System for manuell monitorering av sau

Andreas Jensen Jonassen & Lars Erik Vassbotn

Masteroppgave i Datateknologi og Informatikk
Veileder: Svein-Olaf Hvasshovd
Trondheim, Juni 2022

NTNU

Abstract

To ensure animal welfare for grazing sheep on rangeland, the Norwegian authorities require sheep farmers to carry out weekly supervisions in the rangeland. When performing supervisions, it is common for the shepherd to take notes by hand, where the observations that are made as well as details such as an area description and timestamps are noted. At the end of the grazing season, these notes will be collected in a season summary report. The basic idea of this master's thesis project is to digitize the process of creating, sharing and presenting these notes.

This project builds on the work that was started in the preliminary thesis project, where a mobile application was developed to allow sheep farmers and shepherds to record observations made during supervision. The aim of this master's thesis project was to develop a new application that presents and summarizes this supervisory data, and to answer various research questions that involve comparing our system with the use of paper-based notes.

After building an understanding of the domain and examining existing solutions, an extensive planning phase began where a plan for the project as well as the requirements and architecture of the web application were established. The solution was then designed, developed and tested through an iterative and Agile process. All the requirements for the system were met as planned, and the project was carried out without significant hindrances.

Through usability testing, we eliminated large parts of the usability problems the test users experienced with the system, and the final system has many advantages compared to the use of paper-based notes according to the test users. Through the research questions, we discovered that the use of our system, compared to paper-based solutions, is perceived as more efficient while also providing a more satisfactory experience for the user. Our system also facilitates easier sharing of supervisory notes between farmers in the same grazing cooperation, and it frees sheep farmers and shepherds from having to manually register details such as time and GPS coordinates during supervisions.

In order to identify problems that may arise during real-life use of the system, it will be necessary to monitor sheep farmers and shepherds' use of the system over an extended period of time, preferably throughout an entire grazing season. As the grazing season is outside the time limit for this project, this remains as possible future work. There are also additional features that could be useful to include to make the system an «all-in-one» solution for everything related to sheep farming during the grazing season, such as integration of tracking devices or heatmaps to suggest which areas supervision should be performed on.

Sammendrag

For å sikre dyrevelferden for småfe på utmarksbeite, krever norske myndigheter at bønder utfører ukentlige tilsyn i beiteområdet. Ved utføring av tilsyn er det vanlig at gjeteren tar notater for hånd, der observasjonene som blir gjort samt detaljer slik som område og tidspunkt noteres. På slutten av beitesesongen skal disse notatene samles i en oppsummerende sesongrapport. Grunntanken for dette masterprosjektet er å digitalisere prosessen rundt å skape, dele og presentere disse notatene.

Dette prosjektet bygger videre på arbeidet som ble påbegynt i fordypningsprosjektet, der en mobilapplikasjon ble utviklet for å tillate sauebønder og gjetere å registrere observasjoner gjort under tilsyn. Målet for dette masterprosjektet var å utvikle en ny applikasjon som presenterer og oppsummerer denne tilsynsdataen, og å besvare ulike forskningsspørsmål som går ut på å sammenligne vårt system med bruk av papirbaserte notater.

Etter å ha bygget opp en forståelse av domenet gjennom en forstudie og en undersøkelse av eksisterende løsninger, startet et omfattende planleggingsarbeid der en plan for prosjektet samt webapplikasjonens krav og arkitektur ble etablert. Løsningen ble så designet, utviklet og testet gjennom en iterativ og smidig prosess. Alle kravene for systemet ble tilfredsstilt som planlagt, og prosjektet lot seg utføre uten nevneverdige hindringer.

Gjennom brukertesting har vi fått eliminert store deler av brukbarhetsproblemene testbrukerne opplevde med systemet, og det endelige systemet har mange fordeler sammenlignet med bruk av papirbaserte notater ifølge testbrukerne. Gjennom forskningsspørsmålene fikk vi avdekket at bruk av vårt system, sammenlignet med papirbaserte løsninger, oppleves som mer effektivt samtidig som at det er mer tilfredsstillende å bruke. Systemet vårt tilrettelegger også for lettere deling av tilsynsdetaljer mellom bønder i samme beitelag, og det frigjør bønder og gjetere fra å måtte registrere detaljer som tidspunkt og GPS-koordinater under tilsyn.

For å avdekke problemer som kan oppstå under virkelig bruk av systemet vil det være nødvendig å følge bønder og gjeteres bruk av systemet over en langvarig periode, helst gjennom en hel beitesesong. Ettersom beitesesongen er utenfor tidsforløpet til dette prosjektet, gjenstår dette som eventuelt framtidig arbeid. Det finnes også tilleggsfunksjoner som kunne vært nyttige å inkludere for å gjøre systemet til en «alt-i-ett»-løsning for alt relatert til sauehold gjennom beitesesongen, som for eksempel integrasjon av sporingsbjeller eller 'heatmaps' for å foreslå hvilke områder tilsyn burde utføres på.

Innhold

Figurer

Tabeller

Del I:

Introduksjon

I denne delen introduserer vi omfanget av prosjektet, og forskningsmetoden vi har valgt for gjennomføringen av prosjektet.

1 Beskrivelse

Intensjonen med dette masterprosjektet er å utvikle en webapplikasjon som lar sauebønder observere og holde oversikt over saueflokkene sine gjennom beitesesongen. Webapplikasjonen presenterer data registrert via mobilapplikasjonen som ble utviklet i fordypningsprosjektet [1]. I sin helhet skal webapplikasjonen og mobilapplikasjonen sammen svare på forskningsspørsmålene beskrevet i Seksjon 3.1.

For at webapplikasjon skal være nyttig for sauebønder er det viktig at den støtter flere nøkkelfunksjonaliteter og at den har full interoperabilitet med mobilapplikasjonen og dataen som blir samlet inn gjennom den. Disse nøkkelfunksjonene er beskrevet i Seksjon 10.3.2 og mer overordnet i hele Seksjon 10.

Den praktiske delen av oppgaven går ut på å designe og utvikle webapplikasjon, og å koble den opp til backend-systemet som lagrer data registrert i mobilapplikasjonen. Gjennom brukertester skal webapplikasjon gå gjennom flere iterasjoner av redesign og forbedringer basert på tilbakemeldinger fra testbrukerne. Vi har ikke mulighet til å utføre brukertesting på bønder på grunn av uoverensstemmelse med tidspunktet masteroppgaven ble skrevet (januar - 13. juni) og tidsperioden bønder slipper ut sauene sine (mai - juni), noe som ikke gir tilstrekkelig tid til testing på den riktige målgruppen innenfor et realistisk bruksscenario. Istedenfor testing på bønder vil webapplikasjonen bli testet på en variert brukergruppe innenfor et simulert bruksscenario.

Veilederen for oppgaven er professor Svein-Olaf Hvasshovd som bidrar med både formell rådgivning rundt masteroppgaven og innsikt og kunnskap rundt sau og sauegjeting.

Kildekoden for prosjektet kan bli funnet i repoet vårt på NTNU IDI sin offisielle GitLab-instanse via lenken `https://gitlab.stud.idi.ntnu.no/andreajj/master-of-the-sheeps/`. For demo av systemet kan en live versjon av webapplikasjonen bli funnet på `https://sheep-tracker-master.web.app/` med innloggingsinformasjon: `Brukernavn:` «demo—bruker@sauemaster.no», `Passord:` «SauBrukerTest2022%».

2 Omfang og interessenter

Denne masteroppgaven er en halvårig masteroppgave skrevet av to studenter – Andreas Jensen Jonassen og Lars Erik Vassbotn – som henholdsvis går 5-årig master i datateknologi og 2-årig master i informatikk, med professor Svein-Olaf Hvasshovd som veileder. Oppgaven ble skrevet på våren 2022 med innleveringsfrist 13. juni 2022 med en lengde på 20 uker, pluss én uke i tillegg grunnet påske. I tillegg til masteroppgaven ble et fordypningsprosjekt utført gjennom høsten 2021 som en forberedelse for selve masteroppgaven.

Mulige interessenter og målgruppe for prosjektet er hovedsakelig enkeltpersoner som tar del i beitelag – sauebønder og gjetere som gjennomfører tilsyn på vegne av sauebønder (også kalt *tilsynspersoner*). I tillegg er forfatterne av masteroppgaven, Andreas Jensen Jonassen og Lars Erik Vassbotn, interessenter for prosjektet – så vel som veileder Svein-Olaf Hvasshovd.

Andre som også vil ha en interesse i prosjektet er statlige representanter og organer som Statsforvalteren, Mattilsynet og Statens Naturoppsyn. Statsforvalteren har ansvar for å betale ut erstatning til bønder som har mistet dyr til rovdyr [2] og Mattilsynet har ansvaret for dyrevelferd og dermed er interessert i rapporter og dokumentasjon rundt tilsynene [3]. Statens Naturoppsyn (en avdeling i Miljødirektoratet) er interessenter i prosjekter fordi de jobber med statistikk og data rundt døde sau, og er de som blir tilkalt for å finne dødsårsaken til døde sau [4].

3 Valg av forskningsmetode

I denne seksjonen redegjør vi for forskningsmetoden som ble brukt under utførelsen av masterprosjektet. Forskningsmetodikken er basert på teori fra faget *IT3010 Metoder for empirisk forskning innen IT og digitalisering* på NTNU Gløshaugen, hvor boka *Researching Information Systems and Computing* [5] var sentralt pensum. Boka tar for seg seks aspekter – *The Six P's of Research* – som må tas i betrakning ved utførelsen av et forskningsprosjekt:

1. Purpose (norsk: *formål*)
2. Products (norsk: *produkter*)
3. Process (norsk: *prosess*)
4. Participants (norsk: *deltakere*)
5. Paradigm (norsk: *paradigme*)
6. Presentation (norsk: *presentasjon*)

Disse blir gjennomgått i delseksjonene under.

3.1 Formål

Intensjonen med dette prosjektet er å utvikle en applikasjon til nytte for sauebønder. Applikasjonen har som funksjon å presentere data fra tilsyn som har blitt utført av eller på vegne av sauebonden. Selve dataen kommer fra mobilapplikasjonen som ble utviklet som en del av fordypningsprosjektet. Sammen utgjør mobilapplikasjonen og den nye applikasjonen et helhetlig system for å registrere og lese observasjoner gjort under tilsyn på beiteområde. Et viktig formål med utførelsen av prosjektet er å sammenligne vår løsning – applikasjonen som presenterer og oppsummerer digitalt registrerte notater fra tilsyn – med den mer tradisjonelle måten som består av å lage en sesongrapport for hånd ut ifra papirnotater.

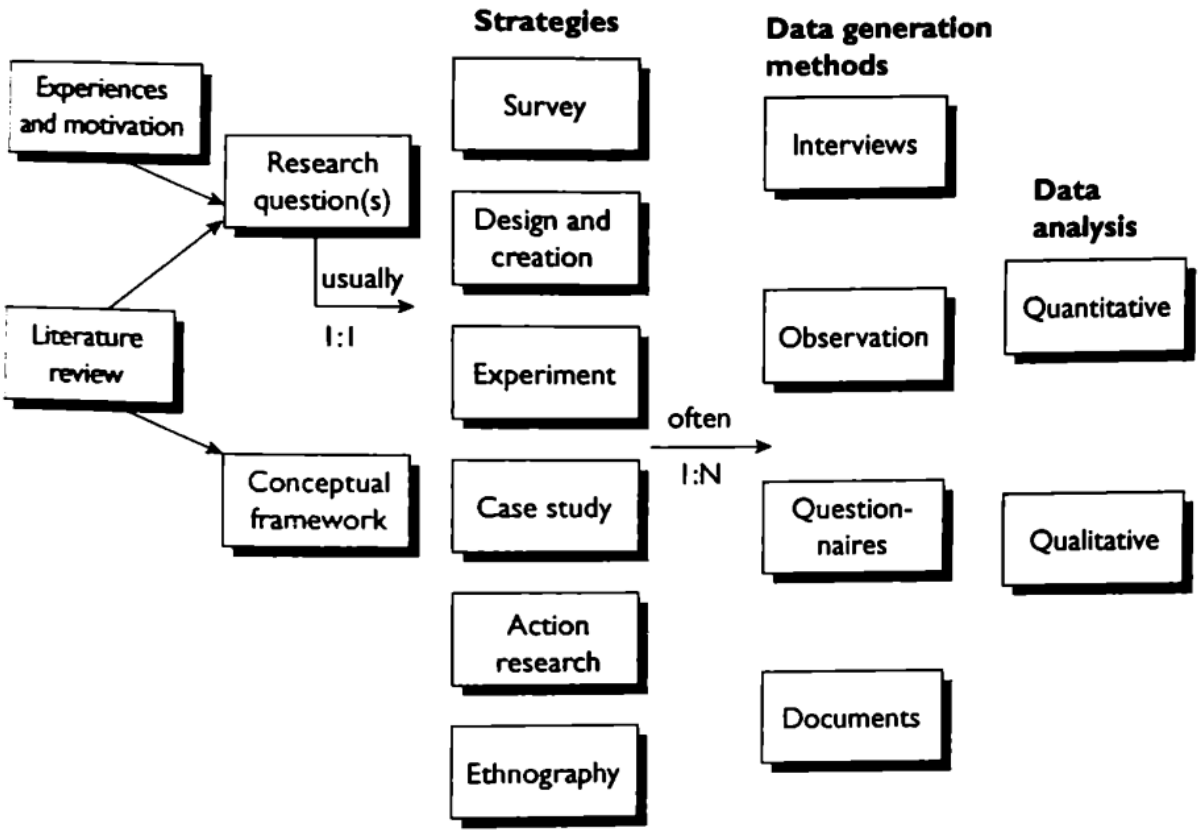
Forskningsspørsmålene vi skal forsøke å besvare gjennom utførelsen av dette prosjektet er som følger:

- **Forskningsspørsmål (FS)1:** Kan vårt digitale system effektivisere å oppsummere beitesesongen sammenlignet med bruk av papirbaserte notater?
- **FS2:** Oppleves vårt digitale system som mer tilfredsstillende for brukeren enn bruk av papirbaserte notater?
- **FS3:** Hvilke fordeler og ulemper tilbyr vårt digitale system en sauebonde framfor dagens papirbaserte løsning?

3.2 Produkter

Det viktigste produktet dette prosjektet vil produsere er en applikasjon innenfor et helhetlig digitalt system, til bruk av sauebønder, som tillater visning og oppsummering av observasjoner fra tilsyn i beitemark. Svar på forskningsspørsmålene og andre eventuelle funn som blir gjort i dette prosjektet er å anse som produkter. Denne masteroppgaven i seg selv er også et produkt av forskningsprosessen.

3.3 Prosess



Figur 1: En modell av forskningsprosessen og dets komponenter. Kilde: [5, side 33].

Som Figur 1 viser finnes det innenfor forskningen et utvalg av strategier (engelsk: *strategies*), metoder for datagenerering (engelsk: *data generation methods*) og metoder for dataanalyse (engelsk: *data analysis*). Ett forskningsspørsmål (engelsk: *research question*) er vanligvis knyttet til én strategi. For hver strategi er det mulig å velge flere måter å generere data på, og valget av hva man bruker for datagenerering vil avhenge av strategien og hva man forsøker å forske på. En datagenereringsmetode kan enten produsere kvantitativ eller kvalitativ data [5, side 36]. Kvantitativ data er simpelthen numerisk data (for eksempel antall sekunder brukt på å utføre en oppgave), mens kvalitativ data er all annen form for data som ikke representeres numerisk.

Dataen som blir generert kan enten analyseres på kvantitativt eller kvalitativt vis. Kvantitativ dataanalyse går ut på å behandle resultatene matematisk, for eksempel med statistiske metoder som å se på medianverdier for å tolke numerisk data. Det er også mulig å ta i bruk kvantitative metoder på kvalitativ data ved å transformere den kvalitative dataen til noe numerisk, for eksempel ved å telle antall ganger et nøkkelord blir nevnt i et intervju [5, side 266].

Kvalitativ dataanalyse går ut på å identifisere de gjengående temaene eller momentene i ulike typer kvalitativ data som er interessante innenfor det gitte forskningstemaet [5, side 267].

I utførelsen av denne masteroppgaven valgte vi *Design and creation* som forskningsstrategi, ettersom hovedmålet vårt er å produsere et nytt IT-system. For datagenerering har vi valgt å ta i bruk alle de fire metodene vist i Figur 1: intervjuer, observasjoner, spørreskjema og dokumenter. Dette er fordi vi så et behov for å ta alle disse metodene i bruk for å vurdere vårt eget system, samtidig som vi sammenlignet det med den eksisterende, papirbaserte løsningen. Bruken av disse metodene kommer til uttrykk i Seksjon 14.4

som omhandler brukertesting.

Når det kommer til dataanalyse har vi tatt i bruk kvalitativ analyse for å tolke resultatene.

Oates [5, side 111] beskriver at *Design and creation*-prosessen består av fem steg:

- **Awareness:** å bygge opp en forståelse av problemet
- **Suggestion:** å drøfte en idé for hvordan problemet kan løses
- **Development:** å arbeide med å gjøre idéen om til noe konkret
- **Evaluation:** å undersøke det man har utviklet for å se på dets nytteverdi og hvorvidt det stemmer overens med den originale ídéen
- **Conclusion:** å samle resultatene fra prosessen og presentere den opparbeidede kunnskapen og erfaringer, samt å se på muligheter for videre forskning

Både prosessen med å utvikle produktet, samt strukturen av denne masteroppgaven, samsvarer med disse fem stegene. I Del II bygger vi opp en forståelse rundt domenet, i Del IV foreslås det en løsning på problemet, og i Del V beskrives prosessen med å utvikle et konkret produkt. I Del VII evaluerer vi løsningen, og i Del VIII presenteres en konklusjon av prosjektet samt muligheter for videre forskning.

3.4 Deltakere

Veilederen vår, professor Svein-Olaf Hvasshovd, var i stor grad deltakende i prosjektet både som en kilde til informasjon om sauehold og generelt rundt utførelsen av masterprosjektet. Vi som masterstudenter (Andreas Jensen Jonassen og Lars Erik Vassbotn) var naturligvis også deltakere. Fem testbrukere har også vært involvert i prosjektet for gjennomføringen av brukertesting. Dette er personer som opptil tidspunktet for gjennomføringen av testen ikke hadde noen innsikt i masterprosjektet. Testbrukerne er valgt ut ifra venner og bekjente for oss – masterstudentene – og har variert bakgrunn når det kommer til type utdanning og dataferdigheter.

3.5 Paradigme

Paradigme omhandler å identifisere hvilket filosofisk paradigme forskningen faller under. Oates presenterer tre mulige paradigmer: *positivism* (norsk: *positivisme*), *interpretivism* (norsk: *interpretivisme*) og *critical research*.

Positivisme anser verden som noe objektivt; hvor alt kan måles og modelleres. Positivistisk forskning går ofte ut på å forsøke å bevise eller motbevise en hypotese, og kvantitativ dataanalyse blir som regel foretrukket [5, side 286].

Interpretivisme forsøker å identifisere, utforske og forklare fenomener sett i en sosial sammenheng. Innenfor interpretivismen er det som regel flere subjektive forklaringer, og det finnes ikke én enkelt sannhet slik som det gjør i positivismen. Interpretivistisk forskning ser på hvordan folk oppfører seg i naturlige omgivelser istedenfor for eksempel et eksperiment i et laboratorie. Her foretrekkes som regel kvalitativ dataanalyse, og forskerne tar med sine egne tidligere erfaringer og biaser med i betrakninger når de studerer resultatene [5, side 293].

