han likte best innenfor IT-fag var å drive på med algoritmer og har derfor valgt å ta en master innenfor dette til høsten.

**Tom Henrik M. Basmo:** Tom har en bachelorgrad i musikkteknologi fra NTNU, og fikk en smakebit på programvareutvikling i studietiden der. Startet å studere ved dataingeniørlinjen fordi han ønsket å forstå mer om hvordan datamaskiner fungerer. Det han syns er mest interessant med programvare er hvordan store komplekse systemer henger sammen, og hvordan man kan forenkle eller effektivisere slike systemer. Etter bacheloroppgaven skal Tom Henrik jobbe som konsulent.

**Hajin Barzingi:** Hajin er utdannet adjunkt i realfag med tilleggsutdanning i blant annet norsk. Da hun startet prosessen med å forberede seg på nye læreplaner, fant hun fort ut at hun ønsket mer dybdekunnskap i programmering og hvordan systemer blir utviklet. Den pedagogiske kompetansen har hun fått god bruk for i arbeid med å lage brukervennlige programmer og

nettsider. Etter bachelorgraden i dataingeniør skal hun bruke kompetansen fra begge utdanningene i rollen som rådgiver innen cybersikkerhet.

**Nikola Dordevic:** Gjennom valgfaget Informasjonsteknologi på videregående fikk Nikola interesse for IT-systemer og spesielt det som ligger bak det brukerne interagerer med. I løpet av studiet fikk han også mer interesse for algoritmer og datastrukturer og hvordan de kan bidra til systemets effektivitet. Etter bachelorstudiet skal han forstype seg mer i algoritmer og datamaskiner ved å ta masterprogrammet Datatekologi ved NTNU.

**Ola Gynnild Berg:** Ola har alltid vært interessert i teknologi, og var interessert i å utforske programmering med ingeniørfaglig praksis. Han finner programutvikling mest interessant, og trives godt med å forstype seg i små og store problemer. Til sommeren skal han jobbe som utvikler i bank, og studere videre ved NTNU.

## 1.2. Oppdragsgiver

Krefregisteret er et av de eldste nasjonale kreftregistrene i verden og har siden 1951 samlet inn data om krefttilfeller. Siden 2002, da Krefregisterforskriften tredde i kraft, har leger ved alle helseinstitusjoner vært pålagte å melde inn krefttilfeller, også når det kun gjelder mistanke om kreft. Dette har ført til at materialet Krefregisteret besitter har en signifikant verdi når det gjelder å etablere ny kunnskap gjennom forskning og å spre kunnskap om kreft for å fremme og utvikle kvaliteten på forebyggende tiltak og helsehjelp som tilbys. Krefregisteret er organisert som selvstendig institusjon under Oslo universitetssykehus HF, med eget styre og 165 årsverk som er organisert i tre fagavdelinger og en stabs/støtteavdeling (Krefregisteret, 2020b). Registerinformatikkavdelingen er den avdelingen som har gitt oss oppdraget med å utvikle et nytt meldingssystem.

## 1.3. Veiledere

### 1.3.1. Kreftregisteret

Jan Franz Nygård

Leder ved Registerinformatikkavdelingen

E-post: [jfn@krefregisteret.no](mailto:jfn@krefregisteret.no)

Andre veiledere:

- Andreas Sørstrøm  
Veileder innhold i skjema/frontend
- Thomas Schwitalla  
Veileder backend
- Gintaras Pikelis  
Veileder Kreftregisterets metadatabase

### 1.3.2. OsloMet – Storbyuniversitet

George Anthony Giannoumis

Førsteamanuensis ved Fakultetet for teknologi, kunst og design

E-post: [george.a.giannoumis@oslomet.no](mailto:george.a.giannoumis@oslomet.no)

Andre veiledere:

- Hedvig Bjørge  
Skriveveileder for bachelorgrupper med engelskspråklig veileder.

## 2. Dagens situasjon

I 2002 trådde Kreftregisterforskriften i kraft, som fastsatte formelle krav både til helseinstitusjonene og til Kreftregisteret. Kravene innebærer blant annet at alle som behandler kreftpasienter er pliktige til å rapportere til Kreftregisteret innen to måneder etter at opplysninger om kreftsykdom er dokumentert, og at helseinstitusjoner skal ha systemer som sikrer at kreftmeldinger blir sendt til Kreftregisteret innen fristen (Kreftregisteret, 2020b, "Forskriftene" seksjon). En kreftmelding, som Kreftregisteret mottar som XML-fil fra sykehus rundt omkring i Norge, består av informasjon som gjelder sykdomsforløpet til en pasient. Det benyttes forskjellige skjemaer avhengig av hvilken fase av behandlingen pasienter befinner seg i, for eksempel utredning av primærsykdom, behandling, etc. (Oslo universitetssykehus HF, u.å.).

I dag bruker Kreftregisteret systemet KREMT – *Kreftregisterets Elektroniske Meldingstjeneste*. KREMT er en webbasert innrapporteringsløsning, der autoriserte brukere har blant annet muligheten til å registrere nye meldinger, se administrativ og klinisk statistikk og se kvitteringer for innsendte meldinger. De fleste helseinstitusjoner i dag bruker KREMT for innrapportering av klinisk informasjon til Kreftregisteret (Kreftregisteret, 2020a, "KREMT" seksjon). KREMT er laget av Kreftregisteret gjennom InfoPath Form Services som er Microsoft sin løsning for å lage skjemaer (Microsoft, u.å.).

## RADIKAL PROSTATEKTOMI Meldes etter avsluttet kirurgisk behandling

### Pasient/behandlingsinstitusjon

Fødselsnummer\*

  Ikke norsk personnummer

Navn\*

Sykehus\*

Avdeling\*?

### Preoperativ informasjon

Har pasienten vært aktivt monitørt?\*

 Ja  Nei

Er det gjort ny vurdering av sykdomsutbredelse (restaging) etter primær diagnose?\*

 Ja  Nei

### Behandling

PSA før prostatektomi og eventuell neoadjuvant endokrin behandling\*

  Ukjent

Er det utført neoadjuvant endokrin behandling?\*

 Ja  Nei  Ukjent

### Radikal prostatektomi

Operasjonsdato (dd.rrr.mmm.åååå)\*

Kirurgi\*

Nervesparende intensjon\*

Er det foretatt lymfadenektomi samtidig?\*

Figur 1: Et utsnitt av skjemaet som brukes til å innhente informasjon om Prostatakreft etter kirurgisk behandling. Et utfylt skjema kalles en kreftmelding.

### 3. Problemstilling

I helsevesenet sendes pasientinformasjon mellom sykehus og fra sykehus til Kreftregisteret i form av XML meldinger i et eget helsenettverk (Norsk Helsenett). Kreftregisteret mottar flere hundretusen slike meldinger hvert år. Ettersom at meldingene inneholder sensitiv informasjon om pasienter, er det eksterne krav om at disse er riktig kodet i henhold til medisinsk terminologi og kodeverk (Direktoratet for e-helse, 2020). Derfor gjennomgås meldingene av medisinske kodere ved Kreftregisteret og informasjon kan måtte rettes eller kodes om.

I dag brukes InfoPath til å lese og vise meldingene fra XML skjemaene. InfoPath er End-of-life, som symboliserer slutten i livssyklusen for et produkt. Det innebærer at produktet ikke lenger blir videreutviklet eller vedlikeholdt. Med tiden vil dette lede til en rekke problemer for systemer som benytter seg av disse produktene. Blant annet vil systemet stå i fare for å møte på programvarefeil, inkompatibilitet med systemarkitektur og dårlig ytelse (Spiceworks, 2016). I tillegg er InfoPath et kommersielt produkt og har blitt utviklet som et selvstendig system med Microsoft sine programmeringsspråk. Dette har ført til at InfoPath er lite fleksibelt og kommuniserer dårlig med fagsystemene til Kreftregisteret som er Java-baserte.

Kreftregisteret har derfor valgt «rip-and-replace»-strategien for å håndtere InfoPath. Rip-and-replace går ut på å lage noe som kan erstatte det eksisterende Legacy-systemet slik at det kan bli utvekslet med minimal dødtid. Problemstillingen blir dermed: *Hvordan lage en robust og skalerbar løsning som er web- og API-basert, og som kan kommunisere med resten av fagsystemene til Kreftregisteret slik at det gir minimal dødtid?*

## 4. Mål og rammebetingelser

Dette delkapittelet gir en forklaring av kravspesifikasjonen og målet for prosjektet vårt. En kravspesifikasjon er en beskrivelse av en løsning med formål om å definere avgrensninger, funksjonalitet og nytteverdi, slik at forståelsen av hva som skal bli utviklet for alle involverte parter, samt brukernes behov blir dekket (Jansen, 2018; Rolstadås & Liseter, 2018).

Kravspesifikasjonen er dermed viktig i arbeidet med å nå målet med prosjektet.

Kravene er utledet i dialog med oppdragsgiver hvor vi har vært interesserte i smertepunktene, det vil si prøve å grundig forstå forretningsproblemet som trenger å løses. I løpet av de første møtene satt vi opp følgende overordnede krav:

- Verdier skal kunne redigeres og så valideres
- Løsningen skal kunne kommunisere med resten av Kreftregisteret sine fagsystemer
- Løsningen skal bare bruke Open-Source teknologier
- Løsningen skal kunne være skalerbar for flere krefttyper
- Ønskelig at man skal kunne gå raskt gjennom skjemaet
- Ønskelig at utviklingen er modulbasert, mulighet for å gjenbruke deler av systemet.

Ettersom vi hadde valgt en smidig metode å arbeide på, formulerte vi nye og flere krav etter hvert som vi utviklet og ble bedre kjent med både skjemaene, systemet og hvordan de medisinske koderne jobbet. Kravene kan deles inn i flere grupper: Funksjonelle, ikke funksjonelle, organisatoriske og eksterne krav. De funksjonelle kravene beskriver hvordan systemet skal oppføre seg og hva den ikke skal gjøre, mens de ikke-funksjonelle sier noe om kvalitetsønsker ved systemet som for eksempel responstid. De organisatoriske kravene handler om kostnader, programmeringsspråk, verktøy osv., mens de eksterne omhandler sikkerhet og lovverk knyttet til systemet (Jansen, 2018). Selv om det eksterne kravet ikke var en del av vår oppgave, mener vi det er riktig å ha det i mente.

## 4.1. Funksjonelle krav

Systemet må:

- tillate redigering av innsendte meldinger.
- validere skjemaer før lagring.
- kunne laste inn data i riktig skjema.
- kunne validere data.
- kunne lagre nye skjemaer med ny id.
- kunne lagre redigerte skjemaer med ny id.
- tillate at brukere kan se gjennom en melding og avslutter uten endringer.
- gi brukere enkel tilgang til knapper som «lagre» og «avslutt».
- vise brukere hvor i skjemaet det er ugyldig data dersom lagring feiler på grunn av det.
- gi brukere en forklaring på hvorfor lagring feiler.
- gi brukere tilbakemelding på om lagring er vellykket.
- vise brukere hvilken type skjema man redigerer/lager.
- kunne håndtere de tre forskjellige meldingstypene innen krettypen prostata.
- kunne hente data fra og validere data mot Krefregisteret metadatabase.

Skjemaene må:

- være enkle å navigere for brukere.
- være mulig å navigere med kun tab.

## 4.2. Ikke-funksjonelle krav

Systemet skal:

- være skalerbart for flere krettyper.
- ha snarveier.
- være modulbasert med mulighet for gjenbruk av kode.
- behandle data innen en minimal responstid.

- ha godt dokumentert kode som gjør det mulig at en utvikler vil kunne forstå, videreutvikle og vedlikeholde koden uten problemer.
- være lett å vedlikeholde.
- data som er pålitelige og tilgjengelige når det er nødvendig.
- kunne håndtere opp mot 25 brukere samtidig.
- kunne behandle opp mot 200 meldinger i løpet av en arbeidsdag.

Skjemaene skal:

- være oversiktlige.
- være enkle å redigere.
- være raske å fylle ut.
- være brukervennlige for medisinske kodere og andre som jobber med kreftmeldinger.
- se moderne ut.
- kunne håndtere opptil 30 brukere samtidig.

#### 4.3. Organisatoriske krav

Systemet skal:

- være open-source.
- kun bruke open-source teknologier.
- kunne kommunisere med resten av Kreftregisterets fagsystemer.
- bruke Java som backend-språk.
- bruke SurveyJS-biblioteket (hvis mulig) og React som frontend-teknologier.

#### 4.4. Eksterne krav

- Systemet skal fylle Normens minimumskrav til informasjonssikkerhet for å sikre konfidensialitet, integritet, tilgjengelighet og robusthet (Direktoratet for e-helse, 2020).

# Prosessdokumentasjon

Dannelse av bachelorgruppe og planlegging av prosjekt startet allerede høsten 2021. Flere i gruppen jobbet deltid eller hadde annen personlig relasjon til aktuelle oppdragsgivere, som førte til at vi fikk flere tilbud. Vi måtte derfor gjennom en utvelgelsesprosess evaluere våre ønsker opp mot prosjekttilbudene. Prosjektet vi valgte å gå for var fra Kreftregisteret. Vi valgte å gå for dette prosjektet fordi det hadde en definert problemstilling som fylte våre ønsker når det gjaldt utvikling av programvare og design, i tillegg til at det virket utfordrende og var noe Kreftregisteret virkelig ville ha nytte av. At prosjektet var innen helseteknologi-feltet, et ukjent felt for samtlige gruppemedlemmer, vekket også vår interesse.

I dette kapittelet vil vi beskrive hvordan vi gikk frem for å løse problemstillingen vi fikk fra Kreftregisteret. Vi vil først presentere metodene, verktøyene og bakgrunnslitteraturen vi brukte, før vi forklarer hvordan disse ble brukt.

## 5. Metoder og verktøy

Etter dialog med oppdragsgiver der vi fikk forståelse over hva oppgaven gikk ut på, kom spørsmålet om hvordan prosjektledelsen skulle utføres. Det ble kjapt åpenbart at smidig metodikk egnet seg bra for denne gruppen og prosjektet vi hadde.

### 5.1. Smidig Metodikk

Smidig metodikk er en populær utviklingsmetode for IT-prosjekter. Metoden kom som en respons på den lite brukersentrerte, dokumentasjonstunge og plan-drevne metoden på 90-tallet som passet bedre for større grupper. Der plandrevne metodikk ofte må ha fullført forhåndsdefinerte steg før man går til neste trinn, kan man i smidig metodikk jobbe parallelt med de forskjellige aspektene ved programvareutvikling. Dersom et prosjekt anvender smidig metodikk kan man for eksempel gå tilbake og endre på kravene ettersom man blir bedre kjent med brukergruppen og/eller produktet. I tillegg foregår planlegging av stegene i ukentlige statusmøter, hvor det diskutes blant annet hva som er gjort, hva som skal gjøres og utfordringer (Sommerville, 2011, s. 56–74).

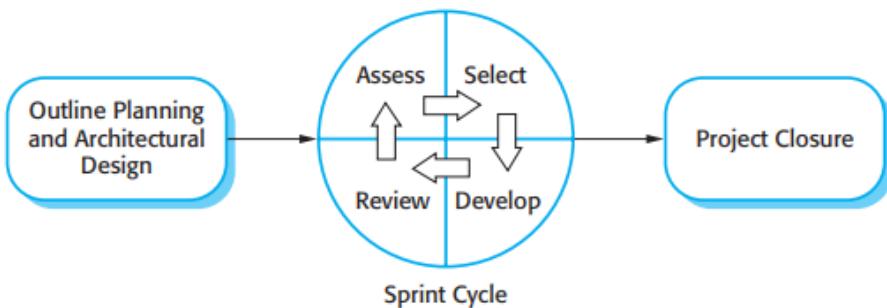
Tett oppfølging av oppdragsgiver er også en sentral del i smidig metodikk. Dette gjøres gjennom en produkteier som kan være oppdragsgiver selv eller en annen som skal representere oppdragsgiver sine interesser. Gjennom en produkteier som er til stede under hele prosessen er det forsikret at produktet møter kravspesifikasjonene. Produkteier kan gi tilbakemeldinger, og komme med eventuelle endringer. Dette åpner for at endringer kan gjøres underveis i utviklingsprosessen (Sommerville, 2011, s. 58).

Valget falt på å utvikle prosjektet ved hjelp av smidig metodikk til tross for kravet om dokumentasjon, ettersom at denne metodikken er best tilpasset mindre grupper og systemer - samt tillater den at det skjer endringer underveis og er derfor mer brukersentrert.

Oppdragsgiver, vår veileder ved Kreftregisteret, ble dermed vår produkteier og deltok på statusmøter en gang i uken. I tillegg til statusmøtene, brukte vi elementer fra både rammeverkene Scrum, Extreme Programming og Kanban for å administrere prosjektet og arbeidsoppgavene. Videre beskrives metodene og verktøyene nærmere.

### 5.1.1. Scrum

Scrum er et populært rammeverk for prosjektledelse innen smidig metodikk og har flere fordeler. Et typisk Scrum prosjekt er inkrementell, som betyr at man starter med den viktigste funksjonen til produktet først, også bygger videre på denne. Ved å levere litt hele tiden, får man raskere tilbakemeldinger og kan justere produktet til å fylle kravene bedre (Sommerville, 2011, s. 72).



Figur 2: I Scrum er prosjektene delt inn i tre faser: planleggingsfasen, utviklingsfasen og avslutningsfasen.

Utviklingsfasen er iterativ. Illustrasjon hentet fra Sommerville (2011, s. 73)

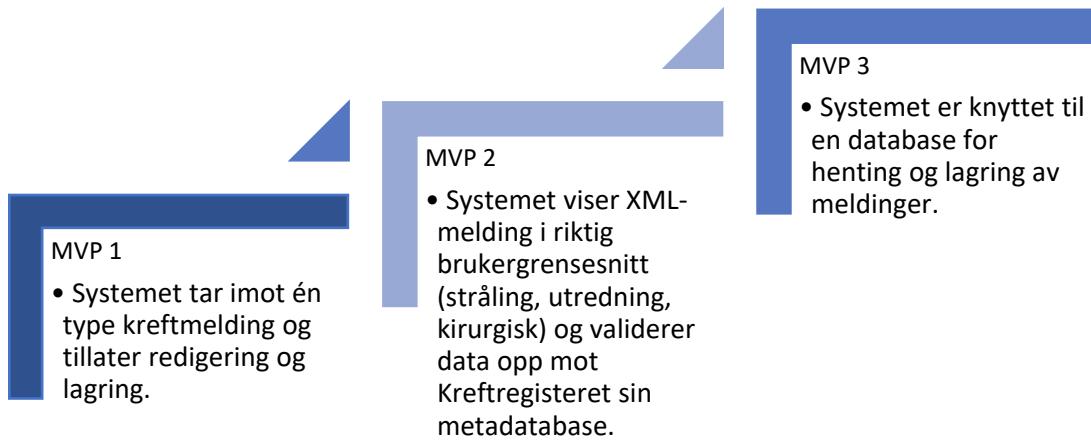
Byggingen skjer etter en planleggingsfase der blant annet produkt-backlog blir definert, altså oppgaver som må gjøres. Planleggingen blir gjort med hele teamet i samarbeid med produkteier og gir dermed oppdragsgiver mulighet for å introdusere nye kravspesifikasjoner. Etter denne fasen kommer de såkalte «sprintene» og byggingen som kan vare fra en til flere uker. I sprintprosessen blir det holdt korte daglige møter kalt stand-up, der framgang blir presentert av hvert medlem, eventuelle problemer som har oppstått i løpet av utviklingen og hva neste steg er. Det er typisk å planlegge for neste sprint i løpet av en sprint og representere oppgavene i form av lapper på en tavle for å opprettholde oversikt for alle medlemmene. På slutten av hver sprint blir de nye funksjonene presentert for produkteier og andre interesser. Når alle krav er utviklet og produktet er ferdig etter flere sprinter, kommer man til avslutningsfasen. Her utarbeides systemdokumentasjon og ferdig produkt leveres til oppdragsgiver (Sommerville, 2011, s. 72-73). Figur 2 illustrerer fasene i et Scrum-prosjekt. I kapittel 6, 7 og 8 beskrives vår prosess innen hver fase nærmere.

### 5.1.2. Scrum master

Et Scrum-prosjekt ledes ofte av en Scrum master. En Scrum master har som ansvar å passe på teamet i de ulike fasene ved å forsikre at møtene blir utført som de skal, verne medlemmer fra eksterne distraksjoner og overse at prosjektet er utført på den mest effektive måten (Sommerville, 2011, s. 74). I vårt tilfelle hadde ingen av oss nok kunnskap om rollen og så derfor ikke nødvendigheten av å ha en Scrum master. Vi ser i ettertid at en gruppe på vår størrelse burde hatt en tydeligere leder som kunne ta en større planleggingsrolle, slik at de andre ble frigjort til å utvikle og fokusere på produktet. Vi var dog heldige som jobbet veldig tett med oppdragsgiver, som ved enkelte tidspunkt tok rollen som Scrum master.

### 5.1.3. Minimum viable product

Som nevnt er et Scrum-prosjekt inkrementell som vil si at man starter med å levere de viktigste funksjonene først også bygger videre på dem. De viktigste funksjonene er ofte kjent som Minimum Viable Product (MVP) eller Enkleste Brukbare Produkt. MVP-er er viktige for å validere en produktidé tidlig i utviklingen. Innen programvareutvikling brukes MVP-er også til å få tilbakemeldinger fra brukere så raskt som mulig for å utvikle produktet i henhold til brukernes behov og ønsker (ProductPlan, 2021).



Figur 3: Et trapediagram med "Minimum Viable Product"-tekstene våre.

I prosessen med å lage MVP-er diskuterte vi våre oppfatninger av hvordan vårt sluttprodukt burde se ut og hvilke delprodukter det innebar. For å definere MVP-ene hadde vi tett dialog med oppdragsgiver og en av de medisinske koderne viste oss hvordan dagens løsning fungerte. I løpet av prosjekttiden utviklet vi til og med MVP 2, og så at å bruke tid på å utarbeide gode MVP-er var viktig. Vi skjønte også etter hvert at det ikke er nødvendig å utarbeide flere MVP-er med en gang når man jobber smidig. Viktigst av alt er å få tak i kjernen av produktet i starten, slik at man raskt kan teste ut funksjonalitet på brukerne og starte dialogen med de om den videre utviklingen.

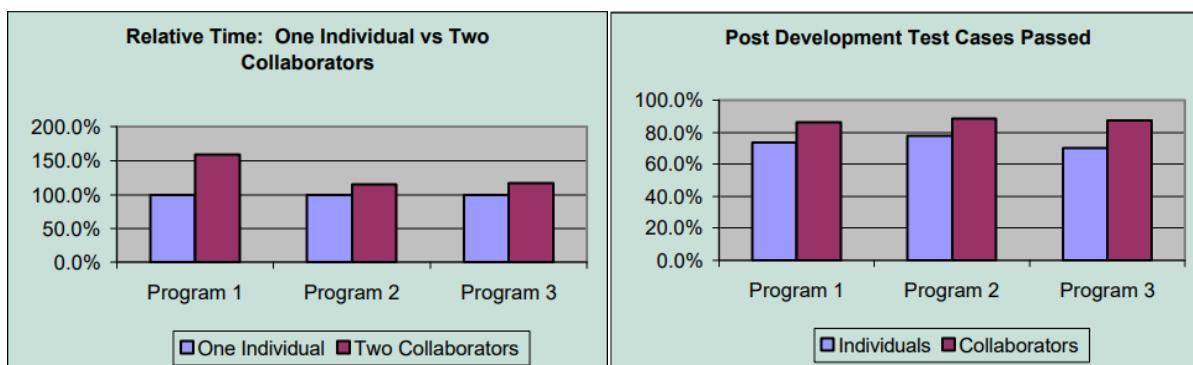
## 5.2. Parprogrammering

Parprogrammering er en aktivitet først introdusert i Extreme Programming (XP). Aktiviteten i seg selv er ganske enkel: To eller flere utviklere jobber sammen om samme bit av kode. De største fordelene med dette konseptet ifølge Sommerville (2011, s. 71-72) er:

- Utviklere kan komme med innspill underveis og diskutere designvalg som er det beste for systemet. Utviklere kan ut ifra dette lære av hverandre og bygge en sterkere teamfølelse.
- I og med at minst to kodere ser på enhver bit av kode sammen, vil dette lede til en uformell inspeksjon av koden, som effektivt kan redusere antall programvarebugs.

- Et grunnleggende konsept i XP er kollektivt eierskap. Det går ut på at alle utviklere føler de kan redigere alle deler av kodebasen, i tillegg til å ha ansvar for helheten. Dette kan kreve kunnskap om systemet, noe parprogrammering danner ved at flere er en del av utviklingsprosessen.

Man kunne tro at parprogrammering var ineffektivt, men i en undersøkelse gjort av Alistair Cockburn og Laurie Williams (2000) i *"The Costs and Benefits of Pair Programming"* viser det seg at utviklere som jobbet i par brukte 15% mer tid enn koderne som var alene, men i gjengjeld hadde koden også 15% mindre programvarebugs. Programvarebugs vil ifølge Cockburn og Williams koste selskapet mer å fikse enn den reduserte tiden det hadde tatt å utvikle systemet på. I tillegg fant de ut at kvaliteten av koden konsekvent var bedre og med færre linjer kode enn hos de som arbeidet alene.



Figur 5: Figuren til venstre viser tiden brukt for å utvikle et system i par og individuelt, mens figuren til høyre viser antall tester bestått i systemene som indikerer kvalitet. Kilde: Cockburn & Williams (2000).

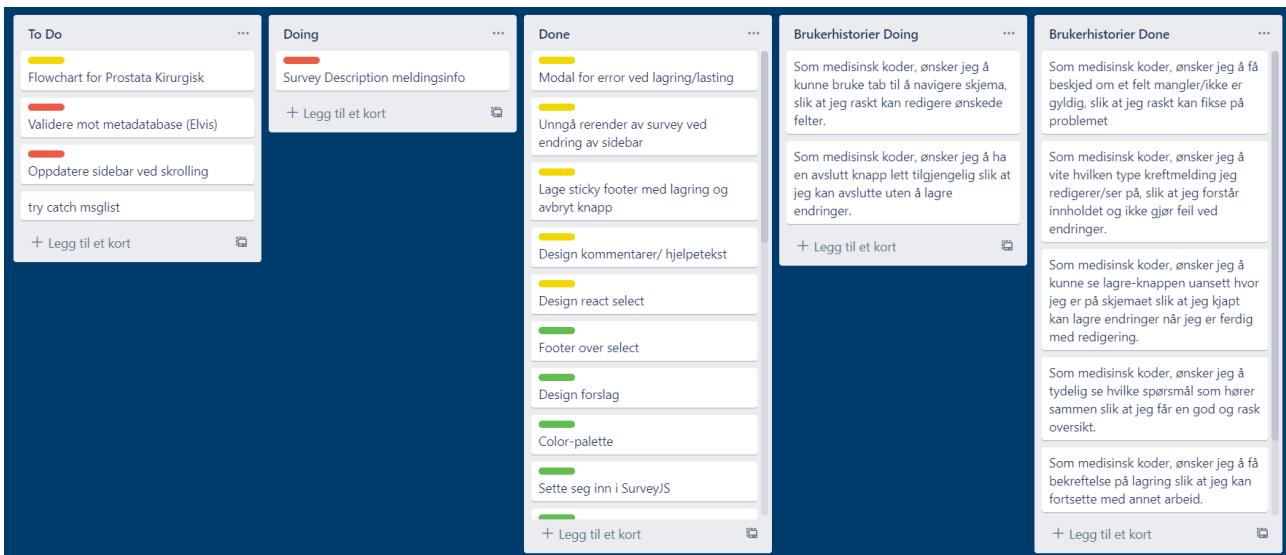
Prosjektet hadde tydelig krav om en backend-del som tok hånd om logikken rundt skjemaene og frontend-del for brukergrensesnittet. Vi bestemte oss for at delene kunne bli utviklet parallelt som i tillegg ville forhindret forvirring med mål og oppgaver. Dette førte til at to fikk hovedansvaret for frontend-delen og tre fikk for backend-delen. Dette åpnet opp muligheten for å utvikle i par og mindre grupper. Grunnet pandemien har gruppen ikke hatt muligheten til å møtes fysisk. Derfor har vi brukt Microsoft Teams som en substitusjon for et fysisk arbeidssted. Teams tilbyr evnen til å “dele skjermen” med hverandre, en funksjon som etterligner måten par ville ha arbeidet sammen. Vi opplevde at å programmere i par var mer effektivt enn å sitte

alene, spesielt når vi møtte på utfordringer. Sammen kunne vi både lære av hverandre i tillegg til å oppdage mulige løsninger og bugs raskere.

### 5.3. Prosjekttavle

Ofte kan det være vanskelig å oppfatte hvilket stadium prosjektet man arbeider på er i. Hvilke oppgaver som gjenstår og hvilke som er fullført er i utgangspunktet usynlige for gruppemedlemmene. Derfor finnes det metoder som Kanban-tavle som prøver å løse disse problemene. Tavlen har som hovedoppgave å gi et overordnet blikk over prosjektet i hele perioden den blir utviklet. Det skal være arbeidsflyt i tavlen slik at det er tydelig for utviklerne hva som må gjøres, hva som blir gjort og hva som er ferdig (Rehkopf, u.å.).

I prosjektet vårt valgte vi å bruke Trello.com, en nettapplikasjon for å sette opp virtuelle oppslagstavler. Ved å bruke en slik tavle kunne vi sette prioritet på oppgaver, dele ut ansvar og enkelt få oversikt over hvilke oppgaver som krevde oppmerksomhet. Figur 6 viser Kanban-tavlen for frontend-teamet. For bedre bilde og eksempel fra backend, se vedlegg E – Kanban-tavler.



Figur 6: Vi brukte Trello for å visualisere oppgaver og prioritet. Her også med brukerhistorier.

## 5.4. Brukerhistorier

Å beskrive funksjonelle krav med brukerhistorier er en metodikk innen smidige utviklingsmetoder. En brukerhistorie oppsummerer enkelt hvem en funksjon er viktig for, hva funksjonen gjør og hvorfor funksjonen er viktig. Denne metoden sikret at brukerens behov ble ivaretatt, ga grunnlag for hva som skulle prioriteres, og ga gruppen en enda bedre forståelse av hva som skulle utvikles. Hver brukerhistorie fikk også et akseptkriterium. Et akseptkriterium beskriver hva som må oppfylles for at en brukerhistorie er ferdig utviklet og er dermed god hjelp i testing av funksjonene (EnTur, u.å.).

| Brukerhistorie   | Akseptkriteriet   |
|--|---|
| Som medisinsk koder, ønsker jeg å få beskjed om et felt mangler/ikke er gyldig, slik at jeg raskt kan fiksere på problemet.                                      | Gitt at jeg har redigert en kreftmelding og ønsker å lagre, så skal jeg automatisk sendes til det feltet hvor det er feil data eller data mangler når jeg trykker på lagre-knappen.   |
| Som medisinsk koder, ønsker jeg å vite hvilken type kreftmelding jeg redigerer/ser på, slik at jeg forstår innholdet og ikke gjør feil ved endringer.            | Gitt at jeg har åpnet en kreftmelding for redigering, så skal jeg kunne se en overskrift som beskriver kreftmeldingstypen øverst i skjemaet.  |
| Som medisinsk koder, ønsker jeg å kunne se lagre-knappen uansett hvor jeg er på skjemaet slik at jeg kjapt kan lagre endringer når jeg er ferdig med redigering. | Gitt at jeg har åpnet en kreftmelding og redigert kun innhold på øvre del av skjemaet, så skal jeg, dersom jeg kun skulle redigere det øverste feltet, kunne trykke på en lagre-knapp som er lett tilgjengelig uten å skrolle.                |
| Som medisinsk koder, ønsker jeg å tydelig se hvilke spørsmål som hører sammen slik at jeg får en god og rask oversikt.   | Gitt at jeg har åpnet en kreftmelding, så skal jeg se underoverskrifter for hver del og hver del skal ha tydelige rammer som skiller de fra andre bugst deler.  |
| Som medisinsk koder, ønsker jeg å få bekreftelse på lagring slik at jeg kan fortsette med annet arbeid.  | Gitt at jeg har åpnet en kreftmelding, gjort endringer i den som jeg vil lagre og trykket på lagre-knappen, så skal jeg få en pop-up som gir meg bekreftelse på at jeg fylt ut riktig og at en ny kreftmelding er lagret med endringene mine. |

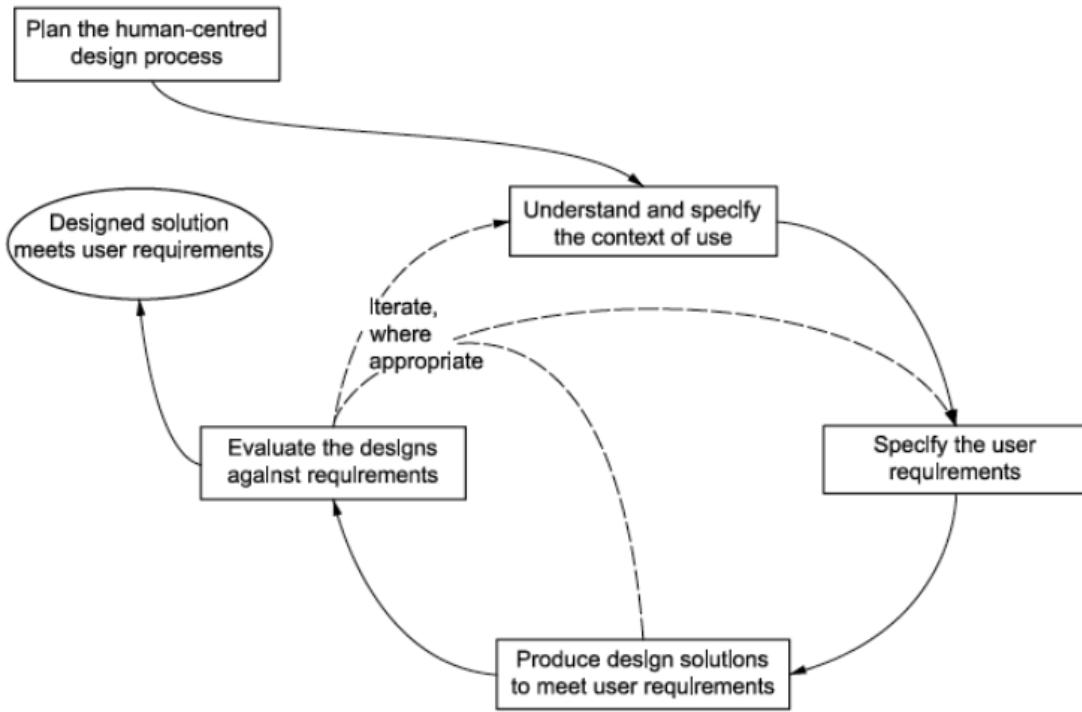
|  |   |
|--|---|
| Som medisinsk koder, ønsker jeg å få beskjed dersom lagring ikke er vellykket og årsaken til det, slik at jeg kan rette på eventuelle feil eller løse problemet. | Gitt at jeg har åpnet en kreftmelding, gjort endringer i den som jeg vil lagre og trykket på lagre-knappen, så skal jeg få en pop-up som gir meg bekreftelse på om lagring er vellykket. Dersom lagring ikke er vellykket, skal jeg få beskjed på årsak og kunne gå tilbake til skjemaet og rette på eventuelle feil. |
|--|---|

## 5.5. Bakgrunnslitteratur

### 5.5.1. Brukersentrert design og teori

Ifølge ISO 9241-210:2010(E) er brukersentrert design en tilnærming til interaktive systemer med mål å produsere de brukbare, trygge og funksjonelle. Dette gjøres ved å fokusere på brukerne av systemet, deres behov og krav, samt de menneskelige faktorene og brukervennlighetskunnskap. Med det store fokuset på brukerne og deres krav og behov, vil løsningen bli mer funksjonell, tilgjengelig og ivareta brukernes helse (National Institute of Standards and Technology, 2021).

En bruikersentrert designprosess går godt sammen med Scrum, da den også har en planleggingsdel, iterativ utviklingsdel og avslutningsdel. Etter at en plan for hvordan designprosessen skal foregå er lagt, går man inn i utviklingsprosessen. Her starter man med å forstå konteksten systemet skal brukes i og utarbeide personas. Så går man over til å spesifisere krav og behov fra bruker før man lager et eller flere designforslag basert på kravene. Designforslagene blir så evaluert mot kravene gjennom blant annet brukertesting. I siste prosess kan ulike ting avdekkes, som for eksempel kan det dukke opp nye krav og man kan gå tilbake til andre faser av prosessen (National Institute of Standards and Technology, 2021). Se illustrasjon i figur 7.



Figur 7: Illustrasjon av hvordan fasene i en brukersentrert designprosess kan henge sammen i tillegg til prosessens fleksibilitet.

Da design ble diskutert med produkteier var det viktigste at skjemaene var kompakte slik at brukerne slapp å skrolle så mye, og at skjemaene fikk farger som var behagelige å jobbe med. For å øke brukervennligheten av skjemaene, bestemte vi oss for å bruke en brukersentrert tilnærming i designutviklingen. Metoden passet godt med prosjektets helhetlige smidige prosessmetode også.

#### 5.5.1.1. Forstå og spesifisere brukskonteksten

Gjennom samtaler med en bruker og produkteier, fikk vi samlet inn mer informasjon om brukerne og konteksten. Produkteier tilbød oss å observere brukerne for å forstå dem bedre når koronasituasjonen hadde dempet seg. Koronasituasjonen varte dessverre hele informasjonsinnsamlingsperioden og dermed måtte vi heller ta flere samtaler digitalt. Gjennom samtalene fikk vi nyttig informasjon om konteksten og brukerne. Informasjonen ble brukt til å blant annet lage personas. Personas er en fiktiv person som representerer en eller flere brukere. Gjennom en persona kan man trekke beslutninger bort fra ens egne opplevelser og bruke den fiktive personens personlighet, opplevelser, atferd og mål som målestokk (Travis, u.å.).

Brukergruppen har tittelen «medisinske kodere» og består av mellom 25-30 personer hvorav 75% er kvinner. Aldersspennet strekker seg fra 20-70 år, så det er en ganske heterogen gruppe med forskjellig bakgrunn, kunnskap og arbeidsoppgaver. Hver person har sin krefttype som de fokuserer på og spesialiserer seg innenfor. I tillegg til arbeidet med kreftmeldinger har flere av de medisinske koderne arbeidsoppgaver innen analyse og organisering. Alle koderne har to 27 tommer skjermer på kontorene sine som er koblet mot en dokkingstasjon. Noen ganger kan en bruker jobbe med ti kreftmeldinger samtidig. Det er derfor ønskelig at hele skjemaet er på en side og kompakt, med minst mulig behov for å skrolle.

#### 5.5.1.2. Spesifiser krav/behov

I den første kravlisten, var det kun to krav som omhandlet brukervennlighet: «verdier skal kunne redigeres og så valideres» og «ønskelig at man skal kunne gå raskt gjennom skjemaet». Ettersom vi fikk mer innsikt i brukergruppen og konteksten skjemaene skulle bli brukt i, ble det lagt til flere krav som omhandlet brukervennlighet og design.

Funksjonelle krav:

Systemet må:

- tillate redigering av innsendte meldinger.
- tillate at brukere kan se gjennom en melding og avslutter uten endringer.
- gi brukere enkel tilgang til knapper som «lagre» og «avslutt».
- vise brukere hvor i skjemaet det er ugyldig data dersom lagring feiler på grunn av det.
- gi brukere en forklaring på hvorfor lagring feiler.
- gi brukere tilbakemelding på om lagring er vellykket.
- vise brukere hvilken type skjema man redigerer/lager.

Skjemaene må:

- være enkle å navigere for brukere.
- være mulig å navigere med kun tab.

Ikke-funksjonelle krav:

Skjemaene skal:

- være oversiktlige.
- være enkle å redigere.
- være raske å fylle ut.
- være brukervennlige for medisinske kodere og andre som jobber med kreftmeldinger.
- se moderne ut.

Da vi begynte å se på mulige designforslag, la vi raskt merke til at vi hadde behov for enda mer konkretisering av kravene, fra brukerens perspektiv. Vi stilte oss selv spørsmål som «var funksjonene viktige for personas-ene Andam og Siri?», «hva gjorde funksjonene for dem?» og «hvorfor var funksjonene viktige for dem?». Dette førte til at vi utarbeidet brukerhistorier som ga oss et klarere bilde av de funksjonelle kravene med brukeren i sentrum. Brukerhistoriene tydeliggjorde også hvilke funksjoner som burde prioriteres fra brukerens perspektiv.

Brukerhistoriene kan leses i delkapittel 5.4.

#### 5.5.1.3. Designforslag

Før vi utarbeidet designforslag, analyserte vi nåværende design opp mot gode designprinsipper for skjemaer for å fordype oss i relevant teori. Dette ga oss et bedre innblikk i hvordan skjemaene blir fylt ut og brukt av de medisinske koderne, og hvilke eventuelle utfordringer de kunne oppleve. Det hendte at vi la til flere krav ettersom vi oppdaget mangler gjennom analysen.