Critical research kan minne om interpretivistisk forskning, men har et fokus på å undersøke og kritisere sosiale fenomen slik som maktforhold og tradisjoner [5, side 296]. *Critical research* kan for eksempel gå

ut på å undersøke hvordan økonomiske mekanismer har gitt oppstandelse til en skjevhet i maktforholdet mellom ulike grupper.

Forskning som benytter *Design and research*-strategien kan være vanskelig å plassere innenfor ett av disse paradigmene. Vi vil likevel argumentere for at dette prosjektet faller nærmest positivismen, ettersom sosiale faktorer i stor grad har uteblitt fra vår forskning.

3.6 Presentasjon

Presentasjon omhandler hvor og i hvilken form forskningsresultatene redegjøres for. Dette prosjektets produkt og resultater presenteres innenfor denne masteroppgaven.

Del II:

Bakgrunn

For å kunne forbedre et eksisterende system eller eksisterende løsninger er det viktig å forstå hvordan de fungerer og hvorfor de eventuelt ikke fungerer optimalt. For å forstå systemene er det også viktig å forstå hvordan sauehold og tilsyn fungerer i Norge i dag. I denne delen presenteres en forstudie om lover og praksis rundt sauehold i Norge, samt en undersøkelse av eksisterende løsninger som kan være relaterte til vår egen løsning.

4 Teori - 'state of the art'

I denne seksjonen vil vi ta for oss hvordan sauehold og tilsyn fungerer i Norge i dag, hvilke krav som stilles for tilsyn og utbetaling av erstatning, og hvordan tap av sau skal håndteres.

4.1 Sauehold i Norge

Sau er blant de eldste husdyrene i verden, og funn av sau i Norge kan spores tilbake til 1500-1400 år før vår tidsregning [6]. Sau er viktige dyr innenfor norsk landbruk og kan bli funnet over hele landet. Sauehold gir to kilder til inntekt for sauebonden; i Norge står kjøttproduksjon for 3/4 av inntektene, mens resterende 1/4 av inntektene kommer fra ullproduksjon [6]. Produksjon av sauemelk er ikke vanlig i Norge [7].

Norsk lov stadfester at småfe (fellesbetegnelse for sau og geit [8]) skal «holdes på egnet beite» i minst 16 uker [9, § 24], gitt at forholdene tillater det. Tall fra SSB [10] viser at nesten to millioner sau ble sluppet ut på beite i 2020, og at dette tallet har holdt seg relativt stabilt det siste tiåret. Beitesesongen tar sted på sommerhalvåret – den nøyaktige varigheten av beitesesongen avhenger av lokale klima- og værforhold og vurderinger sauebonden gjør rundt dette. Ved beite slippes dyrene som regel ut i *utmark*, som er naturlig terreng i fjell- og skogområder [11]. På beite vil sauen danne flokker, vandre i terrenget og fôre seg av vegetasjon. Når sau er på ute på beite slipper altså bonden å forsyne dem med fôr, som utgjør en besparelse for bonden.

Atferdsmessig kan sau beskrives som rolige flokkdyr med sterke flokkinstinkt [7]. Sau vil som regel være motivert til å unngå å fraskilles flokken. Arbeidet med sanking av sau på slutten av beitesesongen tar nytte av sauens flokkatferd til å lede sauen inn på ønsket område, for eksempel med bruk av gjeterhund.

4.1.1 Krav om tilsyn og rett til erstatning ved rovviltangrep

Norske myndigheter krever at dyr på utmarksbeite skal sees etter minst én gang per uke [9, § 19]. Minimumskravet på ett tilsyn i uka er gjeldende i områder «uten særskilt risiko»; tilsyn må forekomme hyppigere enn dette om det mistenkes at dyrene er utsatt for et høyere farenivå.

I Norge kan dyreeieren motta erstatning for husdyr som skades eller blir drept av rovvilt [12, § 1]. For å ha krav på erstatning må det ved undersøkelse kunne fastslås med sannsynlighetsovervekt at årsaken til skadene ble påført av rovvilt [12, §§ 6 og 8]. Under utførelse av tilsyn er det derfor hensiktsmessig å dokumentere ulike funn som enten direkte eller indirekte kan tyde til tilstedeværelsen av rovdyr på beiteområdet. Dette kan for eksempel være bittmerker på sau eller konkrete observasjoner av rovdyr. Desto flere funn som kan bevise rovviltangrep eller tilstedeværelse av rovvilt, desto bedre grunnlag har dyreeieren for å kunne motta erstatning for skadde eller døde dyr.

Det er statsforvalteren som er ansvarlig for å behandle søknader om erstatning og håndtere utbetaling av erstatning [12, § 15].

4.1.2 Tap av sau

Det hender at rovdyr trenger seg inn på beiteområdene til sauen, og dette fører til betydelige sauetap hvert år. Jerv og gaupe er dyrene som tar flest saueliv i Norge. I 2021 var jerv skyldig for 42,4 % av sauetap, mens gaupe knyttes til 24,5 % av tapene [13]. Andre rovdyr som er representert i statistikk for angrep på sau er kongeørn, bjørn og ulv. I 2021 ble det betalt erstatning for 2 525 sau og 14 358 lam grunnet rovviltangrep. Dette utgjorde beløp for til sammen omtrent 44 millioner norske kroner [13].

Ved funn av skadde eller drepte dyr der det er sannsynlighet for at rovdyr har vært ansvarlige skal eieren varsle Statens Naturoppsyn (SNO) om dette. Lokalt rovviltpersonell fra SNO vil da foreta en feltundersøkelse for å samle funn som kan fastgjøre om skaden eller dødsfallet skyldes rovdyrangrep. Dette personalet rapporterer videre til den regionalt ansvarlige innenfor SNO [14]. Ved en sikker konklusjon eller i det minste sannsynlighetsovervekt om at rovdyr var ansvarlig vil dette gi dyreeier rettighet til erstatning [15].

Resten av dødsfallene blant sau på beite omtales som *normaltap* (tapet av husdyr som erfaringsmessig inntreffer i besetningen på utmarksbeite uten forekomst av rovvilt [12, § 2]). Tall fra 2019[1] viser at normaltapet lå på 21 665 sau og 62 784 lam[2]. Innenfor normaltap er det mange mulige dødsårsaker. Sykdom, flåttangrep og ulykker som for eksempel påkjørsel eller fall er alle årsaker som kan føre til tap av saueliv [17]. Tap av husdyr som faller innenfor normaltap vil ikke føre til utbetaling av erstatning.

4.2 Markering av sau

Sau på beite er utstyrt med øremerke, sett i Figur 2, og slips, sett i Figur 3. Begge disse utgjør hver sin funksjon, som vi vil utrede for her.

4.2.1 Øremerker

Øremerker brukes for å identifisere sauen, så vel som at de fungerer som visuelle indikatorer for gårdstilhørlighet under beitesesongen. I Norge er det påbudt å markere sau med et elektronisk øremerke og et visuelt øremerke, 30 dager etter fødsel [18].

De elektroniske øremerkene benytter RFID-teknologi [19] og kan derfor leses ved hjelp av elektroniske verktøy på kort avstand – typiske RFID-lesere som brukes i Norge kan lese i en avstand på 30-40 centimeter [20]. Lam som skal slaktes før de blir 12 måneder gamle er unntatt kravet om elektronisk øremerke, men må fortsatt bære et visuelt øremerke.

Forskriften om merking, registrering og rapportering av småfe bestemmer at det visuelle øremerket skal oppgi følgende informasjon [18, § 3]:

1. Mattilsynet: MT.
2. Nasjonalitetsidentifikasjon: NO.
3. Dyreholdets spesielle identitetsnummer tildelt av Mattilsynet: Syv siffer.
4. Individnummer: Fem siffer, der første siffer er fødselsårets siste siffer, og de følgende sifrene er dyrets individnummer.

Et eksempel på et visuelt øremerke for sau og lam vises i figur 2.

[1] Tall fra 2019 brukes fordi statistikk for 2020 var utelatt fra Mattilsynets årsrapport 2020. Årsrapport for 2021 har ikke blitt publisert per dato 4. februar 2022.

[2] Dette er utregnet ved å ta det totale antallet sau og lam tapt i året ifølge Mattilsynets årsrapport [16], og trekke fra antallet sau som det ble betalt erstatning for grunnet rovviltangrep det samme året [13].
24 555 sau tapt totalt - 2 890 sau betalt erstatning for = 21 665 sau i normaltap.
77 467 lam tapt totalt - 14 683 lam betalt erstatning for = 62 784 lam i normaltap.

Figur 2: Bilde av øremerke for sau og lam. Kilde: [21]

Det visuelle øremerket kan ha valgfri farge [18, § 7], utenom hvit og lakserød. Det er også mulig å ha tofargede øremerker. I praksis koordinerer gårder som deler beiteområde om valg av farge på øremerkene, slik at hver gård i området får sin egen unike farge. Dermed kan gårdstilhørligheten til sau observert på beite lett etableres. Om to gårder som *ikke* deler beiteområde har valgt samme farge for det visuelle øremerket har dette ingen konsekvenser, fordi sau fra de to gårdene ikke vil omgås hverandre.

4.2.2 Bjelleslips

Sauebønder benytter fargekodede bjelleslips for å markere søyer (voksen hunnsau). Slipset er festet rundt nakken på søya, og fargen på slipset forteller hvor mange lam søyen har med seg ut på beite. Formålet med slipset er å gjøre det lettere å få oversikt over flokken – hvis det observeres en søye med et slips som tilsier at det har to lam, men søya kun har ett lam på følge, vil dette antyde at ett av lammene er på avveie. Det er ikke noe krav fra myndighetene om å benytte bjelleslips, men det brukes likevel fordi det bidrar til å gjøre utføring av tilsyn og sanking av sau på beite mer praktisk. Søye med slips rundt nakken vises i Figur 3.

Interesseorganisasjonen Norsk Sau og Geit (NSG) har siden 2011 kommet med en offisiell anbefaling til standardisering av bruk av fargekodede slips [22]. Anbefalingen knytter visse farger til spesifikke antall lam:

- **Rødt slips:** ingen lam.
- **Blått slips:** ett lam.
- **Gult slips eller uten slips:** to lam.
- **Grønt slips:** tre lam.

En slik standard gjør det lett å få oversikt over slipsenes betydning, uansett hvor man er i landet. Siden 2019 har standarden blitt tatt i bruk i alle Norges regioner[3].

[3]NSG Oppland benyttet en annen fargekoding fram til desember 2019 [23].

Figur 3: Søye med blått slips. Kilde: Norsk Sau og Geit [22].

4.3 Utføring av tilsyn

Som nevnt i Seksjon 4.1.1 er det påbudt å utføre tilsyn i beiteområdet minst én gang i uka, og hyppigere tilsyn enn dette om forholdene måtte kreve det (for eksempel et område med historikk av mye rovdyr). Under tilsyn skal beiteområdet og sau inspiseres [12, Retningslinjer til § 4, bokstav c]. Dette tolkes av bønder ikke nødvendigvis som et krav om å se til hele sauebestanden sin hver uke, men som et krav om at tilsyn skal forekomme innenfor beiteområdet minst én gang i uka.

I tillegg til å sikre velferden til dyrene, hjelper observasjoner hentet fra tilsynsrunder med å få oversikt over hvor sauen befinner seg i løpet av beitesesongen. Dette er til hjelp ved sesongslutt når sauen må sankes inn igjen.

4.3.1 Notater og rapport fra tilsyn

Det finnes ingen konkret standard på hva som skal dokumenteres under utførelse av tilsyn. Det er svært viktig at tilsynspersonen som utfører tilsynet i det minste noterer tilfeller av skade eller død blant sauebestanden hvor rovvilt kan være ansvarlig, ettersom slike forhold snarest skal meldes til Statens naturoppsyn. Å ikke tilfredsstille dette kravet fører til avkortning eller bortfall på erstatning [12, § 10].

Veileder Svein-Olaf Hvasshovd har under møte med relevant fylkeskommunal etat (Vedlegg A) fått bekreftet at følgende informasjon om utført tilsynsrunde er ønskelig:

- Dato og tid på tilsynsrunde.
- Antall voksne dyr og lam.
- Farge på øremerke på observerte dyr.
- Forekomst av skadde dyr.
- Forekomst av døde dyr.

- For hver observasjon skal posisjon til observasjonen noteres.

Det kan være vanskelig å observere fargen på øremerket til dyr som er langt unna. Øremerkene er relativt små, og fargen på disse lar seg derfor lettest observere om gjeteren er nærme dyrene eller benytter seg av en kikkert.

Utover de nevnte punktene vil det også være hensiktsmessig å notere spor av, tegn på eller konkrete observasjoner av rovdyr. Dette kan bidra til å gi grunnlag for saksbehandler til å kunne komme med en konklusjon om at rovdyr sannsynligvis var ansvarlige for skader på, eller tap av, sau.

Ved endt beitesesong burde notater fra alle utførte tilsyn i løpet av sesongen slås sammen til en sesongrapport. I tråd med at hva som skal registreres ikke har noen formell standard, er det heller ikke en standardisert rapportmal som skal utfylles på slutten av sesongen. Det er likevel hensiktsmessig å få med oppsummeringer av de punktene som er beskrevet over; altså oppsummerte tall på observerte sau og lam, tall på skadde eller døde dyr, dato/tid for hver tilsynsrunde og detaljer som kan peke mot tilstedeværelse av rovdyr i området.

4.3.2 Beitelag

Å utføre ukentlige tilsynsturer for å sjekke på sauene kan være krevende for en enkelt bonde å utføre alene. Derfor er det vanlig for sauebønder å være en del av et beitelag, der medlemmene av beitelaget samarbeider om å utføre tilsyn og sanking [24]. Medlemskap i beitelag gjør at ansvaret om å utføre tilsyn delegeres på flere personer. Beitelag må godkjennes av kommunen [24].

Det er flere insentiv for bønder til å bli medlem av et beitelag. Et beitelag kan søke om midler til å drifte beitelaget, som kan brukes til å forbedre beitebruket. Basert på risikoen i beiteområdet – som vurderes ut ifra observasjoner gjort under tilsyn – kan beitelaget også søke om midler til å iverksette tiltak som kan sikre velferden til dyrene, for eksempel gjerder eller utstyr for elektronisk sporing av sau [25].

4.4 Sanking

Når det går mot slutten av beitesesongen skal sauene sankes inn igjen. Det vil være regionale forskjeller på når sanking tar sted, men Miljødirektoratet anser 10. september som normal sankedato for sau [26, side 11]. Dato for sanking vil også påvirkes av forholdene; forskrift om velferd for småfe [9, § 27] kravsetter at dyrene sankes før snøfall eller frost ventes i beiteområdet. Et annet forhold som kan påvirke sankedato er risikoen rovvilt utgjør for dyrene – om risikoen anses som høy, er tidlig innsanking et mulig tapsreduserende tiltak [27, § 5]. Forskriften bestemmer at sats for kompensasjon for tidlig nedsanking er 7 kroner per dyr, per dag [27, § 10].

Observasjoner av saueflokker fra tilsyn og hvor disse befinner seg kan være til stor hjelp for innsankingsarbeidet på slutten av beitesesongen.

5 Eksisterende løsninger

I denne seksjonen ser vi på eksisterende løsninger på markedet som kan hjelpe med monitorering av sau, og vurderer hvorvidt de er egnet til å bidra til å tilfredsstille kravet om manuelt tilsyn.

5.1 Papirskjema

Som diskutert i Seksjon 4.3.1 er det ingen etablert standard for hva som skal dokumenteres fra et tilsyn, eller hvordan det skal dokumenteres. Om gjeteren velger å ta i bruk penn og papir for å dokumentere tilsynet, er det flere papirskjemaer tilgjengelig fra ulike organisasjoner eller statlige/kommunale etater. Et eksempel er papirskjemaet tilgjengelig fra interesseorganisasjonen Norsk Sau og Geit (Vedlegg B), et annet eksempel er et skjema tilgjengelig fra statsforvalteren.no (Vedlegg C). Disse to skjemaene har svært mye til felles:

- En logg over enkelttilsyn, med felt for **dato**, **rute** og et stort **fritekstfelt for observasjoner**
- En tabell hvor hele beitesesongen oppsummeres
- Navn på beitelag eller organisasjonsnummer for beitelaget
- Navn på gjeter
- Felt for å registrere hvor mye tid man selv har brukt på tilsyn

Den største forskjellen mellom de to skjemaene er hvilke detaljer som er med i tabellen som oppsummerer beitesesongen. Begge skjemaene har felt for å sammenligne antall dyr sluppet og sanket, men kun skjemaet fra statsforvalteren har felt for å oppgi sikre eller antatte tapsårsaker for tapte dyr.

5.1.1 Drøfting av egnethet til papirskjema

Når man sammenligner disse skjemaene med det veileder Svein-Olaf Hvasshovd oppgir som ønskelig informasjon fra en tilsynsrunde, som diskutert i Seksjon 4.3.1, er det tydelig at omtrent all den informasjonen som skal fylles ut i fritekstfeltene på skjemaet uteblir fra oppsummeringsdelen. Dette er for eksempel farger på øremerker, antall dyr observert eller antall skadde dyr. Oppsummeringen dreier seg i stor grad kun om hvor mange dyr som har dødd, og i tilfellet av statsforvalterens skjema, hva dødsårsakene var.

Hver boks for å utfylle detaljer fra enkelttilsynet er relativt liten, og det kan være utfordrende å fylle ut alle detaljene Svein-Olaf Hvasshovd mener er relevante innenfor én enkelt boks – det er mulig man må ta i bruk flere av disse boksene på skjemaet for ett tilsyn.

Et punkt Svein-Olaf Hvasshovd nevnte som viktig var lokasjonen til observasjonen, og dette kan være vanskelig å registrere med penn og papir. Utstyr for avlesing av GPS-koordinater kan benyttes for dette, men å notere ned GPS-koordinater for hånd er upraktisk. Ellers kan en tekstlig beskrivelse av området noteres, som da sannsynligvis vil være forståelig kun for den som selv har utført tilsynet samt de ekstremt lokalkjente. Et digitalt system for å skrive ned detaljer, som automatisk logger GPS-koordinatene til enhver observasjon, ville vært svært nyttig.

Det finnes få praktiske begrensninger for hva man kan notere ned på papir, så bruk av papirskjema burde kunne dekke alle praktiske behov under utføring av tilsyn. Papirskjemaer vil for mange ansees som «gode nok», men bruk av papirskjema har visse svakheter. Som tidligere nevnt kan det kreve bruk av tilleggsutstyr som for eksempel GPS ved siden av. En annen svakhet er at alt noteres ned som fritekst, og med mindre tilsynspersonen selv etablerer og følger en viss standard for notatene sine, vil notatene være på ukonsekvent format som gjør at det kreves mer innsats når notatene skal bearbeides senere. Det er også mulighet for at noens håndskrift er vanskelig å tyde for andre, som kan redusere effektiviteten av arbeidet med å lage en oppsummeringsrapport på slutten av sesongen.

5.2 Elektroniske sporingsenheter

Det eksisterer produkter fra flere produsenter som har som funksjon å tilrettelegge for fjernovervåking av sau via elektroniske enheter som registrerer sauens koordinater og sender disse til sauebonden. Implementasjonen av dette kan variere, men er som oftest basert på små batteridrevne enheter som festes rundt halsen til dyret, og periodisk innhenter geolokasjon fra GPS og deretter sender denne dataen videre over satellitt eller mobilnettet. Her presenteres to av disse løsningene, fra henholdsvis Findmy og Telespor.

5.2.1 Findmy

Findmy er et norsk selskap som selger sporingsteknologi for å spore vandrende husdyr som sau og reinsdyr [28]. Findmys hovedprodukt er *E-bjella*, som er en elektronisk sporingsenhet som festes rundt halsen på dyret og benytter satellitter for å kommunisere posisjonen sin [29]. Bjellen melder ifra om sin posisjon i et spesifikt tidsintervall, og dyreeieren kan følge med på dyrets bevegelser gjennom en PC eller en smarttelefon. I tillegg har E-bjeller andre nyttige funksjoner som en bevegelsesalarm som melder i fra om dyret ikke har beveget seg på en stund, en urovarsler som melder ifra om unormal atferd i flokken, og støtte for såkalt *geofencing* som sier ifra når dyret går inn i eller forlater et spesifikt område definert av dyreeieren. Fordi FindMy avhenger av satellittbasert kommunikasjon, kreves det klar sikt til himmelen fra der sauen står for at bjellen skal kunne melde sin posisjon.

Nåværende versjon av produktet, *E-bjella Modell 2* er priset til 1980 kroner per enhet (eksklusive merverdiavgift) [30]. Det kommer også med en årlig brukeravgift per bjelle på 229 kroner. Batteriet på enheten er utskiftbart og har ifølge produsenten levetid på cirka tre år på beitebruk på sau. Nytt batteri koster cirka 99 kroner eksklusive merverdiavgift.

5.2.2 Telespor

Telespor AS er et norsk selskap som «utvikler og selger produkter og tjenester for elektronisk overvåking av husdyr på beite» [31]. Deres hovedprodukt er en sporingsenhet ved navn *Radiobjella*. I tillegg til å kontinuerlig rapportere posisjonen til dyret med et gitt intervall, er enheten utstyrt med en bevegelsessensor som kan varsle om dyret ikke har beveget seg på en stund. Posisjonene som blir sendt av enheten kan kobles til «Norgeskart friluftsliv»-applikasjonen [32] slik at dyreeier kan følge med på dyrene under beitesesongen.

Telespors Radiobjelle tar i bruk LTE-M og Narrowband IoT-teknologi for kommunikasjon, som i bunn og grunn betyr at det benytter seg av det tilgjengelige 4G-nettet [33]. Dette byr på både fordeler og ulemper sammenlignet med Findmys satellittbaserte løsning: Telespors løsning krever ikke klar sikt til himmelen og tillater for to-veis kommunikasjon slik at enheten kan hente inn oppdaterte innstillinger hver gang den kommuniserer. Ulempen er at Telespors løsning, i motsetning til Findmys satellittbaserte løsning, krever at det er mobildekning i området dyret befinner seg - noe som ikke er garantert i utmarka.

Nåværende versjon av Radiobjella er 4. generasjonsmodell, med en enhetspris på 989 kroner uten abonnement, hvor ett batteri er inkludert [34]. Sesongabonnement kommer i tillegg og koster 99 kroner for fem måneder. Batteriet er utskiftbart og batteribytte anbefales etter endt sesong. Nytt batteri koster 60 kroner fra Telespors nettbutikk.

5.2.3 Drøfting av egnethet til elektroniske sporingsenheter

Bruk av sporingsenheter oppfyller ikke kravet om utførelse av manuelt tilsyn. Sporingsenheter kan likevel hjelpe med å effektivisere utførelse av tilsynsrunder, ettersom det gir en indikator på hvor sauen(e) befinner seg. I en undersøkelse utgitt i 2010 utført av Trøndelag Forskning og Utvikling AS (TFoU)

på oppdrag fra Statens landbruksforvaltning (i dag Landbruksdirektoratet), ble det sett på effekter av elektronisk overvåkning gjennom bruk av radiobjeller [35]. Undersøkelsen så på bruk av radiobjeller i flere fylker over hele landet. Undersøkelsen konkluderte blant annet at bruk av radiobjeller kan ha «direkte, indirekte og langsiktig indirekte tapsreduserende og dyrevelferdsmessig effekt». Det ble også funnet at radiobjeller øker gjenfinningsgraden av de tapte sauene som har påmontert bjelle, og at bruken av radiobjeller har en indirekte gjenfinningseffekt også på dyrene som ikke er utstyrt med bjelle fordi de gjør tilsynene mer treffsikre.

Den største ulempen med elektroniske sporingsenheter som Findmy og Telespor er kostnadene assosiert med å kjøpe bjeller, batterier og abonnement for å kunne spore hver enkelt sau. Med batteri og abonnement inkludert koster begge produktene over en tusenlapp. Ifølge interesseorganisasjonene Norsk Sau og Geit er verdien på en tapt søye omtrent 3500 kroner, mens verdien på et tapt lam er rundt 1800 kroner [36]. Samme kilde oppgir at kjøttverdien til både søye og lam ligger på rundt 1000 kroner. Dette gir uttrykk for at slike sporingsenheter er relativt dyre sett i forhold til verdien til dyret. Det er derfor ikke hensiktsmessig for en sauebonde å utstyre hvert dyr med slik sporingsteknologi.

Findmy anbefaler at 25 % av dyrene spores [28]. Rapporten fra undersøkelsen til TFoU kom ikke med en spesifikk anbefaling på hvilken dekningsgrad med radiobjeller sauebønder burde ha, ettersom faktorer som terrenget i beiteområdet og sauerase spiller inn.

Selv om elektroniske sporingsenheter ikke tilfredsstiller kravet om manuelt tilsyn, har bruken av slike enheter altså likevel positive effekter på sauens velferd og effektiviteten til tilsyn.

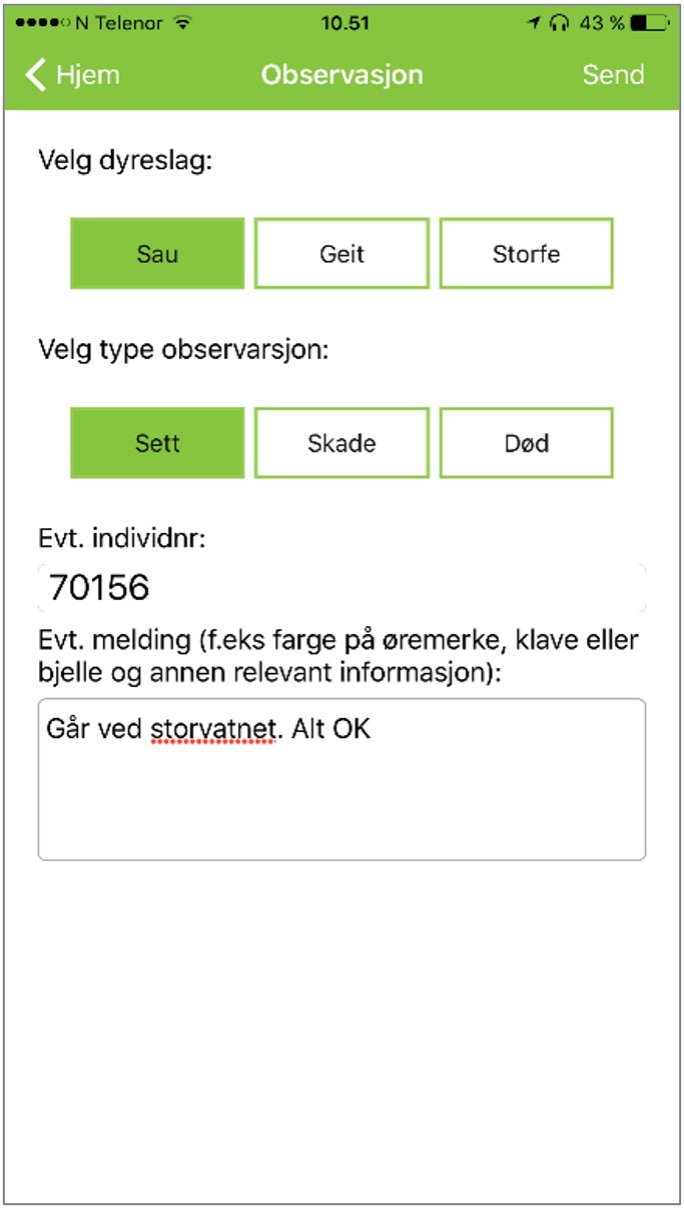
5.3 Eksisterende mobilapplikasjoner

I denne delseksjonen vil vi se på applikasjoner som kan ligne på den vi selv lagde i fordypningsprosjektet. Dette vil altså være applikasjoner som lar en bruker opprette notater som plasseres på et kart, slik at et notat kan knyttes til en geografisk plassering. For at en slik lignende applikasjon skal kunne være relevant innenfor dette prosjektet er det nødvendig at slike eventuelle applikasjoner enten tillater å eksportere notatene ut av appen, for presentasjon et annet sted, eller at disse notatene kan deles med andre brukere. En annen funksjonalitet som er ønskelig er at applikasjonen kan brukes uten en permanent nettilkobling, ettersom det ble etablert i fordypningsprosjektet at å ha nettilkobling ikke er en selvfølge ute i beitemarka.

5.3.1 BeiteSnap

BeiteSnap er en applikasjon fra det norske selskapet Fant AS, som beskriver applikasjonen som «et komplett verktøy for alle husdyr på beite» [37]. Applikasjonen er ikke lenger tilgjengelig for nedlasting, og per dags dato framstår det som at alle tjenester relatert til BeiteSnap er nedlagt. Applikasjonen var tilgjengelig for Android og iOS. Ressurser og informasjon om BeiteSnap er likevel fortsatt tilgjengelig på produktets nettside `https://www.beitesnap.no/`. Ettersom appens brukermanual fortsatt er tilgjengelig på BeiteSnaps nettsider [38], er det mulig å få en forståelse for hva slags funksjonalitet applikasjonen hadde. Det er ut ifra denne manualen at vi vil bedømme BeiteSnap i denne delseksjonen, ettersom applikasjonen ikke lenger er tilgjengelig for nedlasting er manualen vår beste informasjonkilde.

Antallet funksjoner som tilbys avhenger av om man er en gratisbruker eller en betalende bruker. Gratisbrukere har tilgang til å sende inn observasjoner, som altså er detaljer om dyr som sees ute på beite. Observasjoner kan registreres med eller uten bilde. Detaljene en observasjon kan inkludere sees i Figur 4.



Figur 4: Registrering av en observasjon i BeiteSnap. Kilde: [38].

Resten av funksjonaliteten til applikasjonen er forbeholdt betalende brukere, som er ment til å være bønder og gjetere hyrt inn av bøndene. Den betalte versjonen var priset til 1200 kroner pluss merverdiavgift per år, og én betalt bruker kunne ha syv "underbrukere". Dette er brukere – som oftest gjetere hyrt inn av bonden – som kan utføre tilsyn i applikasjonen, og data fra utførte tilsyn av disse underbrukerne vil bli sendt til hovedbrukeren.

En bonde må tegne et polygon over kartet som definerer beiteområdet sitt. Alle observasjoner som gjøres innenfor dette området vil føre til at bonden som «eier» det beiteområdet mottar et varsel. Det er en mulighet for at flere bønder har overlappende beiteområder, og appen vil da varsle alle bøndene om observasjonen på det overlappende beiteområdet. Varslinger om observasjoner gjort på bondens beiteområde kan enten vises som en liste i kronologisk rekkefølge eller som markører på et interaktivt kart.

Det er en kommentarfunksjon på enkeltobservasjoner som tillater bønder å samhandle om hvordan de skal håndtere observasjonen. Kommentarfunksjonen vises nederst på Figur 5. Her vises også at hver

observasjon kan markeres som «ferdig behandlet», og informasjon om observasjonen kan eksporteres som en e-post.

Figur 5: Visning av detaljer om en observasjon. Kilde: [38].

BeiteSnap-applikasjonen har også en funksjon for å ta notater. Dette er fritekst-notater som lagres og presenteres i kronologisk rekkefølge etter når de ble tatt, slik at bonden for eksempel kan holde styr på diverse detaljer som kanskje er av interesse senere.

BeiteSnap har mange funksjoner som dreier seg om utføring av tilsyn eller er relatert til tilsyn. Alle disse funksjonene er å finne innenfor en meny som heter «Registreringer». Her er det blant annet mulig å utføre «tilsyn», som betyr at brukerposisjonen kontinuerlig vil lagres inntil tilsynet stoppes. Det finnes flere kategorier av tilsyn som kan registreres. Utførte tilsyn vises i en liste, og underbrukere/gjetere som har utført tilsyn på hovedbrukerens vegne vil også vises. En viktig funksjonalitet som diskuteres i brukermanualen til BeiteSnap er å lage rapporter, som blir diskutert som en kommende funksjonalitet. Rapporter skal presentere alle de utførte tilsynene, og det er mulig å bestemme hva som skal inkluderes i

rapporten. Det er mulig for brukeren å registrere datoene for beiteslipp og sanking, og disse vil inkluderes i rapporten. Hvordan rapportene er utformet vises ikke, og det er uklart om denne funksjonaliteten senere ble ferdigstilt.

Til sist skal det nevnes at BeiteSnap tilrettelegger for nedlasting av kartområder for når brukeren ikke har dekning. Brukermanualen beskriver denne funksjonaliteten som at brukeren ikke behøver å gå gjennom noen spesiell prosess for å laste ned kartområder selv; alle kart som vises for brukeren mens de har nettilgang vil også bli lastet ned i bakgrunnen for offline-bruk.

5.3.1.1 Drøfting av egnethet til BeiteSnap

I starten av deleseksjonen ble det introdusert tre kriterier som kunne gjøre lignende applikasjoner til mulige alternativ til vår egen løsning. Det første var muligheten for å ta notater som kan plasseres geografisk, den andre var å kunne eksportere eller dele disse notatene, og den tredje var å støtte nedlasting av kartområder for situasjoner der brukeren ikke har nettilgang. I vår redegjørelse av BeiteSnap har vi vist at den tilfredsstiller alle disse kravene. BeiteSnap framstår som en applikasjon fullpakket med funksjonaliteter som også burde dekke de fleste behovene til en sauebonde generelt. Det at Fant AS har opphørt salg og drift av denne applikasjonen betyr likevel at denne applikasjonen ikke lenger er en mulig løsning for å drive med tilsyn av sau på beite.

Om BeiteSnap fortsatt hadde vært i drift i dag, er det likevel noen viktige måter den hadde skilt seg fra vår eksisterende (mobilapplikasjonen fra fordypningsprosjektet) og framtidige løsning som skal utarbeides i dette prosjektet (en webapplikasjon). For det første er vår løsning tiltenkt som en mobilapplikasjon og en webapplikasjon med ulike ansvar og funksjonaliteter, mens BeiteSnap samler all funksjonaliteten i en mobilapplikasjon. For det andre er vår mobilapplikasjons brukergrensesnitt designet for å imøtekomme behovene til gjetere som potensielt bruker kikkert under utføring av tilsynet sitt, og dermed kan ha behøv for en mobilapplikasjon som kan fungere godt selv om brukeren ikke ser på skjermen så ofte mens de fyller ut observasjonene sine. For det tredje er telling av antall sau og lam, samt registrering av slipsfarger, en viktig del av vår applikasjon. Dette lar seg kun gjøre som fritekst i BeiteSnap, som ikke er spesielt intuitivt.

BeiteSnap tilbyr likevel noen interessante funksjoner som kan være interessante å tenke på i sammenheng med vår egen løsning, som for eksempel varslinger når nye observasjoner kommer inn, samt muligheten for å registrere slipp- og sankedato slik at dette vises på rapporten.

BeiteSnap gir oss innsikt i hvordan andre har kommet med løsninger ulik våre egne innenfor samme problemstilling. For eksempel forholder BeiteSnap seg til tilsyn på en annen måte sammenlignet med vår egen mobilapplikasjon. I BeiteSnap virker det som at «tilsyn» konseptuelt kun er sporing av brukerlokasjoner, mens observasjoner kan registreres uavhengig av at de er del av et tilsyn. I vår applikasjon er «tilsyn» mer sentrale, ettersom alle observasjoner tilhører et tilsyn.

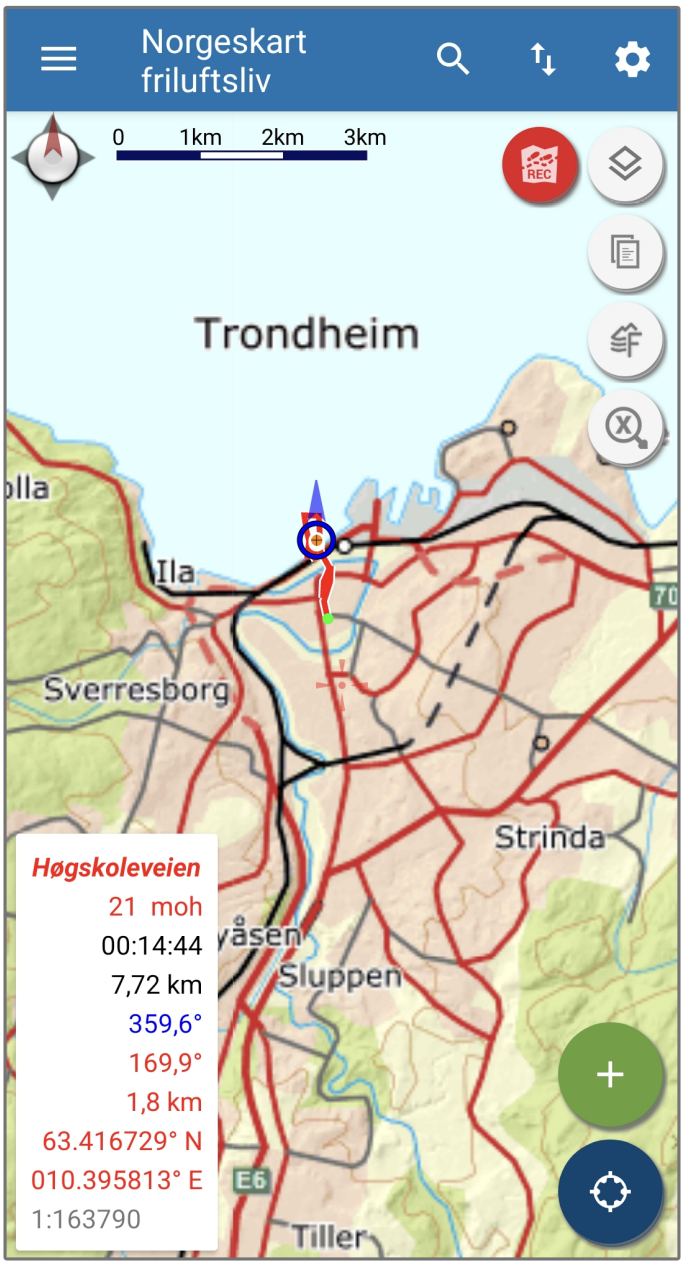
5.3.2 Norgeskart Friluftsliv

Norgeskart friluftsliv er en applikasjon fra Ture Apps AS[4] [5]. Applikasjonen er tilgjengelig på både Android og iOS. Per dato 4. mai 2022 var siste oppdatering til applikasjonen utgitt 17. april 2022, som tyder på at applikasjonen fortsatt blir vedlikeholdt.