**UTREDNING**

Meldes etter avsluttet utredning

**Pasient/behandlingsinstitusjon**

Fødselsnummer

2312313

 Ikke norsk personnummer

Navn

dsdasda

Sykehus

Annet

Avdeling?

Onkologisk avdeling

Spesifiser

dsdasdas

**Funn i utredningen**

- Primærtumor (med eller uten metastase(r))
- Kun metastase(r)

*Utredning av primærtumor og samtidig utredning av regionale lymfeknutemetastaser/fjernmetastaser før igangsetting av primærbehandling*

*Utredning av regionale lymfeknutemetastaser og/eller fjernmetastaser uten samtidig utredning av primærtumor*

**Sykehistorie**

PSA-verdi ved diagnostidspunkt  
 Ikke     Ukjent  
 tatt

Var vannlatingsproblemer en del av årsak til utredningen?  
 Ja     Nei     Ukjent

Var kreftsymptomer (smerte, anemi eller annet) en del av årsak til utredningen?  
 Ja     Nei     Ukjent

Funksjonstilstand/WHO-status ved diagnose  
 4: Totalt stillesittende eller i seng hele daç

Funksjonstilstand/WHO-status ved diagnose\*

Velg...

Prostatavolum\*  
 ml     Ukjent

**Diagnostikk av primærtumor**

Er det gjort MR prostata?\*  
 Ja     Nei

Er det gjort annen bildediagnostikk av primærtumor?\*  
 Ja     Nei

Celle-/vevsprøver\*  
 Ja     Nei

Dato sykdommen ble bekreftet/diagnosedato (dd.mm.åååå)\* ?  
 dd/mm/yy

**Patologilaboratorium\***

Laboratorium  
 Velg...     Ikke  
 relevant

**Sykdomsutbredelse etter ferdig primærutredning og før primærbehandling**

Palpatorisk T-stadium **høyre** side (laveste stadium velges ved tvil)\*  
 Velg...

Palpatorisk T-stadium **venstre** side (laveste stadium velges ved tvil)\*  
 Velg...

Samlet palpatorisk T-stadium

Totalvurdering av klinisk T (laveste stadium velges ved tvil)\*

Figur 8: Utsnitt av skjema Prostata Utredning. Dagens skjemadesign er grå og kan oppfattes som kjedelig.

#### 5.5.1.3.1. Struktur

Brukere kan ofte bli demotiverte av lange skjemaer og ofte kan disse skjemaene forkortes dersom man kun spør om absolutt nødvendig informasjon. Det er også i god stil med GDPR-reglene å kun innhente absolutt nødvendig informasjon (European Union, 2021). Men rekkefølgen du spør i er også viktig. Det du spør om bør komme i en logisk rekkefølge for å minske unødvendige misforståelser fra brukerens side. En metode som er vanlig å bruke sammen med logisk rekkefølge er Gestalts nærhetsprinsipp. Gestalts nærhetsprinsipp sier at elementer som er ordnet nær hverandre vil bli oppfattet som mer beslektede enn de som er plassert lenger fra hverandre (Dahl, 2020). Gruppering av relaterte felt bidrar til at brukeren får bedre forståelse av informasjonen som må fylles ut. Sammen med gruppering, kan man bruke «En kolonne»-prinsippet for å øke effektivitet. Prinsippet sier at skjemaer med en kolonne er enklere å skanne gjennom enn to-kolonner. Dette fordi brukeren skanner to-kolonne-skjemaer i et Z-mønster, som fører til redusert hastighet på forståelse (Nick Babich, 2020). En illustrasjon av dette vises i Figur 9.



Figur 9: Illustrasjon av hvordan en bruker leser to-kolonne skjema og en-kolonne skjema.

Som en kan se fra utsnittet fra dagens skjema i figur 8, er det noen mangler med tanke på struktur. Først og fremst er det en blanding mellom en-kolonne og to-kolonne løsninger. Dette er en hindring for brukere som skal fylle skjemaene effektivt og raskt. Undersøkelser viser at man bruker mer tid på å få oversikt over to-kolonne-skjemaer enn en-kolonne-skjemaer. Videre starter skjemaet med å spørre om enkel informasjon før det blir komplisert. Om det er kun det

absolutt nødvendige som spørres om, kan vi ikke svare på siden vi ikke har helsefaglig bakgrunn. Skjemaet virker ellers oversiktlig da det er delt inn i ulike deler. Delene er tydelig markerte med felles overskrift og nærhet.

#### 5.5.1.3.2. Inputfelter

Inputfelter er viktige elementer for informasjonsinnsamling. Gjennom eksempelvis avkrysningsbokser, tekstbokser og nedtrekkmenyer får man samlet inn informasjon fra brukere. For å sikre at man får ønsket informasjon, bør skjemaets inputfelter som tekstbokser gjenspeile forventet informasjon. Inputfelter som nedtrekkmenyer, bør være mulige å søke gjennom dersom de kan inneholde mange verdier. Videre bør en tenke på hvordan man skal vise obligatoriske felter. Konvensjonen er å bruke en asterisk (\*) på de obligatoriske feltene sammen med en beskrivelse av hva tegnet betyr tydelig på siden (Nick Babich, 2020).

Dagens løsninger har en blanding av ulike typer inputfelter. Inputfeltene har en bredde som gjenspeiler forventet input og er markerte med en asterisk dersom de er obligatoriske å fylle ut. Det er brukt en del nedtrekkmenyer for faste verdier. Nedtrekkmenylene er ikke søkbare og kan føre til at det blir litt skrolling og leting.

#### 5.5.1.3.3. Ledetekster

Brukere trenger å bli veiledet i utfylling av skjemaer. Ledetekster gir brukerne den veiledningen de trenger. En god ledetekst forteller brukeren formålet med feltet, opprettholder nytten når fokuset er på selve inputfeltet, og forblir synlig selv etter at inputfeltet er fylt ut (Nick Babich, 2020). Gode ledetekster er også korte og konsise med «sentence case», altså stor forbokstav i setningen. Plasserer man ledetekstene på toppen av inputfeltene de tilhører, vil det hjelpe brukeren å skanne gjennom skjemaet raskere ifølge undersøkelser gjort med Eye-Tracking verktøy av Mattheo Penzo (Nick Babich, 2020; Penzo, 2006).

Skjemaene har gjennom bruk av «sentence case» og gode ledetekster på toppen av inputfeltene gjort det lettere

å skanne gjennom. Dette sammen med Gestalts nærhetsprinsipp har ført til at skjemaet kan oppleves noe oversiktlig og enklere å skanne gjennom til tross for bruk av to-kolonner.

#### 5.5.1.3.4. Handlingsknapper

Knapper er en metode for brukeren å samhandle med systemet. Gjennom et trykk kan brukeren enten forkaste alt eller sende inn. Å gjøre det tydelig og enkelt for brukeren å trykke den ønskede knappen er viktig for å unngå unødvendig feil. Å vise distinksjon mellom primær- og sekundærknapper kan gjøres ved bruk farger og form. Primærknappene, altså de man ønsker at brukeren skal trykke på, bør være fremhevte. Fremhevelsen bør også gjøres i form av plassering. Primærknapper bør være plassert under inputfeltet og være neste på fokus-listen etter siste inputfelt. Sekundærknapper derimot bør plasseres på plasser som gjør dem mindre tilgjengelige. Til sist bør knapper ha beskrivende navn som forteller brukeren hvilken handling som vil skje ved trykk (Nick Babich, 2020).



Figur 10: Nederst i skjemaene kan man finne tre handlingsknapper: Avbryt registrering, midlertidig lagring og send inn skjema.

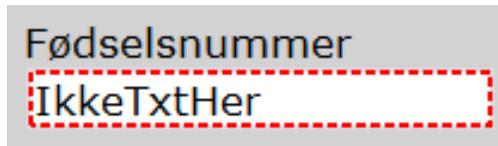
For å komme til handlingsknappene i dagens løsning, må en bruker skrolle helt ned i skjemaet. Som en kan se fra dagens løsning, kan et skjema bli ganske langt og dermed kan skrolling ta mye tid. Distinksjon mellom primær- og sekundærknapper er viktig for at en bruker ikke skal trykke feil, og her har det blitt gjort ved å skape avstand mellom knappene. Visuelt er knappene like med noe ulik bredde. Videre er knappene godt beskrevet, men deler av teksten forsvinner på grunn av overlapping med andre elementer eller for smal bredde i forhold til tekstlengde.

#### 5.5.1.3.5. Validering og respons

Å gi brukeren respons på deres handlinger er viktig for å fremheve at utfylling av et skjema er en dialog mellom systemet og brukeren. Gjennom tekst, bilde, farger og animasjon kan man gi tilbakemeldinger på forskjellige ting. For eksempel bør inputfeltene validere input og gi tilbakemeldinger på feil og fortelle hva som feilet. Dersom skjemaet er langt, kan tilbakemeldinger på hvor i skjemaet brukeren har feil input både motivere og hjelpe brukeren med å fullføre skjemaet. Videre bør brukeren få tilbakemeldinger på om innsending/lagring ble vellykket eller ikke. Dersom det ikke ble vellykket, bør brukeren få beskjed om hvorfor (Nick

Babich, 2020). Validering er også viktig for å sikre dataenes integritet som er et eksternt krav fra Direktoratet for e-helse da disse skjemaene inneholder data av sensitiv karakter. Med integritet menes at dataene skal registreres på rett person, føres i henhold til relevant kodeverk og terminologi, i tillegg til at de er korrekte og om nødvendig oppdaterte. Man skal også hindre at kopier av data blir en kilde til utdatert informasjon (Direktoratet for e-helse, 2020).

I dagens løsning får brukere tydelig beskjed ved feil input med engang. Og når brukeren sender inn et riktig utfylt skjema, blir brukeren sendt til en ny side med beskjeden om at skjema er



Figur 11: Dersom input er feil, blir inputfeltet markert med stiplete rød kant.

sendt inn. Men dersom brukeren har glemt å fylle inn et obligatorisk felt før hen trykker på «Send inn skjema»-knappen, blir brukeren sendt til en ny side med beskjed om at alle felter markert med rød stjerne må fylles ut. Dette er uheldig da brukeren må gjennom flere handlinger og lete etter hvor et felt er glemt.



Figur 12: Brukere som har glemt å fylle ut obligatoriske felt blir sendt til ny side med en feilmelding.

#### 5.5.1.3.6. Assistanse

Komplekse skjemaer kan ofte trenge mer enn bare ledetekster for å assistere brukeren i utfylling. HjelpeTekster blir ofte brukt for å supplere ledetekster. Disse ligger ofte «bak» små «i»-ikoner ved siden av eller under ledetekstene. I hjelpeTekstene er det lurt å bruke klart språk for å sikre brukerens forståelse. Dette betyr at hvert ord skal være den korteste, mest enkle versjonen som er tilgjengelig (Nick Babich, 2020; Tank, 2022).

|  |   |
|--|---|
| <b>PI-RADS høyre</b> side (laveste stadium velges ved tvil) <b>PI-RADS venstre</b> side (laveste stadium velges ved tvil)  |   |
| <span style="border: 1px solid black; padding: 2px;">*</span> <span style="border: 1px solid black; padding: 2px;">?</span><br>Velg... <span style="font-size: small;">▼</span>  | <span style="border: 1px solid black; padding: 2px;">*</span> <span style="border: 1px solid black; padding: 2px;">?</span><br>Velg... <span style="font-size: small;">▼</span> |
| <small>* PI-RADS 5 = Høy sannsynlighet for klinisk signifikant malign tumor som er 15 mm eller større, eller som har ekstraprostatisk vekst.</small>   |   |
| <small>* PI-RADS 4 = Høy sannsynlighet for klinisk signifikant malign tumor som er mindre enn 15 mm og uten suspekt ekstraprostatisk vekst.</small>  |   |
| <small>* PI-RADS 3 = Intermediær sannsynlighet for klinisk signifikant malign tumor, usikker lesjon, 50/50%.</small>   |   |
| <small>* PI-RADS 2 = Lav sannsynlighet for klinisk signifikant malign tumor, benigne funn.</small>   |   |
| <small>* PI-RADS 1 = Svært lav sannsynlighet for klinisk signifikant malign tumor, normale funn.</small>   |   |
| <i>Klinisk signifikant tumor er definert som histologisk Gleason score 7 eller høyere, og/eller volum større enn 0,5 ccm, og/eller ekstraprostatisk vekst.</i>   |   |
| Totalvurdering av klinisk T (laveste stadium velges ved tvil)<br><i>TNM på diagnostidspunktet baseres på all relevant utredning som er gjort før primærbehandling inkludert klinisk undersøkelse, bildediagnostikk, endoskopi, biopsi og kirurgisk eksplorasjon.</i>                     |   |
| T3a – Ekstrakapsuler vekst (unilateral eller bilateral) <span style="float: right;">▼</span>   |   |
| Hvem har gjort totalvurderingen av klinisk T?*<br>Velg... <span style="font-size: small;">▼</span>   |   |
| <small>Er regionale lymfeknutemetastaser påvist (N-sykdom)?<br/> <i>Grensen mellom regionale lymfeknutemetastaser og fjernmetastaser går ved delingsstedet til arteria iliaca communis. Ved tvil om korrekt N-kategori skal den laveste (minst avanserte) kategorien velges.</i></small> |   |
| <input type="radio"/> NX = Ikke undersøkt<br><input checked="" type="radio"/> NO = Ingen regionale lymfeknutemetastaser<br><input type="radio"/> N1 = Regionale lymfeknutemetastaser   |   |

Figur 13: Skjemaet har mye assistanse i form av hjelpeTekster som har fått fokus med en blå farge.

Ettersom at innholdet i skjemaene er av helsefaglig karakter så kan ikke vi si noe om det er tydelige hjelpeTekster eller ei. Det vi kan se av dagens skjema er at hjelpeTekster får mye fokus ved at de er skrevet i en blå farge. Dette kan for en medisinsk koder som har jobbet med samme skjema i mange år og som forstår alle uttrykkene oppleves plagsomt og rotete. Det er gjort forsøk på å rydde opp ved å legge noen hjelpeTekster bak «?»-ikonet. HjelpeTekstene kommer ofte under ledetekstene og er tydelige for hvilke felt de gjelder.

#### 5.5.1.4. Evaluere design

Vi hadde statusmøter med produkteier og en bruker hver uke hvor eventuelle endringer i design fra forrige uke ble presentert. Vi fikk dermed raske tilbakemeldinger på hva som funket og ikke. I tillegg til statusmøtene hadde vi to brukertester hvor vi fikk tilbakemeldinger fra flere brukere, samt brukte vi akseptkriteriene til brukerhistoriene og sammenlignet designet mot.

## 5.5.2. Backend – teori

Målet med en webapplikasjon som kan ha lang levetid, er blant annet å lage et system som har *lav kobling, høy kohesjon*, er modulær og enkel å teste. Flere av valgene man tar tidlig i utviklingsprosessen kan hjelpe mot dette målet. I dette kapitelet vil vi forklare teorien bak metoder, designmønstre og teknikker som er relevante for dette prosjektet.

### 5.5.2.1. Designmønstre

Et designmønster er en generell, gjenbrukbar løsning til et gitt problem innen programvareutvikling. Det kan sees på som en beskrivelse på hvordan man kan løse et problem som kan brukes i mange sammenhenger. Designmønstre er en måte for en utvikler å strukturert tilnærme seg programmering, som kan gjøre utviklingsprosessen mer effektiv, ved å anvende gode løsninger som allerede er testet og utprøvde (Gamma et al., 1995).

#### 5.5.2.1.1. Model-View-Controller

MVC (Model-View-Controller) er et designmønster som berører det arkitektoniske aspektet ved en kodebase. Controlleren har ansvaret for å behandle http-forespørslar, og i praksis gjør metodekall indirekte på modellen, som å endre og opprette instanser av modellen.

Metodekallene controlleren gjør på modellen, skjer ofte gjennom en serviceklasse som har direkte tilgang til en database. Modellen har ansvaret for hvordan dataen selv skal behandles, og kan sees på som en representasjon av hvordan et objekt i den virkelige verden kan representeres, lagres og endres på (Davis, 2008). View-delen av arkitekturen ligger i en webapplikasjon ofte på klient-siden, der klienten henter data som er definert i en modellklasse, via en metode i controlleren som samsvarer med en av http-operasjonene.

#### 5.5.2.1.2. Dependency injection

Dependency injection er et designmønster som gjør det mulig for et objekt å motta andre objekter den er avhengig av, og på denne måten skape et skille mellom hvordan objektene er konstruert, og hvordan de tas i bruk. Grunnen til at vi ønsker å sette avhengigheter på denne måter er blant annet på grunn av «The Dependency Inversion Principle», som sier at høy-nivå moduler ikke burde være avhengige av lav-nivå moduler. Hvis lav-nivå avhengigheter er

«hardkodet» inn i en høy-nivå serviceklasse, kan det gjøre det vanskelig å teste enkeltdeler av applikasjonen samtidig som vedlikehold og videreutvikling kan bli komplisert (Baeldung, 2022).

#### 5.5.2.1.3. REST

REST er et arkitektonisk designmønster for å utvikle API-er. Rest API står for «representational state transfer application programming interface» og er kode som gjør det mulig å utveksle data mellom to forskjellige systemer. Et REST API kan være basert på HTTP metodene for å få tilgang til, opprette eller endre ressurser. Informasjon kan i et RESTAPI sees på som en ressurs, og all data som kan navngis i et system er en ressurs, for eksempel et dokument, en samling dokumenter eller en data som representerer et menneske (bruker, student eller lignende) (Fielding & Reschke, 2014).

#### 5.5.2.1.4. Data transfer object

DTO er et designmønster som er brukt for å innkapsle data. DTO står for *Data transfer object*, og kan minne om en datastruktur i den forstand at det er et objekt som er komponert for å samle data som klienten trenger, for å unngå at klienten må gjøre flere forskjellige kall for å få den dataen som trengs. DTO kan frakoble modellklassen fra det som vises i view-laget av applikasjonen. På denne måten kan man bestemme hvilke data som skal presenteres, uten å måtte endre på databasekall eller selve domenemodellen (Baeldung, 2022).

### 5.5.2.2. Programmeringsprinsipper og teknikker

Kontinuerlig integrasjon (CI) og kontinuerlig levering (CD) er en del av en kultur og et sett med teknikker som gjør det mulig å levere ny funksjonalitet og endringer i kode oftere. Ved hjelp av verktøy som for eksempel GitHub Actions kan vi bygge, teste og distribuere koden idet en endring har blitt gjort i kodebasen. Dette betyr at utviklere ikke manuelt trenger å sjekke at integreringen av nye funksjoner fungerer, men at arbeidsflyten automatisk tester, bygger og distribuerer koden (Sacolick, 2022).

Høy kohesjon er et mål på hvilket ansvar et objekt har, og hvor begrenset eller spesifikt objektets område er. Dette betyr i praksis at metoder som deler klasse bør støtte samme mål

som andre funksjoner i klassen. Funksjoner i en felles klasse eksisterer der for å løse et delproblem som er en del av det hovedproblemet klassen er utviklet for å løse (Sommerville, 2011, s 674).

Lav kobling mellom objekter er et mål på hvor sterkt avhengig et objekt er på et annet objekt. Dersom et objekt er sterkt avhengig av et annet objekt kan det bety at å endre en klasse vil påvirke andre områder av programvaren, og er derfor ønskelig å unngå (Sommerville, 2011, s 674).

### 5.5.2.3. Testing

En av de mest effektive metodene å sørge for kvalitet i systemet er gjennom testing. Testing kan deles hovedsakelig opp i to former: Testing opp mot det oppdragsgiverens kravspesifikasjon og brukerhistorier, og testing som sjekker for programvarebugs.

Testingen gjort i backend fall hovedsakelig under Development testing. Development testing er all testingen som foregår aktivt under utviklingsfasen for et system. Debugging og development testing brukes ofte om hverandre i dette tilfellet. Under denne prosessen prøver utviklere å finne programvarebugs i systemet. Under utviklingen av meldingssystemet brukte backend black-box metoder for å teste etter bugs. Black-box er en måte å teste på der man ikke trenger kjennskap til den interne koden og resultatet av testen sjekkes kun mot input og forventet output. Hovedsakelig ble det utnyttet to former for black-box testing – Enhetstesting og systemtesting (Sommerville, 2011, s 210).

#### 5.5.2.3.1. Enhetstesting

Enhetstesting er en metode der små deler av systemet blir testet individuelt, oftest klasser eller metoder. Prosessen inneholder oftest en oppstartsfas, der klasser eller variabler blir initialisert. Deretter kalles metodene og til slutt testes en påstand som skal bestemme om testen er bestått. Enhetstester er oftest automatiserte, for å forsikre at oppdateringer i koden ikke tilfører nye programvarebugs (Sommerville, 2011, s. 212-213).

#### 5.5.2.3.2. Systemtesting

Systemtesting er det naturlige steget etter enhetstesting, etter at komponentene har bestått testen individuelt, settes de sammen og sjekkes helhetlig opp mot systemet. Målet med systemtesting er å finne uforventede interaksjoner mellom de ulike komponentene når alt kjøres samtidig. Systemtesting skjer mot slutten av en sprint når alle oppgavene har blitt gjort (Sommerville, 2011, s. 219).

#### 5.5.2.3.3. Mock-objekter

Output i de fleste av testene i backend har vært produkter av mock objekter. Mock-objekter er objekter som deler samme interface med ordentlige objekter. Siden objektene deler samme interface, kan simulering av ordentlige kjøringer finne programvarebugs ut ifra interaksjoner med objektene. I tillegg vil det ta tid å hente ordentlige objekter fra en database, derfor brukes mock-objekter til å eliminere tiden det tar å hente ekte data (Sommerville, 2011, s. 213).

## 6. Planleggingsfasen

Gjennom utarbeidelse av prosjektskisse, kontrakt og forprosjektrapport fikk vi en oversikt over nødvendige ressurser, kostnader og planer knyttet til prosjektet (Se vedlegg A - Prosjektskisse, B - Prosjektkontrakt og C - Forprosjektrapport). Det ble tidlig bestemt at all teknologi involvert i prosjektet skulle være gratis og Open-Source, da Kreftregisteret hadde et prinsipp på dette.

Videre så vi på ressurser som kompetanse, maskinvare og programvare som var nødvendig for oppgaven og så at programvarene hadde gratisversjoner eller ble dekket gjennom OsloMet sine eksisterende programvare-avtaler. Maskinvarer stod vi studentene for selv. Når det gjelder kompetanse måtte vi bruke tid på å lære oss hva gode prinsipper for skjemadesign er, designmønstre for robusthet og hvordan de ulike teknologiene fungerer og bør brukes sammen. Kompetansen vi så på som nødvendig for å utføre denne oppgaven har vi skrevet om i blant annet kapittel 5.4. Brukersentrert design og 5.5. Backend – teori.

### 6.1. Teknologier

I dette delkapittelet vil vi kort beskrive de programmeringsspråkene, rammeverkene, bibliotekene og programvarene som ble brukt i prosjektet.

#### 6.1.1. Frontend

##### 6.1.1.1. SurveyJs og React

Kreftregisteret hadde tidligere undersøkt forskjellige rammeverk og biblioteker for utvikling av skjemaer. *SurveyJs* var blant disse, men de hadde ikke ressurser til å undersøke det nærmere. Vi ble derfor bedt om å undersøke dette biblioteket. SurveyJS er et webutviklingsbibliotek for undersøkelser skrevet i JavaScript. Biblioteket støtter flere frontend rammeverk som Angular 2, jQuery og React.js (Devsoft Baltic OÜ, u.å.). SurveyJS sine hovedfunksjoner er at den har mange spørsmålstyper, støtter flere sider, kan endre innholdet og logikken i skjemaet dynamisk, flerspråklig og Bootstrap støtte. Etter å ha sjekket ut andre alternativer som Forms og Formik, valgte vi å fortsette med SurveyJS fordi det viste seg å være enkelt å skrive lefftattelig kode sammen med React, og at biblioteket gjør det enkelt å utvikle betingete visninger av inputfelt.

#### 6.1.1.2. Node.js

Node.js er et «runtime-system» som gjør det mulig å kjøre JavaScript-programmer på tjenersiden. Hvis man ønsker å bruke Open-Source Node.js-biblioteker, kan man bruke NPM (Node package manager), som er en database som inneholder javascriptbibliotek. NPM har også et kommandolinjeverktøy som kan brukes for å behandle avhengigheter og bibliotek på en enkel måte.

#### 6.1.1.3. Reactstrap og Axios

Vi brukte Reactstrap og Axios som tredjeparts-biblioteker. Dette er biblioteker som inneholder ferdige løsninger og logikk for å gjøre utviklingsprosessen enklere. Reactstrap, som er en variant av Bootstrap for React, er et komponentbibliotek som inneholder forhåndsbygde grensesnitt-komponenter som utvikleren kan ta i bruk og endre. Axios er et JavaScript-bibliotek brukt for å lage http-forespørsler.

#### 6.1.1.4. Figma

Figma, er en nettbasert løsning for prototyping av brukergrensesnitt og ble brukt til å skissere fargepalett-forslag, personas og designforslag. Figma gjør det mulig å brukertest ideer uten å implementere det i kode.

### 6.1.2. Backend

#### 6.1.2.1. Java

*Java* ble brukt som programmeringsspråk for backend. Dette fordi både Kreftregisteret og prosjektgruppen hadde erfaring med språket og fordi de ytterlige systemene Kreftregisteret opererer med er hovedsakelig skrevet i Java. Det gjør at en eventuell integrasjon med deres systemer vil være gjennomførbar. Vi har brukt den siste LTS-versjonen av Java, da den kan være gunstig for fremtiden. Blant annet vil nyere versjoner ha bedre sikkerhet, ytelse, ha fjernet avhengigheter eller andre pakker i grunnbiblioteket med «end-of-support» og være enklere å

oppdatere, spesielt for store systemer (Krill, 2021). For testing valgte vi testrammeverket *JUnit*, som gjør det mulig å enkelt skrive tester for mindre deler av applikasjonen.

### 6.1.2.2. Spring Boot

For å utvikle webapplikasjoner er det vanlig å bruke rammeverk med pakketerte løsninger på vanlige problemer. I planleggingsfasen var vi litt usikre på hvilket rammeverk som ville passe best for vår situasjon, da det finnes flere rammeverk for å utvikle webapplikasjoner. Ettersom Kreftregisteret har kompetanse på rammeverket *Spring Boot*, og to av gruppemedlemmene skulle ta et fag som omhandler rammeverket, ble det et enkelt valg. Ingen av gruppemedlemmene hadde arbeidet med dette før, men ettersom alle i gruppen er kjent med Java og har utviklet webapplikasjoner i andre programmeringsspråk, viste det seg å være en fin læringsprosess. Der vi møtte på problemer eller ikke skjønte hvordan elementer i Spring Boot fungerte, googlet vi informasjon eller så etter svar i dokumentasjon.

### 6.1.2.3. JAXB

Kreftregisteret mottar meldinger i form av XML-filer og det var derfor behov for en måte å konvertere XML-filene til datastrukturer som programmet vårt kunne anvende. Vi kunne ha sendt XML-filene direkte til klienten, men da flyttet vi problemet med å konvertere filene til brukbar data over på klienten. *JAXB* (Java Architecture for XML Binding) er et bibliotek som gjør det mulig å lage POJO-er, også kjent som Java-klasser, ut ifra et XML-skjema (XSD fil) (JAXB Users Guide, 2009). For å holde styr på avhengigheter og biblioteker i Java, ble det brukt *Maven*, som er et verktøy for å beskrive hvordan systemet skal bygges sammen med dets avhengigheter.

## 6.1.3. Prosjektstyringsverktøy

I planleggingsfasen undersøkte vi ulike problemsporing og tidsstyringsverktøy som Jira, Github Issues og Trello. Vi spurte Kreftregisteret om de ønsket at vi skulle bruke noen spesifikke prosjektstyringsverktøy, men tiden som ville gått med på å lære seg et verktøy som Jira, var ikke verdt det mente de. Vi brukte derfor Trello da vi hadde litt erfaring med verktøyet. Trello gjør det mulig å organisere prosjekter og sette oppgaver som «lapper» på tavler med egendefinerte områder. Vi lagde en Kanban-tavle med områdene «To Do», «Doing» og «Done», som forklarer

status på mål og oppgaver. På denne måten fikk prosjektet en åpenhet, der status på oppgavene ble transparente.

IntelliJ IDEA er et kraftig utviklingsmiljø. Alle i prosjektgruppen hadde erfaring med dette utviklingsmiljøet siden OsloMet har lisens på det. GitHub ble også brukt for å samarbeide på kodebasen. GitHub gjør det mulig å holde oversikt over versjonshistorikken for prosjektet og har tjenester som forenkler arbeidet gjennom kommandolinjeverktøyet Git. GitHub har også flere funksjoner som forenkler distribueringsprosessen, blant annet GitHub Actions, som gjør det mulig å automatisere en respons på oppdatert kodebase, f.eks. tester.

#### 6.1.4. Andre verktøy

| Utredning                        |               |                |          |            |                                   |                     |              |            |             |
|----------------------------------|---------------|----------------|----------|------------|-----------------------------------|---------------------|--------------|------------|-------------|
| Meldes etter avsluttet utredning |               |                |          |            |                                   |                     |              |            |             |
| Feltavn                          | Visningsverdi | Lagrings-verdi | Datatype | Kontroller | Betinget formattering (vis/skjul) | Regler/valideringer | Obligatorisk | XML Schema | Hjelpetekst |

Figur 14: Utsnitt av oppsettet til Excel-dokumentet

Kravspesifikasjonene fra Kreftregisteret fikk vi i Excel-dokumenter. Dokumentene er delt inn i kolonnene felt- og lagringsnavn, visnings- og lagringsverdi, datatype, kontroller, betingelser om visning, validering, om feltet er obligatorisk og hjelpe tekst, og hver rad inneholder informasjon om et felt i skjemaet. Radene er videre gruppert inn i sider. I figur 14 kan du bl.a. se en kolonne som heter "XML Schema". Denne kolonnen viser til hvor i XML-skjemaet verdien til et felt blir lagret, altså lagringsnavnet til feltet.

## 6.2. Fremdriftsplan

Som en del av forprosjektrapporten, utarbeidet vi et utkast til en fremdriftsplan. Vi valgte å gå for Gantt-diagram, da disse diagrammene oppleves som intuitive, er enkle å forstå og gir en rask oversikt over prosjektets tidsplan, aktiviteter og milepæler. Gantt-diagrammer er også enkle å vedlikeholde og endre, noe vi forventet at vi måtte gjøre ettersom at vi valgte en smidig prosjektmetodikk (Wålberg, 2020). Figur 15 viser vårt førsteutkast på en fremdriftsplan. Den ble skrevet på engelsk for at vår engelsktalende veileder skulle forstå. Videre planer ble utarbeidet på norsk da han kan lese og forstå norsk.

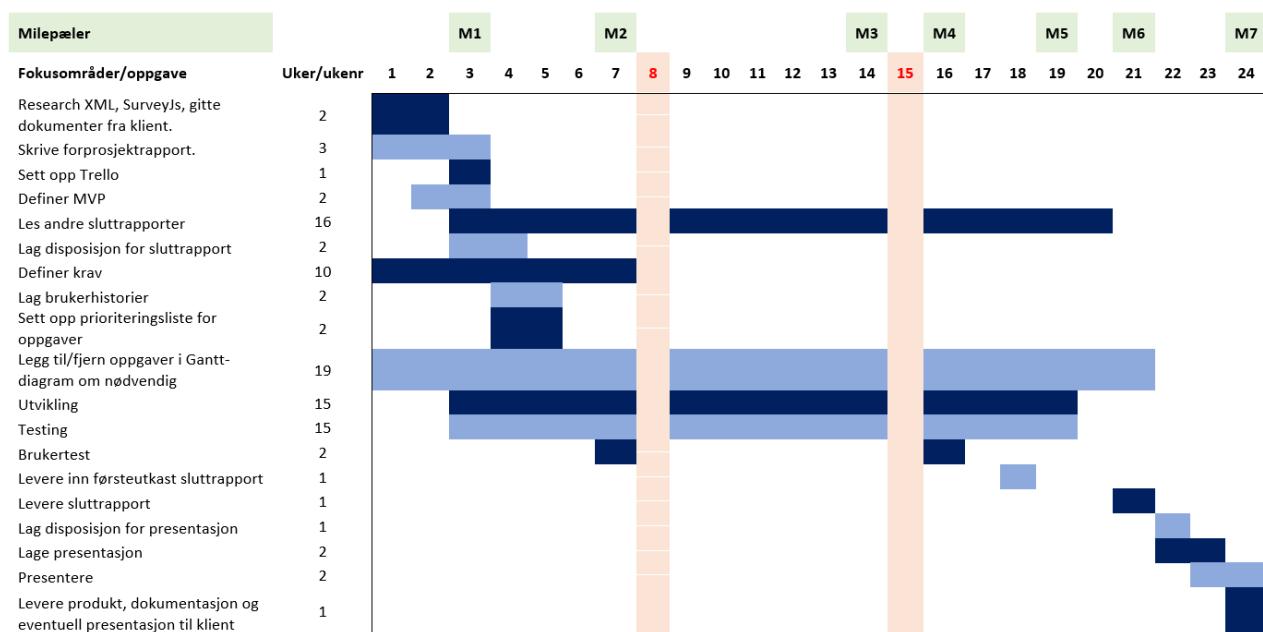
| Development schedule - Rough |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 9 | 10 | 11 | 12 | 13 | 14 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
|------------------------------|--|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Week                         | Main focus   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 1-3                          | Research XML, SurveyJS, given documents from client. Write "forprosjektrapport". Workshop with client. Set up a YouTrack/Trello for Scrum-method. Define MVP for project.        |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 4-7                          | Outline documentation (which will be written as we go along). Create use cases based on the MVPs and appoint prioritization. Appoint responsibilities. Set up detailed schedule. |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 8                            | Winter break   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 9-14                         | Improve and add to product through iterative process. Usertest product.  |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 15                           | Easter break   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 16-20                        | Improve and add to product through iterative process. Final touch-up documentation and finish project report.  |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 21                           | Hand in final product  |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 22                           | Prepare presentation   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 23                           | Presentation of product  |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |

Figur 15: Første fremdriftsplan

Som en kan se var denne tidsplanen noe uoversiktig da hver aktivitet inneholdt flere aktiviteter og lite som viste hvordan oppgavene burde prioriteres. På tross av dette ga planen oss en rask oversikt over hovedelementene i prosjektet som var nødvendig for å vite hvor og hvordan vi skulle starte.

## 6.3. Milepæl

En milepæl er en planlagt, målbar hendelse, en form for delmål som må oppnås for å kunne nå prosjektets mål (Rolstadås, 2019). Vi så for oss at milepælene skulle være basert på MVP (Minimum Viable Product). For å lage MVP-er, måtte vi gjennom et par andre aktiviteter og vi ventet derfor med å sette milepælene inn i førsteutkastet av Gantt-diagrammet. Da MVP-ene var klare var det naturlig for oss at de som ble betraktet som oppnåelige innenfor tiden, også ble våre delmål/milepæler for produktet. Figur 16 viser et bildeutsnitt av sisteutkastet av Gantt-diagrammet med milepælene over den uken måloppnåelsen skulle være nådd (se vedlegg D – Fremdriftsplaner for større bilde).



Figur 16: Fremdriftsplan med milepæler.

Beskrivelse av milepæler:

- Milepæl 1: Utarbeide MVP
- Milepæl 2: MVP 1 – Systemet kan behandle én type kreftmelding og tillater redigering og lagring.
- Milepæl 3: MVP 2 – Systemet viser XML-melding i riktig brukergrensesnitt (stråling, utredning, kirurgisk) og validerer data opp mot Kreftregisteret sin metadatabase.

- Milepål 4: Implementere design
- Milepål 5: Førsteutkast sluttrapport sendt til skriveveileder.
- Milepål 6: Levere sluttrapport
- Milepål 7: Presentere sluttprodukt

## 7. Utviklingsfasen

I utviklingsfasen satte vi oss mål som var tett tilkoblet de forskjellige stadiene av våre MVP-er, ettersom at de var våre delmål. Dette førte til tre sprinter, hvor hver sprint varte i fem uker.

Under er en kort beskrivelse av arbeidet for hver av sprintene, og i vedlegg F – Beskrivelse av Sprint, kan man lese mer om hva som ble gjort i detalj og utfordringer vi opplevde etter hvert.

Gjennom hele utviklingsfasen hadde vi et ganske klart skille mellom backend-teamet og frontend-teamet. Selv om enkelte oppgaver berørte begge grupper, var det i hovedsak delt opp slik:

- Frontend: Jørund og Hajin
- Backend: Tom, Nikola og Ola

### 7.1. Sprint 1

#### 7.1.1. Oppgaver

Frontend:

- Sette seg inn i SurveyJS
- Brukerhistorier
- Prototype i Figma
- Lage farge-palett
- Hente JSON fra backend

Backend:

- Definere modellklasser
- Sette opp prosjektet, definere utviklingsmiljø
- Definere RESTAPI med GET-metode
- Kunne lese .XML filer og konvertere til Javaobjekt

### 7.1.2. Arbeidet

Etter å ha blitt ferdig med planleggingsfasen opprettet vi et GitHub repository med utviklingsmiljø for Spring Boot og React. Vi brukte skriptet «Create-React-App» for å sette opp et React prosjekt og Spring Boot Initializer for å generere de grunnleggende filene som trengs for å kjøre Spring Boot som en webapplikasjon. I forbindelse med dette ble det også lagd en README-fil med beskrivelse av hvordan man kan starte applikasjonen. Videre ble første skjema og kontakt mellom frontend og backend opprettet.

**System-for-behandling-av-XML-meldinger**

Build and run Maven tests passing

For å starte localhost server:

1. Du trenger Java JDK 17
2. Bygg prosjektet i IntelliJ ved å velge Build-> Build project
3. a.Kjør appen ved å starte KrefregisteretAppApplication fra IntelliJ  
b.Eller fra kommandolinje: cd inn til mappen der "pom.xml" ligger og kjør: `mvn spring-boot:run`

Hvis alt fungerer skal man kunne besøke backendserveren på localhost:8080

For å starte react dev server:

1. Du trenger Node version >14 fra terminal:
2. cd inn til mappen der "package.json" ligger
3. kjør `npm start`

Hvis alt fungerer som det skal vil browseren åpne localhost:3000 og vise React startpage.

Figur 17: README.md med steg for å kunne kjøre prosjektet

## 7.2. Sprint 2

### 7.2.1. Opgaver

Frontend:

- Validering av input i skjema for utredning
- Generalisering av inputkontrollere
- Starte på flere skjematyper

Backend:

- Skrive melding sendt fra klient til disk
- Definere RESTAPI med POST-metode
- Validering
- Testing av validering og POST-metode

### 7.2.2. Arbeidet

Ved starten på denne sprinten var vi allerede godt i gang med systemet, og hadde fått utviklet grensesnittet for det første skjemaet, «KliniskProstataUtredning», som henter en melding i JSON-format fra serveren. I et av møtene med oppdragsgiver, spurte Kreftregisteret om vi kunne lage en landingsside eller «dashboard» for prosjektet, det vil si en liste over tilgjengelige meldinger som gjør at bruker kan velge en melding som skal redigeres.

## 7.3. Sprint 3

### 7.3.1. Opgaver

Frontend:

- Sidebar med navigerbare titler
- Validere mot metadatabase
- Forbedre API-kall og responshåndtering
- Footer med handlingsknapper

Backend:

- Forbedre statuskoder sendt til API forespørsler
- API exceptions
- Deployere en demo av prosjektet til Heroku

### 7.3.2. Arbeidet

I den siste sprinten gjorde vi ferdige oppgaver som hadde blitt utsatt på grunn av lav prioritet.

Det ble også brukt tid på refaktorering med tanke på navngivning og gjenbrukbar kode.

## 8. Avslutningsfasen

Den siste fasen av prosjektet gikk til dokumentasjonsarbeid og rydding av kode. For et programvaresystem er pålitelig dokumentasjon viktig, da det kan hjelpe å holde styr på alle aspektene ved systemet og gjøre videreutvikling og vedlikehold enklere. Oppdragsgiver har satt av ressurser til at systemet skal bli videreutviklet i sommer, hvor to fra gruppen blant annet skal utarbeide integrasjon mellom vårt system og deres databasesystem. Vi har derfor satt inn mye arbeid i dokumentasjonen og forsøkt å være grundig i dokumentasjonen av hvordan systemet er bygd opp, i tillegg til erfaringer og utfordringer som dukket opp underveis.

Prosjektrapporten ble til et ganske stort dokument med mange vedlegg etter hvert og dette ga oss mange problemer underveis. Blant annet fordi Word-dokumentet ble delt gjennom Teams og siden alle hadde forskjellige desktop-versjoner fikk vi kompatibilitetsproblemer. Noen ganger opplevde vi at tidligere versjoner overskrev nyere endringer og vi måtte lete gjennom versjonslogger for å finne endringer som var gjort. Også når vi satt inn bilder opplevde vi problemer med at figur-nummere forsvant og ikke fulgte syntaksen. I tillegg til bildeproblemer så har vi hatt vanskeligheter med at overskrifter ikke ville bli oppdatert til riktig format og problemer rundt lagring og forskjellige Word-versjoner.