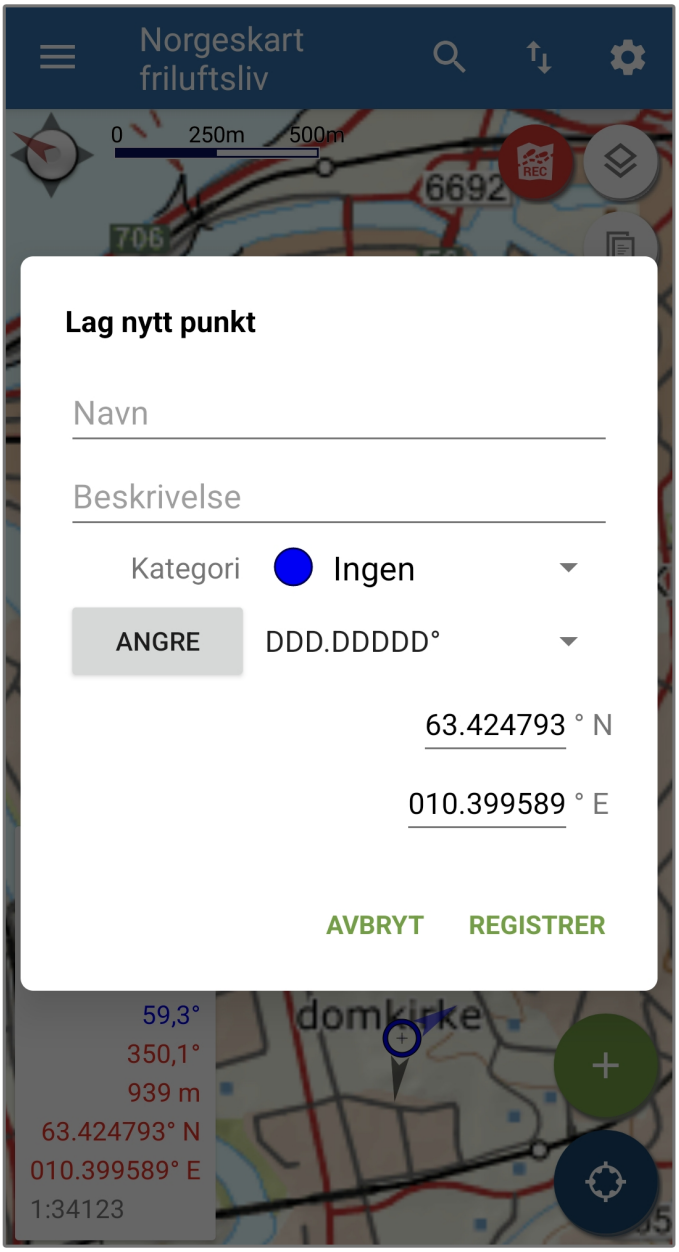
Applikasjonen har et detaljert kart over Norge og har flere kartstiler å velge mellom. Appen støtter å laste ned kartområder for framtidig offline-bruk. Et eksempelbilde av applikasjonen i bruk vises i Figur 6. Noen funksjoner appen tilbyr er blant annet å lage en sporlogg som kontinuerlig lagrer brukerposisjonen

[4] Google Play Store - Norgeskart friluftsliv https://play.google.com/store/apps/details?id=no.avinet.norgeskart
[5] Apple App Store - Norgeskart Outdoors https://apps.apple.com/no/app/norgeskart-outdoors/id1350671128

helt til den avsluttes, så vel som å registrere interessepunkt (vises i Figur 7). Denne dataen lagres lokalt på enheten og kan eksporteres som .gpx-filer i etterkant.



Figur 6: Skjermbilde fra Norgeskart friluftsliv på Android.



Figur 7: Skjermbilde fra Norgeskart friluftsliv på Android - Registrering av nytt punkt

Som nevnt i presentasjonen av Telespor i Seksjon 5.2.2, støtter denne applikasjonen å vise sporingsdata fra Telespors sporingsbjeller. Dette krever et «Dynamiske punkter-abonnement» som koster 50 kroner/år. Egne registreringer gjort i applikasjonen er kun lagret lokalt på enheten; å lagre denne dataen i skyen krever et «Premium»-abonnement som koster 59 kroner/år. Ved å lagre dataen i skyen får man da tilgang til å se registrert data på en webbasert versjon av applikasjonen på `https://norgeskart.avinet.no/`.

5.3.2.1 Drøfting av egnethet til Norgeskart friluftsliv

Mobilapplikasjonen Norgeskart friluftsliv ga oss et godt inntrykk i testingen vår; vi opplevde den som en responsiv applikasjon som har mange tilpasningsmuligheter når det kommer til å vise/skjule ønskede

skjermelementer.

Det aller største problemet med bruken av denne applikasjonen innenfor utføring av tilsyn av sau på beite er at interessepunktene man kan lagre i applikasjonen ikke tillater detaljer som gjerne er ønskelig innenfor den typen arbeid. Som vist i Figur 7 kan brukeren fylle ut navn- og beskrivelsesfelt, så vel som å oppgi en kategori. Kategoriene er tilpasset applikasjonens oppgitte formål som er friluftsliv – med andre ord er kategorier som «campingplass», «fiskeplass» og «utsiktspunkt» til stede, men ingen som har å gjøre med dyr. En bruker som benytter appen for bruk under tilsyn ville derfor være nødt til å bruke beskrivelsesfeltet til å fylle ut det meste av detaljer. Dette feltet er kun på én linje i høyde, istedenfor en større tekstboks som man finner i BeiteSnap sin registreringsskjerm (Figur 5). Dermed blir registreringen uoversiktlig med en gang observasjonen krever mer enn et par ord for å beskrive.

En annen ulempe er at man som nevnt må kjøpe et «Premium»-abonnement for å lagre data i skyen slik at man kan vise dataen på den tilknyttede webapplikasjonen. Ettersom all dataen er i fritekst er det heller ingen konkret måte å lage rapporter ut ifra data registrert via Norgeskart friluftsliv-appen.

5.4 Oppsummering av eksisterende løsninger

Alt i alt er det et behov for en løsning som gjør det lettere å få en oversikt over sau på beite enn det bruk av papirnotater tillater. Elektroniske sporingsbjeller kan være til stor hjelp, men tilfredsstiller ikke kravet for manuelt tilsyn. Ut ifra de eksisterende mobilapplikasjonene vi har sett på er BeiteSnap den som mest ligner på den originale visjonen for det helhetlige systemet vi forestilte oss, men også denne hadde noen ulemper. Blant annet virket det som at rapportfunksjonaliteten til BeiteSnap ikke var fullført når det ble utgitt, og det faktum at BeiteSnap uansett ikke er tilgjengelig for bruk lenger er en åpenbart ødeleggende faktor for å ta den i bruk som løsning.

Redegjørelsen for eksisterende løsninger belyser mangelen på et system som på én side tillater registrering av observasjoner via en mobilapplikasjon, og på en annen side tillater detaljert inspeksjon av utførte tilsyn samt generering av rapporter via en PC-basert løsning.

Del III:

Forarbeid

Dette prosjektet bygger videre på forarbeidet som ble gjort under fordypningsprosjektet i høstsemesteret 2021. Resultatet fra forarbeidet var en mobilapplikasjon for å registrere observasjoner som blir gjort under utførelse av tilsyn. Data som produseres av denne applikasjonen danner grunnlaget for utformingen og utviklingen av den nettbaserte løsningen som blir presentert i dette masterprosjektet. Det er derfor viktig å redegjøre for funksjonaliteten til mobilapplikasjonen og hva slags data den genererer som kan brukes videre i dette prosjektet.

6 Beskrivelse av mobilapplikasjonen

Mobilapplikasjonen er et verktøy for å hjelpe de som utfører tilsyn av sau på beite til å dokumentere observasjoner som blir gjort underveis i tilsynet. Mobilapplikasjonen kjører på både Android og iOS. Istedenfor å måtte skrive ned alle observasjoner som fritekst, som er den vanlige måten det gjøres på når et papirskjema for observasjoner benyttes, har mobilapplikasjonen forhåndsbestemte kategorier av observasjoner. Dette gjør det oversiktlig og effektivt å føre opp riktig type observasjon og korrekte detaljer for observasjonstypen.

For å illustrere nytten til mobilapplikasjonen kommer vi med et eksempel: En observasjon av en saueflokk vil gjerne involvere detaljer som for eksempel slipsfargene observert i flokken eller hvilke øremerkefarger man ser i flokken. For observasjon av rovdyr eller rovdyrspor er ikke disse detaljene relevante, men et felt for fritekstbeskrivelse om nøyaktig hva man så kunne heller vært nyttig.

Å dele observasjoner inn i tydelige kategorier tillater en mer tydelig brukerflyt og mer oversiktlig data. Å bruke en mobilapplikasjon framfor et papirskjema har likevel mange flere fordeler enn bare et penere brukergrensesnitt; en smarttelefon er utstyrt med programvare (klokke) og maskinvare (GPS-modul) som tillater mobilapplikasjonen å automatisk fylle ut detaljer om observasjonene som ellers hadde måtte blitt gjort for hånd (henholdsvis dato/tid for tilsynsrunde og ruten som ble tatt under tilsynsturen). Også det at dataen eksisterer digitalt og kan synkroniseres over internettet er en fordel, da dette tilrettelegger for enkel innsamling, behandling og presentasjon av historisk data – noe som utgjør en viktig del av dette masterprosjektet.

En redegjørelse for de mest relevante funksjonalitetene mobilapplikasjonen tilbyr presenteres i Seksjon 6.1. En oversikt over dataen som blir sendt fra brukerens mobiltelefon inn til backend-systemet, og dermed utgjør den tilgjengelige dataen som kan brukes i webapplikasjonen, presenteres i Seksjon 7.

6.1 Hovedfunksjoner i mobilapplikasjonen

For å beholde fokuset på webapplikasjonen som skal utvikles i dette masterprosjektet, vil kun de viktigste funksjonene i mobilapplikasjonen presenteres her. Med «viktigste» mener vi de funksjonene som er relevante til å registrere data (observasjoner og annen data fra utføring av tilsyn) som kan tas nytte av i webapplikasjonen. Andre funksjoner som kun er relevante internt i mobilapplikasjonen og ikke produserer data som sendes videre til backend-systemet, vil ikke nevnes her.

Hver funksjonalitet beskrives i den grad det er nødvendig; enkle og relativt selvforklarende funksjonaliteter blir beskrevet kort, mens mer innviklede funksjonaliteter utredes for i større grad.

6.1.1 Registrering og innlogging

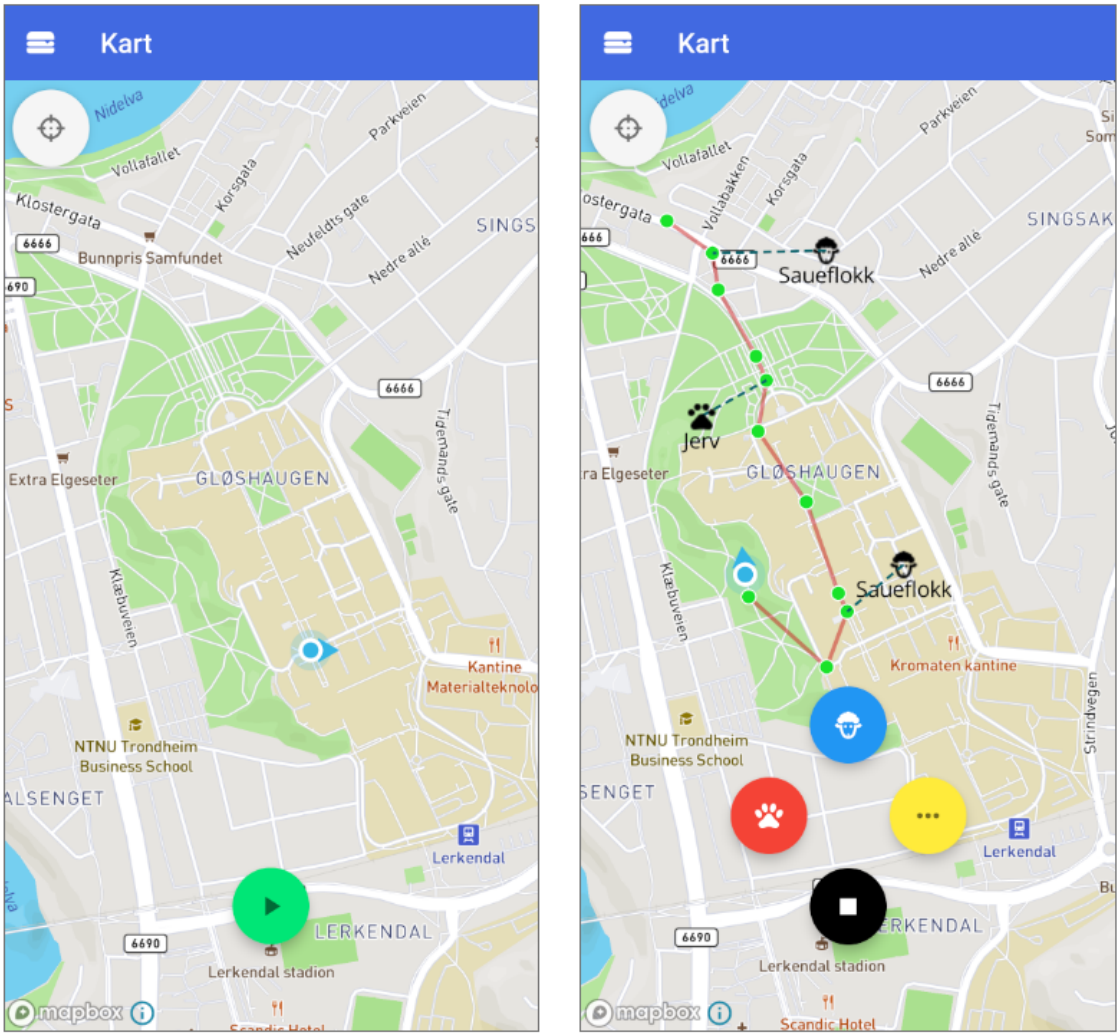
For å benytte mobilapplikasjonen må man være logget inn. Man oppretter en konto ved å oppgi en e-post og et passord, og logger så inn med disse kredentialene.

6.1.2 Kart og brukerposisjon

Startskjermen til mobilapplikasjonen (gitt at brukeren allerede har logget inn) er et kart hvor brukerens nåværende posisjon vises som en prikk på kartet. En start-knapp vises nederst i midten av skjermen og trykkes når brukeren er klar for å starte tilsynsrunden. Denne skjermen vises til venstre i Figur 8.

6.1.3 Utføring av tilsyn

Å trykke på «start»-knappen betegner starten på en ny tilsynsrunde, og brukerens posisjon vil kontinuerlig lagres og bli vist som punkter på kartet. Sporing av brukerens posisjon vil fungere selv om mobilen «låses» og plasseres i lommen. Tilsynsrunden kan til enhver tid stoppes ved å trykke på stopp-knappen, som er synlig etter en tilsynsrunde er påbegynt. Et eksempel på hva som kan sees når tilsyn er i gang vises til høyre i Figur 8.



Figur 8: Skjermbilder fra mobilapplikasjonen – Utføring av tilsyn. Til venstre: Klar for å starte tilsyn. Til høyre: Tilsyn er påbegynt og brukerens rute blir vist som grønne prikker med røde linjer som kobler dem sammen. Brukeren har registrert noen observasjoner og disse vises som svarte markører på kartet.

6.1.4 Registrering av observasjon

Brukeren kan velge mellom tre typer observasjoner: saueflokk, rovdyr eller «annet». Felles for alle disse er at observasjonen må plasseres et sted på kartet. Å plassere en observasjon på kartet vises i Figur 9. Etter å ha valgt riktig posisjon for observasjonen, tas brukeren inn på en egen skjerm for å registrere detaljer om observasjonen. Hver type observasjon har sin egen skjerm med ulike valgmuligheter.

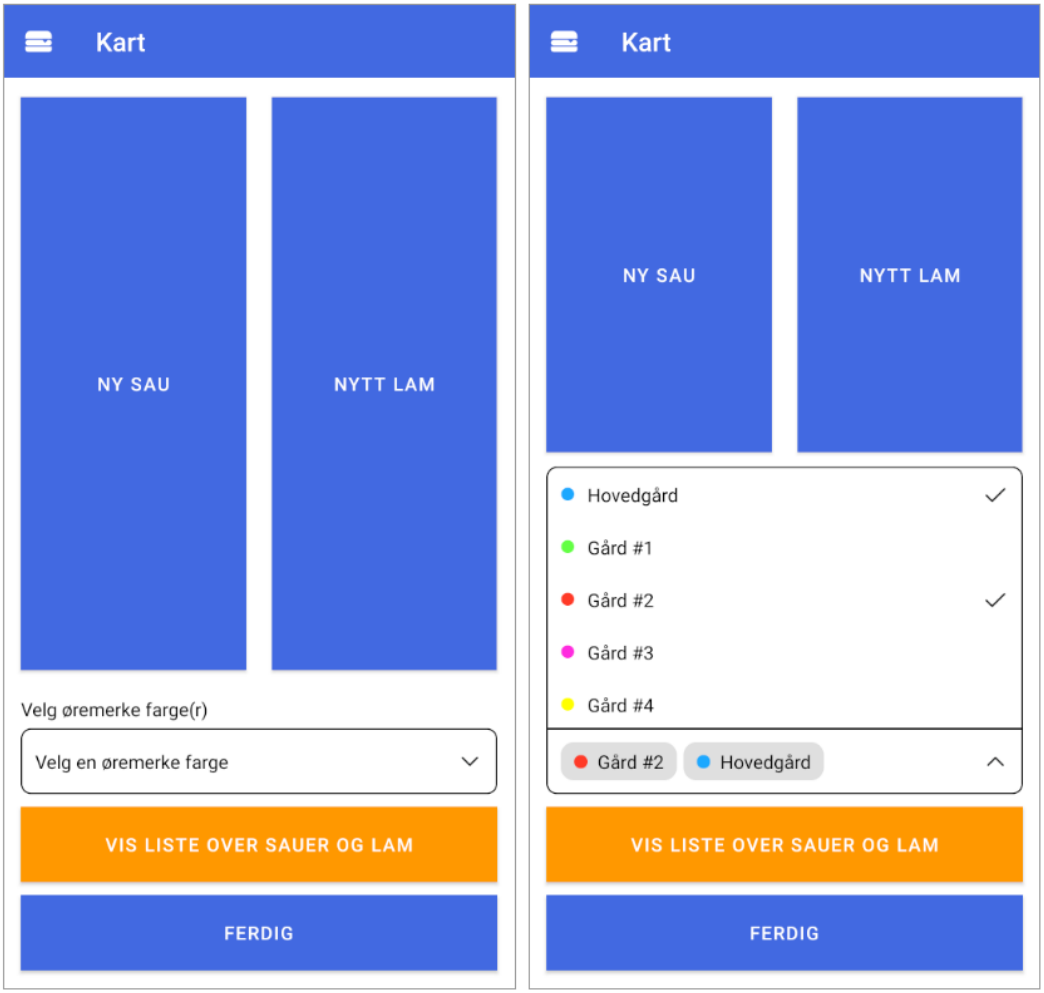
Figur 9: Skjermbilde fra mobilapplikasjonen – Plassere en observasjon på kartet.

6.1.4.1 Registrering av saueflokk

Hvis man skal registrere en saueflokk og plasserer markøren *innenfor* 30 meter fra der man står, presenteres brukeren med en skjerm for å registrere detaljer om saueflokken. 30 meter-kravet kommer av at veileder Svein-Olaf Hvasshovd informerte om at øremerkefarger vil være vanskelig å observere på avstander lengre enn dette. Skjermen for registrering av en saueflokk vises i Figur 10. Sau og lam legges enkeltvis til i flokken ved å trykke på knappene «Ny sau» og «Nytt lam». For hver sau må man registrere slipsfarge på sauen (eller eventuell fravær av slips). En tekst til tale-funksjon hjelper brukeren å gjøre riktig valg ved å uttale hvilken slipsfarge som er aktiv uten at de nødvendigvis behøver å se på skjermen. For lam trengs det ikke noen videre detaljer. Brukeren legger til lam og sau helt til kvantiteten av sau og

lam stemmer for flokken. Før man er ferdig må brukeren velge hvilke øremerkefarger som er representert i flokken via en nedtrekksmeny (fargene skal legges inn på forhånd av brukeren før utføring av tilsyn – se Seksjon 6.1.6). Valg av øremerkefarger i flokken vises i høyre skjermbilde i Figur 10. Øremerker telles altså ikke per dyr, men det oppgis istedenfor hvilke øremerkefarger som er representert i flokken. Når brukeren trykker «ferdig» navigeres brukeren tilbake til kartet og et ikon for saueflokken vil være plassert på kartet.

Om saueflokken plasseres *utenfor* 30 meter fra brukerens posisjon, kan brukeren ikke legge inn noen detaljer på flokken. Det blir kun lagret en flokk uten detaljer, og et ikon for saueflokken blir plassert på kartet.



Figur 10: Skjermbilder fra mobilapplikasjonen – Registrere detaljer om saueflokk.

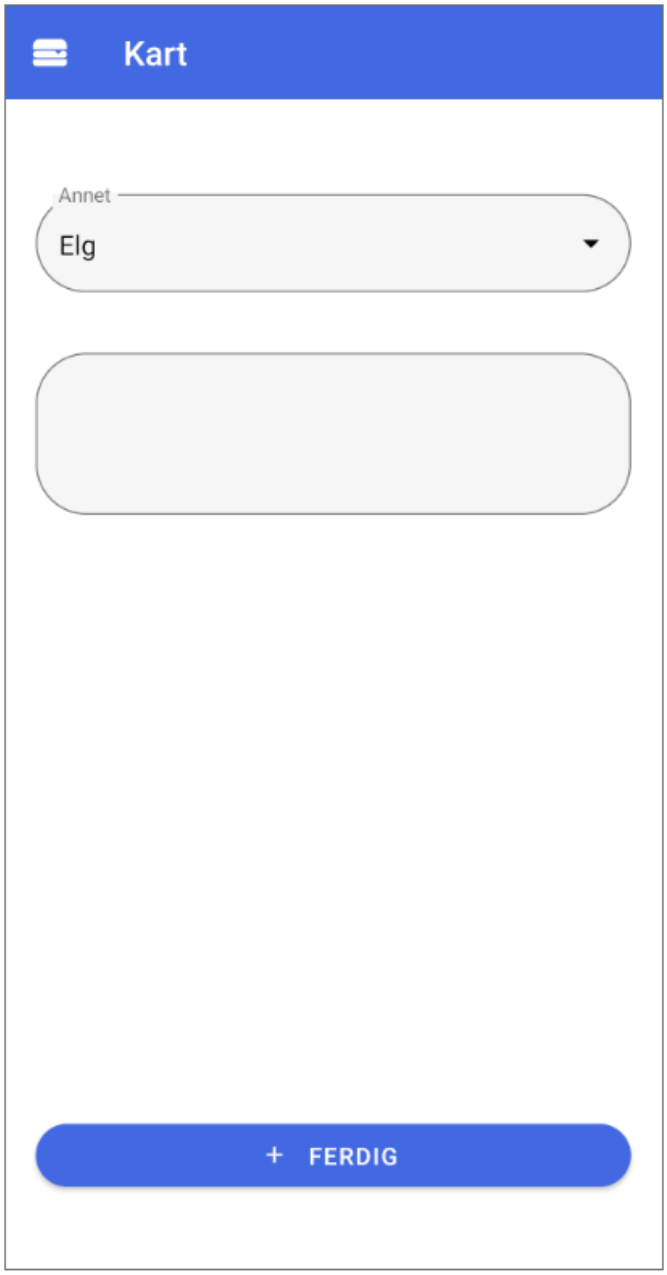
6.1.4.2 Registrering av rovdyr

Etter plassering av markør kan brukeren bestemme hva slags rovdyr de har observert eller sett spor av – for eksempel jerv eller ulv. De rovdyrene som står for flest angrep på sau i Norge er representert som valgmuligheter. Det kan være vanskelig å gjenkjenne nøyaktig hva slags rovdyr man ser tegn til basert på for eksempel fotspor, og man kan derfor velge «annet» som navnet på rovdyret for å etterlate denne tolkningen til andre, eller om man heller ønsker å oppgi en fritekstbeskrivelse av rovdyret. Figur 11 viser skjermen for registrering av rovdyr.

Figur 11: Skjermbilder fra mobilapplikasjonen – Registrere rovdyr.

6.1.4.3 Registrering av annet

I samme stil som registrering av rovdyr, etter plassering av markør får man her valg om ulike typer observasjoner som hverken er rovdyr eller saueflokker. Valgmuligheter her er blant annet elg, reinsdyr og hund. Det finnes også «annet» som valgmulighet hvis det man har observert ikke faller under noen av disse. Uansett hvilken verdi man velger her kan man oppgi en fritekstbeskrivelse. Figur 12 viser skjermen for registrering av «annet».



Figur 12: Skjermbilder fra mobilapplikasjonen – Registrere «annet».

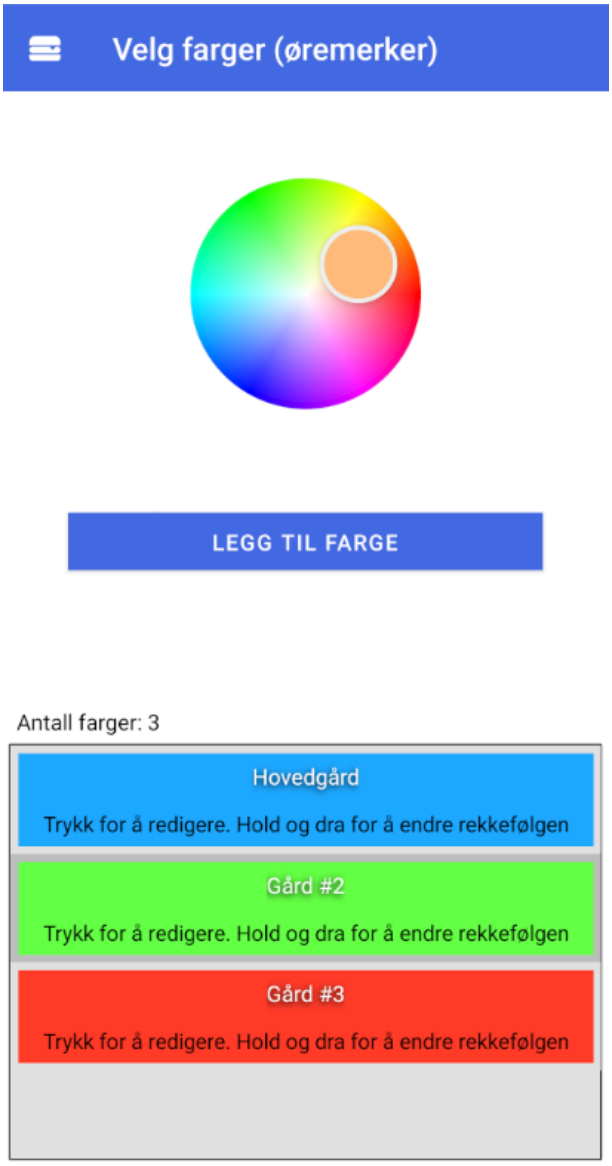
6.1.5 Redigering av observasjoner

Hver observasjon som har blitt opprettet i nåværende tilsynsrunde kan redigeres. Ved å trykke på markøren på kartet tas brukeren inn på den samme skjermen som når observasjonen først ble opprettet, og her kan man gjøre de endringene man måtte ønske.

6.1.6 Lagring av øremerkefarger

Som nevnt i forarbeidet er hver gård pålagt å markere sauene med øremerker, og de velger selv hvilken farge de skal ha på øremerkene. Øremerkene kan være ensfargede eller tofargede. For gårder som slipper sau ut på beite i lag med andre gårder er det gunstig å velge en farge som skiller seg fra nærliggende gårder, slik at fargen lett er identifiserbar for en gjeter som er ute på tilsyn. Fordi øremerkefargene en gjeter vil møte på vil variere fra område til område, må brukeren av mobilapplikasjonen selv lagre hvilke øremerkefarger som er relevante for beiteområdet de kommer til å utføre tilsyn på.

Skjermen for lagring av øremerkefarger tillater brukeren å lagre opp til ti farger. Brukeren bruker fargevelgeren til å peke ut ønsket farge og trykker så på «Legg til farge» for å legge den til i listen over farger. De lagrede fargene kan redigeres eller slettes. Det er også mulighet for å endre rekkefølgen på listen av farger. Fargen som er øverst i lista vil oppgis som «Hovedgård». Disse fargene benyttes i skjermen hvor brukeren oppgir detaljer om en saueflokk, se Figur 13.



Figur 13: Skjermbilde fra mobilapplikasjonen – Fargevelgeren for å angi hvilke øremerkefarger som er relevante for tilsynsrunden.

6.1.7 Redigering av utført tilsynsrunde

Mobilapplikasjonen lar brukeren se en liste over utførte tilsyn. Hver tilsynsrunde kan slettes om ønskelig, som for eksempel hvis brukeren har registrert et tilsyn ved et uhell. Det er også mulig å redigere en tidligere utført tilsynsrunde – her kan man for eksempel slette en observasjon som viste seg å ikke stemme, legge til flere sauer i en tidligere registrert saueflokk eller legge til helt nye observasjoner. Også brukerens posisjon vil loggføres mens man redigerer tilsynsrunden. Å redigere en avsluttet tilsynsrunde fungerer i praksis som å gjenoppta den avsluttede tilsynsrunden – alle observasjoner og brukerposisjoner som var en del av tilsynsrunden vil vises for brukeren, og nye kan legges til.

6.1.8 Synkronisering

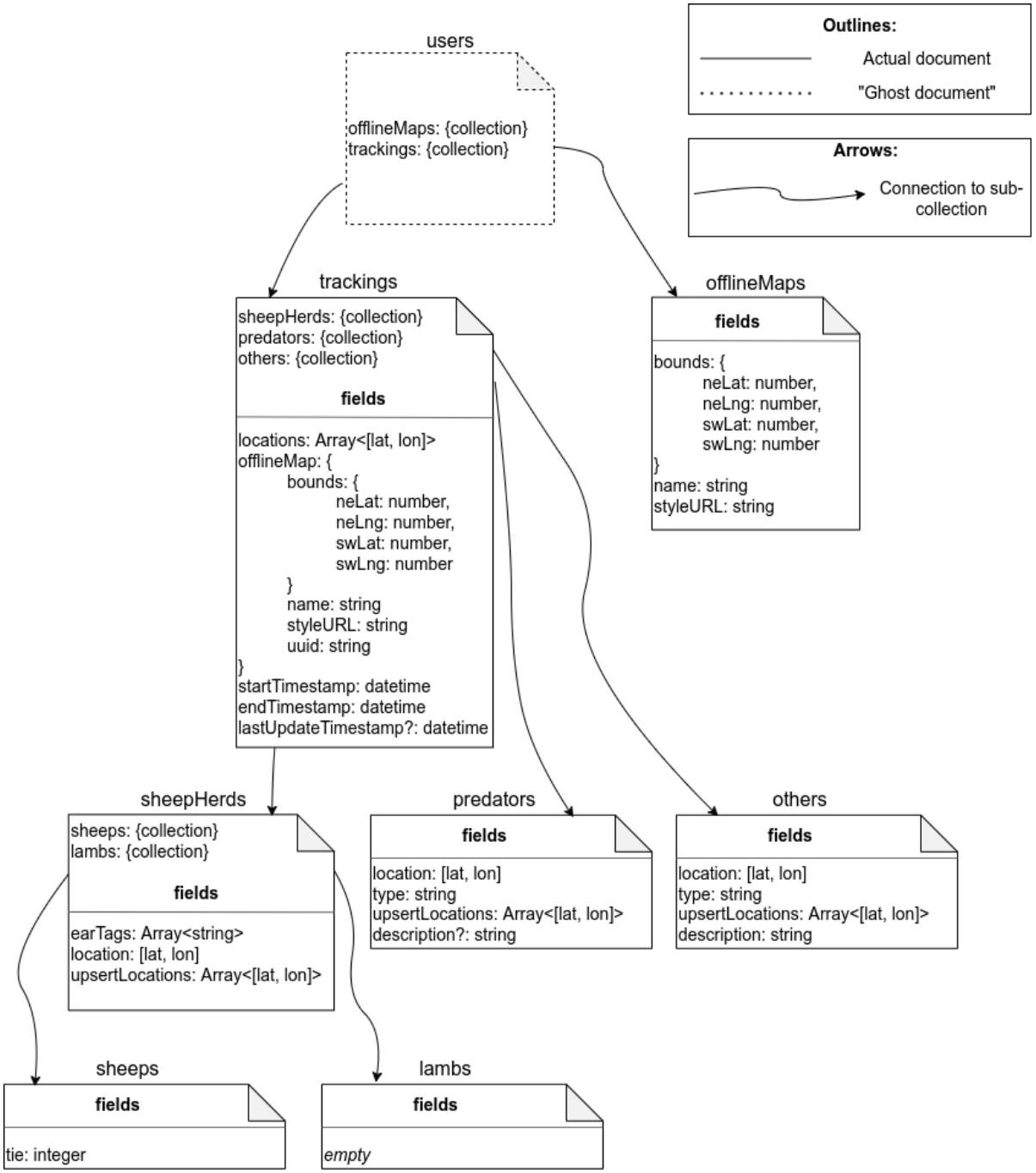
Data fra mobilapplikasjonen vil synkroniseres over internett automatisk – brukeren trenger ikke forholde seg til å utføre synkronisering manuelt. Mobilapplikasjonen er laget for å fungere selv uten nettilgang ettersom det kan være varierende hvorvidt det er dekning med mobilnett i utmarka. Når mobilen får nettilgang igjen, vil dataen sendes inn til backend-systemet i bakgrunnen.

7 Data tilgjengelig fra mobilapplikasjon

For å utvikle et system som tar i bruk data fra mobilapplikasjonen, er det viktig å få en forståelse for hvilken data som er tilgjengelig. Det er nettopp denne dataen som skal presenteres for brukeren i den nettbaserte løsningen denne masteroppgaven fokuserer på. Om det eventuelt skulle vise seg å være data som er nødvendig for å implementere en spesifikk funksjonalitet, men som ikke er tilgjengelig ut ifra datamodellen vår, må dette korrigeres ved å gjøre endringer i mobilapplikasjonen slik at denne dataen blir tilgjengelig.

Datamodellen som brukes for data fra mobilapplikasjonen vises i Figur 14. Dataen er lagret i Firebase sin Firestore-tjeneste, som er en skybasert database. Firestore benytter en dokumentmodell. Data i Firestore er lagret i dokumenter (engelsk: «documents»); hvert dokument er en samling med nøkkel-verdi-par. En samling av flere dokumenter kalles en «collection» – vi kommer til å omtale dette som en «kolleksjon» for å forhindre å forvirre «kolleksjon» med begrepet «samling» som kan brukes i en bredere sammenheng. Hvert dokument kan igjen bestå av såkalte «subcollections» eller subkolleksjoner, undergrupperinger av kolleksjoner. Til sammen utgjør altså databasen et hierarki av kolleksjoner og dokumenter [39].

Hver bruker har en kolleksjon av tilsyn, kalt *trackings* i vår datamodell. Hvert tilsynsdokument har relasjoner til saueflokker, rovdyr og 'annet'-observasjoner (henholdsvis kalt *sheepHerds*, *predators* og *others* i datamodellen). Hver saueflokk består av flere sauer og lam (*sheeps* og *lambs*). I tillegg har hver bruker en kolleksjon med offlinekart-data (*offlineMaps*), som lagrer kartområdene brukeren har lagret for offline-bruk.



Figur 14: Oversikt over all dataen tilgjengelig fra backend-systemet. Dataen blir sendt inn fra mobilapplikasjonen.

Dokumentet på toppnivå, *users*, er et såkalt «nonexistent ancestor document» [40] – et dokument som i praksis ikke eksisterer. Det går ikke an å utføre spørringer på dette dokumentet, men det benyttes fordi dataen må struktureres i et hierarki. «Users» har ingen felt fordi brukerkontoer ikke håndteres i databasen, men heller av Firebase sitt eget autentiseringsplatform Firebase Authentication. «Users» eksisterer derfor for å definere et toppnivå i hierarkiet, ettersom dataen fra én bruker lagres under den brukeren – men selve brukeren er kun en unik ID i systemet og er ikke lagret i et dokument.

Hvert dokument som har en referanse til en subkolleksjon, slik som «sheepHerds» med sine subkolleksjoner

av «sheeps»- og «lambs»-dokumenter, kan inneholde ingen eller flere instanser av disse dokumentene. Dette etterligner en 1:n-relasjon i en relasjonell datamodell. Hvert dokument har også en unik ID, men denne håndteres av Firebase og er ikke representert som et av feltene på dokumentet.

Felt med data av type [lat, lon] står for «latitude» (norsk: breddegrad) og «longitude» (norsk: lengdegrad). Disse er altså felt som inneholder geografiske koordinater hentet fra mobilens GPS.

Feltet **locations** i *trackings*-dokument er en liste med brukerposisjoner under tilsynet. Dette tillater oss å tegne opp brukerens rute under tilsynet.

Feltet **upsertLocations** i *sheepHerds*, *predators* og *others*-dokumentene er en liste med brukerposisjoner *når de respektive observasjonene ble plassert på kartet*. Dette er fordi det skal trekkes en linje fra brukerens posisjon til der observasjonen ble plassert. En linje skal også trekkes fra brukeren hver gang en observasjon blir redigert. Feltet upsertLocations brukes altså for å holde alle disse brukerposisjonene per observasjon.

8 Refleksjon og vurdering av forarbeid

I utførelsen av fordypningsprosjektet fikk vi implementert nesten all funksjonaliteten som var planlagt – hvorfor noe arbeid gjenstod vil vi komme tilbake til i Seksjon 8.1. Det ble utført to faser med brukertesting som resulterte i mange forbedringer når det gjaldt brukskvaliteten på mobilapplikasjonen, og etter forbedringene fra siste fase med brukertesting var implementert konkluderte vi med at brukskvaliteten på applikasjonen var god, og alt i alt var vi fornøyde med hvordan fordypningsprosjektet gikk.

8.1 Viktigheten av formell notasjon av krav

I fordypningsprosjektet hadde vi en beskrivelse av funksjonelle krav, men utredelsen av disse fulgte ikke en formell mal eller struktur og ble istedenfor presentert i tekstparagrafer. Dette, sammen med mangelen på en grundig gjennomgang av om alle de funksjonelle kravene var blitt implementert i en tilfredsstillende grad, gjorde at ett funksjonelt krav hadde blitt gjenglemt og ikke var implementert innen slutten av fordypningsprosjektet. Dette var spesifikt et krav om å kunne registrere døde eller skadde sau i mobilapplikasjonen. En viktig lærepenge vi fikk fra utførelsen av fordypningsprosjektet var derfor å sørge for å være grundig med utredelsen av funksjonelle krav, så vel som å sørge for at implementasjonen av disse følges opp. Dette motiverte oss til å ta i bruk en mye mer strukturert og formell notasjon av funksjonelle krav under utførelsen av dette masterprosjektet, med hensikt om å unngå at et funksjonelt krav unnslipper oss igjen under prosjektforløpet. Påvirkningen av dette kommer fram i Seksjon 10.3.2. Videre motiverte det oss til å utføre en grundig evaluering av implementasjonen av de tekniske kravene mot slutten av dette prosjektforløpet, noe som kommer fram i Seksjon 21.