Med råd fra veileder har vi også forsøkt å prioritere universell utforming av prosjektrapporten, blant annet ved å bruke beskrivende lenker, korrekt formaterte overskrifter, gode kontraster og alternativ tekst som beskriver bilder (Statlig spesialpedagogisk tjeneste, 2020). Bilder/figurer som bakgrunnen på forsiden er blant annet markert som dekorativt for å ikke ødelegge leseflyten for en skjermleser.

# Produktdokumentasjon

Dette kapittelet er delt inn i tre hoveddeler. Den første delen omhandler skjemaet sitt design og hvordan den ble utformet, mens den andre delen beskriver det tekniske bak det man ser og brukeren direkte interagerer med - frontend. Den siste delen, backend, beskriver hvordan systemets tjener-sidekode fungerer, og hvordan systemet behandler XML-filer.

## 9. Design

Et digitalt skjema består av flere interaktive komponenter som har som mål å innhente data fra en bruker. Et typisk skjema ifølge Babich (2020) består av følgende fem komponenter: Struktur, inputfelter, ledetekster, handlingsknapper og tilbakemelding. I tillegg til disse typiske komponentene kan skjemaer også inneholde komponentene assistanse og validering. I utformingen av designforslag har vi forsøkt å ha beste praksis i mente og har gjennom iterative prosesser utviklet et forslag som er brukervennlig, moderne og som med gode valideringsfunksjoner sikrer at informasjonen som gis er riktig. Dette kapittelet presenterer valgene vi tok og hvorfor vi tok de valgene.

### 9.1. Presentasjon av design

Vi brukte Figma, en nettbasert grafikkredigering- og utformingsapplikasjon for brukergrensesnitt for å utarbeide fargepalett-forslag, personas og designforslag. En fargepalett for digitale tjenester er en samling av farger og annet visuelt som blir eller skal bli brukt. Fargepaletten i figur 21 er fra Kreftregisteret.no. Formålet med denne var å se om det var noen farger vi kunne bruke på skjemaet som brukerne allerede var godt kjente med. Ved å bruke farger brukerne allerede kjente, ville de muligens raskere få tillit til skjemaet og ønske å bruke den.

| Inspo fra hjemmesiden kreftregisteret.no  |                                      |   |
|---|--------------------------------------|---|
| Logo  | Farge på logoer og hyperlenker       | <p> verdier:<br>5F6278<br>Størrelse: 14px   |
|  | 6BA4B8                               |   |
|   | Farge på overskrifter                | <h3> verdier:<br>#21314d<br>Størrelse: 20px |
|   | 003E69                               |   |
|   | Farge på footer                      | <h2> verdier:<br>#21314d<br>Størrelse: 26px |
|   | D7F1F1                               |   |
|   | Farge på innholdsliste/dropdown meny | <h1> verdier:<br>003e69<br>Størrelse: 36px  |
|   | F2F0EA                               |   |
|   | Farge på innholdsliste/dropdown meny |   |
|   | E6F5F5                               |   |

Figur 18: Fargepalett laget fra hjemmesiden til Kreftregisteret, [www.Kreftregisteret.no](http://www.Kreftregisteret.no)

### 9.1.1. Personas

Personas er fiktive personer som representerer en bruker eller en brukergruppe. Basert på informasjonen vi fikk gjennom samtaler med brukere og produkteier, lagde vi to personas: Siri og Andam. Disse to personene representerer de medisinske koderne og hvor forskjellige de kan være. Siri er en eldre dame som har jobbet hos Kreftregisteret siden hun fikk fagbrevet sitt i helsefag. Siri er kjent for å være glad i struktur og ikke forandringer. På grunn av hennes kompetanse og sans for struktur, er hun også opplæringsansvarlig i tillegg til andre administrative oppgaver som hun gjør i løpet av en arbeidsdag. Andam fikk opplæring av Siri da han i sin tid var ny hos Kreftregisteret. Andam er utdannet fysioterapeut og jobbet som dette i flere år før han startet i jobben som medisinsk koder og fikk ansvaret for videreutvikling av skjemaer. Andam kalles ofte for IT-eksperten av de andre kollegaene da han alltid finner en løsning på IT-problemene de andre opplever. Mer om våre personas kan leses på de to neste sidene.

Personas-ene Siri og Andam hjalp oss å tenke utenfor boksen og tenke på hvilke funksjoner som faktisk var viktige for brukerne, i tillegg til hvordan de ville forholde seg til endringene - samt var personas-ene nyttige i arbeidet med å skape en balanse mellom det veldig moderne og konservative. Ettersom at det var så mange brukere som hadde lang fartstid hos Kreftregisteret og så mange som fortsatte i jobben selv etter pensjonsalder, var det viktig å skape en god balanse som beholdt folk i jobben. Å beholde dyktige folk i arbeid er viktig.

## 1. Persona



**Siri Jahren**

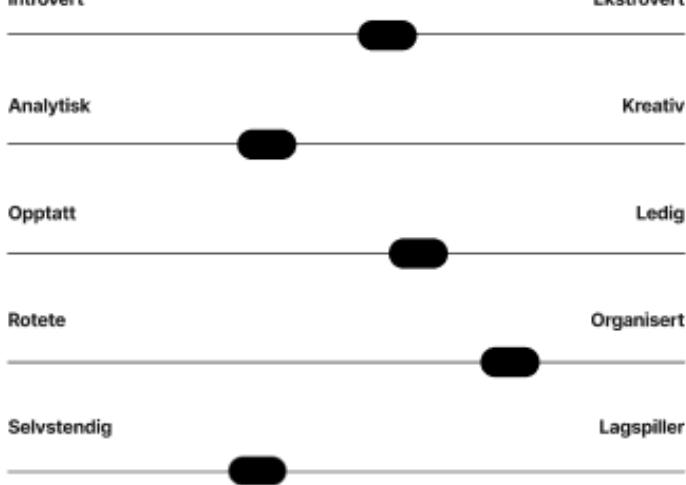
Alder: 58 år  
Yrke: Medisinsk koder  
Bosted: Nesodden  
Utdannelse: Fagbrev i helsefag  
Sivilstatus: Skilt

**Beskrivelse**

Siri har jobbet hos Kreftregisteret siden hun fikk fagbrevet sitt og trives veldig. På jobb er hun svært systematisk og liker å ha kontroll. Hun er ikke veldig glad i endringer og trives best med ting slik som de er. På fritiden inviterer hun gjerne venner og kollegaer på middager og i tillegg til å strikke nye plagg til barnebarna.

- Konservativ
- Livlig
- Ofte stresset

**Personlighet**



| Oppslidret  | Ekstrovert |
|-------------|------------|
| Analytisk   | Kreativ    |
| Oppatt      | Ledig      |
| Rotete      | Organisert |
| Selvstendig | Lagspiller |

**Dataferdigheter**

Kan bruke de verktøyene hun trenger for å utføre jobben sin, men utover det er det kun sosiale medier som Messenger, Instagram og Facebook det går i.

Figur 19: Personas Siri – den konservative som ikke liker forandringer.

## 2. Persona



**Andam Ibrahimovic**

Alder: 31 år  
Yrke: Medisinsk koder  
Bosted: Oslo  
Utdannelse: Fysioterapeut  
Sivilstatus: Samboer

**Beskrivelse**

Andam har opplevd kreft i nære relasjoner og ønsket derfor å bidra til bedre kreftforskning. Han trives i jobben hos Kreftregisteret. Andam bruker mye av fritiden på sykling og svømming, og stiller alltid opp dersom noen trenger hjelp innen fysioterapi.

- Omsorgsfull
- Avslappet
- Intressert

**Personlighet**

Introvert ————— Ekstrovert

Analytisk ————— Kreativ

Opptatt ————— Ledig

Rotete ————— Organisert

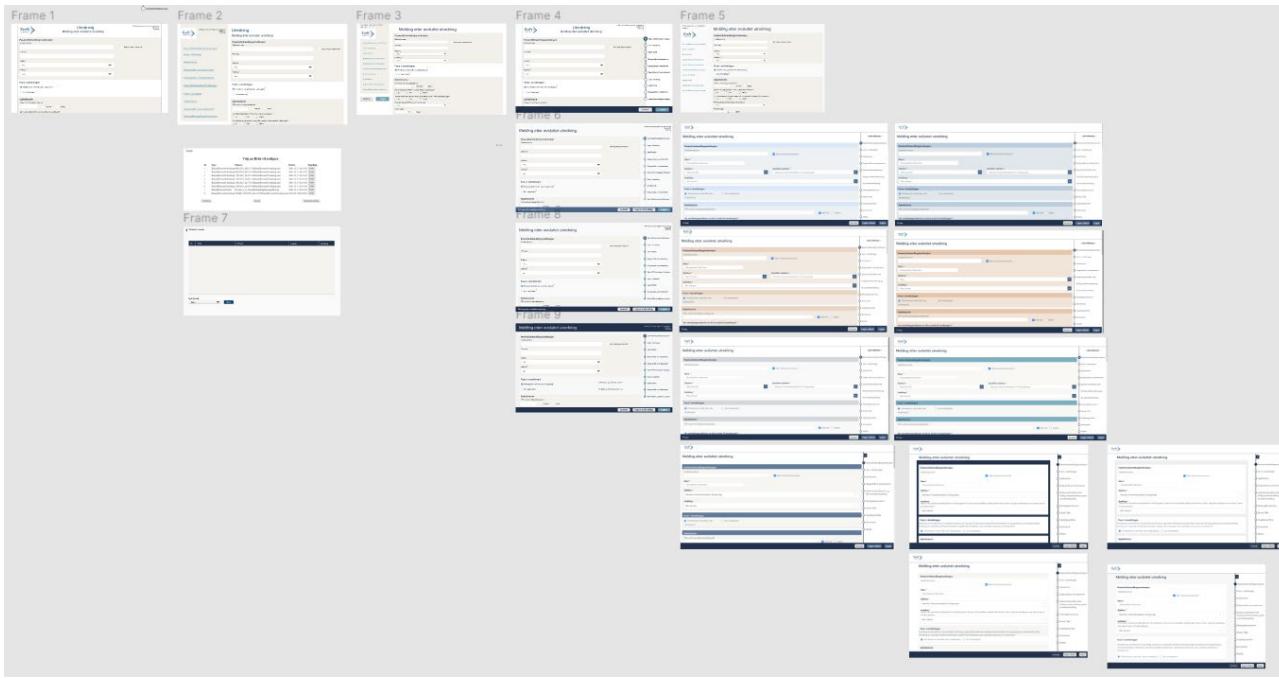
Selvstendig ————— Lagspiller

**Dataferdigheter**

Stor interesse for hvordan systemer henger sammen, både teknologiske og helsemessige. Han liker å utforske nye teknologier.

Figur 20: Personas Andam – den nye, datakyndige medisinske koderen.

### 9.1.2. Designforslag



Figur 21: Oversiktsbilde av designforslagene i Figma.

Selve utformingen var en lang prosess som en kan se fra oversiktsbildet fra Figma ovenfor. Vi startet med å fokusere på bakgrunnsfargen i skjemaet. Valget falt fort på en beige farge fra fargepaletten til Kreftregisteret.no. Fargen overholdt også WCAG-kravene om at skrift og bakgrunnen skal ha en viss kontrast som gjør teksten godt leselig for alle. Etter at bakgrunnsfarge var bestemt, satt vi opp et designforslag hvor vi plasserte inputfeltene i en kolonne-struktur med ledetekstene over for å øke hastigheten for skanning og forbedre leseflyten. Når dette forslaget var godkjent fra produkteier og en bruker, startet vi å se på hvordan vi kunne gjøre handlingsknappene mer brukervennlige. Ved å følge [denne koblingen](#) kan man se prosjektet i Figma.

Vi lagde tre nye forslag til hvordan handlingsknappene kunne plasseres for effektivitet, i tillegg ble det satt forskjellige farger på primærknapp og sekundærknapp for å minske risiko for feiltrykk. I de to første forslagene nedenfor, er knappene alltid synlig uansett hvor brukeren skroller til, mens i siste forslag er den i bunnen av skjemaet som den er i dag også.

Vi la så inn forslag på navigering. Å gi brukerne en metode til for navigering, som ikke er basert

på å skrolle nedover, var noe vi så på som viktig. En av årsakene til dette var for å minske risikoen for belastningsskader som kan komme som følge av bevegelser som skrolling med datamusen eller pekeflaten. Med en høyere andel kvinner enn menn i brukergruppen er risikoen for belastningsskader, som karpaltunnelsyndrom, høyere (NHI, 2018).

**Kreftregisteret**

## Utredning

Melding etter avsluttet utredning

Melding til nasjonalt register for prostatakreft  
Version 4.0  
Utredning

**Pasient/behandlingsinstitusjon**

Fødselsnummer:

Fullt navn:

Sykehus:

Avdeling:

**Funn i utredningen**

- Primærtumor (med eller uten metastase(r))
- Kun metastase(r)

**Sykehistorie**

PSA-verdi ved diagnostidspunkt:

Pasjonalt register for prostatakreft

Utredning

Pasient/Behandlingsinstitusjon

Funn i utredning

Sykehistorie

Diagnostikk av primærtumor

Pasient/Behandlingsinstitusjon

Funn i utredning

Sykehistorie

Diagnostikk av primærtumor

Pasjonalt register for prostatakreft

Utredning

**Avbryt** **Lagre**

Figur 22: Forslag 1 til fastplasserte primær- og sekundærknapper nederst på siden og navigasjonsbar på høyre side.

Melding til nasjonalt register for prostatakreft  
Versjon 4.0

**Kreft registeret**

**Pasient/Behandlingsinstitusjon**

Funn i utredning

Sykehistorie

Diagnostikk av primærtumor

Diagnostikk av primærtumor

Pasient/Behandlingsinstitusjon

Funn i utredning

Sykehistorie

Diagnostikk av primærtumor

Pasient/Behandlingsinstitusjon

**Melding etter avsluttet utredning**

**Pasient/behandlingsinstitusjon**

Fødselsnummer  
  Ikke norsk personnummer

Fullt navn

Sykehus

Avdeling

**Funn i utredningen**

Primærtumor (med eller uten metastase(r))  
 Kun metastase(r)

**Sykehistorie**

PSA-verdi ved diagnosetidspunkt  
  Ikke tatt  Ukjent

Var vannlatingsproblemer en del av årsak til utredningen?  
 Ja  Nei  Ukjent

Var kreftsymptomer (smarter, anemi eller annet) en del av årsak til utredningen?  
 Ja  Nei  Ukjent

Funksjonstilstand/WHO-status ved diagnose

Poststatavolum  
 mL  Ukjent

**Avbryt** **Lagre**

Figur 24: Forslag 2 til fastplasserte primær- og sekundærknapper under navigasjonsbar til venstre.

Melding til nasjonalt register for prostatakreft  
Versjon 4.0  
Utredning

**Kreft registeret**

**Pasient/Behandlingsinstitusjon**

Funn i utredning

Sykehistorie

Diagnostikk av primærtumor

Diagnostikk av primærtumor

Pasient/Behandlingsinstitusjon

Funn i utredning

Sykehistorie

Diagnostikk av primærtumor

Pasient/Behandlingsinstitusjon

**Melding etter avsluttet utredning**

**Variabeler**

Var vannlatingsproblemer en del av årsak til utredningen?  
 Ja  Nei  Ukjent

Var kreftsymptomer (smarter, anemi eller annet) en del av årsak til utredningen?  
 Ja  Nei  Ukjent

Funksjonstilstand/WHO-status ved diagnose

Poststatavolum  
 mL  Ukjent

**Diagnostikk av primærtumor**

Er det gjort MR prostata?  
 Ja  Nei

Er det gjort annen bildediagnostikk av primærtumor?  
 Ja  Nei

Celle-/Vevsprøver  
 Ja  Nei

Dato sykdommen ble bekreftet/diagnoseredato (dd.mm.åååå)

**Patologilaboratorium**

**Avbryt** **Lagre**

Figur 23: Forslag 3 til primær- og sekundærknapp i bunn av skjemaet og fastplassert navigasjonsbar til venstre.

Navigeringen tar utgangspunkt i de ulike delene av skjemaet. Overskriftene fra grupperingene i skjemaet blir brukt som hyperlenker. Hyperlenkene har samme design som hyperlenkene på Krefregisteret.no i figur 27 og figur 26. Ved trykk blir brukeren sendt til riktig sted i skjema. Figur 25 sin navigasjonsbar tar også utgangspunkt i grupperingene som er gjort i skjema, men i form av runde knapper og ikke hyperlenke. Dette forslaget har et mer moderne og lekent uttrykk som inviterer brukeren til å trykke. Plasseringen av navigasjonsbaren har også en god grunn. Ettersom at de fleste er høyrehendte i befolkningen generelt, er tanken at for de brukerne som bruker datamus, vil navigasjonsbaren ikke være så langt unna og minske unødvendig bevegelser. I fremtidige design bør det legges til rette for at brukeren kan flytte navigasjonsbaren til venstre for å inkludere venstrehendte også. I tillegg til å gi en alternativ måte å navigere skjemaet på, gir navigasjonsbaren en oversikt over skjemaet.

### 9.1.3. SurveyJs og brukervennlighet

SurveyJs-biblioteket som er brukt for logikken i skjemaet, hadde en del innebygde funksjoner for å øke brukervennlighet. Blant annet var SurveyJs elementene responsive og tillater at man har flere skjemaer opp samtidig. De andre funksjonene la vi inn i våre designforslag etter behov. Nedenfor blir noen av disse presentert i tillegg til hvilke endringer vi har gjort på de innebygde funksjonene for å øke brukervennligheten.

#### 9.1.3.1. Tilbakemeldinger

For tilbakemeldinger på inputfeltet var vårt forslag å bruke SurveyJs sin løsning med noen få endringer. Dersom en bruker skriver inn tekst istedenfor tall i et inputfelt som krever tall, eller brukeren skriver et tall for mye, vil en feilmelding dukke opp rett etter brukeren flytter seg fra

The figure consists of two side-by-side screenshots of a survey form. Both screenshots show a field labeled 'Pasient/behandlingsinstitusjon' and a required field 'Fødselsnummer \*'. In the left screenshot, a red error box contains the text 'Skriv inn gyldig fødselsnummer'. Below this box, a grey box displays the message 'Dette er ikke et nummer'. In the right screenshot, the same red error box contains the text 'Skriv inn gyldig fødselsnummer'. Below it, a grey box displays the message 'Eksempel på feilmelding'.

Figur 25: Til venstre: SurveyJs sitt innebygde feilmelding-layout. Til høyre: Endringer som ble gjort for å styrke kontrast.

det feltet. Feilmeldingen er tydelig plassert mellom ledteksten og inputfeltet for å tydeliggjøre hvilket felt feilmeldingen tilhører. I tillegg har feilmeldingen er annen farge enn resten av skjemaet og en tydelig kant som gjør den vanskelig å overse. Den originale løsningen fra SurveyJs hadde svak kontrast mellom bakgrunn og tekst og vi endret derfor skriftfargen for å styrke den.

#### 9.1.3.2. Validering

Vi foreslo også å bruke valideringsfunksjonen til SurveyJs. Valideringen av hele skjemaet slår ut når en bruker trykker på «Lagre»-knappen. Dersom en bruker har oversett et eller flere obligatorisk felt, vil brukeren bli sendt til det første feltet som mangler og bedt om å fylle ut. Feltet får autofokus slik at brukeren ikke trenger å trykke på feltet for å fylle den.

Utført dato \*

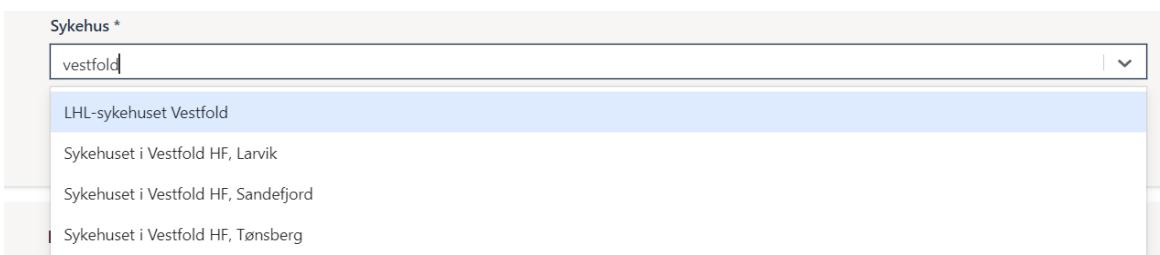
Vennligst svar på spørsmålet.

dd.mm.åååå   Ukjent

Figur 26: Valideringen sender brukeren direkte til første felt som ikke er fylt ut.

#### 9.1.3.3. Søkbare nedtrekkmenyer

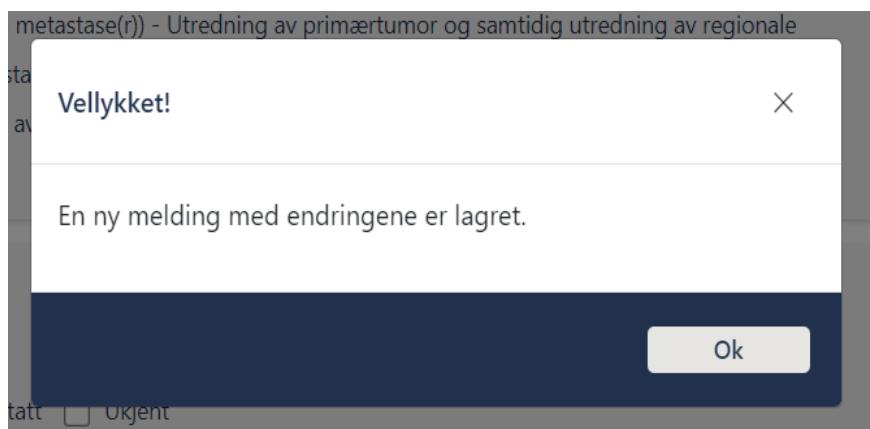
Nedtrekkmenyene kan bli veldig store og omfattende, det er for eksempel over 100 sykehus å velge mellom. Å lete gjennom de hundre sykehusene, er dermed lite brukervennlig. I løsningen vår la vi inn en søke-funksjon i nedtrekkmenyene. Istedeknuten for å skrolle seg ned til riktig sykehus, kan brukeren bare skrive inn de første bokstavene av navnet og få opp forslag som passer til det innskrevne.



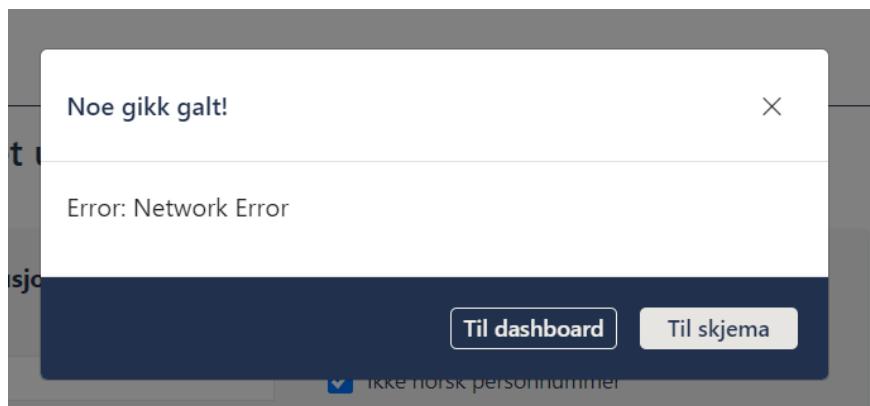
Figur 27: Nedtrekkmenyene har en søkefunksjon som gjør det enkelt å bruke.

#### 9.1.3.4. Modal for bekrefte/tilbakemelding

Når man trykker på standardknappen for lagring av skjema fra SurveyJs, blir man sendt til en ny side med bekrefte/tilbakemelding. Vi fjernet denne knappen og satt inn funksjoner i vår egen knapp som gir bekrefte/tilbakemelding i en modal. Modalen er fra hentet fra biblioteket Reactstrap og kommer med et design og funksjonalitet som bevarer brukervennlighet. Modalen er tastaturvennlig, og designet bygger på Gestaltprinsippene om forgrunn og bakgrunn for å skape riktig fokus. Dersom det skulle oppstått nettverksproblemer får brukeren mulighet til å gå tilbake til skjema og gjøre eventuelle endringer eller gå tilbake til dashboard uten å lagre endringer.



Figur 29: Ved en veldig suksessfull lagring får brukeren opp en modal som bekrefter handlingen.



Figur 28: Ved nettverksfeil får brukeren beskjed og får muligheten til å gå tilbake til skjema eller til dashboardet.

### 9.1.3.5. Trykkbare beskrivelser/ledetekster

I dagens løsning er avkrysningsbokser og radioknapper svært små og en bruker må klare å trykke riktig på inputfeltene for å merke de. SurveyJs sine ledetekster til radioknappene og avkrysningsboksene er trykkbare som gjør det enklere for brukere med datamus eller pekeflate å trykke på disse.

## 9.2. Brukertesting

Ettersom at dette er et skjema som skal bli mye brukt hver dag i løpet av en brukers arbeidshverdag, var det viktig for oss å involvere brukerne og gi de en stemme. Vi utførte til sammen to større brukertester, men hadde i de ukentlige statusmøtene alltid en bruker til stede som var med for å gi tilbakemeldinger fra bruker perspektivet. Nedenfor blir de to større brukertestene forklart nærmere.

### 9.2.1. Første brukertesting

Første brukertest ble utført i samme møte som vi skulle ha demo av løsningen hittil. Målet for den første brukertesten var å få tilbakemeldinger på det visuelle som fargene, plasseringen av handlingsknappene og navigasjonsbaren sin utforming og plassering. Brukertesten foregikk over samhandlingsverktøyet Teams hvor brukerne først fikk en presentasjon av fem designforslag. Her brukte vi Figma sin prototyping-funksjon og la inn noen handlinger en bruker kunne gjøre for å illustrere blant annet navigasjonsbaren. Så fikk brukerne en lenke til en spørreundersøkelse. Spørreundersøkelsen inneholdt bilde av de fem designforslagene og brukerne kunne ved hjelp av en likert-skala dele hvor godt de likte hvert design. På slutten av spørreundersøkelsen ble de stilt spørsmål som gikk direkte på plassering av handlingsknappene og generelle farger. Se vedlegg G – Brukertest 1 – Skjema og vedlegg H – Brukertest 1 – Svar for mer detaljer om spørreundersøkelsen.

Vi fikk inn svar fra åtte brukere og resultatene viste at den mer lekne navigasjonsbaren og handlingknapper i en footer var mest ønsket. Testbrukerne var fornøyde med fargene og mente at de var nøytrale nok.

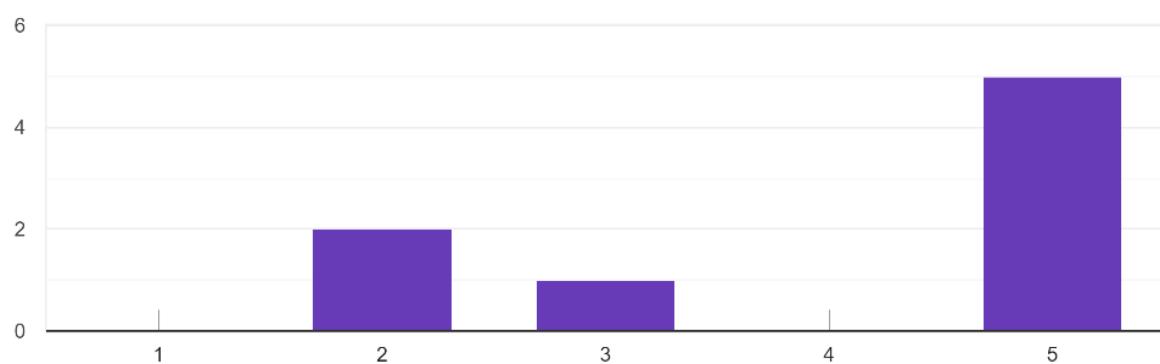
Etter presentasjonen og spørreundersøkelsen fikk testdeltakerne en lenke til selve skjemaet. Denne hadde SurveyJs sitt standard design, altså uten navigasjonsbaren og de fastplasserte knappene. Her fikk deltakerne testet ut de innebygde funksjonene til SurveyJs med respons på feil og navigering med tastatur. Tilbakemeldingen fra brukerne på slutten av brukertesten var at de så frem til et mer moderne uttrykk og et mer brukervennlig skjema. De var mest begeistret

over at man ble sendt direkte til hvilke felt som manglet dersom man trykket «Lagre» uten å ha fylt ut et obligatorisk felt, i tillegg til at handlingsknappene var så lett tilgjengelige.

Etter første brukertesting lagde vi et nytt forslag til et design med navigasjonsbar til høyre og handlingsknapper plassert fast nederst på siden. Den opprinnelige beige fargen fra Kreftregisteret.no ble også byttet ut til med en mildere beige farge som ga et mer moderne uttrykk.

#### Hvor godt liker du dette designet?

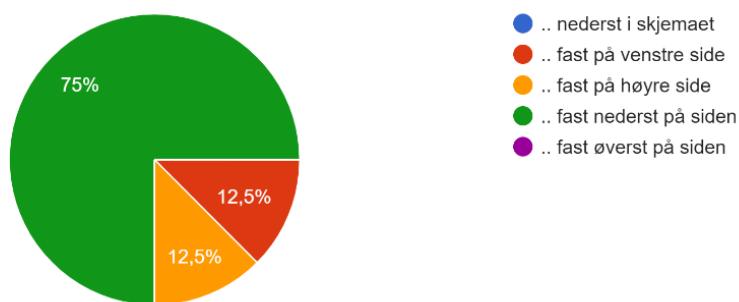
8 svar



Figur 30: Fem av åtte testbrukere likte designforslaget med en leken navigasjonsbar best (Forslag 1 i figur 25).

#### Jeg vil ha lagre og avbryt-knappen..

8 svar



Figur 31: 75% av testbrukerne ønsket seg handlingsknapper fast nederst på siden.

## 9.2.2. Andre brukertesting

Formålet med den andre brukertesting var å undersøke om det nye skjemaet faktisk var mer effektivt å bruke og opplevdes bedre. Dermed skulle testbrukerne få en lenke hvor vi hadde distribuert koden vår, så de kunne utføre noen oppgaver før de ga tilbakemelding gjennom et spørreskjema. Dessverre fant vi for sent ut at distribusjonen av koden ikke hadde gått som planlagt. Vi hadde våre lokale servere kjørende med de nødvendige filene da vi dagen i forveien testet om distribusjonen funket. Så da vi testet igjen rett før brukertesting, viste det seg at det var noe feil i koden som førte til at den kjørte riktig kun hvis du hadde en lokal server med koden kjørende hos deg. Dermed måtte vi i løpet av 10 minutter finne på en ny løsning. Vi hadde to deltakere til denne brukertesten og testen foregikk også denne gangen over Teams.

Testen ble delt inn i to deler, der første del omhandlet endring og innsending av melding.

Deltakerne observerte hvordan vi åpnet en melding og gikk gjennom og endret meldingen, også svarte de på noen spørsmål knyttet til det de observerte. I andre del av testen skulle man bruke navigasjonsbaren aktivt i arbeidet med å få oversikt over hva som står i meldingen før man startet å redigere. Igjen fikk deltakerne noen spørsmål de skulle svare på knyttet til det de observerte.

Tilbakemeldingen fra deltakerne var at de opplevde skjemaet som mer strukturert og generelt mer behagelig å jobbe i på grunn av luft og farger. Videre syntes de at tilbakemelding på lagring var bedre, siden man får opp en modal og ikke blir sendt videre til en annen side for feilmelding/bekreftelse på lagring. Mens en av deltakerne ikke så en personlig nytte for navigasjonsbaren, var den andre mer begeistret og så den som verdifull i fremtidig arbeid. Vi hadde også flyttet noen hjelpeTekster inn i nedtrekkmenyene og dette likte de svært godt da det samsvarer med innholdet til de andre nedtrekkmenyene også. Se vedlegg I – Brukertest 2 – Skjema og vedlegg J – Brukertest 2 – Svar

Etter andre brukertest gjorde vi noen endringer på det visuelle basert på tilbakemeldingene. Inputfeltene fikk en bredde som samsvarer mer med forventet innhold og navigasjonsbaren

følger og markerer hvor brukeren er til enhver tid. Nedenfor kan en se det endelige resultatet fra testingene:

**Totalvurdering av klinisk T (laveste stadium velges ved tvil) \***

TNM på diagnosetidspunktet baseres på all relevant utredning som er gjort før primærbehandling inkludert klinisk undersøkelse, bildediagnostikk, endoskopi, biopsi og kirurgisk eksplorasjon.

TX - Primaertumor kan ikke vurderes

T0 - Primaertumor ikke påvist

T1 - Ingen tumor påvisbar med palpasjon, ultralyd eller annen radiologisk metode, men cancer er påvist ved prostatabiopsier

T2 - Palpabel eller synlig tumor begrenset til prostatakjertelen (innenfor kapselbegrensning)

T3 - Ekstrakapsulær tumorvekst. NB: Innvekt i apex prostata eller inn i (men ikke gjennom) prostatakapselen klassifiseres som T2

Ukjent  
 CT  
 MR  
 PET  
 Lymfadenektomi (diagnostisk i forkant av behandling)  
 Annet

Spesifiser

**Erfjernmetastaser, inkludert fjerne lymfeknudemetastaser, påvist (M-sykdom)? \***

Grensen mellom regionale lymfeknudemetastaser og fjernmetastaser går ved delingsstedet til arteria iliaca communis. Ved tvil om korrekt M-kategori skal den laveste (minst avanserte) delen tilsvare M1.

\* Obligatorisk

Lagre   Lagre utkast   Avbryt

Navigation sidebar:

- >
- Pasient/behandlingsinstitusjon
- Funn i utredningen
- Sykehistorie
- Diagnostikk av primærtumor
- Sykmeldsutbredelse etter ferdig primærutredning og før primærbehandling**
- Klinisk TNM
- Patologilaboratorium
- Oppfølging/tiltak
- Kommentar
- Melder

Figur 32: Siste designforslag som oppfyller brukerkrav og behov.

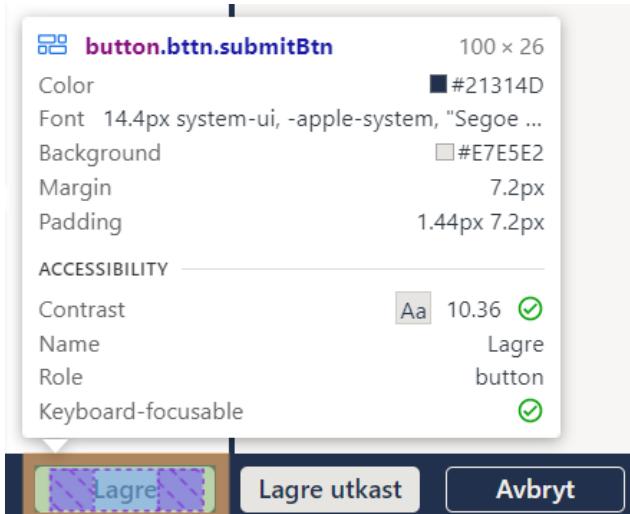
## 9.3. Universell utforming

Å utforme produkter, applikasjoner og tjenester slik at de kan brukes av så mange som mulig på en likeverdig måte, kalles for universell utforming. Universell utforming skal bidra til at alle har like muligheter for å delta i samfunnet og vil motvirke diskriminering basert på for eksempel funksjonshemming (Lid, 2021). I Norge har vi en forskrift som stiller krav til de digitale tjenestene offentlige og private virksomheter tilbyr. 35 av 61 suksesskriterier i standarden Retningslinjer for tilgjengelig webinnhold (WCAG) 2.0 er obligatoriske for alle med unntak for systemer som kun tas i bruk av ansatte ved en virksomhet (UUTilsynet, u.å.).

Selv om det ikke er pålagt for Kreftregisteret å utforme skjemaene med kravene fra WCAG 2.0 fordi den brukes internt av ansatte for å utføre et arbeid, valgte vi å ta hensyn til kravene så mye som vi kunne. Det er tenkt frem i tid at alle helseinstitusjoner som skal sende inn kreftmeldinger skal bruke dette skjemaet for innsending. Da vil de få mange flere forskjellige brukere som gjør at det blir viktig at skjemaene er tilgjengelige. Skjemaer som er designet med universell utforming i mente, sikrer at flere brukere kan bruke tjenesten - samt minsker risiko for at brukere sender inn feil informasjon og er generelt mer brukervennlige.

### 9.3.1. Tilgjengelighet for brukere med synsutfordringer

Skulle en bruker ha utfordringer med synet og ha behov for å forstørre eller forminske skjemaene, har vi sørget for at dette er mulig uten at innhold/funksjonalitet går tapt. Selv om de fleste i målgruppen er kvinner og dermed har lavere risiko for å ha nedsatt fargesyn, så vil de fleste ha godt av at farge ikke er brukt som eneste virkemiddel i tillegg til at vi har forsikret oss om at tekst mot bakgrunn har god kontrast. For de som bruker skjermlesere derimot er det en fordel at brukergrensesnittet gjenspeiler koden og at knapper har navn og rolle. En annen fordel for skjermlesere er at SurveyJs følger korrekt rekkefølge fra <h1> til <h5>.



Figur 33: Tekst mot bakgrunn har god kontrast og knapper og annen interaktive elementer har passende roller og navn.

### 9.3.2. Tastaturnavigasjon

Skjemaene og dens tilhørende funksjoner betjenes via et tastaturgrensesnitt, dette er både helsemessig positivt for å unngå belastningsskader og øker også effektivitet. Med en gang man kommer inn på et skjema, blir skjul/vis knappen til navigasjonsbaren fokusert. Dermed kan brukeren velge å ha den opp eller ikke. For en skjermleser er aria-label lagt til på elementet med teksten «Hide sidebar» dersom navigasjonsbaren er åpen og «Show sidebar» dersom den er lukket. Hvis brukeren velger å lukke navigasjonsbaren, vil brukeren bli sendt direkte til skjema. Dersom bruker ønsker å ha navigasjonsfeltet opp kan hen bruke tab til å navigere og se gjennom skjema, når brukeren har tabbet gjennom vil hen på slutten av navigasjonsbaren bli sendt inn i skjema for eventuelle endringer. Når brukeren bruker tastaturet for å navigere, får inputfeltene tydelig og synlig fokus uten at konteksten endres. Slik vil brukeren alltid vite hvor hen skriver og er i skjemaet.

### 9.3.3. Tilgjengelig modalvindu

Modal-vinduer kan være frustrerende for skjermleser- og tastaturbrukere. For å minske frustrasjon har vi gjort hjelpeTekstene mer tilgjengelige ved at de er synlige i selve skjemaet eller flettet bedre inn i nedtrekkmenyene og unngår at de kommer opp i et modalvindu eller annen

interaktiv måte. Videre har vi sett til at modalvinduet som gir tilbakemelding på lagring er tastaturvennlig og får fokus med tydelig forgrunn og bakgrunn.

## 9.4. Endelig design

Figur 34: Designet er responsiv, gir brukerne bedre navigasjonsmuligheter og tydelige tilbakemeldinger.

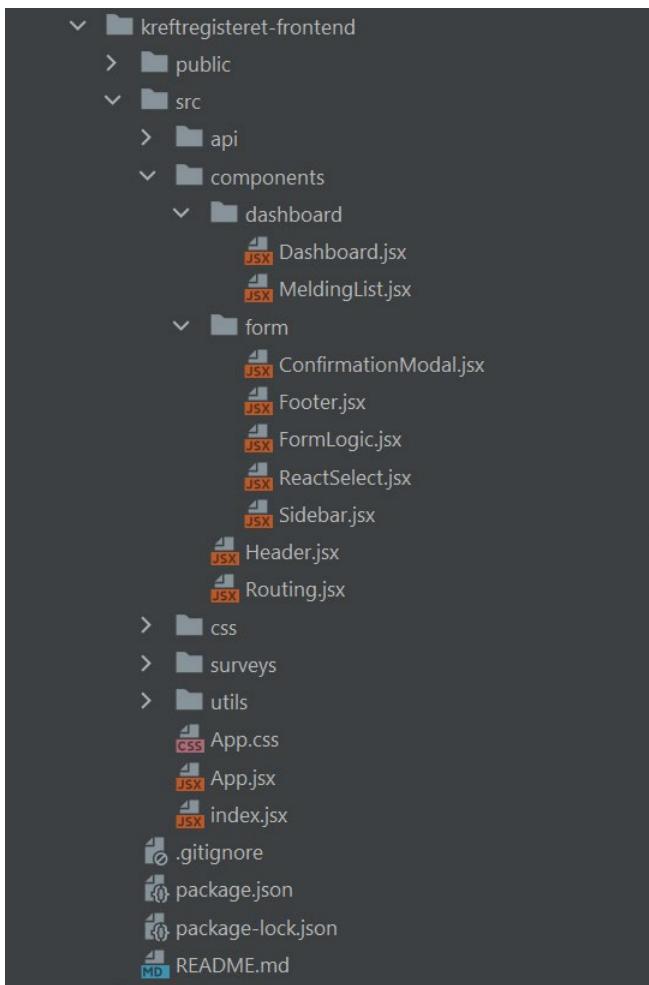
Gjennom en brukersentrert designprosess har vi utarbeidet et design som tar hensyn til de som ikke er glad i endringer, de som trenger endringer og de som ønsker å jobbe til tross for fysiske utfordringer. Vi har tatt med oss de gode designløsningene fra forrige skjema og gjort skjemaene mer brukervennlige. Designet har fokusert på brukernes helse og velvære for å skape noe som gir få belastninger i en arbeidshverdag hvor man må se gjennom og eventuelt redigere 10-20 kreftmeldinger. Designet har også satt søkelys på å minimere brukerfeil som gir en trygghet til brukerne og andre som kommer til å benytte seg av skjemaene – samt bidrar dette til å overholde det eksterne kravet fra Normen om dataenes integritet. Det visuelle uttrykket reflekterer at brukerne jobber med moderne teknologi og gir en avslappende følelse. For flere bilder av det endelige designet, se vedlegg L – Endelig design.