8.2 Fokus på systemets helhet versus fokus på enkeltaspekt

En annen mulig svakhet med utførelsen av forarbeidet var at det fordypningsprosjektet omhandlet – en mobilapplikasjon – kun dreide seg om selve mobilapplikasjonen; vi fikk ikke en følelse for at vi utviklet applikasjonen innenfor konteksten av et helhetlig system (som altså inkluderer webapplikasjonen som utvikles i dette masterprosjektet). Vi fikk kun oppgitt informasjon om hva vi burde gjøre med utviklingen av mobilapplikasjonen fra veileder, og den framtidige webapplikasjonen ble ikke diskutert i noen stor grad. Dette førte dermed til at vi ikke var sikre på de tekniske detaljene rundt masterprosjektets produkt under utførelsen av fordypningsprosjektet. Det finnes både positive og negative aspekter rundt dette.

Et positivt aspekt rundt å kun fokusere på selve mobilapplikasjonen under fordypningsprosjektet var at det minsket sannsynligheten for å bli distrahert av detaljer som ikke var relevante for øyeblikket. Vi

behøvde kun å ta avgjørelser om ting som var relevante for mobilapplikasjonen, og slapp å tenke på ting «i det større bildet» som kunne gjøre planleggingen og den tekniske utformingen av mobilapplikasjonen mer komplisert.

En åpenbar ulempe med å ikke fokusere på framtidige tekniske aspekter av prosjektet (nemlig webapplikasjonen) var at det i en senere anledning kunne oppstå «mismatch» mellom hva mobilapplikasjonen har som funksjonalitet eller lagrer data av, og det webapplikasjonen har som databehov. Dette kunne gi opphav til mulig etterarbeid hvor mobilapplikasjonen måtte endres i etterkant for å passe bedre overens med de tekniske kravene til webapplikasjonen.

På grunn av den begrensede tiden vi hadde til å utføre fordypningsprosjektet, synes vi fokuset på mobilapplikasjonen alene var passende. Under den tekniske planleggingen av webapplikasjonen fant vi likevel ut at det ville bli nødvendig med visse endringer i mobilapplikasjonen i etterkant for å imøtekomme webapplikasjonens databehov, som vil bli utredet for videre i Seksjon 10.2.

Del IV:

Planlegging

Planlegging er kanskje det viktigste steget i hele produktutviklingsprosessen; uten ordentlig planlegging skal det lite til for å enten ikke bli ferdig til fristen eller ende opp med et resultat som ikke oppfyller alle kravene en kunde eller andre interessenter har. I denne delen vil vi gå gjennom utviklingsprosessen, alle de forskjellige kravene og den planlagte programvarearkitekturen.

9 Prosess

Prosessen i et programvareutviklingsprosjekt er viktig ettersom det både bestemmer utviklingsmetodikken og skisserer tidsplanen for prosjektet, noe som bestemmer hvordan teamet skal jobbe og når de burde bli ferdig med spesifikke mål for å kunne nå tidsfristen for prosjektet. I denne seksjonen vil vi gå gjennom nettopp det; metodikken og prosjektets tidsplan.

9.1 Metodikk

For at utviklingen innenfor et prosjekt skal gå så smidig som mulig uten forsinkelser eller feil, er det viktig at man følger et system hvor man har en plan for prosjektet og en struktur og kontroll over prosessene som inngår i utviklingen. Dette er hvor utviklingsmetodikker kommer inn. Dette er rammeverk som hjelper teamet med å planlegge, strukturere og kontrollere prosjektet og prosessene som inngår på en utprøvd måte som har sine kjente fordeler og ulemper.

9.1.1 Utviklingsmetodikker og rammeverk

De mest kjente metodikkene som blir brukt innen industrien er Waterfall [41], DevOps [42], Iterative [43] og Agile [44]. Disse har alle sine fordeler og ulemper, men i dette prosjektet så vil Agile/Iterativ bli brukt, som forklart mer i dybden i Seksjon 9.1.2. Dermed vil bare Agile- og Iterative-implementasjoner bli vurdert i neste avsnitt.

Innenfor Agile og Iterative metodologi er det i all hovedsak Kanban [45], Scrum [45], Systems Development Life Cycle (SDLC) [46] og Extreme Programming [47] som er de rammeverkene/implementasjonene som blir brukt innen industrien. Av disse er det hovedsakelig Kanban og Scrum som blir brukt mest i typiske programvareutviklingsprosjekter som følger en Agile metodikk.

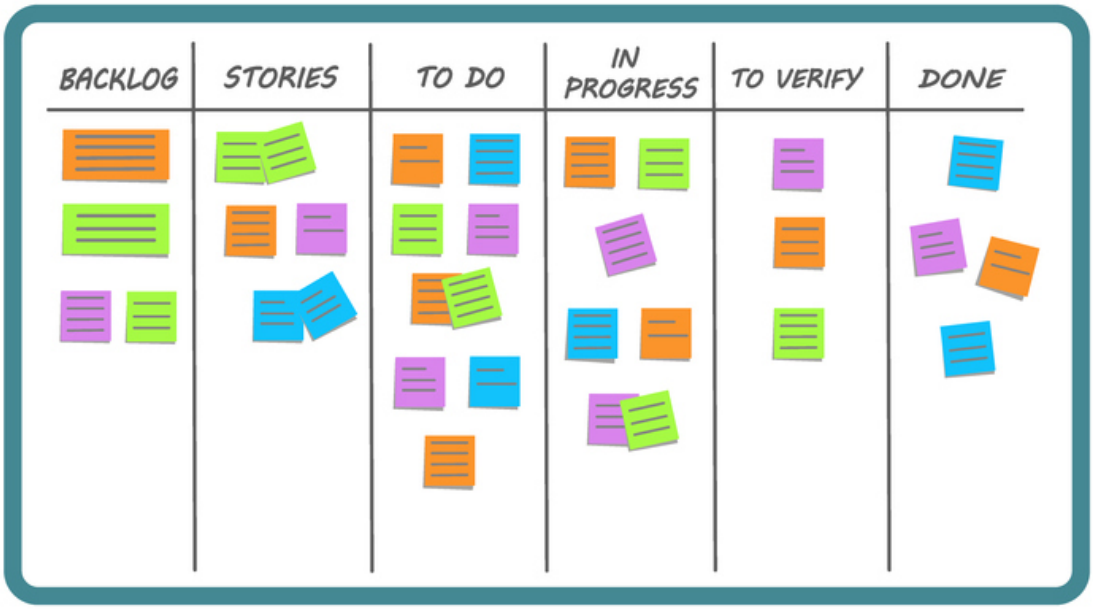
Scrum er et rammeverk som hjelper team med å jobbe sammen og koordinere utviklingen med kundens ønsker. Gjennom «sprintplanlegging» og en «backlog» blir kundens ønsker tatt til følge før man starter en sprint på en fast lengde, hvor oppgavene som ble plukket ut under sprintplanleggingen blir fullført. Under sprinten blir det utført daglige møter kalt «Daily Scrum» som forbedrer teamets koordinasjon og kommunikasjon. Når sprinten er ferdig blir den vurdert gjennom «sprint review» og «sprint retrospective» som skal finne ut hva som gikk bra og hva som kan forbedres. I tillegg til dette er medlemmene ofte tildelt forskjellige roller slik som «Scrum Master», som skal ha en overordnet koordineringsrolle. Fordelene med Scrum blir altså at man får en relativt smidig utvikling (avhengig av lengden til sprintene) samtidig som man øker og forbedrer kommunikasjonen innad i teamet og med kunden, noe som skal føre til et bedre produkt som samstemmer bedre med hva kunden ønsker.

Kanban hjelper team med å definere, administrere og forbedre utviklingsprosessen. Arbeidet, gjerne delt inn i mindre oppgaver, er representert gjennom et «kanban board» som kan være en tavle, vegg, vindu, og så videre, som er delt inn i kolonner med semantisk betydning som vist i Figur 15. Én kolonne kan for eksempel være oppgaver som ikke er startet enda, en annen kan være oppgaver som har blitt startet og den siste kan være oppgaver som er ferdige. Disse oppgavene kan teammedlemmer deretter velge fritt mellom så lenge ingen andre har tatt dem selv, og arbeidet blir dermed veldig smidig og kontinuerlig ved at oppgaver blir fullført etter hvert som medlemmer har tid til å starte og fullføre dem. Kanban kan ligne på Scrum med hvordan oppgaver blir delt opp og smidigheten av hvordan oppgaver blir valgt, men det er noen nøkkelforskjeller mellom dem. I motsetning til Scrum, hvor man har sprinter på fast lengde, er Kanban en kontinuerlig strøm uten faste intervaller. I tillegg til dette har Kanban heller ingen roller, og utrulling av kode blir gjort kontinuerlig istedenfor ved slutten av sprinten. Kanban er også mer smidig enn Scrum ettersom forandringer kan skje når som helst på «kanban board»-et, men i Scrum skal man gjerne ikke forandre noe før neste sprint.

9.1.2 Vår implementasjon av valgt metodikk og rammeverk

Å ta i bruk og implementere en metodikk er viktig for alle prosjekter, men i størst grad når man har et team med flere utviklere. Når et team bare består av to personer, som dette prosjektet gjør, vil det i mindre grad ha en betydning ettersom det er enklere å samarbeide og kommunisere når det kun er to personer. Det er likevel fortsatt viktig å implementere en egendefinert versjon av et rammeverk som passer prosjektets størrelse.

For oss ble Agile metodikk med Kanban-rammeverket den beste løsningen. Å velge en Agile metodikk ga oss friheten og smidigheten som trengs for produktutvikling med en begrenset tidsfrist, hvor hurtig prototyping ble brukt for å raskt komme fram til nødvendige krav og et konsept som fungerte. Ettersom teamet bare består av to personer så ville et rammeverk som Scrum blitt for tidkrevende og kompleks, og ville sannsynligvis ført til mer arbeid enn det man hadde spart gjennom fordelene det gir. Kanban ble dermed det perfekte valget, ettersom det passer bra til produktutvikling uten en kunde med aktive ønsker, et mindre team uten kompleks rolleinndeling og en såpass kort tidsfrist at bruk av sprinter hadde økt effektiviteten minimalt. Kanban gir samtidig enda mer smidighet og fleksibilitet enn det Scrum gir.



Figur 15: Eksempelbilde av et Kanban board. Kilde: [48].

9.2 Prosjektets tidsplan

For å sørge for at man har en klar oversikt over når milepæler bør bli ferdig og hvilke avhengigheter som finnes mellom oppgaver, og totalt sett for å sørge for at man blir ferdig med prosjektet innen fristen, er det viktig å ha en tidsplan.

For dette prosjektet valgte vi å ta i bruk et Gantt-diagram som er et kjent verktøy innenfor programvareutvikling for å få en visuell oversikt over oppgavene som må bli utført over en tidsperiode. Dette diagrammet kan bli sett i Figur 16 og viser den estimerte tidsplanen vår i form av et forenklet Gantt-diagram. Grunnen til at det er forenklet er at Gantt-diagram kan ha funksjonalitet som framgangsvisning,

oppdragsindikator for hvem som gjør hva, og så videre. Dette var ikke nødvendig i et tomannsprosjekt som bare varte i 21 uker. Det skal sies at dette ikke er en plan med faste tidsfrister ettersom det ikke ville vært spesielt Agile, men heller et estimat og et mål for prosjektet.

Tidsplanen er delt inn i ni deler som representerer forskjellige grupper med oppgaver innenfor prosjektet som må bli gjort innen tidsfristen 13. juni. Disse oppgavene kan igjen ha dependencies, hvor en oppgave eller gruppe med oppgaver er avhengig av at en annen oppgave er ferdig før man kan starte arbeidet på den eller dem. I avsnittene nedenfor vil hver del av tidsplanen bli gjennomgått og forklart, men for å få en full oversikt over tidsplanen er det fortsatt viktig at diagrammet ligger i hovedfokus, mens tekstforklaringen blir brukt som supplement til diagrammet.

Den grå delen som kan bli funnet øverst i diagrammet representerer oppgavene assosiert med skrivingen av masteroppgaven, som altså er denne rapporten. Disse fem oppgavene går ut på å skrive forskjellige deler av rapporten. Grunnen til at skrivingen er delt opp i forskjellige oppgaver kommer av at enkelte deler av rapporten ikke kunne bli skrevet før andre oppgaver – som utvikling, brukertest 1 eller brukertest 2 – var ferdig, og kan bli sett i diagrammet som dependencies.

Den lysegrønne delen representerer oppgavene assosiert med planlegging av prosjektet. Denne delen er bare delt opp i to oppgaver: definering av omfang og definering av krav. I tillegg til dette er det også to milestones som representerer at disse oppgavene er ferdig, noe som er en dependency for at designfasen kan starte.

Den lilla delen representerer oppgavene assosiert med å designe webapplikasjonen. Som vist i diagrammet så har den en dependency fra den lysegrønne delen, som tilsier at vi ikke kan starte med designfasen før vi er ferdig med planleggingsfasen. Denne delen er delt opp i fem oppgaver som går ut på å designe et høynivå konsept og flere iterasjoner av ferdigstilte prototyper. I tillegg til dette er det også tre milestones som representerer at de ferdigstilte prototypene er ferdig, som blir brukt som dependencies for når man kan starte med å skrive forskjellige deler i rapporten.

Den gule delen representerer oppgavene assosiert med oppsettet av utviklingsmiljøet og CI/CD-pipelinen. Denne delen er bare delt opp i to oppgaver: oppsettet av basiskode og oppsettet av pipelines. Dette er oppgaver som kan bli gjort under designfasen, og burde være ganske nærme å bli ferdig til utviklingsfasen ettersom det er oppsett som ikke blir påvirket av design og som trengs for å starte utviklingen av webapplikasjonen.

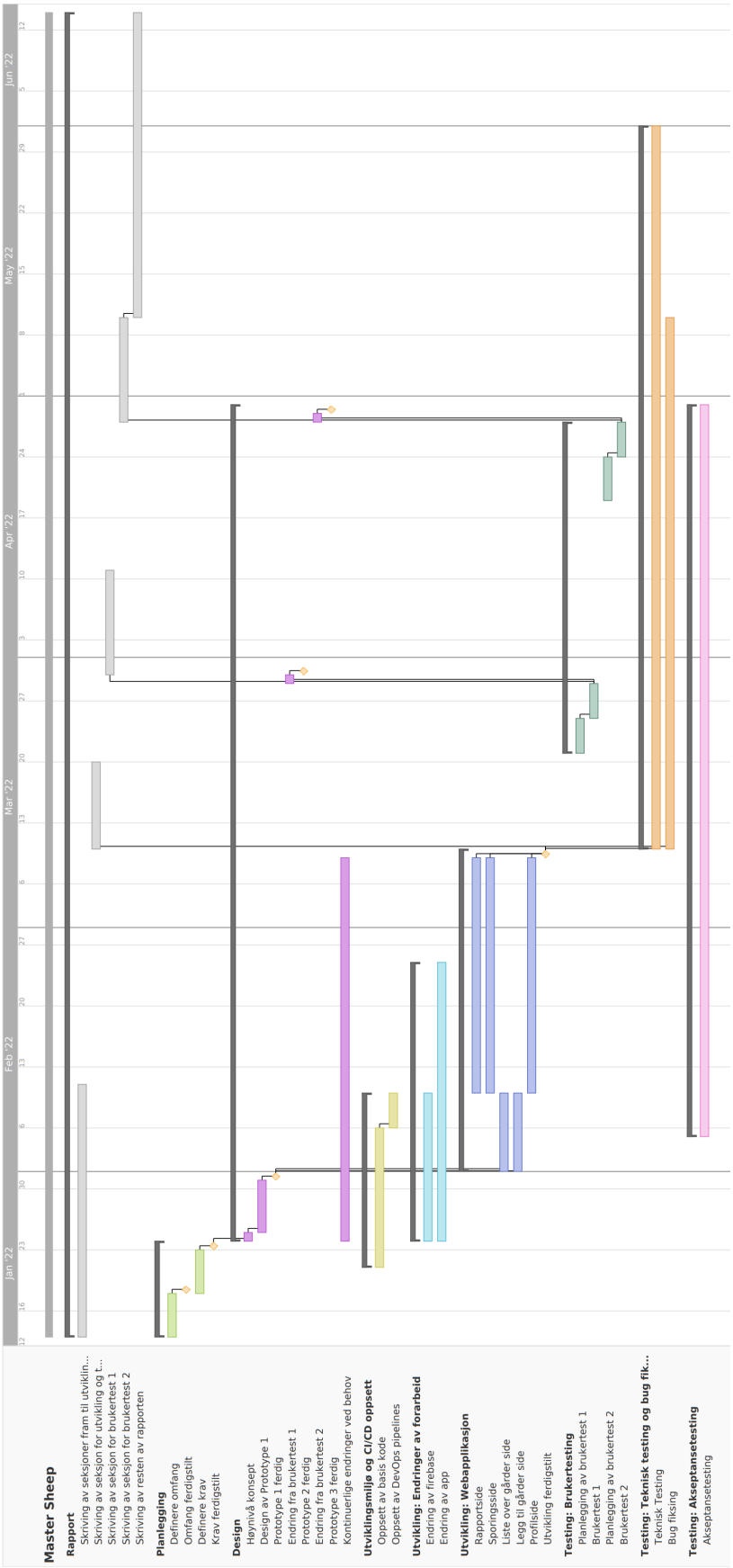
Den lyseblå delen representerer oppgavene assosiert med utviklingen av endringer på systemet fra forarbeidet. Denne delen består også av to oppgaver: endring av mobilapplikasjonen og endring av backend-systemet. Dette er deloppgaver (hovedsakelig endringer i backend-systemet) som til dels måtte bli gjort før vi kunne starte med utviklingen av webapplikasjonen. Det er også deloppgaver (hovedsakelig endringer i mobilapplikasjonen) som kunne gå parallelt med utviklingen av webapplikasjonen.

Den lyseblå delen representerer oppgavene assosiert med utviklingen av webapplikasjonen. Som vist i diagrammet så har den en dependency fra milestonen at prototype 1 er ferdig, altså kan ikke utviklingen starte før første prototype er ferdig. Denne delen er delt inn i fem oppgaver som er hovedkategoriene for de funksjonene som systemet måtte innfri. I tillegg er det også en milestone som er en dependency for oppgaven om å skrive om utviklingen i rapporten og oppgaven om å utføre tekniske tester.

Den mørkegrønne delen representerer oppgavene assosiert med de forskjellige fasene med planlegging og utføring av brukertesting. Oppgavene i denne delen består altså av planlegging og utføring av brukertest-fase 1 og brukertest-fase 2. Disse oppgavene er også en dependency for designendringer og ferdigstilling av nye prototyper.

Den oransje delen representerer oppgavene assosiert med å lage tekniske tester (end-to-end testing). Oppgavene er delt inn i å skrive og kjøre testene og å fikse eventuelle feil som oppdages. Disse oppgavene har en dependency fra utviklingsdelen – de tekniske testene starter altså ikke før utviklingen er ferdig.

Den rosa delen representerer oppgavene assosiert med akseptansetesting. Denne delen har bare én oppgave som er akseptansetestingen, og den blir kjørt kontinuerlig fra litt etter utviklingen har startet, helt til siste prototype har blitt ferdigstilt. Dette kommer av at akseptansetestingen gjøres kontinuerlig fra det første funksjonelle kravet har blitt innfridd til det siste funksjonelle kravet har blitt innfridd.



Figur 16: Tidsplan for prosjektet i form av et Gantt-diagram.

10 Krav

I denne seksjonen vil vi dokumentere krav for systemet, i form av påkrevde bruksmønstre, funksjonelle- og ikke-funksjonelle krav og arkitektoniske betydelige krav. Med «systemet» omtaler vi både webapplikasjonen og mobilapplikasjonen som en helhet, ettersom disse interopererer med hverandre og har høyest nytteverdi når de brukes sammen. Ettersom mobilapplikasjonen allerede er utviklet, vil vi fokusere mest på kravene for webapplikasjonen i denne seksjonen. Vi vil likevel også diskutere mobilapplikasjonen for eventuelle endringer som gjøres på denne, slik at den bedre imøtekommer systemets behov. Mobilapplikasjonen og webapplikasjonen har et felles backend-system, og når vi presenterer krav for spesifikt webapplikasjonen eller mobilapplikasjonen vil disse dreie seg om frontend-aspektene for disse. Krav for webapplikasjonen vil presenteres i Seksjon 10.3, mens krav om endringer og tillegg til mobilapplikasjonen vil presenteres i Seksjon 10.4.

10.1 Elisitering

I løpet av fordypningsprosjektet høsten 2021 og starten av masteroppgaven våren 2022 var det ikke mulighet for å møte bønder og dermed elisitere målgruppen direkte. Heldigvis hadde veilederen vår – Svein-Olaf Hvasshovd – bred kunnskap rundt problemstillingen gjennom flere års erfaring, noe som var veldig verdifullt for planleggingen av prosjektet.

Gjennom de første ukentlige møtene ble de overordnede kravene og funksjonene beskrevet og utarbeidet gjennom veilederens bakgrunnsinformasjon, samt at gruppemedlemmene stilte spørsmål hvor vi elisiterte krav og begrensninger. Dermed fikk vi utarbeidet noen av de funksjonelle og ikke-funksjonelle kravene, i tillegg til de arkitektoniske betydelige kravene (ABK). Det var derimot fortsatt behov for å utarbeide noen av de mindre og mer spesifikke kravene. Dermed ble hurtig prototyping tatt i bruk for å raskt kunne presentere nye funksjoner og implementasjonsdetaljer for veilederen som da kunne gi oss tilbakemeldinger og eventuelle nye krav eller ønsker om funksjonalitet som også burde være med i systemet. Dette var en del av den iterative og Agile prosessen som blir beskrevet i Seksjon 9.

I tillegg til elisiteringsmetodene ovenfor så var brukertesting også en viktig del av formingen av kravene rundt brukergrensesnittet. Her ble det gjort endringer ved hjelp av tilbakemeldingene som ble gitt under brukertestingen.

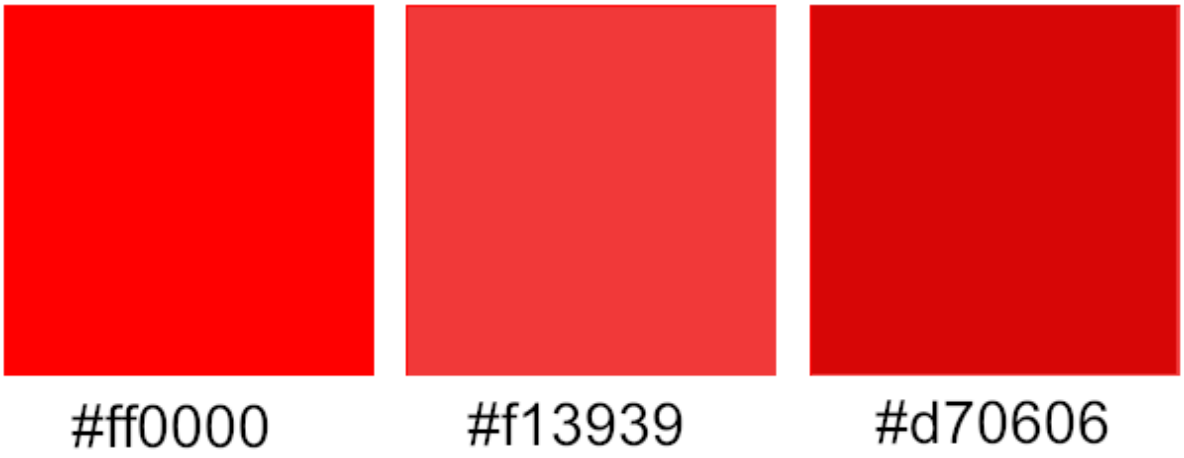
10.2 Før presentasjon av krav: nødvendig endring i mobilapplikasjonen og backend-systemet

Før presentasjonen om funksjonelle krav for webapplikasjonen ser vi på det som nødvendig å først introdusere en viktig endring som måtte implementeres i mobilapplikasjonen og backend-systemet før arbeidet med selve webapplikasjonen kunne starte. Mange aspekter av systemet er påvirket av denne endringen, og vi mener derfor det er nødvendig å presentere den her før vi går inn på krav for selve webapplikasjonen.

Datastrukturen til dataen registrert via mobilapplikasjonen (presentert i Del III), hadde en stor svakhet som krevde endringer før videre arbeid kunne fortsette. Svakheten gjaldt hvordan øremerker er representert som data.

Som diskutert tidligere er det slik at hver gård har en unik farge for sine øremerker, som identifiserer øremerkede dyrs tilhørighet til akkurat den gården. Fargen på øremerkene kan være hva som helst (utenom hvitt og lakserødt), og det viktigste er at fargen er såpass ulik nabogårdene at det er lett å skille et øremerke fra en nabogård sitt. Derfor hadde mobilapplikasjonen en fargevelger der brukeren kunne lagre opp til ti fargeverdier, og benytte disse ved registrering av saueflokker. Hver saueflokk har da en liste med fargeverdier som representerer hvilke gårder som er representert.

Denne funksjonaliteten ble implementert nøyaktig som beskrevet av veileder, men ved påbegynnelsen av hovedprosjektet i denne masteroppgaven viste det seg fort at denne datastrukturen for øremerkefarger var problematisk når man ser på systemet som en helhet.



Figur 17: Ulike eksempler av rødfarger og deres heksadesimale fargeverdier (RGB).

Det hadde vært en mulighet å lage en slags algoritme for å sammenligne heksadesimale fargeverdier og finne ut om de er så nærme hverandre at man kan konkludere at de i bunn og grunn representerer samme farge. Dette løser likevel ikke et fundamentalt problem med hvordan datastrukturen vår fungerer: På webapplikasjonen vår skal brukeren kunne se utførte tilsyn som er relevante for dem. Dette er altså tilsyn hvor brukeren – en sauebonde – sine sauer er representert i minst én saueflokk i tilsynet. Når brukeren benytter webapplikasjonen er det derfor nødvendig å utføre en spørring mot backend-systemet: «ut i fra alle utførte tilsyn, hvilke av dem har data registrert om **mine** dyr?». Hvis vi bruker eksempelet med farge-matching basert på «nærhet» av farger, og lar brukeren av webapplikasjonen registrere at fargen rød er øremerkefargen til brukerens gård, ville denne spørringen da returnert alle tilsyn hvor det har vært minst ett tilfelle av saueflokker hvor sauer har hatt rødt øremerke. Et stort problem med dette er at denne bondens gård sannsynligvis ikke er den eneste i Norge med røde øremerker for dyrene sine, og dermed ville de blitt vist mange tilsyn som er for helt andre gårder i Norge.

På grunn av dette er det en dårlig idé å kun basere seg på farger for å knytte en spesifikk bruker til tilsynene de burde ha tilgang til.

Entiteten «gårder» ble derfor introdusert som en erstatning for farger. Gårder må manuelt registreres inn på webapplikasjonen. En gård har en øremerke-farge (eller tofargede øremerker), men også annen tilhørende data, slik som hvilken bonde som «eier» gården. Dette gjorde det mulig å knytte et tilsyn til én eller flere gårder på en god og oversiktlig måte. I mobilapplikasjonen ble fargevelgeren erstattet av en gård-velger – de spesifikke endringene på applikasjonen presenteres i Seksjon 10.4.

10.3 Kravpresentasjon for webapplikasjonen

I denne underseksjonen vil vi dokumentere krav knyttet til webapplikasjonen. Dette involverer bruksmønster (engelsk: *use cases*), funksjonelle krav, arkitektoniske betydelige krav (engelsk: *architecturally significant requirements*), og ikke-funksjonelle krav (engelsk: *quality attributes*). Disse utgjør konkrete mål for nøyaktig hva som skal utvikles og hva løsningen skal være preget av.

I denne underseksjonen vil begrepet «Systemet» kun referere til webapplikasjonen, og ikke det helhetlige systemet (webapplikasjon og mobilapplikasjon).

10.3.1 Bruksmønster

Bruksmønstrene i Seksjon 10.3.1.1 til Seksjon 10.3.1.12 blir beskrevet gjennom bruksmønstermaler fra boken «Software Requirements 3rd edition» [49] og beskriver funksjonaliteten som kreves av systemet for at det skal være et helverdig fungerende system for interessentene til systemet. Prosessen med å utarbeide kravene er beskrevet i Seksjon 10.1.

10.3.1.1 Innlogging

ID and Name:	Bruksmønster (BM)-1 Innlogging med konto		
Created by:	Andreas & Lars	**Date created:**	03.02.2022
Primary actor:	Sauebonde & Gjeter	**Secondary actors:**	Firebase Authentication og Hosting
Description:	En bruker med en gyldig konto kan logge inn med e-post og passord.		
Trigger	En bruker ønsker å logge inn.		
Preconditions:	PRE-1: Brukeren har en gyldig konto.		
Postconditions:	POST-1: Brukeren er logget inn i webapplikasjonen.		
Normal flow:	**1.0 Logg inn** 1. Naviger til innloggingssiden. 2. Skriv inn e-post og passord. 3 Klikk logg inn.		
Alternative flows:	None		
Exceptions	1.0.E1 Brukeren skriver inn feil passord. 1.0.E2 Brukeren skriver inn feil e-post.		

Tabell 1: Bruksmønster 1 - Innlogging.

10.3.1.2 Registrering

ID and Name:	BM-2 Registrer ny konto		
Created by:	Andreas & Lars	**Date created:**	03.02.2022
Primary actor:	Sauebonde & Gjeter	**Secondary actors:**	Firebase Authentication, Firestore og Hosting
Description:	En sauebonde eller en gjeter skal ha mulighet til å registrere seg med e-post og passord.		
Trigger	Brukeren ønsker å registrere seg.		
Preconditions:	PRE-1: Brukeren er koblet til webapplikasjonen.		
Postconditions:	POST-1: Den nye brukeren er lagret i Firebase Authentication og Firestore.		
Normal flow:	**2.0 Registrer konto** 1. Naviger til registreringssiden. 2. Skriv inn e-post, passord og bekreft passord. 3. Klikk registrer.		
Alternative flows:	None		
Exceptions	2.0.E1 En bruker med den gitte e-posten finnes allerede.		

Tabell 2: Bruksmønster 2 - Registrering.

10.3.1.3 Utlogging

ID and Name:	BM-3 Utlogging av konto		
Created by:	Andreas & Lars	**Date created:**	03.02.2022
Primary actor:	Sauebonde & Gjeter	**Secondary actors:**	Firebase Authentication og Hosting
Description:	En bruker skal kunne logge ut fra kontoen sin.		
Trigger	Brukeren ønsker å logge ut.		
Preconditions:	PRE-1: Brukeren er logget inn.		
Postconditions:	POST-1: Brukeren er logget ut.		
Normal flow:	**3.0 Logg ut** 1. Klikk på profilbilde-ikonet. 2. Klikk på logg ut.		
Alternative flows:	None		
Exceptions	None		

Tabell 3: Bruksmønster 3 - Utlogging.

10.3.1.4 Glemt og tilbakestill passord

ID and Name:	BM-4 Glemt og tilbakestill passord		
Created by:	Andreas & Lars	**Date created:**	03.02.2022
Primary actor:	Sauebonde & Gjeter	**Secondary actors:**	Firebase Authentication og Hosting
Description:	En bruker skal kunne få tilbakestilt passordet sitt hvis de har glemt sitt nåværende passord.		
Trigger	Brukeren har glemt passordet sitt og ønsker å tilbakestille det.		
Preconditions:	PRE-1: Brukeren har en registerert bruker. PRE-2: Brukeren husker e-posten sin.		
Postconditions:	POST-1: Brukerens passord blir endret til det nye passordet		
Normal flow:	**4.0 Glemt passord** 1. Naviger til glemt passord siden. 2. Fyll inn e-posten sin. 3. Klikk tilbakestill passord. 4. Klikker på lenken sendt på e-post til dem. 5. Fyll inn nytt passord og bekreft passord. 6. Klikk tilbakestill passord.		
Alternative flows:	None		
Exceptions	4.0.E1 En bruker med den oppgitte e-postadressen finnes ikke.		

Tabell 4: Bruksmønster 4 - Glemt og tilbakestill passord.

10.3.1.5 Gårdsregistrering

ID and Name:	BM-5 Registrering av gård		
Created by:	Andreas & Lars	**Date created:**	03.02.2022
Primary actor:	Sauebonde & Gjeter	**Secondary actors:**	Firebase Firestore og Hosting
Description:	En bruker skal kunne registrere en ny gård.		
Trigger	En bruker ønsker å registrere en ny gård.		
Preconditions:	PRE-1: En bruker er logget inn.		
Postconditions:	POST-1: En ny gård er lagret i Firebase Firestore.		
Normal flow:	**5.0 Registrer gård** 1. Naviger til gårdregistreringssiden. 2. Velg om du skal ha 1 eller 2 farger. 3. Velg farge 1. 4. Velg farge 2 hvis du skal ha det. 5. Skriv inn navn på gården. 6. Klikk registrer gård.		
Alternative flows:	None		
Exceptions	5.0.E1 En gård med samme navn finnes allerede.		

Tabell 5: Bruksmønster 5 - Gårdsregistrering.

10.3.1.6 Gårdsoversikt

ID and Name:	BM-6 Oversikt over gårder		
Created by:	Andreas & Lars	**Date created:**	03.02.2022
Primary actor:	Sauebonde & Gjeter	**Secondary actors:**	Firebase Firestore og Hosting
Description:	En bruker skal kunne se en liste over sine egne gårder og søke i en liste over alle gårder.		
Trigger	En bruker ønsker å se en liste over sine gårder eller alle gårder som stemmer med søkestrengen.		
Preconditions:	PRE-1: En bruker er logget inn.		
Postconditions:	POST-1: En liste over egne gårder eller en liste over alle gårder som oppfyller søkekriteriet er vist til brukeren.		
Normal flow:	**6.0 Se oversikt over gårder** 1. Naviger til gårdsoversiktsiden. 2a. Se liste over egne gårder. 2b. Søk i søkebaren etter gården(e) du leter etter. 3b. Se liste over gårder som stemmer med søkestrengen.		
Alternative flows:	None		
Exceptions	None		

Tabell 6: Bruksmønster 6 - Gårdsoversikt.

10.3.1.7 Endre/Slette gård

ID and Name:	BM-7 Endring og slettting av gård		
Created by:	Andreas & Lars	**Date created:**	03.02.2022
Primary actor:	Sauebonde	**Secondary actors:**	Firebase Firestore og Hosting
Description:	En bruker skal kunne slette eller endre sin egen gård.		
Trigger	En bruker ønsker å slette eller endre gården(e) sin(e).		
Preconditions:	PRE-1: Brukeren har en gård registrert.		
Postconditions:	POST-1: Gården er slettet fra Firebase Firestore. eller POST-1: Gården har endret farge(r) i Firebase Firestore.		
Normal flow:	**7.0 Slette eller endre gård** 1. Naviger til gårdsoversikt siden. 2a. Klikk slett-ikonet på gården du vil slette. 2b. Klikk endre-ikonet på gården du vil endre. 3a. Klikk bekreft sletting-knappen i modalen. 3b. Endre farge(r). 4b. Klikk bekreft endring-knappen.		
Alternative flows:	None		
Exceptions	None		

Tabell 7: Bruksmønster 7 - Endre/Slette gård.

10.3.1.8 Se tilsyn av saueflokk(er)

ID and Name:	BM-8 Se tilsyn av saueflokk(er)		
Created by:	Andreas & Lars	**Date created:**	03.02.2022
Primary actor:	Sauebonde & Gjeter	**Secondary actors:**	Firebase Firestore og Hosting
Description:	En bruker skal kunne se tilsyn som inneholder deres sauer. Valgt tilsyn skal vise saueflokker, døde sauer, rovdyr, «annet»-observasjoner og gjeterens lokasjoner. I tillegg skal diverse statistikk om tilsynet vises.		
Trigger	En bruker ønsker å se på tilsynene sine.		
Preconditions:	PRE-1: Det finnes tilsyn med sauer som har øremerke tilhørende én av gårdene som er registrert på brukeren.		
Postconditions:	POST-1: En liste over tilsyn er vist til brukeren. POST-2: Et kart med informasjon av valgt tilsyn er vist til brukeren.		
Normal flow:	**8.0 Se tilsyn(er) av saueflokk(er)** 1. Naviger til tilsynssiden. 2. Vent til listen med tilsyn er ferdig med å laste inn. 3. Klikk på et tilsyn. 4. Les diverse statistikk om tilsynet. 5. Se på kartet for å se de ulike observasjonene som har blitt gjort.		
Alternative flows:	None		
Exceptions	8.0.E1 Det finnes ingen tilsyn.		

Tabell 8: Bruksmønster 8 - Se tilsyn av sauflokk(er).

10.3.1.9 Generere rapport

ID and Name:	BM-9 Generering av rapport		
Created by:	Andreas & Lars	**Date created:**	24.02.2022
Primary actor:	Sauebonde	**Secondary actors:**	Firebase Firestore og Hosting
Description:	En bruker skal kunne generere en rapport fra tilsynene.		
Trigger	En bruker ønsker å generere en rapport.		
Preconditions:	PRE-1: Det finnes tilsyn tilknyttet en av gårdene til brukeren.		
Postconditions:	POST-1: En rapport blir generert og vist til brukeren.		
Normal flow:	**9.0 Generering av rapport** 1. Naviger til rapportsiden. 2. Klikk Generer ny rapport. 3. Les rapporten. 4a. Last ned og arkiver rapport. 4b. Arkiver rapport.		
Alternative flows:	None		
Exceptions	9.0.E1 Det finnes ingen tilsyn å generere rapport fra.		

Tabell 9: Bruksmønster 9 - Generere rapport.