## 10. Frontend

Frontend-prosjektet er skrevet med Javascript-bibliotekene React og SurveyJs. Elementene i React komponentene og SurveyJs er styrt med CSS og i tillegg er CSS-rammeverket Reactstrap brukt på blant annet modal-vindu. For kommunikasjon med backend er tredjeparts-biblioteket Axios brukt. I dette kapittelet presenteres det i detalj hvordan brukergrensesnittet er utviklet ved hjelp av disse teknologiene.

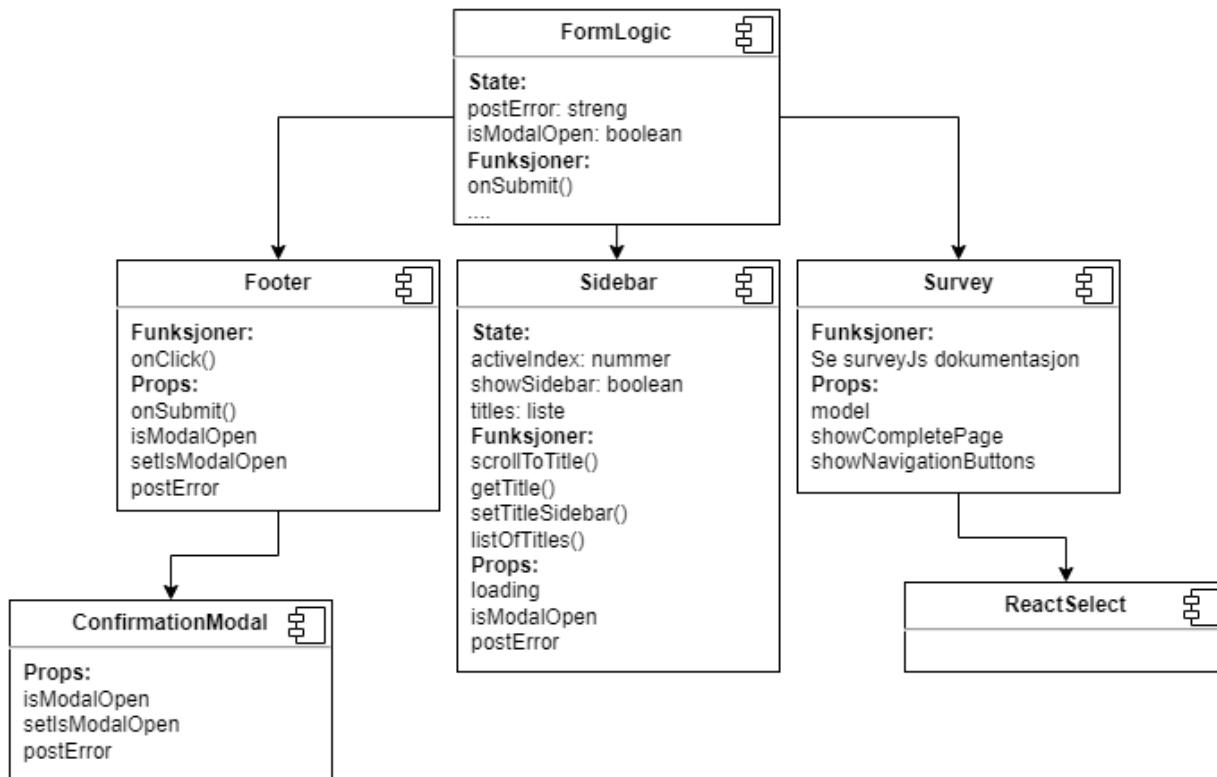
### 10.1. Komponenter og funksjoner

React tillater at man utvikler applikasjoner som en samling av flere komponenter og funksjoner. En React komponent er en del av brukergrensesnittet som kan bli gjenbrukt og som kan håndteres separat. Komponentene holder verdier i objekter som kalles tilstander eller «states».



Figur 35: Mappestrukturen for frontend.

Hver tilstand har en tilhørende funksjon der navngivingen av funksjonene starter med «set» for å endre verdiene den holder. Endring av disse verdiene kan føre til at komponenten kjøres på nytt. Komponentene kan også ta inn verdier, «props» som står for «properties», og returnerer et element som kommer til uttrykk i brukergrensesnittet (Edpresso Team, 2022). I løsningen vår ligger alle komponentene under mappen «components» som igjen er delt inn i undermapper for bedre oversikt og struktur. Under mappen «api» er en gjenbrukbar api-funksjon lagt. Figur 38 viser strukturen i frontend-prosjektet og figur 39 viser hvordan komponentene er satt sammen.



Figur 36: Diagram av hvordan komponentene er satt sammen.

### 10.1.1. GetMeldinger

Metoden GetMeldinger() brukes i «MeldingList»- og «FormLogic»-komponentene og ble derfor etter et par iterasjoner gjort til en gjenbrukbar metode. I FormLogic tar den inn en id som argument, mens den ikke har noen argumenter i MeldingList. Dette fordi i FormLogic skal den hente en spesifikk melding med en id som sendes fra backend, mens i MeldingList skal den få en liste med alle meldingene som er på serveren. Metoden har tre tilstander som den også

returnerer når den brukes. Objektet «data» er responsen fra API-kallet, boolean verdien «loading» forteller om API-kallet er ferdig og objektet «error» inneholder eventuelle feilmeldinger fra API-kallet.

```
const GetMeldinger = (id : string = "") => {
    const [data, setData] = useState( initialState: null );
    const [loading, setLoading] = useState( initialState: true );
    const [error, setError] = useState( initialState: null );

    useEffect( effect: () => {
        const getData = async () => {
            setLoading( value: true );
            setError( value: null );
            try {
                const response = await axios.get( url: URL.url + id );
                response && setData(response.data);
            } catch (err) {
                console.log(err);
                setError(err);
            } finally {
                setLoading( value: false );
            }
        };
        getData();
    }, [deps: [id]]);
    return {
        data, loading, error
    }
};
```

Figur 37: GetMeldinger()-metoden har en valgfri parameter og returnerer data, loading og error.

Metoden bruker Axios, et moderne HTTP-klientbibliotek til å gjøre API-kallene. En av fordelene med Axios er at den er enkel å lese, bruke og fungerer godt med nøkkelordene `async/await`. Ved å pakke API-kallet i nøkkelordene `async/await` sikres det at hovedtråden ikke begynner å kjøre videre før den asynkrone delen av funksjonen er fullført og dermed unngår å bli overbelastet (Gupta, 2021). I løsningen vår har vi også pakket kallet inn i en «try-catch»-blokk for å håndtere

eventuelle feil som oppstår i funksjonen. Metoden vi implementerte er inspirert av Github-prosjektet «custom-use-fetch-hook-react» av brukeren machadop1407 (Machadop1407, 2021).

## 10.1.2. Dashboard

«Dashboard» er en komponent som tillater oss å hente og redigere forskjellige meldinger.

Komponenten er kun ment som et verktøy for utviklere under utviklingen og skal ikke være en del av sluttproduktet når Kreftregisteret skal implementere løsningen i sine systemer. På grunn av dette blir styling gjort direkte i elementet, altså inline styling, eller gjennom App.css filen.

### 10.1.2.1. Utforming av dashboard

The screenshot shows a dashboard interface. At the top left is the Kreftregisteret logo. Below it is a section titled "Velg en fil du vil redigere" (Select a file you want to edit). A table lists four entries:

| Id | Type                     | Filnavn   | Sist endret         | Handling               |
|----|--------------------------|---|---------------------|------------------------|
| 1  | KliniskProstataUtredning | 08022022_kl231751KliniskProstataUtredning.xml       | 2001-12-17 kl.09:30 | <button>Endre</button> |
| 2  | KliniskProstataUtredning | 08042022_kl211251KliniskProstataUtredning.xml       | 2001-12-17 kl.09:30 | <button>Endre</button> |
| 3  | KliniskProstataStraale   | Prostata_4_0_StraalebehandlingEksempelfil.xml       | 2001-12-17 kl.09:30 | <button>Endre</button> |
| 4  | KliniskProstataUtredning | VALID_07032022_kl141937KliniskProstataUtredning.xml | 2001-12-17 kl.09:30 | <button>Endre</button> |

Below the table is a dropdown menu with the following options:

- Velg nytt skjema ▾
- KliniskProstataUtredning
- KliniskProstataKirurgi
- KliniskProstataStraale

Figur 38: Fra dashboardet kan utviklere teste hvordan forskjellige skjemaer ser ut og teste å redigere meldinger.

Dashboardet er ganske enkelt og består av en tabell og en nedtrekkmeny med handlingsknapper. Tabellen viser liste over kreftmeldinger på serveren med spesifikasjonene id, type, filnavn og tidspunkt for sist endret. Helt til venstre kan en utvikler trykke på «Endre» for å

se og/eller redigere kreftmeldingen. Nedtrekkmenyen består av de tre typene melding en prostata kreftmelding kan være. Ved trykk på en av disse, får man et tomt skjema.

#### 10.1.2.2. MeldingList

```
const {data, loading, error} = GetMeldinger();
```

Figur 39: Ved bruk av den egenkomponerte metoden "GetMeldinger()" hentes meldinger fra serveren til dashboardet.

Listen av meldingene blir hentet fra serveren gjennom et API-kall. API-kallet gjøres i den egenkomponerte GetMeldinger()-metoden under mappen «api». Figur 42 viser hvordan returverdiene «data», «loading», «error» fra GetMeldinger()-metoden er med på å bestemme hva komponenten skal returnere. Dersom «loading» ikke er sann og responsen «data» ikke er tom, vil metoden for visning av meldinger, «tableOfMeldinger()», kjøres. Ellers dersom feilmeldingsobjektet «error» ikke er tomt, vil feilmeldingen vises og bruker få beskjed om «Noe gikk galt ved innlasting». Til slutt vil brukeren få beskjed om at meldingene lastes inn dersom «loading» er sann og innlasting pågår.

```
return (
  !loading && data !== null ?
    tableOfMeldinger(data) :
  error !== null ?
    <div>
      <h2>{error.toString()}</h2>
      <p>Noe gikk galt ved innlasting</p>
    </div> :
    <h2>Laster inn meldinger...</h2>
);
```

Figur 40: MeldingList returnerer elementer basert på dataene den får fra API-kallet.

Funksjonen «tableOfMeldinger()» har en parameter og returnerer en <Table>-komponent fra Reactstrap-biblioteket. Argumentet som settes inn forventes å være en liste og dermed skal man kunne gå gjennom listen ved hjelp av map-metoden i Javascript. Map()-metoden gir tilgang til hvert element i listen slik at man kan få hvert element sin verdi som er definert under de ulike attributtene «id», «skjemanavn», «filnavn», «endrettidspunkt».

```

const tableOfMeldinger = (data) => {
  const rows = data.sort((a, b) => a.id > b.id ? 1 : -1).map((item, index) =>
    <tr key={index}>
      <td>{item.id}</td>
      <td>{item.skjemanavn}</td>
      <td>{item.filnavn}</td>
      <td>{formatJsonDate(item.endrettidspunkt)}</td>
      <td...>
    </tr>
  );
}

return (
  <Table hover
    responsive
    size="xl"
    striped>
    <thead style={{textAlign: "left", backgroundColor: Color.king_blue, color: "white"}}...>
      <tbody>
        {rows}
      </tbody>
    </Table>
);
};

```

Figur 41: Funksjonen "tableOfMeldinger()" har en parameter «data» og returnerer en tabell med innhold fra parameteren.

### 10.1.3. FormLogic

Funksjonene for datahåndtering i skjemaene er kodet i FormLogic-komponenet, disse funksjonene er ganske komplekse og blir nærmere forklart under avsnittet «10.3 Datahåndtering i skjema». FormLogic-komponenten er noe ulik de andre komponentene fordi den returnerer tre selvstendige komponenter. Dersom «error»-objektet fra GetMeldinger() ikke inneholder noen feilmeldinger vil den vise <Sidebar>-, <Survey>- og <Footer>-komponenten. Dersom «error» inneholder feilmeldinger, vil et element med feilmeldingen vises.

```
return (
  <>
  {error === null ?
    <div className={"surveyContainer"}>
      <Sidebar
        className={"sidebar"}
        loading={loading}
        isModalOpen={isModalOpen}
        postError={postError}
      />
      <Survey
        model={survey}
        showCompletedPage={false}
        showNavigationButtons={false}
      />
      <Footer
        onSubmit={submit}
        isModalOpen={isModalOpen}
        setIsModalOpen={setIsModalOpen}
        postError={postError}
      />
    </div>
    :
    <div className={"errorContainer"}...>
  }
</>
```

Figur 42: Formlogic returnerer tre komponenter: Sidebar, Survey og Footer.

FormLogic-komponenten er morkomponenten til <Sidebar>-, <Survey>- og <Footer>-komponenten, som vil si at når FormLogic kjører, vil «barna» også kjøre. Selv om barna er selvstendige komponenter, kan de gjennom props fra morkomponenten gi en mer tilpasset oppførsel. Survey-komponenten er fra SurveyJS-biblioteket og viser skjemaer basert på JSON-objektet som blir sendt inn gjennom props-et «model».

### 10.1.3.1. Sidebar

Sidebar-komponenten tar inn tre props: «loading», «isModalOpen» og «postError». «loading» er en boolean-verdi fra GetMeldinger()-metoden og forteller komponenten når skjemaet er lastet inn så den kan returnere brukergrensesnittet sitt. «isModalOpen» er en tilstand med boolean verdier i morkomponenten. Dersom verdien er sann, vil en modal åpnes. «postError» er også en tilstand, men verdien er strenger. Dersom post-metoden i morkomponenten skulle feile, vil feilmeldingen legges inn i tilstanden «postError» som en streng. Dersom «isModalOpen» er sann eller «postError» inneholder noe annet enn en tom streng, vil observasjoner av endringer i DOM-en stoppes. DOM står for «Document Object Model» og referer til blant annet HTML-strukturen på nettapplikasjonen. Foruten tilstandene sidebaren får fra morkomponenten, har den tre tilstander selv: «activeIndex» som har tallverdier, «showSidebar» med boolean verdier og «titles» som er en liste. Hver av tilstandene er med på å styre en funksjon.

#### 10.1.3.1.1. listOfTitles

```
const listOfTitles = (titles) => {
  return (
    <div className={showSidebar ? "sidebar" : "hide"}>
      {titles && titles.map((title, index) => {
        return (
          <div id={`${index}`}
            className={"sidebarNav"}
            key={index}
            onClick={() => { scrollToTitle(title, index); }}
          >
            <button aria-label={`${title.innerText}`}
              className={activeIndex === index ? "titleBtn activeTitleBtn" : "titleBtn"}
            />
            <span className={"sidebarTitle"}
              role={"button"}
            >
              {title.innerText}
            </span>
          </div>
        )})
      </div>
    );
}
```

Figur 43: Funksjonen "listOfTitles" har en parameter "titles" som skal være en liste den skriver ut overskrifter fra.

Funksjonen «listOfTitles» har en parameter «titles» og returnerer et «div»-element. Hvis «showSidebar»-tilstanden er ikke sann vil elementet få et klassenavn med navnet «hide» som betyr at sidebaren skjules ved hjelp av CSS-kode. Når «showSidebar» er sann vil argumentet som tas inn bli sjekket for at den ikke er tom før den går gjennom ved hjelp av map()-metoden. Listen «titles» som tas inn forventes å bestå av alle h4-elementene som er i skjemaet. Dette fordi tittelen på grupperingene i skjemaet er h4-elementer. Elementene blir pakket inn i et div-element med en «onClick»-funksjon. Dette for at bruker skal kunne trykke på både knappen og den beskrivende teksten. I knapp-elementet sjekkes det om activeIndex og index har samme verdi. Dersom de har det vil knappen få andre CSS-verdier som markerer dette.

#### 10.1.3.1.2. scrollToTitle

```
const scrollToTitle = (index) => {
    document.querySelectorAll(selectors: "h4")[index]
        .scrollIntoView(arg: {behavior: "smooth", block: "center"});
};
```

Figur 44: scrollToTitle()-metoden bruker index til å finne riktig plassering.

Når en knapp eller tittel i sidebaren blir trykket på, sendes elementet sin indeks som argument inn i funksjonen «scrollToTitle». «scrollToTitle» har parameteren «index», som vil si et tall som representerer plasseringen til en tittel i dokumentet. Funksjonen bruker «document» som gir tilgang til DOM-treet i applikasjonen, sammen med metodene «querySelectorAll» og «scrollIntoView». Først finner den alle elementene med merkenavnet «h4» også bruker den indeksen til å finne det bestemte elementet i listen. Metoden "scrollIntoView" brukes for å hente elementet inn til sentrum av brukerens visningsområde.

#### 10.1.3.1.3. MutationObserver

```
const observer = (props.isModalOpen === false || props.postError === "") &&
  new MutationObserver( callback: _ => {
    const tempTitles = [...document.getElementsByTagName( qualifiedName: "h4")];
    setTitles(tempTitles);
    observer.disconnect();
  });

(props.isModalOpen === false || props.postError === "") &&
Array.from(rootElement).forEach(target => {
  observer.observe(target, {childList: true})
});
```

Figur 45: MutationObserver hører på endringer i root-elementet "sv\_p\_root" og oppdaterer titler deretter.

Sidebaren er dynamisk og skal fungere med hvilken som helst type skjema. Noen av skjemaene skjuler noen titler basert på brukerinput og derfor må sidebaren kunne lytte og endre seg basert på endringer som skjer DOM-en. For å holde sidebaren oppdatert på endringer i DOM-en bruker vi grensesnittet MutationObserver. MutationObserver-konstruktør har en parameter.

Parameteren er en funksjon også kjent som en «callback»-funksjon fordi den vil kjøre etter at alt annet i hovedfunksjonen har kjørt. MutationObserver har en «observer»-metode som tar to argumenter, «targetNode» og «config». Den første tillater oss å spesifisere hvilken node vi ønsker å observere og den andre lar oss sette regler som at vi ønsker å observere alle barna til noden. MutationObserver har også en metode for å stanse observasjoner som heter «disconnect». I vårt tilfelle hører vi etter endringer i «sv\_p\_root»-noden og dets barn. Får elementet lagt til eller fjernet et barn, betyr det at titlene må oppdateres. Dette gjøres i callback-funksjonen. Her blir en variabel tilordnet en liste med de nye titlene som hentes ved hjelp av DOM-metoden «getElementsByTagName» og tilstanden «titles» endres til å innholde denne listen ved hjelp av tilstandsfunksjonen «setTitles». Når dette er gjort avsluttes observasjonen med «observer.disconnect()».

#### 10.1.3.1.4. setTitleSidebar

```
const getTitle = useCallback( callback: (element) => {
    const parent = element.parentElement;
    if (parent && parent.className === "sv_p_container") {
        const titleFromElement = parent.children[0].children[0];
        for (const title in titles) {
            if (titleFromElement === titles[title]) {
                setActiveIndex(parseInt(title));
                return;
            }
        }
    } else if (parent) {
        getTitle(parent);
    }
}, [deps: [titles]]);

const setTitleSidebar = useCallback( callback: () => {
    const posX = window.innerWidth / 2;
    const posY = window.innerHeight / 2;
    const elem = document.elementFromPoint(posX, posY);
    getTitle(elem);
}, [deps: [getTitle]]);

useEffect( effect: () => {
    document.addEventListener( type: "scroll", setTitleSidebar);
    return() => document.removeEventListener( type: "scroll", setTitleSidebar);
}, [deps: [setTitleSidebar]]);
```

Figur 46: Funksjonen `setTitleSidebar()` sammen med `getTitle()` og eventListeners oppdaterer hvilken overskrift brukeren er på til enhver tid.

Når du skroller i et skjema så må sidebaren oppdateres. For å gjøre dette har vi lagt til en «eventListener» som kjører funksjonen «`setTitleSidebar`» når den oppdager at en bruker skroller. «`setTitleSidebar`» finner midtpunktet til skjermen og deretter sender den elementet som er på dette midtpunktet videre til funksjonen «`getTitle`». «`getTitle`» funksjonen vil rekursivt gå gjennom foreldrelementene til det innsendte elementet. Når funksjonen finner riktig klasse, «`sv_p_container`», vil tittelen vi skal skrolle til alltid være barnebarnet til denne klassen. Derfra

er det bare å løpe gjennom listen av titler og sette hvilken tittel som er i fokus. Alt dette betyr at når du skroller på siden så vil elementet i midten av skjermen diktere hvilken tittel som blir markert i sidebaren.

### 10.1.3.2. Footer

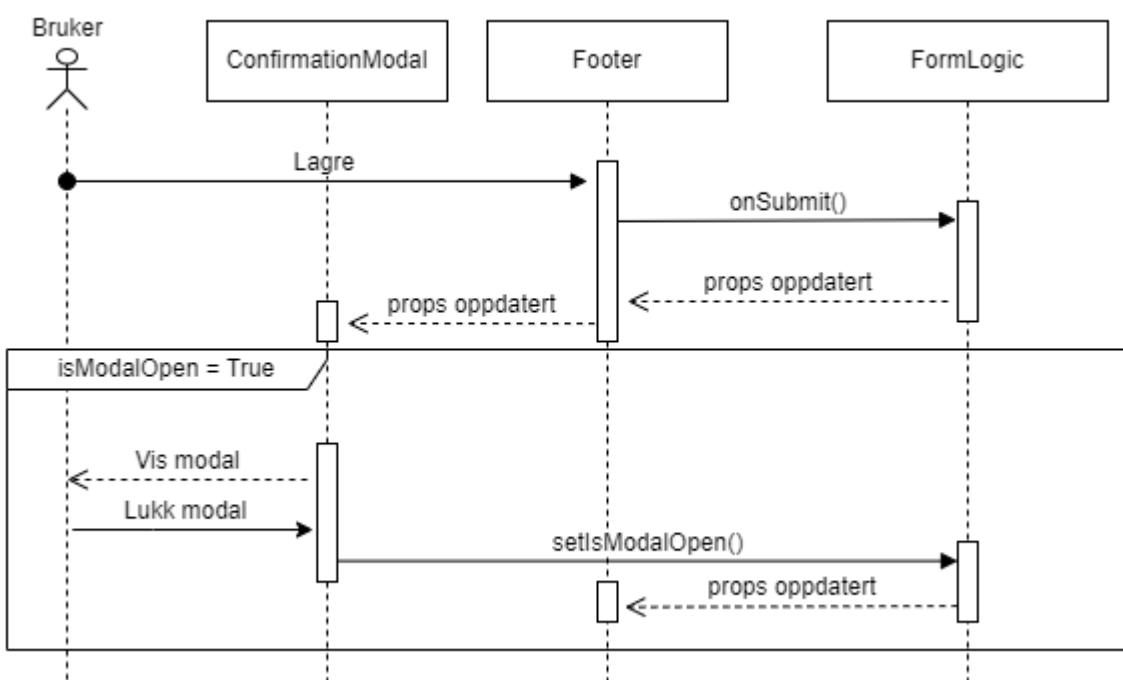
```
return (
  <> <footer
    className="surveyFooter">
      <p className={"asterixMeaning"} aria-label={"Asteriks er obligatorisk"}>* Obligatorisk</p>
      <div className={"actionBtnContainer" }>
        <button
          aria-label={"Lagre"}
          type={"submit"}
          className={"bttm submitBtn"}
          onClick={() => {props.onSubmit() }}
        ...>
        <button
          aria-label={"Lagre utkast"}
          type={"submit"}
          className={"bttm submitBtn"}
        ...>
        <button
          aria-label={"Avbryt"}
          type={"button"}
          className={"bttm cancelBtn"}
          onClick={() => navigate("/")}
        ...>
      </div>
    </footer>
    <ConfirmationModal isModalOpen={props.isModalOpen}
      setIsModalOpen={props.setIsModalOpen}
      postError={props.postError}/> </>
)
```

Figur 47: Footer-komponenten returnerer handlingsknapper og ConfirmationModal-komponentet.

Footer-komponenten returnerer et <footer>-element og <ConfirmationModal>-komponentet. <footer>-elementet inneholder de faktiske elementene som Footer-komponenten viser. Disse elementene er en paragraf som forteller brukeren hva asterisk-tegnet står for og tre handlingsknapper. Første handlingsknapp er Lagre-knappen som kaller på «onSubmit()»-funksjonen til morkomponenten Formlogic. Det vil si at når en bruker trykker på «Lagre» i

footeren, vil morkomponenten få beskjed om at den må kjøre «onSubmit()»-funksjonen som håndterer logikken med innlasting av skjema og eventuell skrolling til felter som er glemt å utfylle. «Lagre utkast»-knappen har ikke blitt implementert da det ikke var en del av oppgaven, men er blitt lagt inn i koden for å bli ferdigutviklet i sommer. Dersom en bruker trykker på «Avbryt»-knappen vil brukeren bli sendt til forsiden ved hjelp av «navigate()»-metoden.

Når en bruker trykker på «Lagre»-knappen vil det trigge «onSubmit()»-funksjonen i FormLogic komponenten som vil endre tilstandene til isModalOpen og postError også sende de oppdaterte tilstandene tilbake igjen. Footer-komponenten sender disse propsene så videre til ConfirmationModal-komponenten. Diagrammet nedenfor gir en illustrasjon av hvordan tilstandene oppdateres mellom komponentene.



Figur 48: Diagrammet viser hvordan props blir oppdatert for å vise ConfirmationModal.

#### 10.1.3.3. ConfirmationModal

ConfirmationModal er en modal fra Reactstrap, et CSS-rammeverk for React. Den inneholder en header, body og footer. I headeren blir tittelen for beskjeden satt, mens bodyen er fylt med beskjeder basert på hva «postError» inneholder av feilmeldingskode. For eksempel vil en bruker

få beskjeden «Det ser ut til at noe gikk galt på serveren» dersom props-en «postError» inneholder feilmeldingskoden 500.

```
<ModalFooter className={"modalFooter"} >
  {props.postError !== "" ?
    <>
      <button
        className={"btn cancelBtn"}
        onClick={() => navigate("/")}>
        Til dashboard
      </button>
      <button
        className={"btn submitBtn"}
        onClick={() => props.setIsModalOpen(false)}>
        Til skjema
      </button>
    </>
    :
    <button
      className={"btn submitBtn"}
      onClick={() => navigate("/")}>
      Ok
    </button>
  }
</ModalFooter>
```

Figur 49: ConfirmationModal returnerer handlingsknapper basert på om postError er tom eller ikke.

I footeren får brukeren opp forskjellige knapper med ulike handlinger basert på om det er en feilmelding i «postError». Når «postError» er tom vil det si at innsending av skjema var vellykket og dermed vil brukeren bli sendt til dashboardet med knappen «Ok». Dersom «postError» inneholder en feilmelding, vil brukeren få muligheten til å gå tilbake til skjema igjen med «Til skjema»-knappen. «Til skjema»-knappen sender gjennom «props.setIsModalOpen» et kall til Footer-komponenten som igjen sender den videre til FormLogic for å oppdatere tilstanden «isModalOpen» med funksjonen «setIsModalOpen». Når den settes til «False», vil modalen skjules.

## 10.2. Skjemaenes oppbygning

Skjemaer i SurveyJS er bygd opp av et JSON-objekt. Ut ifra denne JSON-en vil SurveyJS generere et skjema. Hovedsakelig består denne JSON-en av tre nivåer. Første nivå er hele skjemaet med dets logikk, andre nivå er skjemaets sider og tredje nivå er elementene i sidene.

```
const SurveyJsonUtredning = {
    name: "KliniskProstataUtredning",
    locale: "no",
    title: "Melding etter avsluttet utredning",
    pages: [
        {name: "pasientInstitusjon"}, ...
        {name: "utredning"}, ...
        {name: "sykehistorie"}, ...
        {name: "diagnotiskeUS"}, ...
        {name: "sykdomsutbredelse"}, ...
        {name: "kliniskTNM"}, ...
        {name: "laboratorium"}, ...
        {name: "oppfolging"}, ...
        {name: "kommentarfelt"}, ...
        {name: "melder"} ...
    ],
    triggers: [...],
    calculatedValues: [...],
    showQuestionNumbers: "off",
    clearInvisibleValues: "onHidden",
    checkErrorsMode: "onValueChanged",
    questionsOnPageMode: "singlePage"
};
```

Figur 50: Utsnitt av oppbygningen til skjemaet «KliniskProstataUtredning»

I figur 53 ser vi oppbygningen og logikken til skjemaet «KliniskProstataUtredning» på et lavnivå. Som vi kan se så består skjemaet av: Sider, triggers, kalkulerte verdier og annen metadata. Sidene er hvordan skjemaet blir delt opp. Triggers er funksjoner som utløses når en endring skjer i skjemaet. Kalkulerte verdier er verdier som kan endre seg ut ifra endringer i skjemaet. Metadataen er informasjon om skjemaet og hvordan skjemaet skal oppføre seg.

### 10.2.1. Siders oppbygning

Hver side inneholder en liste over elementer på siden og metadata om siden. Denne metadataen er navn, tittel og eventuelle betingelser om siden skal vises eller ikke. Et element inneholder informasjon om hva det er, hvordan det vises og annen logikk. Det er fire forskjellige typer elementer vi har brukt: Tekstfelter, nedtrekkmenyer, radioknapper og avkryssingsbokser.

```
{
  name: "pasientInstitusjon",
  elements: [
    {name: "fodselsnummerHF"}, ...
    {name: "fodselnummerUtland"}, ...
    {name: "navnPasient"}, ...
    {name: "sykehuskode"}, ...
    {name: "sykehusnavnHFSpesifiser"}, ...
    {name: "avdelingsnavn"}, ...
    {name: "avdelingsnavnHFSpesifiser"}
  ],
  title: {
    no: "Pasient/behandlingsinstitusjon"
  }
},
```

Figur 51: Utsnitt av oppbygningen til siden "pasientInstitusjon"

## 10.2.2. Elementers oppbygning

### 10.2.2.1. Inputfelt

```
{  
    type: "text",  
    name: "fodselsnummerHF",  
    maxWidth: "400px",  
    title: {  
        no: "Fødselsnummer"  
    },  
    enableIf: "{fodselsnummerUtland} empty",  
   isRequired: true,  
    requiredIf: "{fodselsnummerUtland} empty",  
    requiredErrorText: {  
        no: "Skriv inn gyldig fødselsnummer"  
    },  
    validators: [  
        {  
            type: "regex",  
            text: "Skriv inn gyldig fødselsnummer ",  
            regex: "^(0[1-9]|1[0-2]\\d|3[0-1])(0[1-9]|1[0-2])\\d{7}$"  
        }  
    ]  
},
```

Figur 52: Viser oppbygningen av inputfeltet «fodselsnummerHF»

Figur 55 viser hvordan elementet “fodselsnummerHF” er bygd opp. Først kommer typen på elementet, her “text”. Det betyr at dette feltet er et inputfelt hvor en kan skrive inn tekst. Videre kommer navnet på elementet, dette fungerer som en ID for elementet. Neste er tittelen til inputfeltet og deretter har vi logikken bak om feltet er obligatorisk eller ikke. Her har vi to attributter som dikterer dette “isRequired” og “requiredIf”. “isRequired” sier om feltet er obligatorisk, om dette attributtet ikke er spesifisert så er det som standard “false”. “requiredIf” fungerer slik at hvis uttrykket i attributtet returnerer “true” så blir feltet obligatorisk, og hvis det

returnerer "false" blir feltet ikke obligatorisk. Attributtet "enableIf" dikterer om feltet kan fylles ut eller ikke. Dette er fordi feltet ikke skal ha muligheten til å bli fylt ut, men likevel vises. Så har vi teksten som vises dersom feltet ikke blir fylt ut. Sist er validator attributtet, her spesifiserer hva slags type validator det er, feil tekst og selve logikken bak validatoren.

I figur 56 ser vi fire tilstander som viser hvordan feltet «Fødselsnummer» er i skjemaet. Den første tilstanden viser hvordan feltet ser ut til vanlig. Tilstand nummer to er med gyldig input. Tilstand nummer tre viser hvordan det vil se ut med ugyldig input og tilstand nummer fire viser hvordan det blir seende ut når en huker av boksen «Ikke norsk personnummer». Her kan du se hvordan feltet blir gjort grått og stjerna blir fjernet.

The figure consists of four screenshots of a form field for 'Fødselsnummer'.  
1. The first screenshot shows an empty input field with a placeholder 'Fødselsnummer \*'. To the right is a checkbox labeled 'Ikke norsk personnummer' with an empty square.  
2. The second screenshot shows the input field containing the valid number '01010012345'. The checkbox to its right is also empty.  
3. The third screenshot shows the input field containing the invalid number '999999999999'. The entire input field is highlighted with a red box, and the placeholder 'Skriv inn gyldig fødselsnummer' is displayed above it. The checkbox to its right is empty.  
4. The fourth screenshot shows the input field empty again. The checkbox to its right has a checked mark inside a blue square, indicating that the entered value is not a valid Norwegian ID number.

Figur 53: Viser fire tilstander til feltet "Fødselsnummer".

#### 10.2.2.2. Nedtrekkmenyer

```
{  
    type: "dropdown",  
    name: "kliniskVurdertAv",  
    visible: false,  
    visibleIf: "{funnUtredning} = 1",  
    title: {  
        no: "Hvem har gjort totalvurderingen av klinisk T?"  
    },  
   isRequired: true,  
    choices: [  
        {  
            value: "1",  
            text: {  
                no: "MDT (tverrfaglig møte)"  
            }  
        },  
        {value: "2"},  
        {value: "3"}  
    ],  
    optionsCaption: {  
        no: "Velg..."  
    },  
    renderAs: "dropdown-react"  
},
```

Figur 54: Viser oppbygningen av nedtrekkmenyen "kliniskVudertAv"

Figur 57 viser hvordan elementet «kliniskVudertAv» er bygd opp. Oppbygningen er lik et inputfelt, men det er et par forskjeller. Det som er likt er standard informasjon som navn, synligheten til feltet, tittel og om feltet er obligatorisk. Forskjellighetene starter med type, her er det «dropdown» som sier at feltet er en nedtrekkmeny. Videre så har man valgene i nedtrekkmenyen, som det bare er tre av i dette eksemplet. Valgene har lagringsverdi og

visningsverdi. Nest sist er teksten som vises om det ikke er valgt noe ennå og til sist står det at feltet skal vises som «dropdown-react».

«Dropdown-react» er en egendefinert klasse for nedtrekkmenyer i SurveyJS. Denne klassen utvider nedtrekkmeny-klassen til SurveyJS. Dette har vi gjort fordi Kreftregisteret ønsket å ha søkbare nedtrekkmenyer. I figur 58 så kan du se hvordan nedtrekkmenyen vil bli seende ut. I dette eksempelet vil det ikke være så nødvendig å søke siden det bare er tre elementer, men det er likevel greit. For å søke så skriver du mens feltet er i fokus og feltet vil automatisk filtrere etter det som skrives.

Hvem har gjort totalvurderingen av klinisk T? \*

The image shows a dropdown menu with the following options:

- Velg...
- MDT (tverrfaglig møte)
- Utredende lege
- Melder (ikke utredende lege)

Figur 55: Viser hvordan feltet «kliniskVurdertAv» vil bli seende ut

### 10.2.2.3. Radiokapper

```
{  
  type: "radiogroup",  
  name: "kreftsymptomer",  
  title: {  
    no: "Var kreftsymptomer (smerter, anemi eller annet) en del av årsak til utredningen?"  
  },  
 isRequired: true,  
  choices: [  
    {  
      value: "1",  
      text: {  
        no: "Ja"  
      }  
    },  
    {value: "0"},  
    {value: "99"}  
  ],  
  colCount: 3  
},
```

Figur 56: Viser oppbygningen til radioknappgruppen «kreftsymptomer»

Figur 59 viser hvordan elementet «kreftsymptomer» er bygd opp. Selve oppbygningen er helt lik utenom at feltet er av typen «radiogroup» og det er en «colCount». «colCount» antall kolonner på en rad, altså hvor mange knapper det kan være på en rad. Figur 60 viser hvordan dette blir seende ut.

Var kreftsymptomer (smerter, anemi eller annet) en del av årsak til utredningen? \*

Ja       Nei       Ukjent

Figur 57: Viser hvordan feltet "kreftsymptomer" vil bli seende ut

#### 10.2.2.4. Avkryssingsbokser

Figur 61 viser hvordan elementet «vevsproverUS» er bygd opp. Oppbygningen er helt lik som om det skulle vært radioknapper, men er av typen «checkbox» som betyr at det skal være avkryssingsbokser. I figur 62 kan man se hvordan koden ser ut i brukergrensesnittet.

```
{  
    type: "checkbox",  
    name: "vevsproverUS",  
    visible: false,  
    visibleIf: "{vevsprover} = 1",  
    title: {  
        no: "Undersøkelser"  
    },  
   isRequired: true,  
    choices: [  
        {  
            value: "biopsiVevsprover",  
            text: {  
                no: "Biopsi"  
            }  
        },  
        {value: "turpvevsprover"},  
        {value: "annetVevsprover"}  
    ],  
    colCount: 3  
},
```

Figur 59: Viser oppbygningen til avkryssingsboksgruppa "vevsproverUS"

The screenshot shows a light gray input field with a white border. Inside, the title "Undersøkelser \*" is displayed in a dark blue font. Below the title is a list of three items, each preceded by an empty square checkbox. The items are: "Biopsi", "TUR-P", and "Annet".

Figur 58: Viser hvordan avkryssingsboksgruppa blir seende ut

### 10.2.3. Skjemalogikk

For å få feltet "fodselsnummerHF", i figur 63 til å bli tomt når vi trykker på avkryssingsboksen "Ikke norsk personnummer", så må vi legge til en funksjon under triggers. Denne er av typen "setValue", som betyr at den setter verdien i et felt til noe. Videre kommer logikken bak funksjonen, her har vi at hvis «fodselsnummerUtland» ikke er tom, altså fylt inn, så triggges denne. Deretter spesifiseres hvilket felt som vi skal endre verdien til. Vanligvis så vil vi også ha attributtet "setValue" som sier hva vi skal sette inn i feltet, men siden vi skal tømme feltet "fodselsnummerHF", eller sette det til ingenting, så trenger vi ikke det.

```
{  
    type: "setvalue",  
    expression: "{fodselsnummerUtland} notempty",  
    setToName: "fodselsnummerHF"  
},
```

Figur 60: Viser strukturen til et "trigger"-felt.

I tillegg til triggere så har vi kalkulerte verdier. Disse verdiene har to hovedegenskaper: Navn og uttrykk. Basert på uttrykket så beregnes verdien slik at den kan brukes av skjemaet. I figur 64 så kan vi se en eksempel på hvordan det brukes. Det feltet som vises i figuren, «sykdomsutbredelseTitle», beregner tittelen til siden som heter «sykdomsutbredelse». Grunnen til å det gjøres slik, er at felter som håndterer tekst, som «title»-feltet, ikke kan ha uttrykk slik det er i «sykdomsutbredelseTitle».

```
{  
    name: "sykdomsutbredelseTitle",  
    expression: "iif({funnUtredning} = 1, " +  
        "'Sykdomsutbredelse etter ferdig primærutredning og før primærbehandling', " +  
        "'Sykdomsutbredelse ved utredning av metastase(r)'"  
}
```

Figur 61: Viser feltet «sykdomsutbredelseTitle», som er et «calculatedValues»-felt

```
{  
    name: "sykdomsutbredelse",  
    elements: [...],  
    title: "{sykdomsutbredelseTitle}"  
},
```

Figur 62: Viser hvordan siden «sykdomsutbredelse» får tittelen sin.

## 10.3. Datahåndtering i skjema

### 10.3.1. Innlasting av data

For å ha muligheten til å endre på skjemaer så må skjemaet ha muligheten til å laste inn data.

Når vi starter opp et skjema så sjekker det URL-en. URL-en til skjemaet er på formatet “{domene}/{skjematype}/{ID}”. Her er domenet for eksempel “localhost:3000”, som vi brukte under utvikling. Skjematype er hvilken type skjema det er og ID er ID-en til skjemaet vi skal hente data fra. Hvis ID-en er 0 betyr det at det er et tomt skjema. Nå som skjemaet har type og ID, kan det hente data fra API-et. Formatet for dette API-kallet er “{domene}/api/v1/meldinger/{ID}”. For eksempel under utviklingsperioden brukte vi “http://localhost:8080/api/v1/meldinger/3”, når vi skulle hente data for skjemaet med ID 3.

### 10.3.2. Innsetting av data

Når dataen er hentet bruker vi SurveyJS sin innebygde funksjon for å sette inn data i skjemaet. Men for at det skal være mulig må vi formatere JSON-en på en slik måte at den er gyldig. SurveyJS lagrer data som en flat JSON fil, som betyr at JSON-en ikke er nøstet. At et JSON fil er nøstet betyr at dataen i fila er organisert i lag. Dette kommer av at meldingen vi konverterte hadde denne strukturen. For å flate JSON fila, så lager vi først et nytt array for den flate JSON versjonen. Deretter løper vi rekursivt gjennom JSON fila. Her sjekker vi om et element er et objekt eller ikke. Hvis elementet er et objekt, betyr det at elementet inneholder data og vi vil løpe gjennom dette elementet også. Hvis elementet ikke er et objekt så vil elementet bli lagt til i det nye array-et. Når vi har løpt gjennom hele JSON fila vil vi ha en flat versjon av dataen.

```

const flatten = (JSONdata) => {
    for (const key in JSONdata) {
        if (typeof JSONdata[key] !== "object") {
            for (const checkboxGroup in checkboxes) {
                for (const checkbox in checkboxes[checkboxGroup]) {
                    if (key === checkboxes[checkboxGroup][checkbox]) {
                        addCheckboxToFlattenedJSON(JSONdata, checkboxGroup);
                        return
                    }
                }
            }
            flattenedJSON[key] = JSONdata[key];
        } else {
            flatten(JSONdata[key]);
        }
    }
};

;

```

Figur 64: Bilde av funksjonen som flater JSON

Den eneste problematikken ved innlasting av data er avkryssingsbokser. Hvis en avkryssningsboks er alene, som ”Ikke norsk personnummer”, så blir den lagret som et felt med en verdi i SurveyJS, som er likt alle andre felter i SurveyJS. Men hvis det er flere avkryssingsbokser sammen så blir de lagret som en liste som inneholder avkryssningsboksene som er krysset av. For eksempel i figur 67 så er to av tre bokser avkrysset. I eksempelet så vil SurveyJS lagre dette som en liste med de to avkryssete boksene. Dette et problematisk i forhold til hvordan dataen er lagret i meldingene, hvor hver avkryssningsboks har én lagringsverdi.

**Undersøkelser \***

|   |
|---|
| <input type="checkbox"/> Biopsi           |
| <input checked="" type="checkbox"/> TUR-P |
| <input checked="" type="checkbox"/> Annet |

Figur 63: Bilde av mulig avkryssing av en gruppe med avkryssingsbokser

For å håndtere avkryssingsbokser starter vi med å finne alle avkryssingsboksene. Deretter må vi skille på om avkryssingsboksene er i en gruppe eller om de er single. Dersom avkryssingsboksene er i en gruppe legges avkryssingsboksgruppa inn i en liste. Når dette gjøres vil også verdien til hver avkryssingsboks, fra den innsendte meldingen, også lagres. Nå som vi har en liste med avkryssingsboksgrupper, så kan vi bruke denne når vi setter inn og lagrer data. Ved innsetting av data vil vi nå kunne sjekke om en avkryssingsboks er krysset av eller ikke. Lista over avkryssingsboksgruppene blir også bruk ved lagring av data.

```
const findCheckboxes = (JSONdata) => {
    for (const key in JSONdata) {
        if (JSONdata[key].type === "checkbox" && JSONdata[key].choices.length > 1) {
            let checkboxChoices = [JSONdata[key].name];
            for (let i = 0; i < JSONdata[key].choices.length; i++) {
                checkboxChoices.push(JSONdata[key].choices[i].value);
            }
            checkboxes.push(checkboxChoices);
        } else if (typeof (JSONdata[key]) === "object") {
            findCheckboxes(JSONdata[key]);
        }
    }
};
```

Figur 65: Bilde av funksjonen som finner avkryssingsbokser for skjemaet

Videre ved innsetting av data må vi håndtere sykehuskodene. Grunnen er at i dataen vi blir tilsendt så er sykehusene delt inn i regioner. Dette er noe som Kreftregisteret ikke ville ha lenger, men heller ha én nedtrekkliste for alle sykehusene. Én nedtrekkliste for alle sykehusene har ikke blitt implementert tidligere fordi det var ikke mulig å lage en søkbar nedtrekkliste i InfoPath. For å håndtere sykehuskodene finner vi hvilken region som har en sykehuskode og setter den til feltet «sykehuskode». Deretter tømmer vi feltet for hvilken region det er og feltet som hadde koden slik at unødvendig data blir fjernet.

### 10.3.3. Endring og lagring av data

Når en endring blir gjort i skjemaet vil endringen bli lagret med engang. Når endringen skjer, så vil skjemaet registrere hvilket felt som blir endret eller fylt ut. Vi kan da finne det samme feltet i JSON-en vi har og sette inn den nye dataen. Hvis det er en ugyldig verdi som blir fylt inn så vil skjemaet vise en feilmelding, og det vil ikke være mulig å fullføre. Grunnen til at dette gjøres er for å opprettholde den originale datastrukturen, slik at konvertering tilbake til XML vil være lett og at XML-en vil kunne brukes av resten av Kreftregisteret sine systemer (dette gjør det også mulig for midlertidig lagring).

Når man er ferdig med å fylle ut skjemaet og vil lagre, kan man trykke på “Lagre”-knappen nederst på skjermen. Skjemaet vil da sjekke om det er noen felter som mangler, eller felter som har feil. Hvis det er en mangel eller feil vil skjemaet ta deg til første feilen i skjemaet. Ved lagring vil også skjemaet erstatte verdier som er “undefined”, fordi dette skaper problemer ved konvertering tilbake til XML. Hvis det er ingen problemer, vil JSON-dataen postes på API-et slik at serveren kan hente det.

## 10.4. Kommunikasjon med ELVIS

ELVIS er metadatabasen til Kreftregisteret. Databasen inneholder informasjon om variabler og verdier som blir brukt i Kreftregisteret sine systemer. Databasen er tilgengelig for alle brukere via internett og har et API som man kan bruke til å hente og validere data (Kreftregisteret, u.å.).

### 10.4.1. Hente verdier fra ELVIS

Kreftregisteret ønske seg at vi skulle hente verdier fra API-et til metadatabasen, slik at det lett skulle være mulig å legge til og oppdatere verdier som er i skjemaet. Hovedsakelig skulle dette brukes når vi har nedtrekkmenyer. For å gjøre dette brukte vi SurveyJS sin innebygde funksjon. Dette er en funksjon som fungerer slik at når du har «type: dropdown», kan du velge å ha «choicesByUrl». «choicesByUrl» har tre attributter: URL, «valueName» og «titleName». URL henviser til hvor du skal hente fra. «valueName» er verdien du skal hentet fra API-et og «titleName» er teksten som skal vises i nedtrekkmenyen. Se figur 69 for eksempel.

```
{  
    type: "dropdown",  
    name: "labnavnHF",  
    title: {  
        no: "Laboratorium"  
    },  
    enableIf: "{labnavnHFIkkeRelevant} empty",  
   .isRequired: true,  
    requiredIf: "{labnavnHFIkkeRelevant} empty",  
    choicesOrder: "asc",  
    choicesByUrl: {  
        url: "https://metadata.kreftregisteret.no/rest/v1/metadata/values/m_labAngittAvKliniker",  
        valueName: "value",  
        titleName: "description"  
    },  
    optionsCaption: {  
        no: "Velg..."  
    },  
    renderAs: "dropdown-react"  
},
```

Figur 66: Viser hvordan «choicesByUrl» blir implementert i en nedtrekkmeny

#### 10.4.2. Validere verdier mot ELVIS

Kreftregisteret ønsket også at det skulle være mulig at verdier valideres mot ELVIS. Som ved henting av data fra API-et, så har vi bare implementert validering for et felt, altså «labnavnHF». Et slik validerings kall krever hvilken variabel du vil sjekke og verdien dens. I figur 70 kan du se at vi validerer «labnavnHF» feltet. Her kan du se hvordan det foregår: Du sender et kall til API-et med variabel og verdi, og deretter får du enten et «OK» eller «FAIL». «OK» betyr at verdien var gyldig, «FAIL» betyr ugyldig.

```
const checkLabValueURL = "https://metadata.kreftregisteret.no/rest/v1/variables/" +
    "validate/:variable/m_labAngittAvKliniker?value_codes[ ]=" + data.laboratorium.labnavnHF;
fetch(checkLabValueURL) Promise<Response>
    .then(response => response.json()) Promise<any>
    .then(StatusData => {
        if (StatusData.status === "OK") {...}
        else {
            setPostError(StatusData.status);
        }
    }) Promise<any>
    .catch(error => {
        setPostError(error.toString());
    });
setIsModalOpen(true);
```

#### 10.4.3. Problematikk ved ELVIS

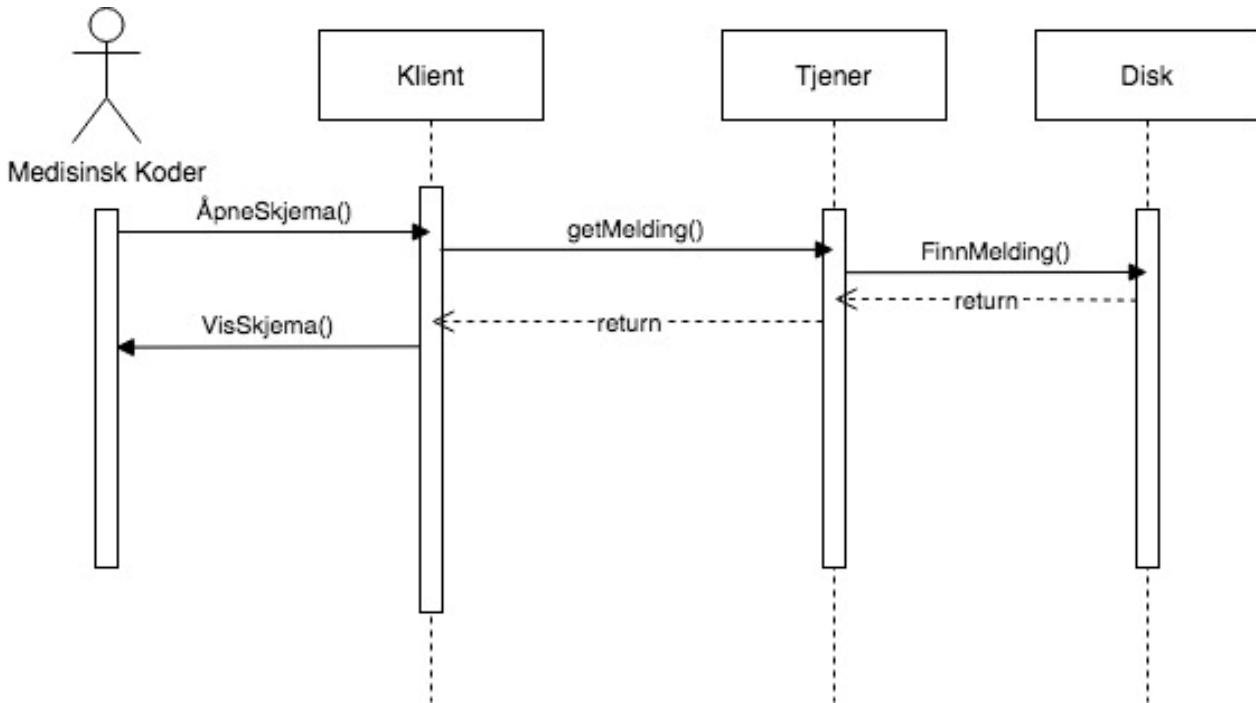
Ettersom vi jobbet med å implementere kall mot ELVIS merket vi at databasen ikke var laget for vår problemstilling. For det første så er ikke navnene til variabler de samme. I figur 69 og 70 så har vi brukt feltet «labnavnHF», som det heter i skjemaet og meldingene. Men i databasen har hver variabel et eget navn som for eksempel «labnavnHF» har det tilsvarende navnet «m\_labAngittAvKlinker». For da å implementere kall for hver variabel måtte vi laget en type ordbok som oversatte hvert variabelnavn fra skjemaet til variabelnavnet i databasen. Dette var uforutsett arbeid og vi, sammen med Kreftregisteret, valgte å implementere kall for bare en variabel.

I figur 69 kan vi se nedtrekkmenyen «labnavnHF». Dette er det enste stedet vi henter data fra API-et til metadatabasen. Grunnen er at API-et ikke hadde noen måte å sende spørrenger og det var derfor ikke mulig å filtrere verdiene fra API-et. I tillegg så har SurveyJS ikke noen innebygd måte å filtrere valg når man henter fra et API. Vi valgte derfor å bare hente fra API-et et sted slik at det skulle være et eksempel på hvordan det gjøres ved eventuell oppdatering på API-et som tillater spørrenger. Vi valgte også feltet «labnavnHF» siden feltet ikke krevde spørrenger for å ha gyldige verdier.

Til å starte med så var ikke kommunikasjonen med ELVIS en del av kravspesifikasjonen. Men under utvikling, da det så ut til at vi hadde tid, kom det ønsket om implementering av kommunikasjonen med vårt system. På det tidspunktet visste ikke partene hva som skulle til for å lage en god løsning. Da det ble klart at en god løsning ikke ville være mulig, grunnet tid og mangler, valgte vi bare å implementere eksempler på hvordan det kan gjøres for en fremtidig utvidelse av ELVIS.

## 11. Backend

Backend-delen av prosjektet er en Spring Boot webapplikasjon, som blant annet bruker bibliotekene JAXB og Jackson. Hensikten med tjenerdelen er å ha et API-endepunkt som klientsiden kan sende data til og hente data fra. Løsningen er utviklet med designmønsteret MVC i mente, som sammen med programmeringsprinsipp som lav kobling og høy kohesjon kan hjelpe mot målet om å lage et robust, skalerbart og modul-basert system. I dette kapittelet presenteres hver del av løsningen for tjenersiden i detalj.



Figur 68: Sekvensdiagram, "Medisinsk koder åpner en kreftmelding"

### 11.1. Controller

Systemet inneholder én Controller i klassen MeldingController. Kontrolleren benytter den sammensatte annotasjonen @RestController. @RestController er sammensatt av @Controller

og @ResponseBody. @Controller lar dermed Spring vite at klassen er en komponent av typen Controller, og @ResponseBody forteller Spring at samtlige metoder benytter annotasjon @ResponseBody som standard (Spring, u.å.). Controllerens hensikt er å motta input fra frontend og returnere forespurt data tilbake gjennom å kalle på metoder i service-komponenten MeldingManager. Den skal også kunne håndtere avvik.

```
● ● ●  
 @GetMapping(path = "/api/v1/meldinger")  
 public ArrayList<MeldingDTO> getAllMeldinger() throws IOException {  
     System.out.println(meldingManager.getMeldingListDTO());  
     return meldingManager.getMeldingListDTO();  
 }
```

Figur 69: Endepunktet som henter samling meldinger

Figur 72 viser til metoden getAllMeldinger(), som returnerer alle tilgjengelige meldinger. Metoden benytter den sammensatte annotasjonen @GetMapping, som er en forkortelse av @RequestMapping (method = RequestMethod.GET). Dette gjør det mulig for frontend å sende en GET-forespørsel til "/api/v1/meldinger", og få en liste med MeldingDTO tilbake. MeldingDTO er et «Data Transfer Object», som er en modifisert versjon av modellklassen Melding som inneholder metadata som ID og dato sist endret. Denne listen inneholder ikke hele skjemaet, og er utviklet på denne måten for å redusere antall nødvendige API-kall. Metoden kaller på meldingManager.getMeldingListDTO(), som viser eksplisitt til at man får tilbake en liste med meldinger. Dette gjør det mulig for frontend å sortere meldingene etter dato.

```
● ● ●  
 @GetMapping(path = "/api/v1/meldinger/{id}")  
 public Melding getMelding(@PathVariable long id) throws IOException {  
     return meldingManager.findMeldingById(id);  
 }
```

Figur 70: Kode for endepunktet for å hente ut en spesifikk melding

Annotasjonen @GetMapping viser til at metoden responderer på GET-forespørslar til "/api/v1/meldinger/{id}". Ved å benytte @PathVariable i metodens parameter, vil verdien til variabelen id automatisk bli hentet ut av den forespurte URI. Et eksempel er å sende en GET-forespørsel til "/api/v1/meldinger/1". Da vil id bli tilordnet verdien «1». Metoden kaller deretter på meldingManager.findMeldingById, som mottar verdien til variabelen id som metodens argument. «findMeldingById» søker om det finnes en lik id som den forespurte, og vil deretter returnere riktig Melding hvis den eksisterer. Hvis den forespurte meldingen ikke finnes, vil metoden returnere null.



```
@PostMapping(path = "/api/v1/meldinger", consumes = "application/json")
public ResponseEntity<String> postMelding(@RequestBody Melding melding) throws
JAXBException, IOException, SAXException {
    meldingManager.writeMeldingToPath(melding); // Validation happens here
    return ResponseEntity.status(HttpStatus.OK).build();
}
```

Figur 71: Koden for endepunktet postMelding.

Metoden postMelding() benytter den sammensatte annotasjonen @PostMapping, som er en forkortelse for @RequestMapping(method = RequestMethod.POST). Dette gjør det mulig for controlleren å håndtere POST-forespørslar. @PostMapping sin parameter «consumes» er satt til «application/json». Dermed vil metoden bare kunne ta imot data i formatet JSON. «postMelding()» sitt parameter melding benytter seg av annotasjonen @RequestBody. Dette gjør det mulig å automatisk konvertere JSON til et objekt av klassen Melding ved bruk av Jackson, gjennom Jackson annotasjon i klassen Melding. Når objektet melding er generert, vil controlleren sende et kall til meldingManager.writeMeldingToPath. Meldingen vil bli validert, før den deretter skrives til filsystemet og legges til i listen over tilgjengelige meldinger. Hvis ikke valideringen kaster et avvik («exception» på engelsk), vil metoden returnere statuskode og status HTTP 200 OK.

```
● ● ●

@ExceptionHandler(value = {SAXException.class, JAXBException.class})
public ResponseEntity<String> handleSAX_JAXBException(HttpServletRequest req,
Exception ex) {
    String errorMessage = ErrorUtils.exceptionToJSON(ex);
    return new ResponseEntity<>(errorMessage, HttpStatus.BAD_REQUEST);
}

@ExceptionHandler(JacksonException.class)
public ResponseEntity<String> handleJacksonException(HttpServletRequest req,
Exception ex) {
    String errorMessage = ErrorUtils.exceptionToJSON(ex);
    return new ResponseEntity<>(errorMessage,
HttpStatus.INTERNAL_SERVER_ERROR);
}

@ExceptionHandler(IOException.class)
public ResponseEntity<String> handleIOException(HttpServletRequest req,
Exception ex) {
    String errorMessage = ErrorUtils.exceptionToJSON(ex);
    return new ResponseEntity<>(errorMessage,
HttpStatus.INTERNAL_SERVER_ERROR);
}

@ExceptionHandler(Exception.class)
public ResponseEntity<String> handleException(HttpServletRequest req,
Exception ex) {
    String errorMessage = ErrorUtils.exceptionToJSON(ex);
    return new ResponseEntity<>(errorMessage,
HttpStatus.INTERNAL_SERVER_ERROR);
}
```

Figur 72: Metodene i controller som håndterer avvik.

Ved å bruke Spring-rammeverket sin innebygde annotasjon @ExceptionHandler kan man ta full kontroll over hvordan avvik håndteres av systemet. Annotasjonen tar imot subklasser av klassen Exception som parameter. Hvis det blir kastet et ikke-håndtert avvik i systemet, vil tilsvarende metode kjøre for å håndtere avviket. Alle metodene vil returnere en JSON med en detaljert feilmelding, og korrekt HTTP-status. Man kunne ha slått sammen metodene «handleJacksonException()», «handleIOException()», men vi har vurdert det hen at det er mer leseelig med to metoder da avvikene ikke har noen direkte sammenheng.

## 11.2. MeldingManager

MeldingManager er en Spring Boot-komponent av typen Service. Hensikten med denne klassen er å ha et interface controller-klassen kan kalle på for å håndtere data. All logikk som berører POJO-en Melding ligger i denne klassen. Derfor kan denne klassen sammen med modellklassen Melding sees på som Model-delen av MVC-arkitekturen. Ved hjelp av «Dependency Injection» og annotasjonen @Autowired blir MeldingManager satt inn i controllerens konstruktør. Ved å opprette MeldingManager-objektet i konstruktøren blir objektet «immutable», som vil si at man ikke kan endre dets avhengigheter mens programmet kjører.

MeldingManager har en ArrayList med alle tilgjengelige meldinger, samt ansvar for å lese, skrive og søke gjennom filsystemet for meldinger i XML-format. MeldingManager kaller også på valideringsmetodene ved skriving av Melding til XML. Ved videre utvikling vil metoder i denne klassen ha databaselogikk i stedet for kall på File-interfacet i Java.

```
● ● ●

@Service
public class MeldingManager {
    private final ArrayList<Melding> meldingList = new ArrayList<>();
    public ArrayList<Melding> getMeldingList() {
        return meldingList;
    }

    private void updateMsgList(Melding melding) {
        //oppdater tidspunktendret
        // "2001-12-17T09:30:47Z"
        //er formatet
        String formattedDate = getDate();
        melding.setEndrettidspunkt(formattedDate);
        melding.setId(createNewID());
        meldingList.add(melding);
    }

    public void addMeldingerFromUtFolderToMeldingList(List<File> list) {
        try {
            list.forEach(file -> {
                if (file != null) {
                    Melding melding = convertFileToMelding(file);
                    System.out.println(melding);
                    if (melding != null) {
                        meldingList.add(melding);
                    }
                }
            });
        } catch (Exception e) {
            System.out.println("er det her vi feiler???");
            e.printStackTrace();
        }
    }
}
```

Figur 73: Metodene i klassen *MeldingManager* relatert til *meldingList*.

Objektet *meldingList* er en *ArrayList* med elementer fra klassen *Melding*. Ved oppstart av systemet vil konstruktøren i *MeldingController* kalle på *addMeldingerFromUtFolderToMeldingList*. Denne metoden gjør om XML-filer til objekter av klassen *Melding* og skriver disse til *meldingList*. Videre vil det bli skrevet til listen ved kall på metoden *updateMsgList()*. Denne kalles av metoden *writeMeldingToPath()*, som blir kjørt når

frontend sender en oppdatert melding gjennom en POST-forespørsel. Man kan tenke på meldingList som en in-memory database over innlastede meldinger.

```
private Long id = 1L;

//sikter på thread-safety
public synchronized Long createNewID() {
    return id++;
}
```

Figur 74: id og hvordan den øker i klassen MeldingManager

For hver innsetting i listen vil det genereres en ny id. Metoden createNewID er synchronized for å forhindre at flere tråder kjører metoden samtidig, og dermed generere forskjellig id på hver tråd. Dette er et forsøk på å gjøre koden trådsikker. Den nylig genererte id-en vil dermed bli gitt som argument i Melding-objektet sin metode setId(). Dette gjør at man kan søke gjennom listen ved bruk av id som nøkkel.

```
public Melding findMeldingById(Long idIn) {
    for(Melding melding: meldingList){
        if(Objects.equals(melding.getId(), idIn)){
            return melding;
        }
    }
    return null;
}
```

Figur 75: Metoden som finner riktig melding ut i fra id

For å søke gjennom listen etter én spesifikk melding brukes findMeldingById(). Ved å benytte en forenklet for-løkke sjekkes det om melding.getId() og metodens parameter; idIn, er like. Hvis melding.getId() og idIn er like, vil den returnere meldingen. Hvis ikke, fortsetter søket. Eksisterer ikke meldingen vil null bli returnert.

```
public void writeMeldingToPath(Melding melding) throws JAXBException,  
IOException, SAXException {  
    JAXBContext jaxbContext = JAXBContextManager.getInstance();  
    Marshaller jaxbMarshaller = jaxbContext.createMarshaller();  
    Schema schema = MeldingValidator.generateSchema(melding);  
    JAXBSource jaxbSource = new JAXBSource(jaxbMarshaller, melding);  
  
    XMLValidator.validate(schema, jaxbSource); // Validering  
  
    String formattedDate = getFileDate();  
    File file = new File(FileManager.getPath() + formattedDate +  
        melding.getSkjemanavn() + ".xml"); // Create filename  
  
    jaxbMarshaller.setSchema(schema); // 2x validering  
    updateMsgList(melding); // Update meldingList with new Melding  
    jaxbMarshaller.marshal(melding, file); // Write to file  
}
```

Figur 76: Metoden `writeMeldingToPath` i `MeldingManager` som håndterer validering og skriving av `melding`.

Controlleren vil kalle «`writeMeldingToPath`» hvis det kommer en gyldig POST-forespørsel. Variabelen «`melding`» i metodens parameter vil da være en ny eller oppdatert melding sendt fra frontend. Metoden lager først en `JAXBContext`, som benyttes for å lage en «`Marshaller`». Riktig Schema blir funnet gjennom å bruke util-klassen `MeldingValidator`. Et schema er et objekt laget fra en XSD-fil, som benyttes for å validere XML-filer. I dette tilfellet vil det da brukes for å validere `Melding`-objektet. Deretter blir variabelen `jaxbSource` laget med attributtene `jaxbMarshaller` og `melding`.

Metoden «`validate`» i util-klassen `XMLValidator` kalles med `schema` og `jaxbSource`. Denne vil da kaste en `JAXBException` eller `SAXException` hvis validering feiler. Hvis validering ikke kaster et avvik, vil metoden reservere en tom fil med formatet dato + skjemanavn + .xml. Hvis `File` ikke kaster et `IOException`, vil `jaxbMarshaller.setSchema` bli kalt med `schema` som parameter. Dette gjør at når `jaxbMarshaller.marshal` kjøres, vil det kjøres enda en validering av `melding` mot `schema`. Dette er for å forhindre at edge-cases lar en korrupt melding bli skrevet til systemet. Til slutt oppdateres `meldingList`, og den opprettede filen blir populert med XML-data.

### 11.3. FileManager

```
● ● ●

public class FileManager {
    /**
     * Get path of utmappe.
     */
    public static String getPath() throws MalformedURLException {
        // Directory name of utmappe
        String utmappe = "utmappe/";
        // Users path
        String path = FileManager.class.getResource("").getPath();
        System.out.println(path);
        // Remove dir "target" and every dir after to get the project root
        // path
        path = path.replaceAll("target.*", "");
        System.out.println(path);
        // Append utmappe
        path = path + utmappe;

        return path;
    }

    /**
     * List all files in utmappe
     * Made for KRG demo for loading files from utmappe
     * @throws MalformedURLException
     */
    public static File[] listFiles() throws MalformedURLException {
        File dir = new File(FileManager.getPath());
        File[] files = dir.listFiles();
        System.out.println("All files in utmappe: " + Arrays.toString(files));
        return files;
    }
}
```

Figur 77: Klassen *FileManager* og dens metoder.

FileManager består av de to metodene `getPath()` og `listFiles()`. Formålet med klassen er å hente ut riktig path til filmappen `utmappe`, som man finner i Kreftregisteret-backend/`utmappe`. Ved videre utvikling vil denne klassen være unødvendig, da man vil benytte en tradisjonell database. Da systemet i nåværende fase benytter seg av å lese og skrive filer i en mappe, må systemet

benytte riktig path. I de tidligere fasene av prosjektet prøvde vi å lese filer fra den relative pathen «./utmappe», men dette viste seg å skape problemer. Ved å kjøre systemet fra en Windows-basert IntelliJ IDE vil dette fungere, men ved videre testing i GitHub Actions, WSL2 og Mac-baserte systemer viser det seg at den relative path vil endre seg fra system til system.

```
String path = FileManager.class.getResource("").getPath();
// Strengen path vil nå være
// /C:/Users/olagb/IdeaProjects/System-for-behandling-av-XML-
meldinger/krefregisteret-backend/target/test-
classes/com/Krefregisteret/KrefregisteretApp/utils/
```

Figur 78: Utsnitt som viser den midlertidige verdien til path i getPath.

Metoden getPath() prøver å løse dette ved å ta utgangspunkt i klassens egen path. Maven sin standard output-mappe er target, og FileManager sin path vil dermed være i targetmappen, selv om selve javaklassen ligger i src. Dermed kan man manipulere strengen path til å fjerne alt som er etter «target», og legge til strengen «utmappe».

```
path = path.replaceAll("target.*", "");
// /C:/Users/olagb/IdeaProjects/System-for-behandling-av-XML-
meldinger/krefregisteret-backend/
path = path + utmappe;
// /C:/Users/olagb/IdeaProjects/System-for-behandling-av-XML-
meldinger/krefregisteret-backend/utmappe/
```

Figur 79: Utsnitt av getPath som viser hvordan man manipulerer strengen path til å peke til utmappe.

Slik kan getPath() returnere systemets korrekte path til utmappe. Dette har vist seg å fungere både i WSL2, Mac samt Windows. Det fungerer også i de automatiserte testene i GitHub Actions. Klassens andre metode, listFiles(), har da muligheten til å benytte getPath() for å liste alle de eksisterende filene i utmappen. Metoden getPath() blir også brukt i MeldingManager.writeMeldingToPath for å finne riktig path i den nye XML-filen skal skrives til.

## 11.4. Validering

```
> tree
.
├── KliniskProstataKirurgi_v4_0.xsd
├── KliniskProstataStraalebehandling_v4_0.xsd
├── KliniskProstataUtredning_v4_0.xsd
└── Prostata_4_0_KirurgiEksempelfil.xml
    ├── Prostata_4_0_StraalebehandlingEksempelfil.xml
    └── Prostata_4_0_UtredningEksempelfil.xml

0 directories, 6 files
~/prostatapakke
```

Figur 80: Oversikt over relevante filer for validering i prostatapakke.

Ved starten av samarbeidet med KRG ble vi gitt en zip-mappe ved navn prostatapakke. Relevant for valideringen inneholdt pakken tre sett med XML-filer som samsvarer med XSD-filer. En XSD-fil (XML Schema Definition) inneholder hvilke elementer en gitt XML-fil skal inneholde, samt regler slik at hvert element skal være gyldig. Dette gjør det mulig å validere at en XML-fil følger skjemaets regler, og inneholder tilsvarende elementer som beskrevet i XSD-filen. Slik kan XML-filen **Prostata\_4\_0\_KirurgiEksempelfil.xml** valideres mot **KliniskProstataKirurgi\_v4\_0.xsd**, men ikke mot **KliniskProstataStraalebehandling\_v4\_0.xsd** og **KliniskProstataUtredning\_v4\_0.xsd**.

```
● ● ●

public class XMLValidator {

    public static final HashMap<String, String> XSD_MAP = new HashMap<>() {{
        put("KliniskProstataKirurgi", "KliniskProstataKirurgi_v4_0.xsd");
        put("KliniskProstataStraale", "KliniskProstataStraalebehandling_v4_0.xsd");
        put("KliniskProstataUtredning", "KliniskProstataUtredning_v4_0.xsd");
    }};
}
```

Figur 81: Utsnitt av `HashMap XSD_MAP`, som brukes for å finne riktig XSD fil.

I prosessen fant vi ut at XML-filens skjemanavn ikke samsvarer med XSD-filens navn. Dermed har vi i util-klassen `XMLValidator` laget en statisk `HashMap XSD_MAP`. Key samsvarer med

skjemanavn, og value er XSD-filens navn i filsystemet. Dette gjør det mulig å hente XSD-filens navn for å hente ut riktig XSD-fil i filsystemet. Videre i valideringsprosessen vil XSD-filene brukes for å lage tilsvarende Schema-objekter. Sett i ettertid kunne vi ha laget tre sett med schema-objekter i value, slik at systemet ikke trenger å lage ett nytt schema for hver gang en melding skal valideres.

```
● ● ●

public class MeldingValidator {
    private MeldingValidator() {}

    public static Schema generateSchema(Melding melding) throws SAXException,
    IOException {
        final String PROSTATAPAKKE = new
ClassPathResource("Prostatapakke").getURL().getPath() + "/";

        File XSD = new File(PROSTATAPAKKE +
XMLValidator.XSD_MAP.get(melding.getSkjemanavn()));
        SchemaFactory factory =
SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
        Schema schema = factory.newSchema(XSD);
        return schema;
    }
}
```

Figur 82: Metoden som lager schema ut i fra type melding.

For å generere schema bruker vi util-klassen «MeldingValidator» i pakken «utils/xml», som inneholder metoden «generateSchema». Formålet med metoden er å returnere riktig Schema-objekt ut ifra hvilken melding som blir gitt som argument. Systemet inneholder mappen Prostatapakke i «src/resources» og man kan bruke den innebygde Spring-klassen «ClassPathResource» for å finne riktig sti til mappen. Vi benytter «SchemaFactory» for å sette «schemaLanguage» til W3C\_XML\_SCHEMA\_NS\_URI, som tilsvarer <http://www.w3.org/2001/XMLSchema>. Deretter kan vi lage schema-objektet gjennom å bruke «factory.newSchema(XSD)», og returnere korrekt XSD-fil til meldingen.

```
public class XMLValidator {  
    public static void validate(Schema schema, JAXBSource jaxbSource) throws  
    IOException, SAXException {  
        Validator validator = schema.newValidator();  
        // Throws SAXParseException if not validated  
        validator.validate(jaxbSource);  
    }  
}
```

Figur 83: Metoden validate som validerer meldingen.

Selve valideringen mot schema foregår i XMLValidator.validate. Metoden tar i mot Schema og JAXBSource som blir hentet fra metoden writeMeldingToPath() i MeldingManager. Objektet jaxbSource inneholder meldingen vi skal validere som source. Schema sin metode newValidator() returnerer en validator for schema. Ved å bruke validator.validate(jaxbSource) blir jaxbSource, som inneholder meldingen, validert mot riktig schema. Hvis denne kaster SAXParseException har valideringen feilet. Med tanke på videre utvikling kunne metoden ha returnert en boolean verdi for å gjøre det mer leselig, men samtidig er det et avvik som må håndteres når en innkommende melding feiler valideringen.

## 11.5. Testing

Formålet med å teste en applikasjon er å få en pekepinn på at funksjonalitet oppfører seg som forventet. Applikasjonen har visning, endring og lagring av kreftmeldinger som hovedoppgave, og det er derfor viktig å dekke denne funksjonaliteten med testing. Det ble også testet for API-forespørsler ved hjelp av JavaScript, som tester interaksjonen mellom klient og server.

### 11.5.1. Enhetstesting

For enhetstestene brukte vi test rammeverket JUnit. Enhetstestingen for MeldingManager starter med oppstarts metoden kalt `setup()`. Denne er annotert med `@BeforeEach`, som signaliserer at denne metoden skal bli kjørt en gang før alle andre i test-klassen den befinner seg i.

```
●●●

@BeforeEach
public void setup() throws IOException {
    meldingManager.addMeldingerFromUtFolderToMeldingList(List.of(FileManager.listFiles()));

    File XMLKirurgi = new
        File("src/test/java/com/Kreftregisteret/KreftregisteretApp/utils/xml/testfiles/Prostata_4_0_KirurgiEksempelfil.xml");
    File XMLStraale = new
        File("src/test/java/com/Kreftregisteret/KreftregisteretApp/utils/xml/testfiles/Prostata_4_0_StraalebehandlingEksempelfil.xml");
    File XMLUtredning = new
        File("src/test/java/com/Kreftregisteret/KreftregisteretApp/utils/xml/testfiles/Prostata_4_0_UtredningEksempelfil.xml");

    XML = new File[] {XMLUtredning, XMLStraale, XMLKirurgi};

    File InvalidFile = new File("src/test/java/com/Kreftregisteret/KreftregisteretApp/utils/xml/testfiles/InvalidFile");
}
```

Figur 84: Oppstartsmetoden for Enhetstesting i MeldingManager.

Her blir flere Mock-objekter og en instanse av MeldingManager initialisert. Det finnes et mock-objekt for hver meldingstype og de er identiske til skjemaeer systemet ville mottatt i praksis. For å effektivisere kallene, er disse lagt til i en array. Videre opprettes en tom, ugyldig fil ment for å gjenskape en feil i systemet. Disse filene blir brukt i blant annet `convertFileToMelding()` testen, annotert med `@Test` for å signalisere at dette er en av testene.

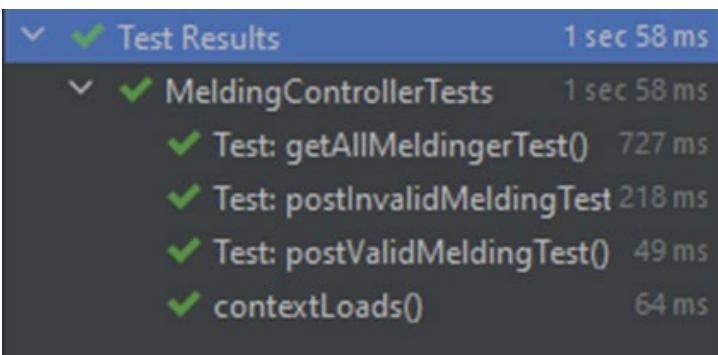
```
● ● ●
```

```
@Test
void convertFileToMelding() {
    // ===== Test 1: Valid file =====
    for (File file: XML) {
        assertNotNull(meldingManager.convertFileToMelding(file));
    }

    // ===== Test 2: Invalid file =====
    Assertions.assertThrows(Exception.class, () -> {
        meldingManager.convertFileToMelding(InvalidFile);
    });
}
```

Figur 85: Enhetstest av convertFileToMelding() i MeldingManager.

I første omgang blir hvert Mock-objekt testet i parameterne til convertFileToMelding()-metoden inn i en løkke. Hvis metoden ikke returnerer null, betyr det at testen ikke har feilet og den godkjennes. Andre test sjekker at det blir kastet et avvik dersom en ugyldig fil blir konvertert til en meldings objekt. JUnit sørger for at prosessen blir automatisert, og kan kjøres kjapt på et par sekunder hver gang det gjøres endringer i metodene. Figuren under er en eksempelkjøring på hvordan JUnit automatiserer testing



Figur 86: Eksempel på automatisert kjøring med JUnit.

### 11.5.2. Andre testmetoder

For å forsikre at API metodene funker som de skal, har vi implementert FetchAPI-tester med «Spring Boot Test»-rammeverket sammen med JUnit. Et eksempel er postValidMeldingTest()-

metoden for MeldingController-klassen. Her sjekkes det for at det er mulig å poste en gyldig melding til systemet.

```
● ● ●

final String POST_MELDING_ENDPOINT = "/api/v1/meldinger";

@Autowired
private MeldingController controller;

@Autowired
private MockMvc mockMvc;

@Test
@DisplayName("Test: postValidMeldingTest()")
public void postValidMeldingTest() throws Exception {
    final String VALID_POST_JSON = "{\"@type\":\"KliniskProstataUtredning\", \"meldingsinformasjon\":{\"skjem...\"}}";
    this.mockMvc
        .perform(post(POST_MELDING_ENDPOINT)
            .contentType(MediaType.APPLICATION_JSON)
            .content(VALID_POST_JSON))
        .andDo(print())
        .andExpect(status().isOk())
}
```

Figur 87: Test av gyldig POST-forespørsel.

Først defineres API-endepunktet globalt i klassen. Deretter instansieres Spring Boot test objektet «mockMvc» i tillegg til kontrolleren «controller». Test-objektet sjekker da HTTP-responsen til å sende en gyldig melding i form av JSON med «VALID\_POST\_JSON». Hvis responsen er 200 (OK), godkjennes testen, hvis responsen er 400 (Bad Request), mislykkes testen. Hele prosessen er også automatisert med JUnit på samme måte som nevnt i kapittelet over.

```
MockHttpServletResponse:
    Status = 200
    Error message = null
    Headers = [Vary:"Origin", "Access-Control-Request-Method", "Access-Control-Request-Headers"]
    Content type = null
    Body =
    Forwarded URL = null
    Redirected URL = null
    Cookies = []
```

Figur 88: Respons fra server etter å ha sendt en gyldig POST-forespørsel.

## 11.6. CI/CD

I utviklingsfasen har vi benyttet oss av GitHub for å arbeide på samme kodebase. Vi har prøvd å distribuere oftest mulig slik at man ikke ender opp med store tidkrevende konflikter mellom utviklerne sine lokale kodebaser. I tillegg har vi ved bruk av GitHub Actions definert workflows. Workflows er spesialiserte YAML-filer som beskriver jobs som kjøres på GitHub sine servere. Dette ga gruppen muligheten til å automatisere bygging av frontend, kopiere frontend til Spring, samt å verifisere at testene i backend ikke feiler. Det er også distribuert en modifisert branch av prosjektet på skytjenesten Heroku.



```
● ● ●

#!/bin/bash

# Deletes "static" folder in "resources" if it exists,
# and cp the "build" folder to "resources" and rename it to "static"

rm -rf ./krefregisteret-backend/src/main/resources/static \
&& cp -R ./krefregisteret-frontend/build ./krefregisteret-
backend/src/main/resources/static
```

Figur 89: Shell-script som kopierer inn frontend i mappen resources/static i Spring.

Mappen static i Spring resources inneholder et «static build» av frontend. En static build er en optimalisert statisk versjon av frontend og består av nødvendige filer frontend trenger for å vise korrekt brukergrensesnitt hos brukeren. Ved å legge disse filene i mappen static, vil Spring automatisk returnere index.html ved en GET-request til «/». Ved oppdateringer i frontend, vil det bygges en ny optimalisert statisk versjon av frontend filene. For å oppdatere mappen når det er distribuert endringer, benyttes shell-scriptet mv\_build\_to\_resources.sh. Skriptet sletter først den eksisterende mappen, for deretter å kopiere «build» i frontend til «static» i resources.

```

● ● ●

name: Build static react build in resources
on:
  push:
    branches:
      - main
    paths:
      - 'krefregisteret-frontend/**'
jobs:
  react-build:
    runs-on: ubuntu-18.04
    steps:
      - name: Checkout
        uses: actions/checkout@v3

      - name: Setup node 16
        uses: actions/setup-node@v3
        with:
          node-version: '16'

      - name: Make static react build
        working-directory: ./krefregisteret-frontend
        run: |
          npm install
          npm run build

      - name: Copy to resources
        run: |
          chmod +x ./mv_build_to_resources.sh
          ./mv_build_to_resources.sh
        shell: bash

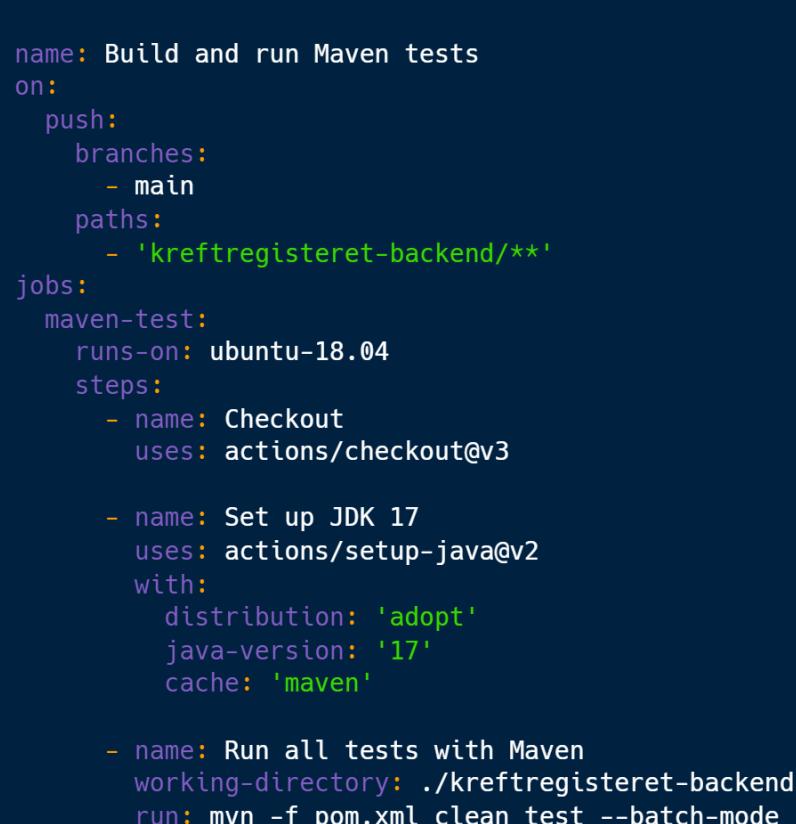
      - name: Commit and push static build
        run: |
          git config user.name github-actions
          git config user.email github-actions@github.com
          git add -f ./krefregisteret-backend/src/main/resources/static
          git commit -m "🤖 Generated static build"
          git push

```

Figur 90: «Github Workflow» som genererer static build fra frontend, og kopierer mappen inn i Spring.

For å generere et «static build» og kjøre shell-skriptet som flytter build til Spring, benyttes `react_static_build.yaml`. Denne «Github Workflow»-en kjøres kun når det er distribuert en endring i branchen `main`, og det er gjort endringer i mappen `Krefregisteret-frontend`.

Workflowen har en jobb som kjører på ubuntu-18.04, og henter ut git-repoet gjennom kommandoen Checkout. Deretter kjøres et node 16 setup som gjør det mulig å kjøre npm. Npm install og npm run build blir kjørt for å installere avhengigheter, og til slutt lage en static build. Når npm run build er ferdig, kjøres mv\_build\_to\_resources.sh for å kopiere den nye statiske versjonen til Spring. Til slutt blir endringene lagt til en commit, og pushet til git-repoet.



```
name: Build and run Maven tests
on:
  push:
    branches:
      - main
    paths:
      - 'krefregisteret-backend/**'
jobs:
  maven-test:
    runs-on: ubuntu-18.04
    steps:
      - name: Checkout
        uses: actions/checkout@v3

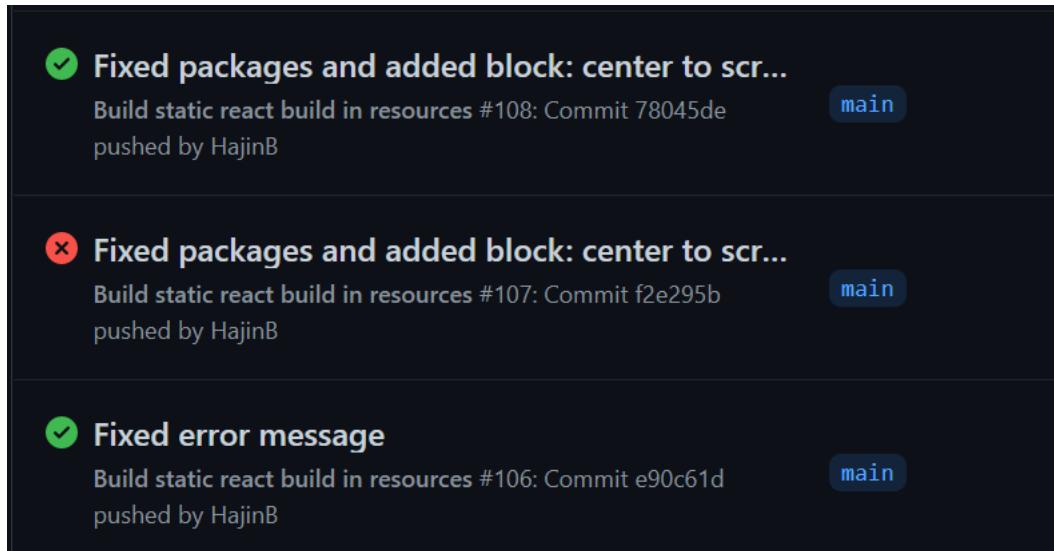
      - name: Set up JDK 17
        uses: actions/setup-java@v2
        with:
          distribution: 'adopt'
          java-version: '17'
          cache: 'maven'

      - name: Run all tests with Maven
        working-directory: ./krefregisteret-backend
        run: mvn -f pom.xml clean test --batch-mode
```

Figur 91: Workflow som kjører maven-tester i GitHub.

For å holde kontroll på hvilke maven-tester som feiler, lagde vi en workflow-fil spring\_test.yaml. Denne kjører når det er distribuert endringer i branchen main, i filer under mappen Krefregisteret-backend. Den har en jobb, som kjører på ubuntu-18.04. Første steg er å hente git-repoet gjennom kommanodet Checkout, før man setter opp Java JDK 17. Maven sine avhengigheter er cachet, slik at man ikke trenger å installere alle maven-avhengigheter på nytt

med mindre det er endringer i pom.xml. Til slutt kjøres mvn -f pom.xml clean test –batch-mode. Kommandoen benytter eksplisitt systemets pom.xml fil, fjerner filene fra forrige build, og kjører alle testene i batch-mode. Batch-mode forhindrer unødvendig logging og vil aldri stoppe for å spørre om input fra bruker.

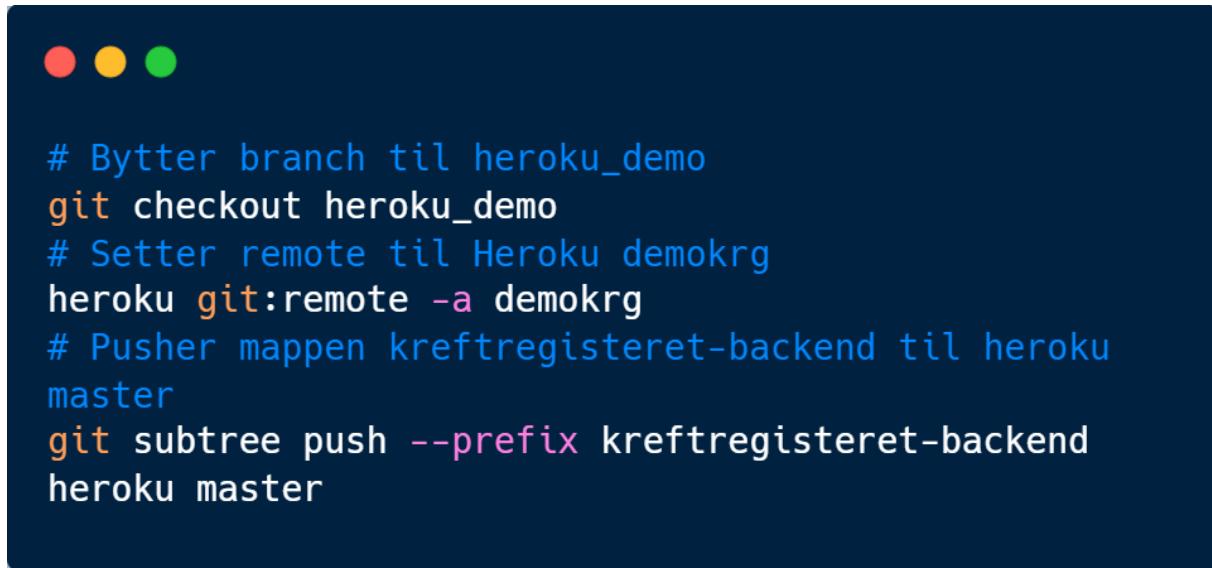


Figur 92: Eksempel på workflow som blir kjørt uten feil, og workflow som har feilet.

Ved distribusjon til GitHub vil man raskt få respons om en workflow har feilet. Dette vil til enhver tid bety at gruppen har oversikt over hvilken versjon som kan kjøres uten feil, og versjoner av prosjektet som har feil som hindrer kjøring av workflows. I figur 95 er en oversikt over tre commits, der commit f2e295 gjør at en workflow feiler. Dermed kan utviklere raskere finne ut av hvilken versjon som innførte feilen, og utelukke at tidligere versjoner har innført feil i koden.

Git-branchen «heroku\_demo» inneholder en spesialisert versjon av backend som kan kjøres i skytjenesten Heroku. Dette lar oss vise en demonstrasjon av prosjektet, ved at brukeren besøker <https://demokrg.herokuapp.com/>. Hovedforskjellen på main-branchen og heroku-branchen er at man unngår å skrive filer til utmappe i Heroku, men skriver de direkte inn i meldingList i minnet i stedet for å skrive de til fil. Dette er gjort for å gjøre prosjektet minst

mulig ressurskrevende, samt at meldingene som blir modifisert vil slettes når Heroku-dynoen skrur seg av.



```
# Bytter branch til heroku_demo
git checkout heroku_demo
# Setter remote til Heroku demokrg
heroku git:remote -a demokrg
# Pusher mappen krefregisteret-backend til heroku
# master
git subtree push --prefix krefregisteret-backend
heroku master
```

Figur 93: Script som laster opp prosjektet til Heroku.

For å laste opp prosjektet til Heroku, har vi valgt å bruke terminalen i Bash eller PowerShell. Ved å besøke root av prosjektet i terminalen, har man mulighet til å endre branch til heroku\_demo. Etterpå forsikrer man seg at man har satt opp riktig remote, altså Heroku remote demokrg. Branchen inneholder både mappen krefregisteret-backend og krefregisteret-frontend, men selve Spring-prosjektet er i krefregisteret-backend. Derfor kan man benytte git kommandoen subtree for å bare pushe krefregisteret-backend, og unngå unødvendige filer i Heroku.

# Konklusjon

*Hvordan lage en robust løsning som er web- og API-basert og kan kommunisere med resten av fagsystemene til Kreftregisteret slik at det gir minimal dødtid?*

Systemet vi har utviklet har en backend-del som anvender teknologier Kreftregisteret allerede bruker i dag på sine egne fagsystemer. Dette kan gjøre systemet enkelt å integrere med øvrige fagsystemer. Teknologiene er også moderne og av nyeste versjoner som minsker fare for krevende oppdateringer og sikkerhetshull i nærmeste fremtid. På frontend-siden har vi også benyttet oss av populære teknologier med god dokumentasjon og support-funksjoner. I tillegg har brukergrensesnittet tatt hensyn til brukerne av systemet som kan gi dem en enklere overgang til vår løsning sine skjema. Systemets frontend er også forsøkt utviklet så dynamisk og fleksibelt at det kan ta imot gamle meldinger fra det gamle systemet og utvikles til å passe med nye typer kreftmeldinger. Dette gjør det mulig å integrere systemet internt, for å deretter endre skjemaene på KREMT-sidene. Kort fortalt har løsningen ved hjelp av teknologiene SurveyJs, React, Axios, REST, Spring Boot, Java og JAXB, prøvd å gjøre det enkelt å kommunisere med Kreftregisteret sine fagsystemer. Løsningen som et modulbasert system skal gjøre det lettere å erstatte Kreftregisterets eksisterende løsning.

Målet vårt om å *utvikle et robust og skalerbart, modulbasert system for behandling av kreftmeldinger* er møtt, blant annet fordi Kreftregisteret har bekreftet sin plan for videreutvikling av systemet. Forhåpentligvis vil løsningen vår forbedre dagens situasjon for brukerne av systemet når det settes i produksjon.

# Refleksjon

Gjennom sprintene har vi møtt på interessante problemstillinger og utfordringer som vi ikke har hatt erfaring med tidligere. I prosjektarbeidet har vi hatt jevnlige møter og kommunisert ofte innad i gruppen ved hjelp av tekst, bilder, fjernmøter og fysiske møter.

På grunn av omfanget av dette prosjektet har kommunikasjon kanskje vært det viktigste delen i prosessen. Vi har vært heldige med tanke på hvor konkrete Kreftregisteret har vært med sine krav og ønsker. Dette har gjort det enkelt å komme i gang, og å spørre de spørsmål om vi har vært på rett vei. De ukentlige statusmøtene med både gruppen og oppdragsgiver, som er typiske innenfor smidig metodikk, har vi sett på svært nødvendige for å fortsette å holde på den samme forståelsen for hva sluttproduktet skal bli og sammen planlegge de neste stegene. I tillegg til å øke vår felles forståelse, så har møtene også hatt en oppfølgingseffekt.

Selv om vi rakk å bli ferdig med alle oppgaver som var viktig for løsningen og de fleste oppgaver definert som «nice-to-have», kunne det ha vært gunstig å ha utnevnt en prosjektleder, eller en person med fokus på å deletere oppgaver, og se til at disse oppgavene ble gjort ferdig til riktig tid.

Hvilke oppgaver som burde prioriteres først og sist ble for oss ganske greit å bestemme, da vi hadde ganske konkrete definisjoner for MVP. Fordi vi brukte god tid på å definere oppgaver å arbeide med i planleggingsfasen, gikk det for det meste fint å kontinuerlig jobbe med konkrete arbeidsoppgaver. Enkelte oppgaver kunne ha blitt definert mer i detalj, da på et punkt i utviklingen arbeidet to gruppemedlemmer med den samme spesifikke kodebiten. Fordi et gruppemedlem hadde en oppgave som krevde at en annen oppgave var ferdig, ble den funksjonen som blokkerte sitt eget arbeid gjort ferdig først. Denne situasjonen kunne ha blitt unngått ved å prioritere de oppgavene som eventuelt kunne ha blokkert annet arbeid.

I løpet av utviklingsprosessen ble vi litt usikre på om arbeidsfordelingen mellom backend og frontend hadde blitt litt skeivt fordelt. Mot slutten av prosjektet hadde backend-teamet gjort ferdig så å si alle oppgavene som var nødvendig for prosjektet, og begynte å arbeide med

refaktorering og deployering – oppgaver som egentlig var definert som «nice to have», og derfor ikke viktig for selve produktet. Det som skulle vise seg å være mest fremmed for gruppemedlemmene som jobbet med backend, var hvordan vi kunne skrive tester, da dette var noe vi ikke hadde gått i dybden på tidligere.

På grunn av at Kreftregisteret ønsket en modulbasert løsning, det vil si en løsning som enkelt kan importeres og integreres i deres ytterlige systemer, var det ikke prioritert å utvikle et databasesystem. Istedentfor å bruke en database som SQL eller MongoDB, skrev vi rene XML-filer til disk. Ettersom de fleste løsninger som fungerer bra sammen med Spring Boot er basert rundt databaser, viste det seg å være ganske vanskelig å lagre og lese rene filer til disk i en webapplikasjon som ble distribuert til Heroku. Tiden vi brukte på å forstå feilene som oppsto i forbindelse med dette kunne ha blitt brukt på å opprette en database for å håndtere lagringen av kreftmeldingene.

# Videre anbefalinger

## Frontend

For å ta vare på alle de forskjellige menneskenes behov og krav, bør man gjennom profesjonelle verktøy sjekke at skjemaet er i tråd med WCAG 2.0-kravene og fra 2023 at de også er i tråd med de nye kravene. Spesielt er dette viktig når skjemaene skal distribueres til andre helseinstitusjoner og brukerne dermed får et større mangfold. I tillegg er det noen bruksmønstre man kan jobbe videre med, blant annet en som bidrar til å sikre at dataene som sendes inn ivaretar integritet, altså at det er sikkert at de er riktige: «En bruker bør få valget om å se gjennom innsendingen før de sender den inn for å sikre dataenes integritet.». Et annet bruksmønster som sikrer mer integritet er «En bruker som prøver å sende inn et skjema uten endringer bør få beskjed om dette og lagring bør stanses, for å unngå unødvendige duplikater. Dette bruksmønstre kan bidra til at kopier av data ikke blir en kilde til utdatert informasjon. For å øke brukervennligheten bør en også implementere løsninger for bruksmønstret «En bruker som prøver å sende inn et skjema uten endringer bør få beskjed om dette og lagring bør stanses, for å unngå unødvendige duplikater.».

## Backend

Systemet er forsøkt utviklet modulbasert, det vil si at et av målene med utviklingen har vært å oppnå høy kohesjon og lav kobling. På grunn av Kreftregisterets ønsker om å integrere vår løsning i deres ytterlige systemer, har informasjonssikkerhet som en del av vår modul ikke vært en prioritet. Dette har vært drøftet i møter med KRG, der de ikke ønsket at vi skulle bruke tid på sikkerhet. Dette blant annet fordi skjemaene kommer til å bli brukt i et lukket nettverk, der sikkerhetsinfrastrukturen allerede eksisterer. Vår løsning er ikke en frittstående webapplikasjon, men et sett med moduler som vil bli del av et større system. Dermed bør vår løsning gjennomgås av KRG for å videre sikre systemet, og tilpasse modulene til deres eksisterende infrastruktur.

API-et kan videre utvides med funksjonalitet for å lage nye meldinger. Dette kan løses ved å sende et JSON-skjema ut ifra forespurt meldingstype, der verdiene er blanke. Frontend vil da ha mulighet til å endre JSON, og sende tilbake en utfylt melding på samme vis som endring i nåværende løsning.

Ved implementering i den eksisterende infrastrukturen i KRG bør metodekallene som går til FileManager erstattes med metodekall direkte til Kreftregisterets databaseløsning. Å integrere database med vår løsning vil være en av arbeidsoppgavene til to av gruppemedlemmene som skal jobbe hos Kreftregisteret sommeren 2022.

# Kilder

Babich, N. B. (2020, 20. juni). *Form Design Best Practices: Structure, Inputs, & Labels | Adobe XD*.

Adobe. Hentet 22. mai, 2022, fra <https://xd.adobe.com/ideas/principles/web-design/best-practices-form-design/>

Cockburn, A. & Williams, L. (2000, februar). The Costs and Benefits of Pair Programming.

ResearchGate. Hentet 22. Mai 2022, fra  
[https://www.researchgate.net/publication/2333697\\_The\\_Costs\\_and\\_Benefits\\_of\\_Pair\\_Programming](https://www.researchgate.net/publication/2333697_The_Costs_and_Benefits_of_Pair_Programming)

Baeldung. (2022, 31. mars). *The DTO Pattern (Data Transfer Object)*. Hentet 23 mai, 2022, fra

<https://www.baeldung.com/java-dto-pattern>

Crusoveanu, L. (2022, 19. mai). *Intro to Inversion of Control and Dependency Injection with Spring*. Baeldung. Hentet 23. mai, 2022, fra <https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring>

Dahl, C. D. (2020, 5. oktober). *Gestaltprinsipper i UI-design*. Vidi - Digitalbyrå i Bergen og Oslo.  
Hentet 22. mai, 2022, fra <https://vidi.no/blogg/gestaltprinsipper-i-ui-design/>

Davis, I. D. (2008, 9. desember). *What Are The Benefits of MVC?* Internet Alchemy. Hentet 23. mai, 2022, fra <https://blog.iandavis.com/2008/12/what-are-the-benefits-of-mvc/>

Det Norske Akademi for Språk og Litteratur. (u.å.). *Aktør*. Det Norske Akademis ordbok. Hentet 22. mai, 2022, fra <https://naob.no/ordbok/akt%C3%B8r>

Devsoft Baltic OÜ. (u.å.). *SurveyJS - Survey and Form JavaScript Libraries*. SurveyJS. Hentet 22. mai, 2022, from <https://surveyjs.io/> Hentet 22 mai, 2022, fra <https://surveyjs.io/>

- Direktoratet for e-helse. (2020, 4. februar). *Normen - Norm for informasjonssikkerhet og personvern i helse- og omsorgssektoren*. Ehelse. Hentet 22. mai, 2022, fra  
<https://www.ehelse.no/normen/normen-for-informasjonssikkerhet-og-personvern-i-helse-og-omsorgssektoren#1.5%20Om%20Normens%20krav>
- Edpresso Team. (2022, 23. mai). *What is a React component?* Educative: Interactive Courses for Software Developers. Hentet 23. mai, 2022, fra  
<https://www.educative.io/edpresso/what-is-a-react-component>
- EnTur. (u.å.). *Brukerhistorier*. Entur Designsystem. Hentet 22. mai, 2022, fra  
<https://design.entur.org/kom-i-gang/for-designere/brukerhistorier/>
- European Union. (2021, 22. oktober). *Art. 5 GDPR – Principles relating to processing of personal data*. General Data Protection Regulation (GDPR). Hentet 22. mai, 2022, fra <https://gdpr-info.eu/art-5-gdpr/>
- EverybodyWiki Bios & Wiki. (2021, 7. februar). *SurveyJS*. Hentet 22. mai, 2022, fra  
<https://en.everybodywiki.com/SurveyJS>
- Fielding, R. T. F., & Reschke, J. F. R. (2014, juni). *RFC 7231 - Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. Datatracker. Hentet 23. mai, 2022, fra  
<https://datatracker.ietf.org/doc/html/rfc7231#section-4>
- Gamma, E., Helm, R., Johnson, R. E., Vlissides, J., & Booch, G. (1995). *Design Patterns* [E-book]. Addison-Wesley. Hentet 23. mai, 2022, fra  
<https://books.google.nl/books?id=tmNNfSkfTlcC>

Gupta, M. G. (2021, 13. desember). *All you need to know about Async Await In JavaScript*.

Medium. Hentet 23. mai, 2022, fra <https://medium.com/technofunnel/javascript-async-await-c83b15950a71>

Jansen, A. (2018, 16. november). *Kravdefinisjon og kravspesifikasjon*. NDLA. Hentet 22. mai,

2022, fra <https://ndla.no/nb/subject:1:9d6d3241-014d-4a5f-b0bc-ae0f83d1cd71/topic:3:193104/topic:2:123578/resource:1:123591>

Krefregisteret. (u.å.). *ELVIS - Hjelp*. ELVIS. Hentet 23. mai, 2022, fra

<https://metadata.krefregisteret.no/help>

Krefregisteret. (2020a, 15. april). *Innrapportering*. Hentet 22 mai, 2022, fra

<https://www.krefregisteret.no/Registrene/Innrapportering/>

Krefregisteret. (2020b, 4. august). *Om Krefregisteret*. Hentet 22 mai, 2022, fra

<https://www.krefregisteret.no/Generelt/Om-Krefregisteret/>

Krill, P. K. (2021, 14. september). *JDK 17: The new features in Java 17*. InfoWorld. Hentet 23.

mai, 2022, fra <https://www.infoworld.com/article/3606833/jdk-17-the-new-features-in-java-17.html>

Lid, I. M. (2021, 27. desember). *Universell utforming*. Store norske leksikon. Hentet 22. mai,

2022, fra [https://snl.no/universell\\_utforming](https://snl.no/universell_utforming)

Machadop1407, M. (2021, 23. august). *GitHub - machadop1407/custom-use-fetch-hook-react*.

GitHub. Hentet 23. mai, 2022, fra <https://github.com/machadop1407/custom-use-fetch-hook-react>

Microsoft. (u.å.). *Introduction to InfoPath Forms Services*. Hentet 22. mai, 2022, fra

<https://support.microsoft.com/en-us/office/introduction-to-infopath-forms-services-e599b4f0-1d1d-4249-8251-5d28afb648d8>

Microsoft. (2015, 6. februar). *Update on InfoPath and SharePoint Forms*. Microsoft 365 Blog.

Hentet 22. mai, 2022, fra <https://www.microsoft.com/en-us/microsoft-365/blog/2014/01/31/update-on-infopath-and-sharepoint-forms/>

National Institute of Standards and Technology. (2021, 3 mai). *Human Centered Design (HCD)*.

NIST. Hentet 22 mai, 2022, fra <https://www.nist.gov/itl/iad/visualization-and-usability-group/human-factors-human-centered-design>

NHI. (2018, 20 februar). *Karpaltunnelsyndrom*. Hentet 22. mai, 2022, fra

<https://nhi.no/sykdommer/hjernenervesystem/nerveskader/karpaltunnelsyndrom/?page=2>

Oslo universitetssykehus HF. (u.å.). *Melding til Kreftregisteret*. eHåndbok. Hentet 22. mai, 2022,

fra <https://ehandboken.ous-hf.no/document/71461>

Penzo, M. P. (2006, 12. juli). *Label Placement in Forms*. UXmatters. Hentet 22. mai, 2022, fra

<https://www.uxmatters.com/mt/archives/2006/07/label-placement-in-forms.php>

ProductPlan. (2021, 12. august). *Minimum Viable Product (MVP)*. Hentet 22. mai, 2022, fra

<https://www.productplan.com/glossary/minimum-viable-product/>

Rehkorf, M. R. (u.å.). *What is a kanban board?* Atlassian. Hentet 22. mai, 2022, fra

<https://www.atlassian.com/agile/kanban/boards>

Rolstadås, A. (2019, 18. juli). *Milepål (prosjektledelse)*. Store norske leksikon. Hentet 22. mai,

2022, fra <https://snl.no/milep%C3%A6l - prosjektledelse>

Rolstadås, A., & Liseter, I. M. (2018, 24. april). *kravspesifikasjon*. Store norske leksikon. Hentet 22. mai, 2022, fra <https://snl.no/kravspesifikasjon>

Sacolick, I. (2022, 15. april). *What is CI/CD? Continuous integration and continuous delivery explained*. InfoWorld. Hentet 23. mai, 2022, fra

<https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html>

Sommerville, I. S. (2015). *Software Engineering* (9th ed.). Pearson.

<https://engineering.futureuniversity.com/BOOKS%20FOR%20IT/Software-Engineering-9th-Edition-by-Ian-Sommerville.pdf>

Spring. (u.å.). *RestController (Spring Framework 5.3.20 API)*. Spring Framework. Hentet 23. mai, 2022, fra <https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/RestController.html>

Staff, K. A. W. (2006, 14. juni). *Is Your Team Too Big? Too Small? What's the Right Number?*

Knowledge at Wharton. Hentet 22. mai, 2022, fra

<https://knowledge.wharton.upenn.edu/article/is-your-team-too-big-too-small-whats-the-right-number-2/>

Statlig spesialpedagogisk tjeneste. (2020, 1. oktober). *Universell utforming av dokumenter | www.statped.no*. Statped. Hentet 23. mai, 2022, fra

<https://www.statped.no/temaer/universell-utforming/universell-utforming-av-dokumenter/>

Tank, A. T. (2022, 16. mars). *46 Form Design Best Practices | Form Design Examples*. Jotform.

Hentet 22. mai, 2022, fra <https://www.jotform.com/form-design/>

Travis, D. T. (u.å.). *Personas*. Userfocus. Hentet 22. mai, 2022, fra

<https://www.userfocus.co.uk/consultancy/personas.html>

UUTilsynet. (u.å.). *Kva seier forskrifta?* Tilsynet for universell utforming av ikt. Hentet 22. mai,

2022, fra <https://www.uutilsynet.no/regelverk/kva-seier-forskrifta/153>

W3C. (u.å.). *Understanding Success Criterion 1.4.3 | Understanding WCAG 2.0*. W3. Hentet 22.

mai, 2022, fra <https://www.w3.org/TR/UNDERSTANDING-WCAG20/visual-audio-contrast-contrast.html>

Wålberg, J. A. W. (2020, 16. januar). *Hva er et Gantt-diagram?* Metier OEC. Hentet 22. mai,

2022, fra <https://www.prosjektbloggen.no/hva-er-et-gantt-diagram>

# Vedlegg A – Prosjektskisse

## Prosjektskisse for bachelorprosjekt ved OsloMet våren 2022

12.11.2021 Oslo

**Tittel:** System for behandling av XML-meldinger for Kreftregisteret

Ola G. Berg, s341874  
Tom H. Basmo, s340432  
Jørund T. Løvlien, s341822  
Nikola Dordevic, s341839  
Hajin Barzingi, s238727

**Oppdragsgiver:**

|                          |                    |                          |
|--------------------------|--------------------|--------------------------|
| Kreftregisteret:         | Org.nr 993 467 049 | Sentralbord: 22 45 13 00 |
| Postadresse:             | Besøksadresse:     |                          |
| Postboks 5313 Majorstuen | Ullernchausseen 64 |                          |
| 0304 Oslo                | 0379 Oslo          |                          |

**Kontaktperson:**

|                            |                                |
|----------------------------|--------------------------------|
| Jan F. Nygård              | Tlf: 22 92 87 10               |
| Leder, Registerinformatikk | E-post: jfn@kreftregisteret.no |

**Beskrivelse:**

Kreftregisteret ble opprettet i 1951 og er et av de eldste nasjonale kreftregistre i verden.

Kreftregisteret samler informasjon om alle krefttilfeller i Norge.

I helsevesenet så sendes pasientinformasjon mellom sykehus og fra sykehus til Kreftregisteret i form av XML meldinger i et eget helsenettverk (Norsk Helsenett). Kreftregisteret mottar flere hundretusen slike meldinger hvert år.

De aller fleste meldinger skal gjennomgås av medisinske koder, og av og til må informasjon i disse meldingene rettes eller kodes om.

I dag brukes det et kommersielt program (Infopath fra Microsoft), til å lese XML meldingene og vis på skjerm til de medisinske koderne. Infopath er “end of life” og videreutvikles ikke lenger. Det er også et nokså “frittstående” program, og kommuniserer ikke godt med Kreftregisteret andre fagsystemer som er java baserte.

Kreftregisteret spør om et web / api basert systemet som kommuniserer med resten av fagsystemene deres, og som viser det medisinske innholdet fra XML i en webleser, der innholdet kan redigeres, og sendes tilbake til fagsystemet. For å løse oppgaven, har vi visualisert den i fem underoppgaver:

1. Ta i mot en XML-fil
2. Lese den opp i et web GUI => [React](#)
3. Ha mulighet til å endre => [Surveyjs](#)
4. Validere => mot XSD og met et API-kall mot metadatabasen ved Kreftregisteret
5. Lagre som en XML => [Hibernate](#)

Målet vårt er å lage en løsning som fungerer for én enkelt krefttype (muligens flere). Verktøyene vi har sett oss ut er React og surveyjs i frontend, og Java EE med Hibernate i backend.

I denne oppgaven må vi spesielt tenke på scalability. Det finnes over 200 forskjellige XML-maler, og denne løsningen skal enkelt kunne implementeres videre. Vi får av oppdragsgiver i nærmeste fremtid en pakke med filer, definisjoner og XSD (XML Schema Definition), slik at vi bedre kan forstå utgangspunktet for oppgaven.

# Vedlegg B – Prosjektkontrakt

## AVTALE OM PROSJEKTOPPGAVE

Den 26 november 2021 ble følgende avtale inngått mellom OsloMet – storbyuniversitetet, med registrert forretningsadresse Pilestredet 46, 0167 OSLO og organisasjonsnr. 997 058 925 (heretter **OsloMet**),

og

Oslo Universitetssykehus HF - Krefregisteret, med registrert forretningsadresse Ullernchausseen 64, 0379 Oslo og organisasjonsnr. 993 467 049 (heretter **Krefregisteret**)

Og følgende studenter (heretter "Studentene")

- Ola Gynnild Berg, med adresse Carl Dons Veg 7, 7049 TRONDHEIM og studentnummer s341874
- Hajin Barzingi, med adresse Trondheimsveien 107, 0565 OSLO og studentnummer s238727
- Tom Henrik Melting Basmo, med adresse Otto Songs vei 14, 0681 OSLO og studentnummer s340432
- Nikola Dordevic, med adresse Øvre Guldalsgate 14C, 1467 STRØMMEN og studentnummer s341839
- Jørund Topp Løvlien, med adresse Bølerskogen 101, 0691 OSLO og studentnummer s341822

### 1. BAKGRUNN

- 1.1. Studentene er studenter ved OsloMet og skal i forbindelse med sine studieløp forfatte og innlevere en prosjektoppgave innenfor informasjonsteknologi (heretter "Prosjektoppgaven"). Prosjektoppgavens arbeidstittel er "System for behandling av XML-meldinger for Krefregisteret"
- 1.2. Krefregisteret driver virksomhet innenfor helse og helseregistre og ønsker å samarbeide med OsloMet om tilretteleggingen av Studentenes arbeid med Prosjektoppgaven. Krefregisterets virksomhet er regulert i Krefregisterforskriften og Helseregisterloven.

### 2. VEILEDNING

- 2.1. Studentene skal ha to veiledere, én fra OsloMet og én fra Krefregisteret.
- 2.2. Veiledning fra Krefregisteret skjer ved faste ukentlige møter på opptil 1 time, samt workshops når det er hensiktsmessig.

### 3. FINANSIERING MV.

- 3.1. Med mindre annet er avtalt, bærer partene sine egne kostnader i forbindelse med Prosjektoppgaven.
- 3.2. OsloMet skal stille ordinære studiefasiliteter, herunder tilgang til egnede lokaler og datanettverk, til rådighet for Studentene i tråd med OsloMets til enhver tid gjeldende

retningslinjer.

- 3.3.** Kreftregisteret skal for egen regning stille til rådighet programvare som studenten ikke har tilgang til gjennom OsloMet, og som er nødvendig for å gjennomføre Prosjektoppgaven.

**4. LAGRING AV PROSJEKTRELATERTE DATA**

- 4.1.** Partene skal hver for seg påse at data av betydning for Prosjektoppgaven behandles og lagres i henhold til kravene til informasjonssikkerhet som følger av regelverket.

- 4.2.** Partene er enige om at elektroniske primærdata skal lagres på GitHub.

**5. RETTIGHETER TIL RESULTATER MV.**

- 5.1.** Studentene har opphavsrett til Prosjektoppgaven. Det samme gjelder alle resultater av arbeidet med Prosjektoppgaven, så fremt resultatene er frembrakt av Studentene alene. Rettighetene til alle resultater, metoder, dokumenter, dataprogrammer, prototyper og annet materiale og andre produkter som utarbeides eller utvikles av Studentene sammen med Kreftregisteret og OsloMet i forbindelse med arbeidet med Prosjektoppgaven, tilfaller Studentene, Kreftregisteret og OsloMet i fellesskap (dvs. med 1/3 som fordeles likt mellom Studentene, 1/3 til Kreftregisteret og 1/3 til OsloMet), med mindre annet er avtalt.

- 5.2.** Kreftregisteret kan fritt, og uten vederlag, benytte Prosjektoppgaven og resultatene av arbeidet med Prosjektoppgaven i egen virksomhet.

- 5.3.** OsloMet eier originaleksemplarene av Prosjektoppgaven med vedlegg. OsloMet kan fritt og uten vederlag benytte Prosjektoppgaven og resultatene av arbeidet med Prosjektoppgaven i sin egen virksomhet, herunder i forsknings- og undervisningsøyemed. For publisering av Prosjektoppgaven gjelder bestemmelsene i punkt 7.1-7.3.

- 5.4.** Bestemmelsene ovenfor gjelder bare så langt de er forenlig med ufravikelige regler etter norsk lov, herunder åndsverklovens regler om ideelle rettigheter.

- 5.5.** Hver part har ansvar for å klarere tredjemenns eventuelle rettigheter til materiale som bringes frem av vedkommende part som bakgrunnsmateriale for Prosjektoppgaven.

- 5.6.** Studentene har plikt til å melde fra til Kreftregisteret og OsloMet om mulig patenterbare oppfinnelser som blir frembrakt i forbindelse med arbeidet med Prosjektoppgaven. Hvis en slik oppfinnelse er frembrakt på grunnlag av faglig bistand fra veilederne eller øvrige innsatsfaktorer fra de øvrige partene, skal partenes respektive andeler i rettighetene gjenspeile deres innsats.

**6. KONFIDENSIALITET**

- 6.1.** Kreftregisteret skal bevare taushet om all informasjon mottatt fra Studentene, og skal ikke formidle slik informasjon til ansatte og andre representanter, personer og enheter enn dem som behøver å kjenne til informasjonen for at rettigheter og plikter etter Avtalen skal kunne utøves. Dette gjelder likevel ikke informasjon som

- (i) er alminnelig kjent
- (ii) en part kan dokumentere at parten kjente til før den ble mottatt fra den annen part
- (iii) en part kan dokumentere at er utviklet av ansatte eller andre som ikke hadde kjennskap til informasjonen mottatt fra den annen part
- (iv) en part kan dokumentere at er blitt gjort tilgjengelig for parten fra en tredjepart som hadde en rett til å frigi denne informasjonen.

Studentene skal bevare taushet om informasjon mottatt fra **Krefregisteret**, der **Krefregisteret** har informert Studentene om at informasjonen er konfidensiell.

## **7. PUBLISERING MV.**

- 7.1.** Alle forskningsresultater fra arbeidet med Prosjektoppgaven skal være gjenstand for publisering uten restriksjoner, i tråd med universitets- og høgskoleloven § 1-5 (6).
- 7.2.** Publisering kan skje i OsloMets institusjonelle arkiv på internett eller i hvilken som helst publikasjon valgt av Studentene.
- 7.3.** Hvis det er saklig grunn til det, kan partene avtale at offentliggjøring og publisering av hele eller deler av Prosjektoppgaven eller resultatene av arbeidet med Prosjektoppgaven skal utsettes i inntil tre år. Partene skal likevel i størst mulig utstrekning legge til rette for at Studentene kan bruke Prosjektoppgaven i jobbsøknader eller i et senere doktorgradsarbeid.

## **8. AVTALENS VARIGHET**

- 8.1.** Avtalen varer fra Avtaledato, og løper inntil Prosjektoppgaven er ferdig utarbeidet, innlevert og sensurert.
- 8.2.** Hvis Studentene avslutter arbeidet før Prosjektoppgaven er innlevert, kan hver av de øvrige partene si opp Avtalen med virkning fra utløpet av inneværende studiesemester.
- 8.3.** Avtalen pkt. 5-7 gjelder videre selv om Avtalen forøvrig er oppfylt eller sagt opp.

## **9. TVISTELØSNING**

- 9.1.** Avtalen er underlagt og skal fortolkes i samsvar med norsk rett.
- 9.2.** Eventuelle tvister mellom partene skal søkes løst ved forhandlinger. Hvis forhandlingene ikke fører frem, kan hver part bringe tvisten inn for de alminnelige domstoler.
- 9.3.** Oslo er avtalt verneting.

\*\*\*

Avtalen er undertegnet i 3 eksemplarer, hvorav partene beholder ett eksemplar hver.

Avtalen er inngått på den dato som fremgår innledningsvis i Avtalen.

**For OsloMet – storbyuniversitetet**



Laurence Habib  
Instituttleder,  
institutt for  
informasjonsteknologi,  
OsloMet -- storbyuniversitet

**For Kreftregisteret**

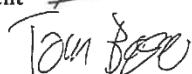
  
9/12/2024

Jørund Topp Løvlien

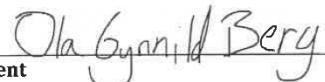
Student



Student



Student



Student



Student

# Vedlegg C – Forprosjektrapport

## Forprosjektrapport

System for behandling av XML-meldinger  
av Ola G. Berg, Tom H. Basmo, Nikola Dordevic, Jørund T. Løvlien & Hajin Barzingi



**Dato:** 24.01.2022

**Versjon:** 1.2

## Sammendrag

---

Gruppen består av fem dataingeniør-studenter som har fått et oppdrag fra Kreftregisteret, med Jan Franz Nygård, leder for Registerinformatikkavdelingen, som kontaktperson.

Oppdraget går ut på å lage et nytt system for å ta imot, vise og redigere XML-meldinger som blir sendt inn fra Norsk Helsenett. Kreftregisteret har allerede et slikt system, men teknologien som er brukt er utdatert og vanskelig å integrere med andre systemer. Registerinformatikkavdelingen har gjort undersøkelser tidligere og kommet frem til at en ny løsning med teknologiene Java EE, React, SurveyJS og Hibernate vil være den beste. Vi har undersøkt teknologiene som er anbefalt og også sett på alternativer. Basert på analysene, Kreftregisterets ønske og kompetanse innenfor de ulike teknologiene, er React og SurveyJS valgt som teknologi for å kode frontend og brukergrensesnitt, mens Java med Spring Boot og Hibernate er valgt for å kode backend-løsningen. Java Spring Boot er et rammeverk basert på Spring som gjør det enkelt å utvikle REST API og mikrotjenester. Valget falt på denne teknologien fordi det skal være greit å skalere og på grunn av "AutoConfigure" er raskt å komme i gang med.

Løsningen som blir utviklet for Kreftregisteret vil blant annet bidra i arbeidet med å modernisere et utdatert system og være enklere å vedlikeholde og videreutvikle. I tillegg vil løsningen effektivisere arbeidet til de medisinske koderne.

## Innhold

---

|                                   |    |
|-----------------------------------|----|
| Sammendrag.....                   | 2  |
| 1. Presentasjon.....              | 4  |
| 1.1. Gruppen .....                | 4  |
| 1.2. Oppdragsgiver .....          | 4  |
| 1.3. Beskrivelse av prosjekt..... | 4  |
| 1.4. Framdriftsplan.....          | 5  |
| 2. Dagens situasjon.....          | 6  |
| 3. Mål og rammebetingelser.....   | 7  |
| 4. Løsninger/alternativer.....    | 8  |
| 4.1. Frontend.....                | 8  |
| 4.2. Backend.....                 | 9  |
| 4.3. Valgt løsning.....           | 9  |
| 5. Analyse av virkninger.....     | 10 |
| 6. Referanser .....               | 11 |

## 1. Presentasjon

### 1.1. Gruppen

Gruppen består av følgende fem dataingeniør-studenter:

- |                              |                                |
|------------------------------|--------------------------------|
| • Ola G. Berg<br>s341874     | • Tom H. Basmo<br>s340432      |
| • Nikola Dordevic<br>s341839 | • Jørund T. Løvlien<br>s341822 |
| • Hajin Barzingi<br>s238727  |                                |

### 1.2. Oppdragsgiver

Oppdragsgiver er Kreftregisteret. Kreftregisteret ble opprettet i 1951 og er et av de eldste nasjonale kreftregistre i verden. I tillegg til forskningsarbeid og ansvar for screeningprogrammer, samler de inn data og lager statistikk over alle krefttilfeller i Norge.

Vår kontaktperson ved Kreftregisteret er leder for registerinformatikkavdelingen, Jan Franz Nygård.

### 1.3. Beskrivelse av prosjekt

Kreftregisteret ønsker et web/API-basert system som kommuniserer med resten av fagsystemene deres, og som viser det medisinske innholdet fra XML-meldinger i en webleser, der innholdet kan redigeres og sendes tilbake til fagsystemet.

For å løse oppgaven, har vi visualisert den i fem underoppgaver:

1. Ta imot en XML-fil => Java
2. Lese den opp i et web GUI => React
3. Ha mulighet til å endre => Surveyjs
4. Validere => mot XSD og mot et API-kall mot metadatabasen ved Kreftregisteret
5. Lagre som en XML => Hibernate

Målet vårt er å lage en løsning som fungerer for én enkel krefttype (muligens flere). Verktøyene vi har sett oss ut er React og Surveyjs i front-end, og Java Spring Boot med Hibernate i back-end.

#### 1.4. Framdriftsplan

|       |  | Developement schedule - Rough |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |  |
|-------|--|-------------------------------|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|--|
| Week  | Main focus   | 1                             | 2 | 3 | 4 | 5 | 6 | 7 | 9 | 10 | 11 | 12 | 13 | 14 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |  |
| 1-3   | Research XML, SurveyJS, given documents from client. Write "forprosjeksrapport". Workshop with client. Set up a YouTrack/Trello for Scrum-method. Define MVP for project.        |                               |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |  |
| 4-7   | Outline documentation (which will be written as we go along). Create use cases based on the MVPs and appoint prioritization. Appoint responsibilities. Set up detailed schedule. |                               |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |  |
| 8     | Winter break   |                               |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |  |
| 9-14  | Improve and add to product through iterative process. Usertest product.  |                               |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |  |
| 15    | Easter break   |                               |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |  |
| 16-20 | Improve and add to product through iterative process. Final touch-up documentation and finish project report.  |                               |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |  |
| 21    | Hand in final product  |                               |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |  |
| 22    | Prepare presentation   |                               |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |  |
| 23    | Presentation of product  |                               |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |  |

## 2. Dagens situasjon

I helsevesenet sendes pasientinformasjon mellom sykehus og fra sykehus til Kreftregisteret i form av XML meldinger i et eget helsenettverk (Norsk Helsenett). Kreftregisteret mottar flere hundre tusen slike meldinger hvert år.

De aller fleste meldinger skal gjennomgås av medisinske kodere, og av og til må informasjon i disse meldingene rettes eller kodes om.

The form is titled "Melding til nasjonalt register for prostatakreft" (Version 4.0) and "Kirurgi". It is specifically for "RADIKAL PROSTATEKTOMI" and "Meldes etter avsluttet kirurgisk behandling".

**Pasient/behandlingsinstitusjon**

Fødselsnummer\*   Ikke norsk personnummer  
Sykehus\*  Velg... Avdeling\*?

**Preoperativ informasjon**

Har pasienten vært aktivt monitoret?\*  
 Ja  Nei  
Er det gjort ny vurdering av sykdomsutbredelse (restaging) etter primer diagnose?\*  
 Ja  Nei

**Behandling**

PSA før prostatektomi og eventuell neoadjuvant endokrin behandling\*  
  Ukjent  
Er det utført neoadjuvant endokrin behandling?\*  
 Ja  Nei  Ukjent

**Radikal prostatektomi**

Operasjonsdato (dd.mm.aaaa)\*    
Kirurgi\*    
Nervesparende intensjon\*    
Er det foretatt lymfadenektomi samtidig?\*

**Patologilaboratorium\***

Laboratorium   Ikke relevant  
Velg...

Figur 1: Skjermbilde av dagens løsning med InfoPath

I dag brukes det et kommersielt program (InfoPath fra Microsoft) til å lese og vise XML meldingene til de medisinske koderne. InfoPath er "end of life" og videreutvikles ikke lenger. Det er også et nokså "frittstående" program, og kommuniserer ikke godt med Kreftregisterets andre fagsystemer som er Java-baserte.

### 3. Mål og rammebetingelser

---

Målet er å lage et system som:

- Tar imot XML-meldinger
- Viser dataene fra meldingene i et grensesnitt
- Muliggjør redigering av data fra grensesnittet
- Validatorer endret data
- Kommuniserer med resten av fagsystemene til Kreftregisteret som er Java-baserte
- Er smidig og effektiv å bruke for brukerne

Kreftregisteret har gitt noen rammebetingelser, slik som at løsningen kun skal bestå av Open-Source teknologier, nettsesarbasert front og at systemet skal være skalerbart for å kunne bli brukt med flere krefttyper. I første omgang skal løsningen kun fokusere på en krefttype, prostata. Løsninger for hvordan systemet skal få inn XML-meldingene eller hvor de skal lagres (database), skal ikke utvikles da dette i dag blir gjort av andre fagsystemer.

Utviklingen vil være modulbasert, i det tilfellet ”Proof of Concept” er vellykket og Kreftregisteret har lyst til å ta i bruk og videreutvikle systemet.

Kreftregisteret har tidligere gjort egne undersøkelser og sett på løsninger for et nytt system, og løsningsmodellen som foreslås å bruke er React og SurveyJS til front-end, Java EE og Hibernate til back-end. React er en teknologi de allerede bruker på andre områder og systemene er stort sett utviklet med Java. De er åpne om å bruke andre teknologier enn SurveyJS og Java EE, så lenge de er Open-Source. I kapittel 4 kan en lese om de andre teknologiene som ble vurdert og hva den endelige løsningen ble.

## 4. Løsninger/alternativer

Kreftregisteret sin løsningsmodell innebærer en nett løsning med React, SurveyJS (Javascript klient) og Java (tjener), der kommunikasjonen foregår over HTTP. Fordelen med denne løsningen er at den gjør det mulig å bruke deler av løsningen andre steder, da klienten og tjeneren er to frittstående deler. Ettersom at Kreftregisteret driver med mye statistisk arbeid er det en fordel at denne løsning muliggjør å generere statistikk. For å sikre at den løsningen som er anbefalt er den riktige, har vi sett på andre teknologiske alternativer for tjener og for å lage skjema.

### 4.1. Frontend

Vi har valgt React da dette bygger på det allerede eksisterende systemet til Kreftregisteret.

For å gjøre koden gjenbrukbar og enkel å vedlikeholde trenger vi et rammeverk for å lage skjema, og har vurdert disse tre rammeverkene:

- [SurveyJS](#)
- [Formik](#)
- [Formily](#)

SurveyJS er et rammeverk laget av Devsoft Baltic. SurveyJS library er gratis og open /source. Devsoft tilbyr "SurveyJS Pro" med ekstratjenester som vi hverken skal betale for eller bruke. Grunnen til at vi har valgt SurveyJS er på grunn av hvor enkelt det er å skrive lefftattelig kode sammen med React, og at det er svært lett å utvikle betingete visninger av inputfelt. Man skriver alle spørsmålene, feltene og logikken i et JSON-objekt, og renderer det i React (Devsoft Baltic OÜ, n.d.).

Formik er et rammeverk laget av Formium. Formik brukes av NASA, Nasdaq, Docker og flere store bedrifter. For å lage et skjema i Formik må man lage en "Form" på tilnærmet lik måte som i vanilla HTML5. Dette gjør at det blir mye boilerplate, og mer krevende å holde oversikt over betingelsene og logikken i skjemaet. Vi vurderer det slik at det kan bli krevende å vedlikeholde over tid på grunn av dette (Formium, n.d.).

Formily er et rammeverk laget av Alibaba. Formily sitt hovedfokus ligger i hvordan man kan lage logikk mellom de forskjellige feltene uten at ytelsen reduseres. Når man implementerer mange betingelser, kan det fort gå på bekostning av hastigheten. I følge Formily (2021), er det største problemet med rammeverket at det er svært vanskelig å sette seg inn i noe som vil bety at det krever mye ressurser og tid for å senere videreutvikle og vedlikeholde.

Kort oppsummert er fordelene og ulempene ved de tre rammeverkene slik:

| Rammeverk       | Fordeler   | Ulempar  |
|-----------------|--|--|
| <b>SurveyJS</b> | <ul style="list-style-type: none"> <li>• Skrives i JSON</li> <li>• Forholdsvis lett å vedlikeholde</li> <li>• Svært enkelt å binde felt med hverandre med betingelser</li> <li>• Lazy loading for store skjemaer</li> <li>• Kan lages i GUI</li> </ul> | <ul style="list-style-type: none"> <li>• Mindre brukt enn andre rammeverk</li> <li>• Krever at man skal lære seg et nytt rammeverk</li> <li>• Finnes rammeverk med bedre ytelse (Formily)</li> </ul> |
| <b>Formik</b>   | <ul style="list-style-type: none"> <li>• Mye brukt i industrien</li> <li>• Stabilt</li> <li>• Ikke vanskelig å sette seg inn i</li> </ul>  | <ul style="list-style-type: none"> <li>• Vanskeligere å binde mange felt sammen med betingelser</li> <li>• Dårlig ytelse</li> </ul>  |
| <b>Formily</b>  | <ul style="list-style-type: none"> <li>• Stabilt og brukt av Alibaba</li> <li>• Veldig god ytelse</li> <li>• Lett å binde felt sammen med betingelser</li> <li>• Man kan designe skjema i GUI</li> </ul>   | <ul style="list-style-type: none"> <li>• Krever svært mye opplæring</li> <li>• Vanskelig å vedlikeholde med mindre man investerer mye tid på å lære seg rammeverket</li> </ul>                       |

#### 4.2. Backend

I backend har vi valgt Java, da dette er brukt i de eksisterende systemene til Krefregisteret.

Det finnes mange måter å utvikle webapplikasjoner i Java, blant annet Spring og Java EE. Java EE virker å være litt mer komplekst enn den andre nevnte, det trenger mer boilerplate kode og mer konfigurering. Spring Boot er en variant av Spring som har betydelig mindre konfigurering og boilerplate, men kommer med den ulempen at det kan være tregere (Brown, 2021). Siden Spring Boot er et rammeverk basert på Spring, er det enkelt å utvikle REST API og mikrotjenester.

I backend er det valgt Spring Boot fordi det skal være greit å skalere, er raskt å komme i gang med på grunn av Spring Boot sin “AutoConfigure”-funksjon og fordi Krefregisteret har kompetanse på dette.

#### 4.3. Valgt løsning

Løsningen som er valgt består av følgende teknologier:

- Java 17
- Spring Boot
- Hibernate
- React
- SurveyJS

Virkningen av å bruke et rammeverk over å bruke kun JavaScript for å utvikle klientsiden vil være at resultatet blir kode som er enkel å vedlikeholde. Virkningen av å bruke et annet programmeringsspråk, som for eksempel Python ville vært at utviklingstiden kunne blitt redusert da læringskurven på språket og utviklingen av REST API er enklere. Men fordi skalering av Python-

systemer ikke er like enkelt, i tillegg til at kompetansen hos oppdragsgiver ligger i Java, ville å velge Java hjelpe Kreftregisteret å integrere vår løsning i overordnede systemer hos Kreftregisteret.

## 5. Analyse av virkninger

---

Et nytt, moderne system har mange fordeler ved seg for Kreftregisteret. En liste over fordeler ved det nye systemet kan leses nedenfor:

- En modernisert versjon av tidligere løsning som er enkel å integrere med eksisterende systemer.
- Open-Source med fri lisens.
- Krever lite opplæring for brukeren.
- Uavhengig av avviklede InfoPath.
- Spesialtilpasset Kreftregisteret sine arbeidsoppgaver.
- Enkel å vedlikeholde og videreutvikle siden teknologiene som er brukt er Open-Source og er populær blant flere bedrifter og dermed utviklere.
- Bidrar til å effektivisere arbeidet de medisinske koderne gjør.

Noen ulemper med løsningen vil være:

- Må lære seg SurveyJS
- Tar tid, ressurser og opplæring for å videreutvikle løsningen
- Kan komme uforutsette kritiske feil som fører til nedetid og bruk av ressurser

## 6. Referanser

---

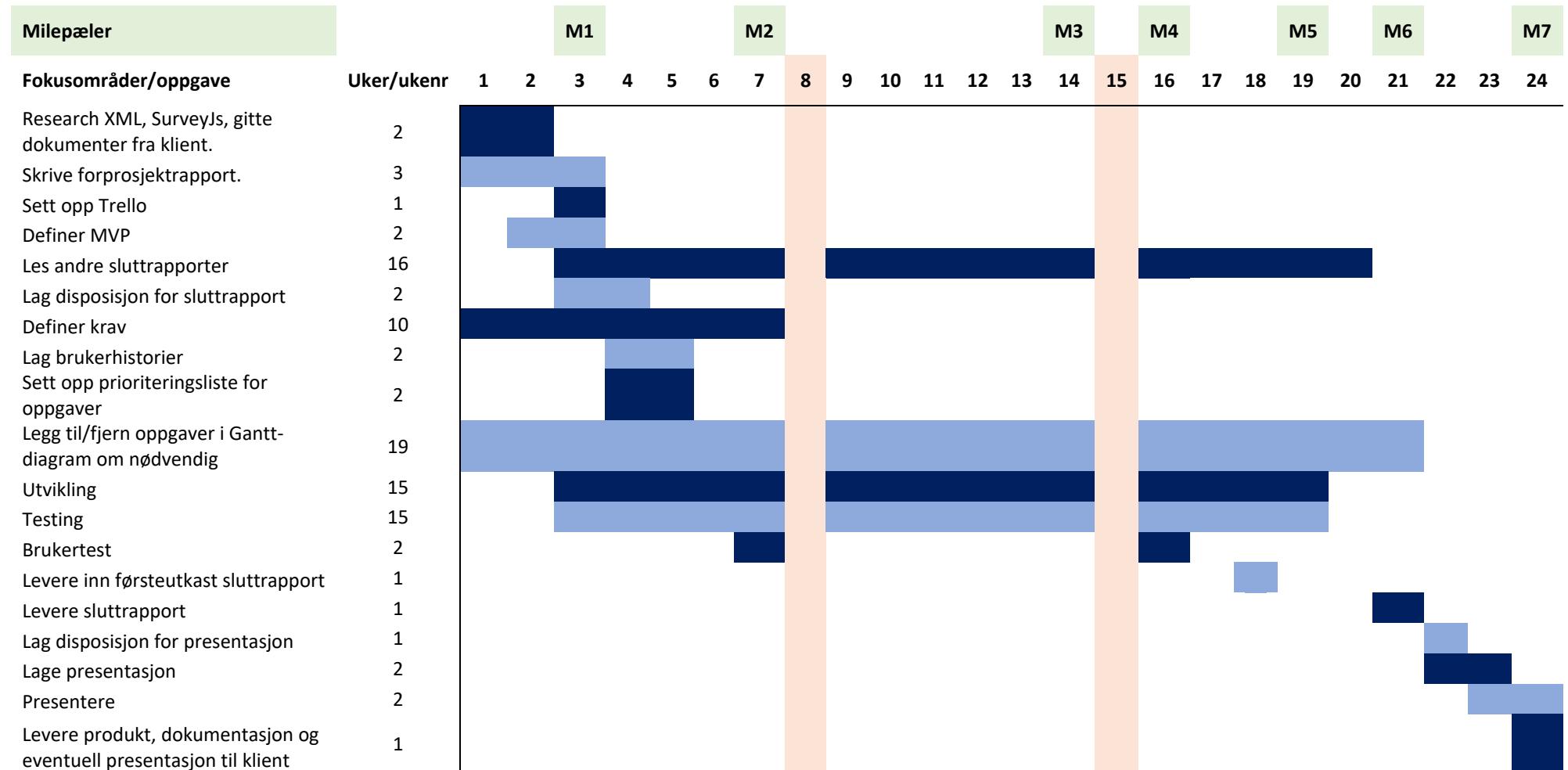
Brown, J. (2021, November 24). *Java EE vs. Spring: Which is more popular?* Xperti.io. Hentet 23. Januar 2022, fra [https://xperti.io/blogs/java-ee-vs-spring/#Java\\_EE\\_vs\\_Spring](https://xperti.io/blogs/java-ee-vs-spring/#Java_EE_vs_Spring)

Formium. (n.d.). *Overview / Formik*. Formik.Org. Hentet 23. Januar 2022, fra <https://formik.org/docs/overview>

Formily. (2021, November 24). *Competitive Product Comparison*. Formilyjs.Org. Hentet 24. Januar 2022, from <https://formilyjs.org/guide#competitive-product-comparison>

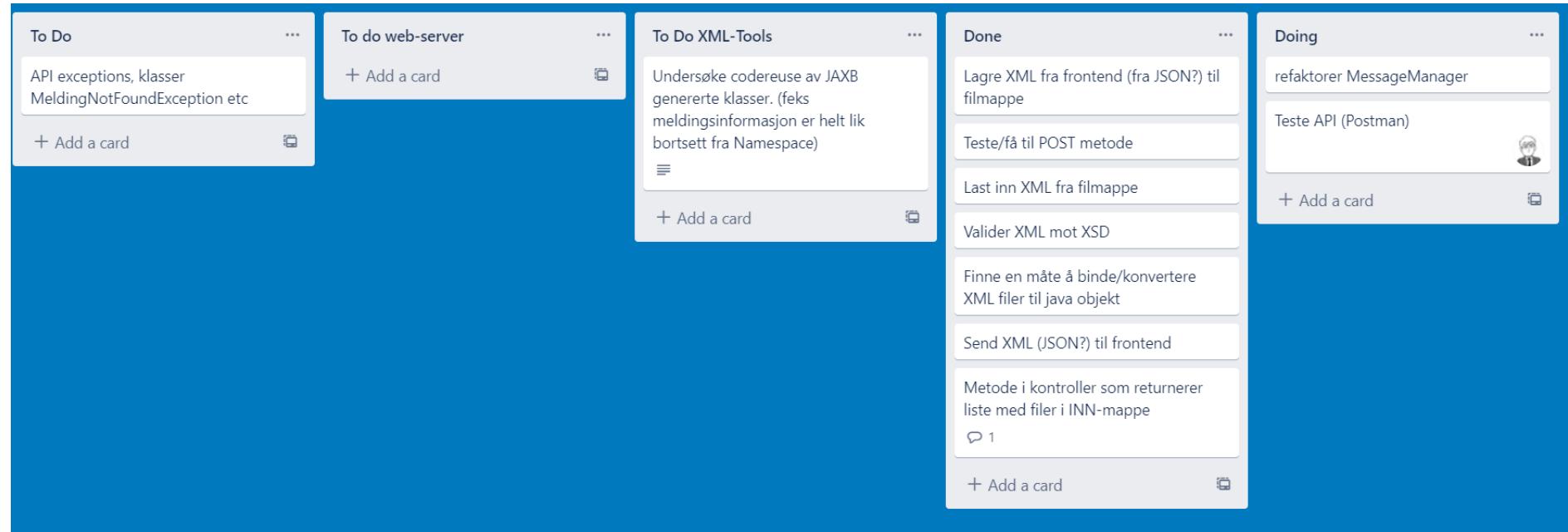
# Vedlegg D – Fremdriftsplaner

| Developement schedule - Rough |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 9 | 10 | 11 | 12 | 13 | 14 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
|-------------------------------|--|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Week                          | Main focus   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 1-3                           | - Research XML, SurveyJS, gitte dokumenter fra klient. Skrive forprosjektrapport. Workshop med klient. Sett opp en YouTrack/Trello for Scrum-metode. Definer MVP for prosjektet. |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 4-7                           | Outline documentation (which will be written as we go along). Create use cases based on the MVPs and appoint prioritization. Appoint responsibilities. Set up detailed schedule. |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 8                             | Winter break   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 9-14                          | Improve and add to product through iterative process. Usertest product.  |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 15                            | Easter break   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 16-20                         | Improve and add to product through iterative process. Final touch-up documentation and finish project report.  |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 21                            | Hand in final product  |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 22                            | Prepare presentation   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 23                            | Presentation of product  |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |

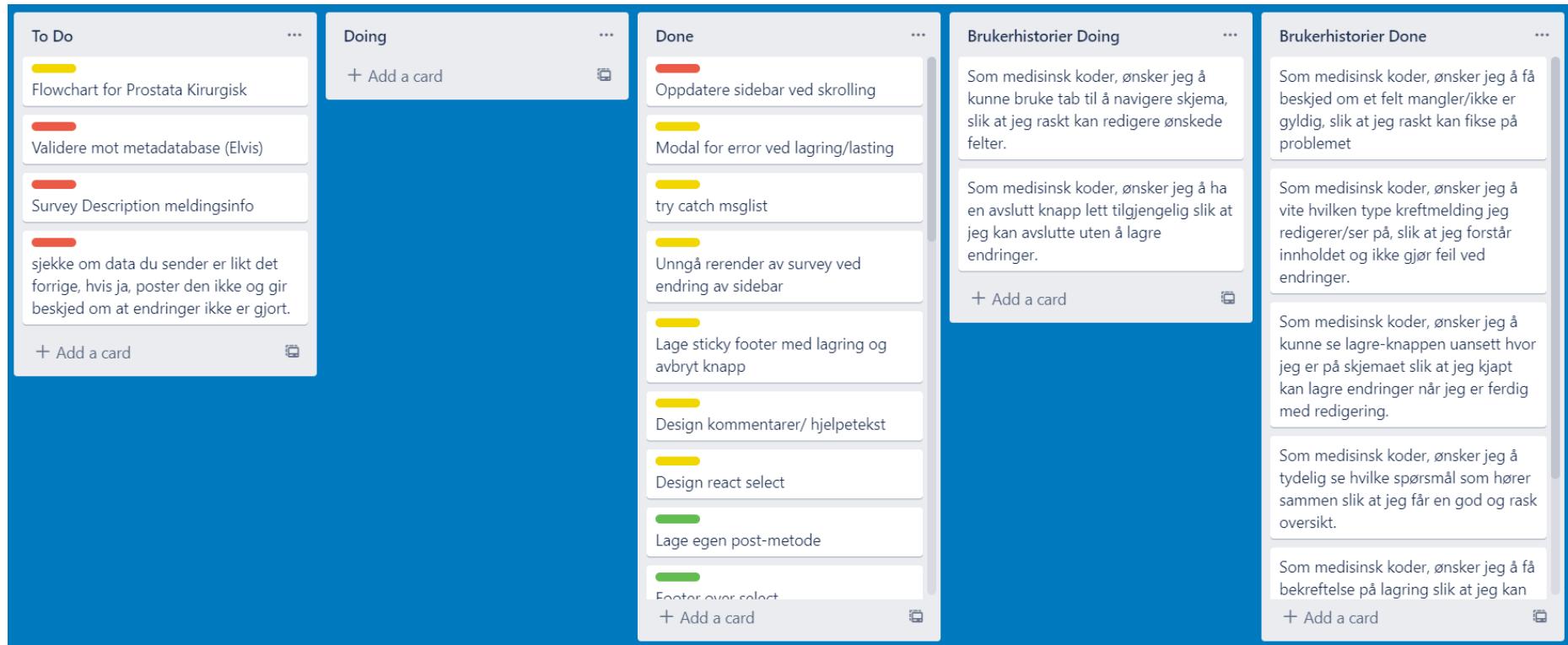


# Vedlegg E – Kanban tavler

## Backend:



## Frontend:



# Vedlegg F – Beskrivelser av Sprint

## F.1. Sprint 1

### F.1.1. Oppgaver

Frontend:

- Sette seg inn i SurveyJS
- Brukerhistorier
- Prototype i Figma
- Lage farge-palett
- Hente JSON fra backend

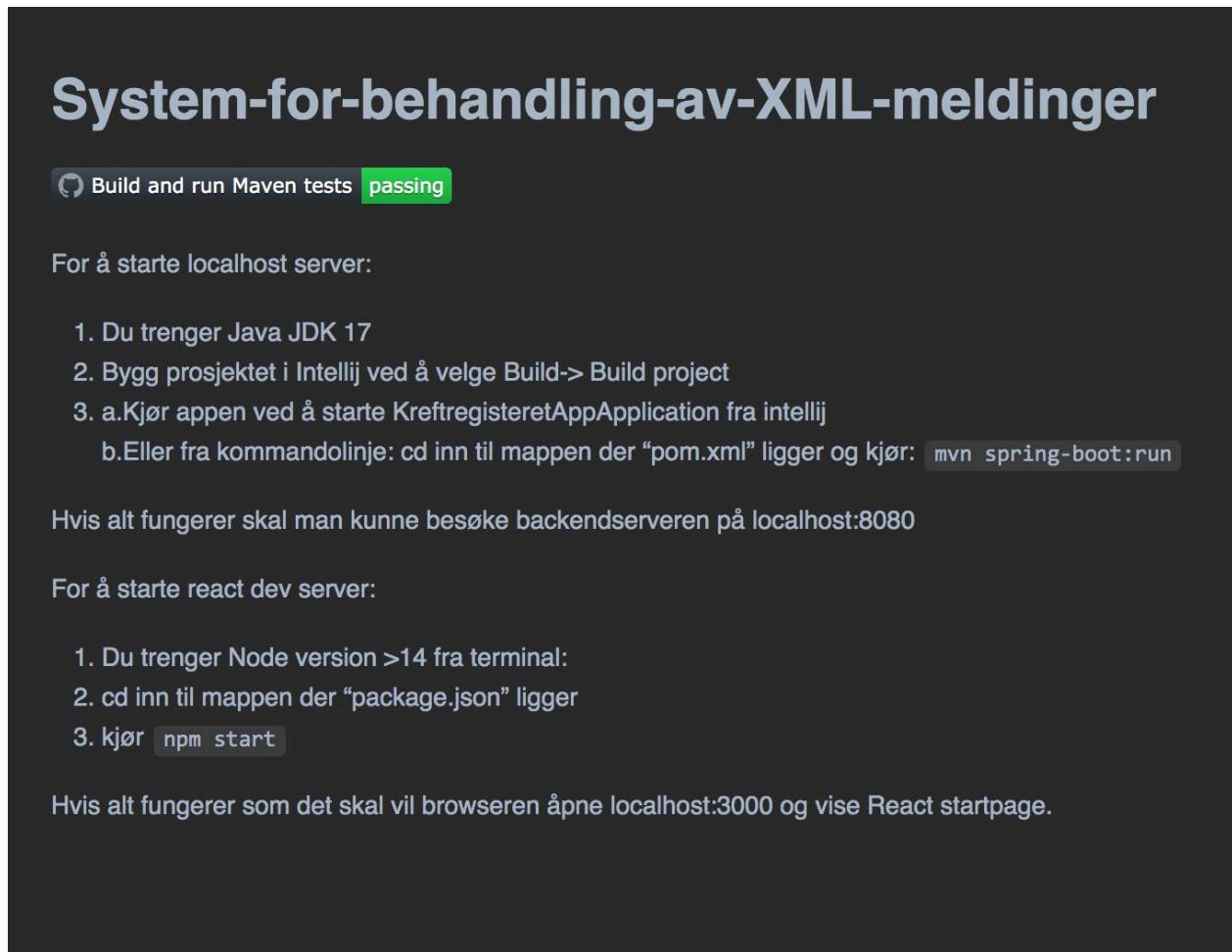
Backend:

- Definere modellklasser
- Sette opp prosjektet, definere utviklingsmiljø
- Definere RESTAPI med GET-metode
- Kunne lese .XML filer og konvertere til Javaobjekt

### F.1.2. Arbeidet

Etter å ha blitt ferdig med planleggingsfasen opprettet vi et GitHub repository med utviklingsmiljø for Spring Boot og React. Vi brukte skriptet «Create-React-App» for å sette opp et React prosjekt og Spring Boot Initializer for å generere de grunnleggende filene som trengs

for å kjøre Spring Boot som en webapplikasjon. I forbindelse med dette ble det også lagd en README-fil med beskrivelse av hvordan man kan starte applikasjonen.



Figur 94: README.md med steg for å kunne kjøre prosjektet

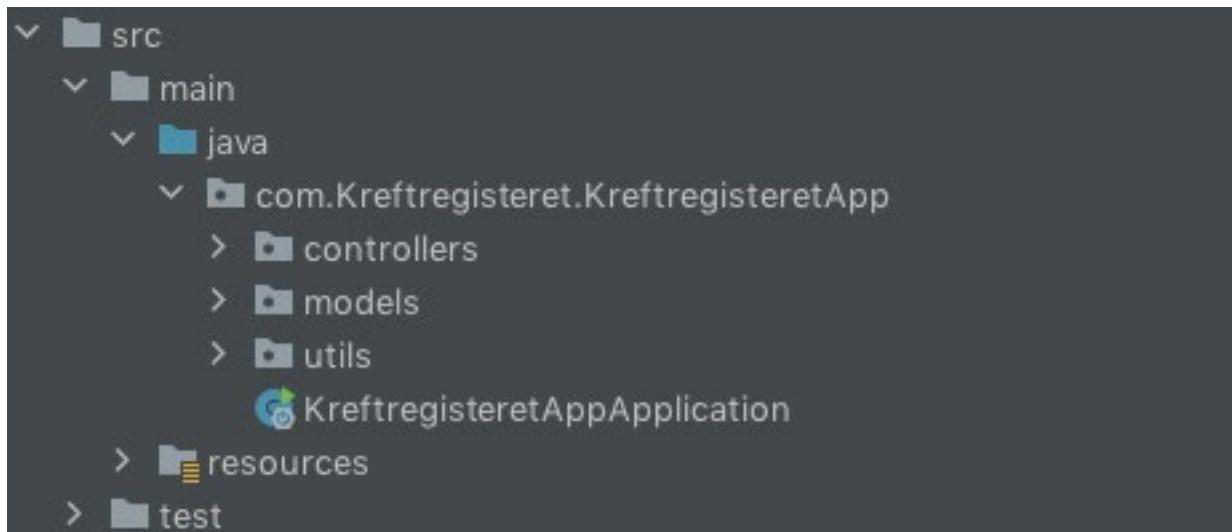
### F.1.2.1 Frontend

Vi startet med å lage brukerhistorier som kunne fungere som delmål og prototype av første skjema for å ha et eksempel å snakke rundt. Prototypen for meldingstypen «Prostata Utredning» ble gjort ferdig i løpet av den første uken og en del tid gikk med på å lære oss SurveyJs og diskutere om vi skulle implementere det i TypeScript. Vi kom frem til at React var best ettersom at dataene fra Kreftmeldingene kunne skape problemer med typene som måtte defineres i Typescript. I løpet av denne sprinten satte vi opp kommunikasjon med backend for å finne ut hvordan data kunne settes inn i skjemaene.

### F.1.2.2. Backend

Rammeverket vi brukte på backend, Spring Boot, var nokså nytt for alle på teamet, men siden alle var kjent med Java som programmeringsspråk i tillegg til å ha jobbet med REST API og backend-teknologier som Python og C#, gikk det forholdsvis greit å hoppe rett inn i utviklingen.

Vi opprettet tre mapper: Controllers, Models og Utils for å kunne dele applikasjonen opp i lag.

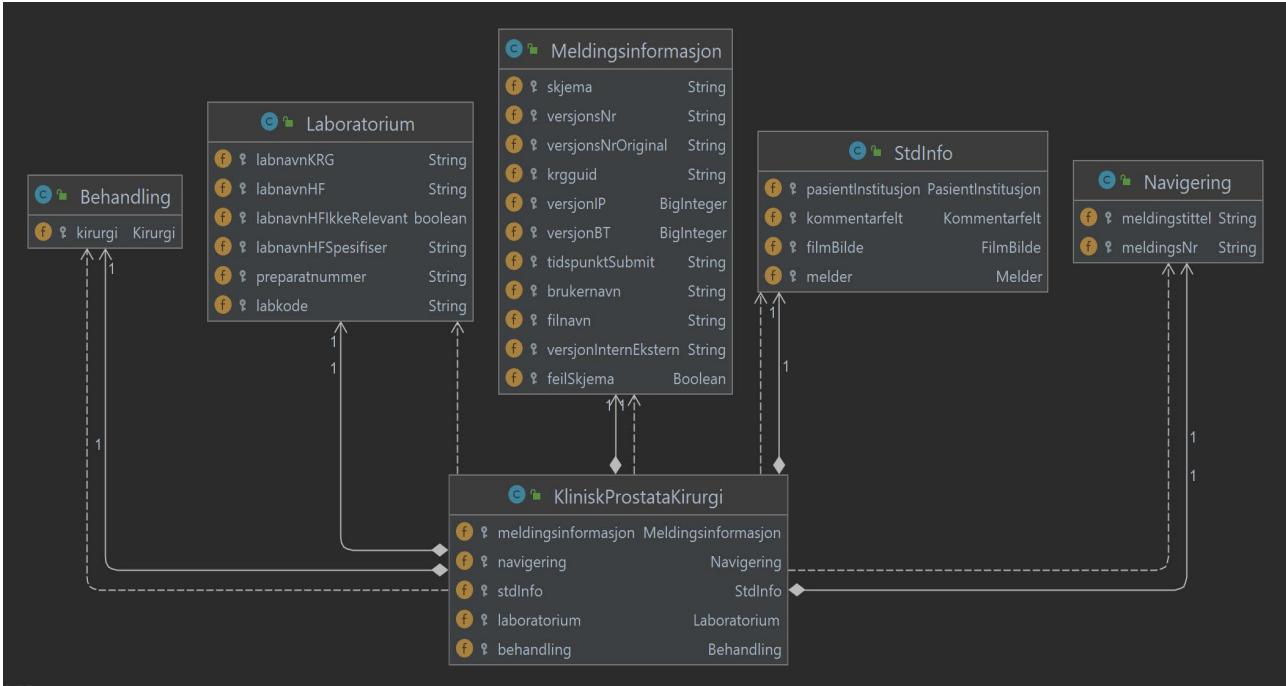


Figur 95: Mappestrukturen i Javaprosjektet.

Siden Kreftregisteret behandler mange forskjellige typer meldinger, ønsket vi å benytte polymorfisme for å knytte de sammen. Første iterasjon av systemet brukte et *interface*, «IMelding», for å få til dette, som vil si at hver klasse implementerer dette interface-et. Ved å bruke polymorfisme kan Controller klassen som inneholder REST API-metodene referere til «IMelding», som vil gjøre controller-klassen agnostisk for hvilken meldingstype som kommer inn eller returneres.

Det var et ønske tidlig i prosessen om å skrive grundige tester og i første sprint brukte vi Postman og JUnit for å undersøke om koden som kjører etter en HTTP-forespørsel fungerer som forventet. For å gjøre hverandre kjent med kodebasen etter hvert som vi programerte, hadde vi parprogrammeringssesjoner over Teams. Dette var en fin måte å lære på og kvalitetssikre koden.

I denne delen av prosessen var også sikkerhet en prioritering for oss. Grunnen var at prosjektet vårt behandler skjemaer med sensitive data og moderne informasjonsteknologi krever en høy stand for sikkerhet. Det er viktig at systemer som behandler sensitiv informasjon er varsomme og robuste, spesielt da Kreftregisteret er en statlig organisasjon.



Figur 96: Klassediagram av klassen *KliniskProstataKirurgi*

For å skape modellklasser som representerer meldingene ble det brukt JAXB. JAXB har et kommandolinjeverktøy som heter XJC som kompilerer et XML-skjema til javaklasser med annotasjoner som gjør det mulig å betrakte XML-filer som objekter. I forbindelse med dette forsto vi at hver meldingstype er en del av et komplisert hierarki av klasser, et eksempel er gitt i figur 19.

### F.1.3. Utfordringer

#### F.1.3.1. Frontend

Det var til tider utfordrende å jobbe med React, ettersom at det var nytt for flere i gruppen. Tidlig i prosessen var vi usikre på om frontend burde motta meldingene som XML-filer og

konvertere de til JSON hos klienten, men ettersom konverteringen virket triviell å gjøre med JAXB på backend, ble det bestemt at klienten skal kunne motta data som JSON.

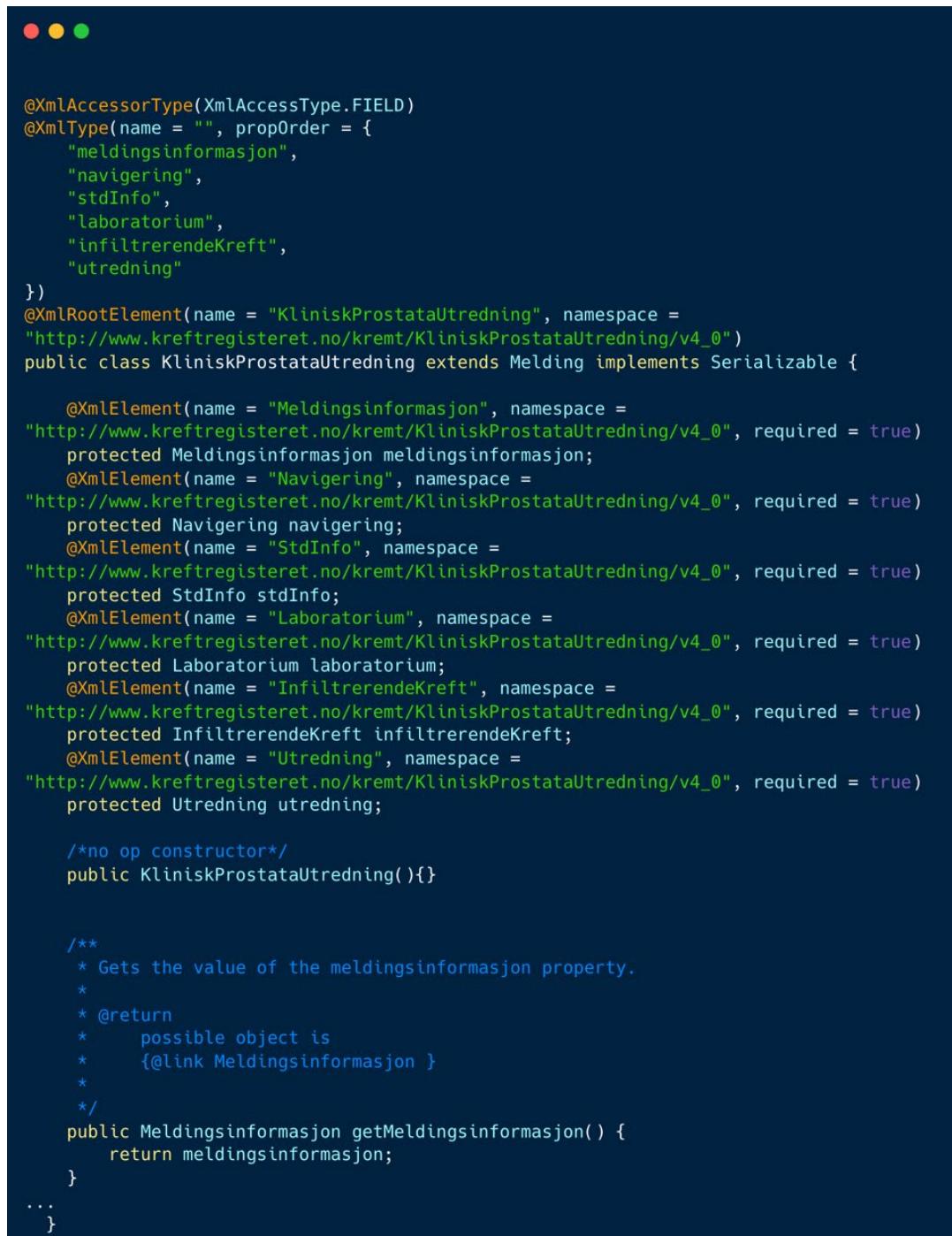
SurveyJS har et web-GUI der man kan lage og forhåndsvise skjemaer, men dersom skjemaene ble for store så stoppet GUI-et å laste. Så vi begynte å gjøre endringer direkte i fila til skjemaet. Dette var bedre siden GUI-et var tregt og uoversiktig ettersom skjemaet ble større. JSON-en vi fikk fra backend passet ikke direkte inn i skjemaet og vi måtte derfor dyppdykk inn i JSON-en og hvordan SurveyJS håndterer data for å lage metoder for innsetting av data.

### F.1.3.2. Backend

Vi forsøkte å definere modellklasser manuelt først, det vil si modellere XML-filene som klasser. Samtidig som dette undersøkte vi rundt hvordan vi ønsket å behandle XML-filene, og da oppdaget vi JAXB, et populært bibliotek for å jobbe med XML filer i Java. Først hadde vi problemer med å inkludere JAXB i systemet vårt, da dette ikke lengre er inkludert i Java sin grunnbibliotek siden versjon 8. Dette løste vi ved å manuelt importere JAXB-biblioteket i Maven som passer den javaversjonen vi bruker, Spring Boot 3 og Java 17.

Som nevnt tidligere brukte vi JAXB for å generere modellklasser. Det som frembringes ved hjelp av JAXB er en rekke Javaklasser, som stemmer med XML-skjema hierarkiet. Et problem som resultat av generering av filene ved hjelp av JAXB er at hvert skjema har egne «targetNamespace». Det vil at meldingstypen Utredning har et klassemedlem Meldingsinformasjon med namespace «Utredning», og meldingstypen Kirurgi har et klassemedlem Meldingsinformasjon med namespace «Kirurgi». XSD-filene (XML Schema) vi fikk fra Kreftregisteret tar i bruk «Heterogeneous Namespace Design» (L. Costello, 2015), som vil si at navngivingen i hvert skjema bruker forskjellige navn, selv om de refererer til samme klasse. Dette fører til at modellklassene får en del duplikater. Hvis vi skulle unngått dette, og følge DRY-prinsippet, burde vi ifølge dokumentasjonen (JAXB Users Guide, 2009) skrive om XML-skjema

definisjonen, noe som vi ikke har tilgang til å gjøre, da XSD filene definerer standarden for disse datatypene.



The screenshot shows a Java code editor with a dark theme. At the top, there are three colored circular icons (red, yellow, green). The code itself is a Java class named `KliniskProstataUtredning` that implements the `Serializable` interface. The class contains several fields annotated with `@XmlElement`, each with a specific namespace defined in the XML schema. The fields include `Meldingsinformasjon`, `Navigering`, `StdInfo`, `Laboratorium`, `InfiltrerendeKreft`, and `Utredning`. The code also includes a constructor and a getter method for the `Meldingsinformasjon` field, which returns an object of type `Meldingsinformasjon`.

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
    "meldingsinformasjon",
    "navigering",
    "stdInfo",
    "laboratorium",
    "infiltrerendeKreft",
    "utredning"
})
@XmlRootElement(name = "KliniskProstataUtredning", namespace =
"http://www.kreftregisteret.no/kremt/KliniskProstataUtredning/v4_0")
public class KliniskProstataUtredning extends Melding implements Serializable {

    @XmlElement(name = "Meldingsinformasjon", namespace =
"http://www.kreftregisteret.no/kremt/KliniskProstataUtredning/v4_0", required = true)
    protected Meldingsinformasjon meldingsinformasjon;
    @XmlElement(name = "Navigering", namespace =
"http://www.kreftregisteret.no/kremt/KliniskProstataUtredning/v4_0", required = true)
    protected Navigering navigering;
    @XmlElement(name = "StdInfo", namespace =
"http://www.kreftregisteret.no/kremt/KliniskProstataUtredning/v4_0", required = true)
    protected StdInfo stdInfo;
    @XmlElement(name = "Laboratorium", namespace =
"http://www.kreftregisteret.no/kremt/KliniskProstataUtredning/v4_0", required = true)
    protected Laboratorium laboratorium;
    @XmlElement(name = "InfiltrerendeKreft", namespace =
"http://www.kreftregisteret.no/kremt/KliniskProstataUtredning/v4_0", required = true)
    protected InfiltrerendeKreft infiltrerendeKreft;
    @XmlElement(name = "Utredning", namespace =
"http://www.kreftregisteret.no/kremt/KliniskProstataUtredning/v4_0", required = true)
    protected Utredning utredning;

    /*no op constructor*/
    public KliniskProstataUtredning(){}
}

/**
 * Gets the value of the meldingsinformasjon property.
 *
 * @return
 *     possible object is
 *     {@link Meldingsinformasjon }
 */
public Meldingsinformasjon getMeldingsinformasjon() {
    return meldingsinformasjon;
}
...
}
```

Figur 97: Eksempel på klasse generert ved hjelp av XJC, hvor namespace for klassemeldem er spesifikt for klassen.

## F.2. Sprint 2

### F.2.1. Oppgaver

Frontend:

- Validering av input i skjema for utredning
- Generalisering av inputkontrollere
- Starte på flere skjematyper

Backend:

- Skrive melding sendt fra klient til disk
- Definere RESTAPI med POST-metode
- Validering
- Testing av validering og POST-metode

### F.2.2. Arbeidet

Ved starten på denne sprinten var vi allerede godt i gang med systemet, og hadde fått utviklet grensesnittet for det første skjemaet, «KliniskProstataUtredning», som henter en melding i JSON-format fra serveren. I et av møtene med oppdragsgiver, spurte Kreftregisteret om vi kunne lage en landingsside eller «dashboard» for prosjektet, det vil si en liste over tilgjengelige meldinger som gjør at bruker kan velge en melding som skal redigeres.

#### F.2.2.1. Frontend

Ettersom oppdragsgiver ønsket en landingsside der brukeren kan velge en melding ut ifra en liste med meldinger, satte vi opp en prototype for hvordan dette skulle se ut også lagde en midlertidig løsning. Arbeidet med flere skjematyper gikk enklere enn for det første skjemaet, da teamet hadde lært seg verktøyene for å utvikle grensesnittet.

#### F.2.2.2. Backend

For å møte kravene fra oppdragsgiver, la vi til 2 nye endepunkt i Controller-en, /api/meldinger, som gir ut informasjon om alle meldinger som er tilgjengelige, og /api/meldinger/{id}, som henter ut all data relatert til den spesifikke meldingen.

I løpet av forrige sprint startet vi på programmeringen av klassen «MeldingManager», som er en serviceklasse som har ansvaret for å lagre meldinger til disk, og opprettholde en liste over tilgjengelige meldinger. Fordi prosjektet vårt skal resultere i moduler som kan innlemmes i et større system, ønsket ikke oppdragsgiver at vi implementerte en databaseløsning, men å skrive filene til disk. Denne listen kan sees på som en database i RAM, slik at vi ikke henter ut meldingene fra disk hver gang klienten trenger de. I praksis så var listen et HashMap med meldingen som verdien, og en ID som nøkkel. Vi måtte finne en løsning på å unikt identifisere meldinger, slik at klienten unikt kan referere til en melding – og dette ble gjort ved hjelp av en ID skapt i serviceklassen MeldingManager.

Idet vi lagde modellklassene, kom vi opp i en diskusjon om hvorvidt vi skulle anvende interface eller abstract klasser for alle typer meldinger som går gjennom systemet. Forskjellen på et interface og en abstract klasse er at en abstract klasse kan ha funksjonalitet som subklassene kan implementere eller overskrive. En interface kan kun definere funksjonalitet, ikke implementere det. Vi landet derfor på å bruke en abstrakt klasse, Melding, som alle meldingstypene må anvende.

*Heroku* en «Platform as a Service», gjør det mulig å distribuere web-applikasjoner, og ble for vårt formål brukt for å demonstrere applikasjonen for Kreftregisteret. Heroku kobler seg til Github, og pakketerer løsningen slik at hvem som helst kan besøke en demoversjon av applikasjonen på en gitt url.

### F.2.3. Utfordringer

#### F.2.3.1. Frontend

SurveyJS har en innebygd logikk for å kunne mate inputkontroller med data. Men enkelte av disse kontrollene, for eksempel avkryssingsboks ville ikke enkelt innpasses med meldingene. Dette førte til at vi skrev en tilpasset funksjon som behandler avkryssingsboksene.

SurveyJS har også innebygd funksjon for å anvende REGEX. Hvis man hadde «inputType = number», så kan man skrive inn tall pluss komma og punktum. Men dette fungerte ikke siden SurveyJS konverterer komma til punktum slik at det blir datatype nummer. Dette betyde at da vi sjekket om det var gyldig input så var det ikke lenger komma, men punktum og REGEX-valideringen feilet, siden vi sjekket om inputen hadde komma. Hadde også lignende problem med ledene 0. Når man skrev inn et personnummer med ledene 0, så ble denne borte når JS konvertere tallet til Number, siden når det blir skrevet inn så er det en streng. Første problemet ble løst ved å sjekke om inputen hadde punktum istedenfor komma og det andre ble løst ved å sette feltet til å ha «inputType = text», som er standard.

#### F.2.3.2. Backend

Ettersom JUnit var et forholdsvis nytt testrammeverk for gruppen, tok det litt tid å sette oss inn i test-vokabularet og mock-testing, som var nødvendig for å kunne grundig teste metodene i controlleren.

Fordi vi ønsket å distribuere systemet til Heroku, fant vi ut at å skrive til og hente fra lokal disk var vanskeligere enn det det først så ut som. De fleste webapplikasjoner anvender en eller annen form for databaseløsning, som kan enkelt integreres med Heroku. Men i vårt tilfelle måtte vi finne en bra måte å kunne stabilt, konsist og sikkert skrive XML-filene til disk. (Mer om dette i produktpresentasjon)

Under sprint 2 hadde vi diskusjoner om det var korrekt å anvende POST-operasjon i vår sammenheng, eller om en PATCH-operasjon hadde blitt mer effektiv å bruke. En PATCH-operasjon vil kun sende inn de feltene/elementene som har blitt endret på, for å dermed endre et helt dokument. Etter litt diskusjon med oppdragsgiver kom vi frem til at hver melding som rettes/endres på skal lagres som et nytt dokument, dette for å ha muligheten for å lagre versjoner av dokumentet.

Fordi vi ønsket å bruke en abstrakt klasse, måtte vi finne måter å fortelle JAXB og Spring Boot hvilke klasser som er del av denne abstrakte klassen, fordi JAXB i utgangspunktet ikke har innebygd støtte for polymorfisme. Dette gjøres ved å bruke annotasjoner som forteller JAXB hvilke klasser som tar del av denne abstrakte klassen.

## **F.3. Sprint 3**

### **F.3.1. Oppgaver**

Frontend:

- Sidebar med navigerbare titler
- Validere mot metadatabase
- Forbedre API-kall og responshåndtering
- Footer med handlingsknapper

Backend:

- Forbedre statuskoder sendt til API forespørslar
- API exceptions
- Deployere en demo av prosjektet til Heroku

### **F.3.2. Arbeidet**

#### **F.3.2.1. Frontend**

For å utvikle sidebar med navigerbare titler ble det utført en brukertest, der brukerne fikk si hvilken av de forskjellige varianter av designet de likte best. Brukerne ga også tilbakemeldinger på hvordan de ville ha handlingsknappene. Disse og resten av designet ble i denne sprinten implementert.

Krefregisteret jobber kontinuerlig med en metadatabase, som inneholder metadata relatert til alle verdier og alle skjema som de håndterer. På grunn av at vårt bruksområde av skjema og verdier er nokså «nytt» i forhold til hvordan Krefregisteret har behandlet dette tidligere, viste

det seg at metadatabasen i utgangspunktet ikke var lagd for å håndtere validering av enkelt-verdier slik vi trengte det.

#### **F.3.2.1. Backend**

I den siste sprinten gjorde vi ferdige oppgaver som hadde blitt utsatt på grunn av lav prioritet.

Det ble også brukt tid på refaktorering med tanke på navngivning og gjenbrukbar kode.

### **F.3.3. Utfordringer**

#### **F.3.3.1. Frontend**

Å overskrive CSS for inputkontrollene i SurveyJS var ikke så enkelt som vi skulle tro. Blant annet fordi SurveyJS har dynamiske ID-er på elementene og vi måtte derfor bruke klassenavn for å referere til inputkontrollene. Klassenavnene var på flere underelementer også og gjorde det problematisk å kun endre en ting ved et element, uten å endre på underelementene med samme klassenavn. For eksempel har hvert kort klassenavn «sv\_row» og hvert felt inne i kortene har også klassenavn «sv\_row». Løste dette ved å finne elementet med «sv\_row» klassenavn som hadde «sv\_p\_root» som foreldrenode.

Et annet problem med design av skjemaet var å sette inn informasjonsknapp. SurveyJS har ikke funksjonalitet for dette som er dynamisk nok og designet for vårt behov. Forklaringene til Kreftregisteret er også lange og gjorde at designet så rotete ut. Vi minsket derfor skriftstørrelsen betraktelig på disse, slik at de ikke fikk så mye fokus. Tanken er at en som jobber med dette skjemaet flere ganger om dagen, ikke nødvendigvis trenger disse beskrivelsene.

Å finne en løsning for å følge med på endringer i DOM-en også oppdatere navigasjonsbaren deretter, viste seg også å være utfordrende. Fordi løsningene ofte førte til at ytelsen til applikasjonen gikk betraktelig ned og det kunne vi ikke tillate. Etter mye leting fant vi «MutationObserver» som hører etter endringer uten å påvirke ytelse.

#### **F.3.3.2. Backend**

I den første sprinten valgte vi å lagre tilgjengelige meldinger i et *HashMap*, en datastruktur der hver verdi er knyttet til en nøkkel, men under videreutviklingen av modellklassene, fant vi ut at vi kan la den abstrakte klassen *Melding* inneholde denne unike *IDen*. På denne måten trengte vi ikke lengre å bruke *HashMap*, og på grunn av å forenkle systemet endret vi denne samlingen meldinger til å være en *ArrayList* som inneholder meldinger.

På grunn av at klassedefinisjonene egentlig er noe som er definert i XSD filer, tidligere utviklet av Kreftregisteret, vil det være navnekonvensjoner som er gitt på norsk. På grunn av dette måtte vi ta noen valg om hvordan vi ville navngi nye felt, der vi landet på at vi fortsetter å bruke engelske navn på metoder og felt, men at selve ressursen, det vil si XML-meldingene Kreftregisteret behandler, med medisinsk terminologi, vil fortsatt være på norsk.



# Vedlegg G – Brukertest 1 - Skjema

Spørsmål Svar 8 Innstillinger

## Design av nytt meldingsskjema

Meldingsskjemaene skal få et nytt utseende og du kan være med å bestemme hvordan de skal se ut!

Hvor godt liker du dette designet?

\*

**Utredning**  
Melding etter avsluttet utredning

Pasient/behandlingsinstitusjon  
Fødselsnummer   Ikke norsk personnummer

Fullt navn

Sykehus   
Vlg...

Audeling   
Vlg...

**Funn i utredningen**

Primærtumor (med eller uten metastase)?    
 Kun metastase(s)?

**Sykehistorie**

PSA-verdi ved diagnostikkspunkt   Ikke testt.  Ukjent

Var varianterproblemer en del av årsak til utredningen?  Ja  Nei  Ukjent

Var kreftsymptomer (smerte, anemi eller annet) en del av årsak til utredningen?  Ja  Nei  Ukjent

1

2

3

4

5

Svært lite



Svært godt

Hvor godt liker du dette designet? \*

Melding til Kreftrådet register for primærtumor  
Mengde 4.3



PasientBehandlingsinstitusjon

Funn i utredning

Sykehistorie

Diagnostikk av primærtumor

Diagnostikk av primærtumor

PasientBehandlingsinstitusjon

Funn i utredning

Sykehistorie

Diagnostikk av primærtumor

PasientBehandlingsinstitusjon

Avbryt

Lagre

## Melding etter avsluttet utredning

### Pasient/behandlingsinstitusjon

Fødselsnummer

Ikke norsk personnummer

Fullt navn

Sykehus

Vælg...

Avtale:

Vælg...

### Funn i utredningen

Primærtumor (med eller uten metastase(r))

Kun metastase(r)

### Sykehistorie

PSA-verdi ved diagnostispunkt

Vælg... Ikke satt Ukjent

Var varmeflyttsymptomer en del av årsak til utredningen?

Ja  Nei  Ukjent

Var kreftsymptomer (smerte, anemi eller annet) en del av årsak til utredningen?

Ja  Nei  Ukjent

Funksjonsstatus/WHO-status ved diagnose

Vælg...

Postatavolum

Vælg... mL Ikjent

1

2

3

4

5

Svært lite

Svært godt

Hvor godt liker du dette designet? \*

Melding til pasient/raportør fra pasienten  
Versjon 4.0



PasientBehandlingsinstitusjon

Funn i utredning

Sykehistorie

Diagnostikk av primærtumor

Diagnostikk av primærtumor

PasientBehandlingsinstitusjon

Funn i utredning

Sykehistorie

Diagnostikk av primærtumor

PasientBehandlingsinstitusjon

## Melding etter avsluttet utredning

### Pasient/behandlingsinstitusjon

Fødselsnummer

Ikke norsk personnummer

Fullt navn

Syklus

Audeling

Voli...

### Funn i utredningen

Primærtumor (med eller uten metastase(r))  
+

Kun metastase(r)  
+

### Sykehistorie

PSA-verdi ved diagnostikkpunkt

Ikke satt

Ukjent

Var vannfløtingsproblemer en del av årsak til utredningen?

Ja       Nei       Ukjent

Var kreftsymptomer (smerte, anemi eller annet) en del av årsak til utredningen?

Ja       Nei       Ukjent

Funksjonsnivå/WHO-status ved diagnose

Voli...

Postavolum

ml      l      litron

1

2

3

4

5

Svært lite



Svært godt

Hvor godt liker du dette designet? \*

Kreft  
registeret

## Utredning

Melding etter avsluttet utredning

Melding til nasjonalt register for prosesskraft  
Versjon 4.0  
Utredning

Pasient/Behandlingsinstitusjon

Fødselnummer:

Fullt navn:

Sykehus:

Vælg...

Avteling<sup>2</sup>:

Vælg...

Funn i utredningen:

Primærtumor (med eller uten metastase(r))  
 Kun metastase(r)

Sykehistorie

PSA-verdi ved diagnosetepunkt:

Avbryt

Lagre



1      2      3      4      5

Svært lite



Svært godt

Hvor godt liker du dette designet? \*

+++

Kreft  
registeret

**Utredning**  
Melding etter avsluttet utredning

Pasient/behandlingsinstitusjon

Fødselnummer  
  
 Ikke norsk personnummer

Full navn

Sykehus  
Velg...

Andeling  
Velg...

Funn i utredningen

Primærtumor (med eller uten metastase(s))  
 Kun metastase(s)

Sykehistorie

PSA-verdi ved diagnostispunkt  
  Ikke satt  Usjett

Melding til pasientregister for prosesskoden  
Versjon 4.0  
Utredning

PasientBehandlingsinstitusjon  
Funn i utredning  
Sykehistorie  
Diagnostikk av primærtumor  
Diagnostikk av primærtumor  
PasientBehandlingsinstitusjon  
Funn i utredning  
Sykehistorie  
Diagnostikk av primærtumor  
PasientBehandlingsinstitusjon

1            2            3            4            5

Svært lite



Svært godt

Jeg vil ha lagre og avbryt-knappen.. \*

- .. nederst i skjemaet
- .. fast på venstre side
- .. fast på høyre side
- .. fast nederst på siden
- .. fast øverst på siden

Jeg vil ha mer nøytrale farger

- Ja
- Nei
- Ikke så viktig for meg

Kommentarer og/eller forslag

Kort svartekst

---

# Vedlegg H – Brukertest 1 Svar

| Tidsmerke                  | Hvor godt liker du dette designet? | Jeg vil ha lagre og avbryt-knappen.. | Jeg vil ha mer nøytrale farger | Kommentarer og/eller forslag |
|----------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|--------------------------------------|--------------------------------|------------------------------|
| 18.03.2022<br>kl. 09.23.51 | 2                                  | 3                                  | 2                                  | 5                                  | 4                                  | .. fast nederst<br>på siden          | Nei                            |                              |
| 18.03.2022<br>kl. 09.24.14 | 3                                  | 4                                  | 3                                  | 5                                  | 4                                  | .. fast nederst<br>på siden          | Ikke så viktig<br>for meg      |                              |
| 18.03.2022<br>kl. 09.24.35 | 3                                  | 3                                  | 5                                  | 3                                  | 3                                  | .. fast nederst<br>på siden          | Ikke så viktig<br>for meg      |                              |
| 18.03.2022<br>kl. 09.25.27 | 4                                  | 5                                  | 3                                  | 2                                  | 1                                  | .. fast på<br>venstre side           | Ikke så viktig<br>for meg      |                              |
| 18.03.2022<br>kl. 09.25.43 | 3                                  | 3                                  | 4                                  | 2                                  | 2                                  | .. fast nederst<br>på siden          | Ikke så viktig<br>for meg      |                              |
| 18.03.2022<br>kl. 09.28.09 | 2                                  | 2                                  | 3                                  | 5                                  | 4                                  | .. fast på høyre<br>side             | Ikke så viktig<br>for meg      |                              |
| 18.03.2022<br>kl. 09.42.24 | 2                                  | 1                                  | 4                                  | 5                                  | 4                                  | .. fast nederst<br>på siden          | Nei                            |                              |
| 18.03.2022<br>kl. 09.44.24 | 2                                  | 3                                  | 3                                  | 5                                  | 4                                  | .. fast nederst<br>på siden          | Nei                            |                              |

# Vedlegg I – PP Brukertest/demo

## Demo av nytt meldingsskjema

18.03.2022

### Innhold

Hjem er vi  
Design  
Demo  
Veien videre

# **Hvem er vi?**

**Fem dataingeniørstuderter fra OsloMet**

**Jørund Ola, Nikola, Tom og Hajin**

# Design – Forslag til løsninger

Fortsatt lang med mye skrolling

**Løsning** Trykkbare overskrifter

Mye info

**Løsning** Fokus på designprinsipper for å få rask oversikt: Inputfeltet på en linje med overskrifter på toppen.

Ser utdatert ut og UU

**Løsning** Moderne og behagelige farger plukket fra Kreftregisteret sine sider og øker kontraster.

## Design - nåværende

Utfordringer/problemer:

- For lang → Mye skrolling
- Mye info → Vanskelig å få en rask oversikt
- Ser utdatert ut → Ikke i henhold til UU-standard

Funksjonstilstand/WHO-status ved diagnose\*  
Velg...

Prostatavolum\*  
ml  Ukjent

**Diagnostikk av primærtumor**

Er det gjort MR prostata?\*  
 Ja  Nei

Diagnosereddato (dd.mm.aaaa)\* ?

Diagnosedato (dd.mm.aaaa)\* ?

Ikke relevant

**primærutredning og før primærbehandling**

Samlet palpatorisk T-stadium  
Velg...

Palpatorisk T-stadium venstre side (laveste stadium velges ved tvil)\*  
Velg...

Totalvurdering av klinisk T (laveste stadium velges ved tvil)\*  
TNM på diagnosetidspunktet baseres på all relevant utredning som er gjort før primærbehandling inkludert

## **Tilbakemelding fra dere brukere**

<https://forms.gle/sgvdeCGgHSHH4nCM7>

**Demo**

## **Veien videre**

- Effektivitet
- Testing
- Utvikle resterende skjemaer
- Design

**Takk for oss**

# Vedlegg J – Brukertest 2 - Skjema

Del 1 av 2

## Brukertest meldinger: Del 1 - Endre/rette på melding

For å gjøre arbeidet med å se gjennom og rette/endre meldinger mer effektivt trenger vi hjelp fra deg som bruker. Selv om skjemaet ser ferdig ut, så er det fortsatt mulig å gjøre endringer og gjøre den enda bedre, så ikke nøy med tilbakemeldingene.

Testen består av to deler. I denne delen av testen får du først se en demonstrasjon av skjemaet også svarer du på noen spørsmål knyttet til skjemaet.

Hvilke endringer la du merke til fra dette skjemaet og tidligere skjema? \*

Lang svartekst

Hva mener du var bra med det nye skjemaet? Begrunn svaret. \*

Lang svartekst

Hva var bedre med det forrige skjemaet?

Lang svartekst

Hvilke eventuelle utfordringer så du med det nye skjemaet for arbeidet ditt? \*

Lang svartekst

## Brukertest Meldinger: Del 2 - Navigering

⋮

I denne delen skal du få se en demonstrasjon av hvordan man kan bruke sidebaren til å navigere. Du skal så vurdere bruken av denne navigasjonsbaren.

Hvilke fordeler og ulemper har sidebaren for arbeidet ditt? \*

Lang svartekst

Hvor sannsynlig er det at du kommer til å bruke sidebaren? \*

1      2      3      4      5

Ikke sannsynlig

Svært sannsynlig

Andre tilbakemeldinger/ønsker

Lang svartekst

# Vedlegg K – Brukertest 2 - Svar

| Tidsmerke               | Hvilke endringer la du merke til fra dette skjemaet og tidligere skjema?    | Hva mener du var bra med det nye skjemaet? Begrunn svaret.  | Hva var bedre med det forrige skjemaet? | Hvilke eventuelle utfordringer så du med det nye skjemaet for arbeidet ditt? | Hvilke fordeler og ulemper har sidebaren for arbeidet ditt?  | Hvor sannsynlig er det at du kommer til å bruke sidebaren? | Andre tilbakemeldinger/ønsker  |
|-------------------------|---|---|---|--|--|--|--|
| 25.04.2022 kl. 10.24.27 | Layout (farger), navigasjon   | Lagring følger vist bilde. Navigasjon   | Lite                                    | Noen felt tok mye bredde. Justerbart?  | Raskt å finne det du er ute etter. Ingen ulemper.  | 5  | Tror det vil være sannsynlig at jeg tabber gjennom felt, men klikke med musepeker på sidebar                 |
| 25.04.2022 kl. 10.27.59 | Freshere design (tidl. skjema ser utdatert ut). er litt mer strukturert ut. | Lysere side, mer luft, fint med navigasjon på høyre side, bedre med endring i nedtrekksmeny (eks. T2 - [tekst enn [tekst] (T2)]). | Skjulte informasjon underveis.          | Pr. nå ser jeg ingen utfordringer.   | Fordelen med sidebaren er at det gjør skjemaet mer oversiktlig for de som fyller ut skjemaet , men "heading" burde utherves etterhvert som man jobber seg gjennom skjemaet så det også gir en oversikt over hvor langt man har kommet i utfylling.<br>Ulempen er at den tar plass, men den kan vises/skjules. Optimalt sett burde den tilpasses slik at den alltid er synlig hvis mulig, men det er nok en individuell smakssak. | 2  | Internt blir nok ikke sidebaren benyttet så mye siden vi allikevel må se over alt som er fylt ut ved mottak. |

# Vedlegg L – Endelig design



## Melding etter avsluttet utredning

### Pasient/behandlingsinstitusjon

Fødselsnummer

Ikke norsk personnummer

Navn \*

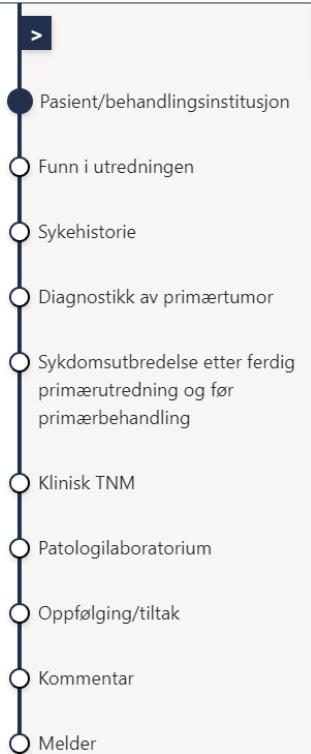
Sykehus \*

Avdeling \*

Dette er en standard avdelingsliste for Kreftregisteret. Dersom din spesifikke avdeling ikke finnes i listen, velg den avdelingen som passer best ut fra alternativene

### Funn i utredningen

- Primærtumor (med eller uten metastase(r)) - Utredning av primærtumor og samtidig utredning av regionale lymfeknutemetastaser/fjernmetastaser før igangsetting av primærbehandling
- Kun metastase(r) - Utredning av regionale lymfeknutemetastaser og/eller fjernmetastaser uten samtidig utredning av primærtumor



Saklig informasjon

\* Obligatorisk

Lagre

Lagre utkast

Avbryt

**Sykehistorie**

PSA ved diagnostispunkt  
  Ikke tatt  Ukjent

Var vannlatingsproblemer en del av årsak til utredningen? \*

Ja  Nei  Ukjent

Var kreftsymptomer (smarter, anemi eller annet) en del av årsak til utredningen? \*

Ja  Nei  Ukjent

Funksjonstilstand/WHO-status ved diagnose \*

4: Totalt stillesittende eller i seng hele dagen

Prostatavolum (mL)  
  Ukjent

**Diagnostikk av primærtumor**

MR prostata \*

Ja  Nei

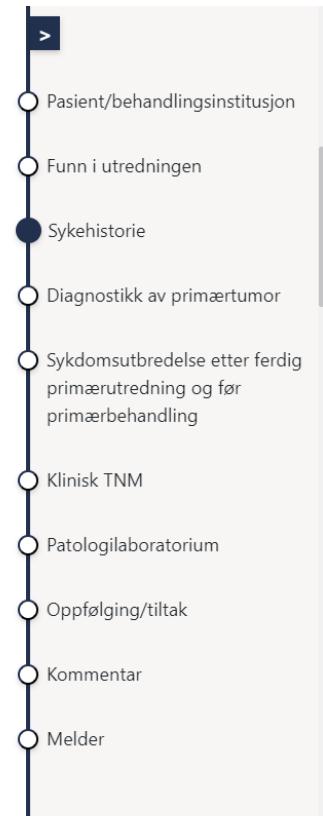
Utført dato \*  
  Ukjent

Er det gjort annen bildediagnostikk av primærtumor? \*

Ja  Nei

\* Obligatorisk

Lagre Lagre utkast Avbryt



**Celle-/vevsprøver \***

Ja  Nei

**Undersøkelser \***

Biopsi  TUR-P  Annet

**Dato sykdommen ble bekreftet/diagnosedato \***  
 Dato sykdommen ble bekreftet er prøvetakingsdato for første positive histologi/cytologi. Dersom det ikke er tatt vevsprøve, benyttes den dato man ved hjelp av andre undersøkelser (bildediagnostikk, blodprøve etc.) bestemmer seg for at pasienten har kreft

13.08.1957

**Sykdomsutbredelse etter ferdig primærutredning og før primærbehandling**

**Palpatorisk T-stadium høyre side (laveste stadium velges ved tvil) \***

T4 - Innvekst i omliggende vev

**Samlet palpatorisk T-stadium**  
 T4

**PI-RADS høyre side (laveste stadium velges ved tvil) \***

4 - Høy sannsynlighet for klinisk signifikant malign tumor som ...

**MR-basert T-stadium høyre side (laveste stadium velges ved tvil) \***

T4 - Innvekst i omliggende vev

**Samlet MR-basert T-stadium**  
 T4

**Palpatorisk T-stadium venstre side (laveste stadium velges ved tvil) \***

T4 - Innvekst i omliggende vev

**PI-RADS venstre side (laveste stadium velges ved tvil) \***

4 - Høy sannsynlighet for klinisk signifikant malign tumor som er ...

**MR-basert T-stadium venstre side (laveste stadium velges ved tvil) \***

T4 - Innvekst i omliggende vev

\* Obligatorisk

Lagre Lagre utkast Avbryt

## Samlet MR-basert T-stadium

T4

### Totalvurdering av klinisk T (laveste stadium velges ved tvil) \*

TNM på diagnostidspunktet baseres på all relevant utredning som er gjort før primærbehandling inkludert klinisk undersøkelse, bildediagnostikk, endoskopi, biopsi og kirurgisk eksplorasjon.

T4 - Tumor er fiksert eller vokser inn i nabostruktur(er) annet enn sædblære(r) | ▾

### Hvem har gjort totalvurderingen av klinisk T? \*

Utredende lege | ▾

### Er regionale lymfeknutemetastaser påvist (N-sykdom)? \*

Grensen mellom regionale lymfeknutemetastaser og fjernmetastaser går ved delingsstedet til arteria iliaca communis. Ved tvil om korrekt N-kategori skal den laveste (minst avanserte) kategorien velges

- NX = Ikke undersøkt
- N0 = Ingen regionale lymfeknutemetastaser
- N1 = Regionale lymfeknutemetastaser

### Utredningsmetode (velg viktigste undersøkelse som basis for cN) \*

- Ukjent
- CT
- MR
- PET
- Lymfadenektomi (diagnostisk i forkant av behandling)
- Annet

### Er fjernmetastaser, inkludert fjerne lymfeknutemetastaser, påvist (M-sykdom)? \*

Grensen mellom regionale lymfeknutemetastaser og fjernmetastaser går ved delingsstedet til arteria iliaca communis. Ved tvil om korrekt M-kategori skal den laveste (minst avanserte) kategorien velges

- Ikke undersøkt
- M0 = Ingen fjernmetastaser

\* Obligatorisk



Lagre

Lagre utkast

Avbryt

M0 = Ingen fjernmetastaser  
 M1 = Fjernmetastaser

**Lokalisasjon \***

Ikke-regionale lymfeknuter (M1a)  
 Skjelett (M1b)  
 Annet (M1c)

**Utredningsmetode \***

Ukjent  
 CT  
 MR  
 PET  
 Skjelettscintigrafi  
 Røntgen thorax  
 Røntgen korsrygg/bekken  
 Cytologi  
 Biopsi  
 Annet

**Klinisk TNM**

|    |    |    |
|----|----|----|
| cT | cN | cM |
| 4  | 0  |    |

**Patologilaboratorium**

\* Obligatorisk



Lagre

Lagre utkast

Avbryt

**Patologilaboratorium**

Laboratorium  
Haukeland universitetssjukehus  Ikke relevant

Preparatnummer

**Oppfølging/tiltak**

Antihormonell behandling alene

Behandlingssted \*  
Utlandet (spesifiser)

Spesifiser

Behandlingsdato

**Kommentar**

Kommentarer til utfylling av meldingen

\* Obligatorisk

**Lagre** **Lagre utkast** **Avbryt**

Utlandet (spesifiser) | ▾

Spesifiser  
String

Behandlingsdato  
13.08.1957

**Kommentar**  
Kommentarer til utfylling av meldingen

**Melder**

Meldedato \*  
13.08.1957

Melders navn \*  
serdtfy

Melder ID

>

- Pasient/behandlingsinstitusjon
- Funn i utredningen
- Sykehistorie
- Diagnostikk av primærtumor
- Sykdomsutbredelse etter ferdig primærutredning og før primærbehandling
- Klinisk TNM
- Patologilaboratorium
- Oppfølging/tiltak
- Kommentar**
- Melder

Lagre   Lagre utkast   Avbryt

\* Obligatorisk

## Feilmeldinger for obligatoriske felter som ikke er fylt ut og automatisk skrolling til manglende felt.

Lokalisasjon \*

Vennligst svar på spørsmålet.

Ikke-regionale lymfeknuter (M1a)  
 Skjelett (M1b)  
 Annet (M1c)

Utredningsmetode \*

Vennligst svar på spørsmålet.

Ukjent  
 CT  
 MR  
 PET  
 Skjelettscintigrafi  
 Røntgen thorax  
 Røntgen korsrygg/bekken  
 Cytologi  
 Biopsi  
 Annet

Klinisk TNM

T N M

\* Obligatorisk

Lagre Lagre utkast Avbryt

>

- Pasient/behandlingsinstitusjon
- Funn i utredningen
- Sykehistorie
- Diagnostikk av primærtumor
- Sykdomsutbredelse etter ferdig primærutredning og før primærbehandling
- Klinisk TNM
- Patologilaboratorium
- Oppfølging/tiltak
- Kommentar
- Melder

## Modal med bekrefteelse

Kreftregisteret

### Melding etter avsluttet utredning

Pasient/behandlingsinstitusjon

Fødselsnummer

Navn \*

Eksempelnavn Navnesen

Sykehus \*

Akershus Universitetssykehus HF, Kongsvinger

Avdeling \*

Dette er en standard avdelingsliste for Kreftregisteret. Dersom din spesifikke avdeling ikke finnes i listen, velg den avdelingen som passer best ut fra alternativene

Ikke relevant

**Funn i utredningen**

Primærtumor (med eller uten metastase(r)) - Utredning av primærtumor og samtidig utredning av regionale lymfeknutemetastaser/fjernmetastaser før igangsetting av primærbehandling

Kun metastase(r) - Utredning av regionale lymfeknutemetastaser og/eller fjernmetastaser uten samtidig utredning av primærtumor

\* Obligatorisk

Lagre   Lagre utkast   Avbryt

Vellykket!

En ny melding med endringene er lagret.

Ok

>

Pasient/behandlingsinstitusjon

Funn i utredningen

Sykehistorie

Diagnostikk av primærtumor

Sykdomsutbredelse etter ferdig primærutredning og før primærbehandling

Klinisk TNM

Patologilaboratorium

Oppfølging/tiltak

Kommentar

Melder

## Responsiv design uten navigasjonsbar

Sykdomsutbredelse etter ferdig primærutredning og før primærbehandling

Palpatorisk T-stadium høyre side (laveste stadium velges ved tvil) \*

T4 - Innvekst i omliggende vev | ▾

Palpatorisk T-stadium venstre side (laveste stadium velges ved tvil) \*

T4 - Innvekst i omliggende vev | ▾

Samlet palpatorisk T-stadium

T4

PI-RADS høyre side (laveste stadium velges ved tvil) \*

4 - Høy sannsynlighet for klinis... | ▾

PI-RADS venstre side (laveste stadium velges ved tvil) \*

4 - Høy sannsynlighet for klinisk s... | ▾

MR-basert T-stadium høyre side (laveste stadium velges ved tvil) \*

T4 - Innvekst i omliggende vev | ▾

MR-basert T-stadium venstre side (laveste stadium velges ved tvil) \*

T4 - Innvekst i omliggende vev | ▾

\* Obligatorisk Lagre Lagre utkast Avbryt

## Responsiv design med navigasjonsbar

### Radikal prostatektomi

**Pasient/behandlingsinstitusjon**

Fødselsnummer \*

Ikke norsk personnummer

**Navn \***

**Sykehus \***

 | ▾

**Avdeling \***

Dette er en standard avdelingsliste for Krefregisteret.  
Dersom din spesifikke avdeling ikke finnes i listen,  
velg den avdelingen som passer best ut fra  
alternativene

 | ▾

>

- Pasient/behandlingsinstitusjon
- Preoperativ informasjon
- Behandling
- Radikal prostatektomi
- Patologilaboratorium
- Kommentar
- Melder

### Preoperativ informasjon

Har pasienten vært aktivt monitorert? \*

Ja  Nei

Er det gjort ny vurdering av

\* Obligatorisk

Lagre Lagre utkast Avbryt