10.3.1.10 Liste over tidligere rapporter

ID and Name:	BM-10 Liste over tidligere rapport		
Created by:	Andreas & Lars	**Date created:**	24.02.2022
Primary actor:	Sauebonde	**Secondary actors:**	Firebase Firestore og Hosting
Description:	En bruker skal se en liste over tidligere rapporter som de skal kunne lese, laste ned eller slette.		
Trigger	En bruker ønsker å lese/laste ned/slette en tidligere rapport.		
Preconditions:	PRE-1: Det finnes tidligere genererte rapporter.		
Postconditions:	POST-1: En tidligere rapport blir vist til brukeren.		
Normal flow:	**10.0 Visning av tidligere rapporter** 1. Naviger til rapportsiden. 2. Klikk på en tidligere rapport i listen over tidligere rapporter. 3a. Les rapport. 3b. Last ned og arkiver rapport. 3c. Slett rapport.		
Alternative flows:	None		
Exceptions	10.0.E1 Det finnes ingen tidligere genererte rapporter.		

Tabell 10: Bruksmønster 10 - Liste over tidligere rapport.

10.3.1.11 Endre eller slette profildata

ID and Name:	BM-11 Endre eller slette profildata		
Created by:	Andreas & Lars	**Date created:**	24.02.2022
Primary actor:	Sauebonde og gjeter	**Secondary actors:**	Firebase Authentication, Firestore og Hosting
Description:	En bruker skal kunne endre eller slette sin profildata.		
Trigger	En bruker ønsker å endre eller slette sin profildata.		
Preconditions:	PRE-1: Brukeren er logget inn.		
Postconditions:	POST-1: Profildataen er endret eller slettet i Firebase Authentication og Firestore.		
Normal flow:	**11.0 Endre eller slette profildata** 1. Naviger til profildatasiden. 2a. Endre e-post. 2b. Endre passord. 2c. Slett data. 2d. Slett bruker og data. 3. Lagre endringene.		
Alternative flows:	1. Systemet ber brukeren reautorisere seg gjennom en modal. 2. Skriv inn e-post og passord og klikk reautoriser. 3. Lagre endringene.		
Exceptions	11.0.E1 Profildataen som endres regnes som sensitiv informasjon (som e-post og passord) og brukeren har ikke autorisert eller reautorisert nylig.		

Tabell 11: Bruksmønster 11 - Endre profildata.

10.3.1.12 Se gjeters rute i løpet av tilsynet

ID and Name:	BM-12 Se gjeters rute i løpet av tilsynet		
Created by:	Andreas & Lars	**Date created:**	10.03.2022
Primary actor:	Sauebonde, gjeter og statsforvalteren	**Secondary actors:**	Firebase Firestore og Hosting
Description:	En person ønsker å se en gjeterrute fra et tilsyn, generelt fordi de har lest en rapport.		
Trigger	En person ønsker å se en gjeterrute fra et tilsyn.		
Preconditions:	PRE-1: Tilsynet eksisterer.		
Postconditions:	None		
Normal flow:	**12.0 Se gjete rute av tilsyn** 1. Naviger til view-ruten for tilsynet (for eksempel via lenke i rapport). 2. Se gjeters rute.		
Alternative flows:	None		
Exceptions	None		

Tabell 12: Bruksmønster 12 - Se gjeters rute i løpet av tilsynet.

10.3.2 Funksjonelle krav (FK)

De funksjonelle kravene for systemet er skrevet for å følge en struktur basert på følgende boilerplate (BP):
BP.1: *< systemet > skal < handling > < enhet >.*
BP.2: *< systemet > skal ikke < handling > < enhet > .*
BP.3: *< systemet > skal ha < enhet > .*
BP.4: *< systemet > skal < handling > når < situasjon > .*
BP.5: *< systemet > skal tillate < handling > < enhet > .*
BP.6: *< systemet > skal tillate brukeren < handling > < enhet > .*
BP.7: *< systemet > skal ikke tillate < handling > < enhet > .*
BP.8: *< systemet > skal ikke tillate brukeren < handling > < enhet > .*
(hver BP kan også ha en forklarende bi-setning)

og kan bli funnet i Tabell 13 til Tabell 24. De beskriver den funksjonaliteten som webapplikasjonen må tilby brukeren.

ID	Beskrivelse	Prioritet (L/M/H)
FK1	Systemet skal tillate brukeren å logge inn med e-post og passord.	Høy

Tabell 13: Funksjonelle krav for bruksmønster i Seksjon 10.3.1.1.

ID	Beskrivelse	Prioritet (L/M/H)
FK2	Systemet skal tillate brukeren å registrere en ny konto med e-post og passord.	Høy

Tabell 14: Funksjonelle krav for bruksmønster i Seksjon 10.3.1.2.

ID	Beskrivelse	Prioritet (L/M/H)
FK3	Systemet skal tillate brukeren å logge ut.	Lav

Tabell 15: Funksjonelle krav for bruksmønster i Seksjon 10.3.1.3.

ID	Beskrivelse	Prioritet (L/M/H)
FK4	Systemet skal tillate brukeren å be om å få tilbakestille passordet sitt.	Lav
FK5	Systemet skal tillate brukeren å tilbakestille passordet sitt når de har glemt sitt nåværende passord.	Lav

Tabell 16: Funksjonelle krav for bruksmønster i Seksjon 10.3.1.4.

ID	Beskrivelse	Prioritet (L/M/H)
FK6	Systemet skal tillate registrering av gårder. Hver gård har én eller to farger, og et navn på gården.	Høy
FK7	Systemet skal ikke tillate registrering av gård med samme navn som en eksisterende gård.	Medium

Tabell 17: Funksjonelle krav for bruksmønster i Seksjon 10.3.1.5.

ID	Beskrivelse	Prioritet (L/M/H)
FK8	Systemet skal tillate brukeren å se en liste over hvilke gårder de selv har registrert.	Høy
FK9	Systemet skal tillate brukeren å søke i en liste over alle gårdene registrert i systemet.	Medium

Tabell 18: Funksjonelle krav for bruksmønster i Seksjon 10.3.1.6.

ID	Beskrivelse	Prioritet (L/M/H)
FK10	Systemet skal tillate brukeren å slette en gård de selv har registrert.	Medium
FK11	Systemet skal tillate brukeren å endre fargen(e) og navnet på en gård de selv har registrert.	Medium

Tabell 19: Funksjonelle krav for bruksmønster i Seksjon 10.3.1.7.

ID	Beskrivelse	Prioritet (L/M/H)
FK12	Systemet skal tillate brukeren å se en liste over utførte tilsynsrunder som er relevante for sin gård.	Høy
FK13	Systemet skal tillate brukeren å se en visuell representasjon av en utført tilsynsrunde. Data fra tilsynsrunden skal overlegges på et kart.	Høy
FK14	Systemet skal tillate brukeren, ved kartvisningen beskrevet i FK13, å kunne klikke på enkeltmarkører som representerer observasjoner gjort under tilsynsrunden. Man skal da kunne få utvidet informasjon om observasjonen.	Høy
FK15	Systemet skal tillate brukeren å se bilder fra en registrert observasjon hvis denne inneholder bilder.	Høy
FK16	Systemet skal tillate brukeren å laste ned bilder enkeltvis eller alle på en gang fra observasjoner som inneholder bilder.	Høy

Tabell 20: Funksjonelle krav for bruksmønster i Seksjon 10.3.1.8.

ID	Beskrivelse	Prioritet (L/M/H)
FK17	Systemet skal tillate brukeren å generere en rapport for alle tilsynene utført i løpet av sesongen.	Høy
FK18	Systemet skal tillate brukeren å lagre genererte rapporter som en .pdf-fil.	Høy
FK19	Systemet skal tillate brukeren å arkivere genererte rapporter.	Høy

Tabell 21: Funksjonelle krav for bruksmønster i Seksjon 10.3.1.9.

ID	Beskrivelse	Prioritet (L/M/H)
FK20	Systemet skal lagre en liste over tidligere genererte og arkiverte rapporter.	Høy
FK21	Systemet skal tillate brukeren å lese arkiverte rapporter.	Høy
FK22	Systemet skal tillate brukeren å slette arkiverte rapporter.	Høy

Tabell 22: Funksjonelle krav for bruksmønster i Seksjon 10.3.1.10.

ID	Beskrivelse	Prioritet (L/M/H)
FK23	Systemet skal tillate brukeren å endre profildata som e-post og passord.	Lav
FK24	Systemet skal tillate brukeren å slette dataen sin.	Lav
FK25	Systemet skal tillate brukeren å slette brukerkontoen og dataen sin.	Lav

Tabell 23: Funksjonelle krav for bruksmønster i Seksjon 10.3.1.11.

ID	Beskrivelse	Prioritet (L/M/H)
FK26	Systemet skal tillate en person å se gjeterruten til et tilsyn (uavhengig av om de er logget inn).	Medium

Tabell 24: Funksjonelle krav for bruksmønster i Seksjon 10.3.1.12.

10.3.3 Arkitektoniske Betydelige Krav (ABK)

Arkitektoniske Betydelige Krav er, som navnet tilsier, krav som betydelig påvirker arkitekturen og som dermed enten er kritiske, er av høy risiko, er volatile, involverer kostbar refaktorering eller har lovmessig innvirkning [50].

Ettersom applikasjonen som lages i dette masterprosjektet er relativ simpelt i forhold til hva som ble lagd i forarbeidet er det et lavere antall ABK, men det gjør det enda viktigere at disse blir tatt hensyn til.

10.3.3.1 Skal være enkel å bruke på datamaskiner

Ettersom webapplikasjonen hovedsakelig skal brukes av sauebønder når de sitter inne på kontoret sitt så er det essensielt at den fungerer på en datamaskin, og da hovedsakelig Windows 8.1 (eller nyere) eller MacOS, men det burde også være støtte for Linux. I tillegg er det viktig at webapplikasjonen er enkel å bruke på datamaskinen, dermed tilsier det at det burde være en webapplikasjon som kan brukes i de mest kjente nettleserne som Chrome, Safari, Edge, og Firefox, og ikke en systemapplikasjon.

10.3.3.2 Interoperabilitet med eksisterende applikasjoner

Webapplikasjonen som blir utviklet i masterprosjektet skal i all hovedsak vise data som er samlet inn og lagret i systemer som ble laget under forarbeidet, i tillegg til noe ny funksjonalitet for å kunne legge til gårder i backend-systemet og generere rapporter. Dermed er det helt essensielt at webapplikasjonen har god interoperabilitet med de eksisterende systemene og dataen som er lagret i dem.

10.3.3.3 Data skal ha god integritet, konfidensialitet og tilgjengelighetsbeskyttelse

Dataen som samles inn i det samlede systemet kan inneholde personidentifiserende informasjon. Dermed er det viktig at disse blir tatt vare på ordentlig og har god integritet, konfidensialitet og tilgjengelighets-beskyttelse sånn at de ikke havner i feil hender, noe som kan ha lovmessige betydninger for personen(e) eller selskapet som eier systemet.

10.3.4 Ikke-funksjonelle krav (Quality Attributes)

I Seksjon 10.3.2 ble rent funksjonelle krav som må oppfylles beskrevet, nå skal vi derimot beskrive ikke-funksjonelle krav også kjent som «Quality attributes». Dette er mål eller egenskaper som systemet så langt som mulig skal oppfylle eller ha. I dette systemet så er det «Functional Suitability», «Interoperability», «Usability», «Security» og «Modifiability» fra ISO/IEC 25010 [51] som er de ikke-funksjonelle kravene.

Grunnen til at noen av de mest brukte ikke-funksjonelle kravene – «Reliability» og «Performance» – ikke blir brukt i dette systemet kommer av at det er et system som hovedsakelig skal brukes sjeldent på datamaskinen til bonden. Dermed er det ikke så viktig om systemet har noe mer nedetid enn andre systemer, i tillegg til at det ikke finnes spesielt krevende funksjonalitet i systemet som en datamaskin kan slite med. Dette fører til at «Reliability» og «Performance» ikke er like viktig i dette systemet som i andre.

10.3.4.1 Functional Suitability

Systemet har i seg selv få essensielle funksjoner når man ser bort ifra autorisering og profilhåndtering, men disse funksjonene er essensielle for at systemet skal ha noe nytte. Dermed er det veldig viktig at systemet dekker alle de funksjonelle kravene (functional completeness), at systemimplementasjonen som løser de funksjonelle kravene har den nødvendige presisjonen (functional correctness), og at systemimplementasjonen faktisk fasiliterer til å løse oppgavene/objektene som brukeren vil løse ved hjelp av systemet (functional appropriateness).

Functional Suitability scenario er beskrevet i Tabell 25.

FS1	
Source:	Sluttbruker
Stimulus:	Leser av posisjonen til en flokk i et tilsyn
Artifact:	Applikasjon
Environment:	Normal drift
Response:	Systemet viser posisjonen til alle flokkene i tilsynet
Response Measure:	Applikasjonen viser posisjonen til flokken som brukeren er interessert i med en presisjon på minst 30m nøyaktighet

Tabell 25: Ikke-funksjonelt krav scenario 1 for Functional Suitability.

10.3.4.2 Interoperability

Kompatibilitet mellom systemene er veldig viktig ettersom de mest essensielle funksjonene i webapplikasjonen baserer seg på data fra mobilapplikasjonen. Dermed er det viktig at vi har «Interoperability» mellom mobilapplikasjonen, backend-systemet og webappliakasjonen, sånn at de kan utveksle tilsynsdata.

Interoperability scenario er beskrevet i Tabell 26.

C1	
Source:	Webapplikasjonen
Stimulus	Vil lese all data om tilsyn
Artifact:	Mobilapplikasjon, backend-system
Environment:	Kjent ved kjøretid
Response:	Forespørselen om dataoverføring blir godkjent og dataen blir overført
Response Measure:	100 % av dataen blir overført riktig

Tabell 26: Ikke-funksjonelt krav scenario 1 for Interoperability.

10.3.4.3 Usability

For at et system skal være brukbart er det viktig at applikasjonen er enkel og intuitiv å bruke. Det er spesielt viktig at erstatningssystemer er like enkle eller enklere og mer intuitive å bruke, og ikke minst gir en bedre brukeropplevelse enn det gamle systemet det skal erstatte. Hvis det ikke er tilfellet, vil brukerne sjeldent bytte over med mindre de blir tvunget.

Hvordan man utformer systemet er også avhengig av hvilken målgruppe man retter seg mot. For eksempel vil en applikasjon rettet mot utviklere kunne være «mindre brukervennlig» i tradisjonell sammenheng enn det en applikasjon rettet mot sauebønder trenger å være. Ettersom målgruppen vår kan antas til å ha helt gjennomsnittlige datakunnskaper er det viktig at brukervennligheten er høy og rettet inn mot brukere som ikke er godt kjent med teknologi.

Usability scenarioene er beskrevet i Tabell 27 og Tabell 28.

U1	
Source:	Sluttbruker
Stimulus	Prøver å lese av et tilsyn
Artifact:	Tilsynsdelen av webapplikasjonen
Environment:	Normal drift - kjøretid
Response:	Webapplikasjonen forutser brukerens behov og hjelper dem ved å bruke «Task model»-en [52] til applikasjonen
Response Measure:	Det burde ta brukeren mindre enn 30 sekunder å utføre en oppgave relatert til å inspisere tilsynet, og ved bruk de burde få mer kunnskap av hvordan tilsynsdelen av applikasjonen fungerer

Tabell 27: Ikke-funksjonelt krav scenario 1 for Usability.

U2	
Source:	Testbruker
Stimulus	Utfører brukertest på systemet
Artifact:	Webapplikasjonen og SUS-skjema
Environment:	Brukertest
Response:	Webapplikasjonen forutser testbrukerens behov og hjelper dem ved å bruke «Task model»-en [52] til applikasjonen
Response Measure:	Testbrukerne vurderer brukskvaliteten til systemet, via et SUS (System Usability Scale)-skjema, slik at den gjennomsnittlige SUS-poengsummen er på minst 80 poeng (av 100).

Tabell 28: Ikke-funksjonelt krav scenario 2 for Usability.

10.3.4.4 Security

Sikkerhet er viktig i alle bedrifter i de fleste land i verden, men i EU og Norge er det spesielt viktig at man tar spesielt ansvar rundt håndteringen av personopplysninger. Dette er fordi man er pålagt å følge personopplysningsloven og dermed personvernprinsippene [53]. Derfor er det veldig viktig at sikkerhet er en av de ikke-funksjonelle kravene våre hvis applikasjonen skulle blitt brukt i et virkelig miljø.

Sikkerhetsscenarioene er beskrevet i Tabell 29 og Tabell 30. Disse er i stor grad en konkret implementasjon av det arkitektoniske betydelige kravet om integritet, konfidensialitet og tilgjengelighetsbeskyttelse beskrevet i Seksjon 10.3.3.3.

S1	
Source:	Hacker
Stimulus	Prøver å lese data de ikke har tilgang til
Artifact:	Backend-system
Environment:	Normal drift
Response:	Krev autorisering og autentisering og ha alarmer som tilkaller administratorer
Response Measure:	Det tar hackeren over én time å bryte gjennom sikkerheten og det krever flere personer for å få det til

Tabell 29: Ikke-funksjonelt krav scenario 1 for Security.

S2	
Source:	Hacker
Stimulus	Prøver å modifisere eller slette data de ikke har eierskap over
Artifact:	Backend-system
Environment:	Normal drift
Response:	Krev autorisering og autentisering for skrive-handlinger (writes og deletes)
Response Measure:	Data kan ikke oppdateres eller slettes med mindre handlingen utføres av brukeren som eier dataen, eller en systemadministrator

Tabell 30: Ikke-funksjonelt krav scenario 2 for Security.

10.3.4.5 Modifiability

Ettersom masteroppgaven går ut på rask prototyping (Seksjon 12.3) og Agile utvikling er det viktig at systemet har høy modifiability, ettersom dette tillater oss å enkelt kunne legge til ny funksjonalitet eller modifisere eksisterende funksjonalitet. I tillegg så vil det gjøre det enklere å eventuelt kunne gjøre justeringer hvis det skulle blitt brukt i et virkelig miljø.

Modifiability scenario er beskrevet i Tabell 31.

M1	
Source:	Utvikler
Stimulus	Legger til ny funksjonalitet
Artifact:	Kildekode
Environment:	Utviklingstid
Response:	Koding, testing, og utrulling
Response Measure:	Tar mindre enn 1-2 uker å fullføre den nye funksjonaliteten

Tabell 31: Ikke-funksjonelt krav scenario 1 for Modifiability.

10.4 Kravpresentasjon for mobilapplikasjonen

Følgende underseksjon omhandler kravene for funksjonalitet som må legges til eller endres på den eksisterende mobilapplikasjonen. Ettersom applikasjonen allerede eksisterer vil vi ikke gå gjennom kravene fra når mobilapplikasjonen først ble utviklet ettersom dette ble gått gjennom i fordypningsprosjektet [1] og er utenfor omfanget til dette masterprosjektet. En god forståelse av mobilapplikasjonens virkemåte kan etableres ved å lese Del III.

Nye bruksmønstre og funksjonelle krav vil presenteres her. Ettersom applikasjonen i stor grad allerede er utviklet vil ikke ikke-funksjonelle krav eller arkitektoniske betydelige krav (ABK) introduseres her ettersom disse har å gjøre med fundamentale aspekter av mobilapplikasjonen og dermed allerede er etablert.

I denne underseksjonen vil begrepet «Systemet» kun referere til mobilapplikasjonen, og ikke det helhetlige systemet (webapplikasjon og mobilapplikasjon).

10.4.1 Bruksmønster

De nye bruksmønstrene for mobilapplikasjonen blir beskrevet i underseksjonene 10.4.1.1 - 10.4.1.5 og bruker samme mal som i Seksjon 10.3.1.

10.4.1.1 Velge lokalt lagrede gårder

ID and Name:	BM-13 - Legge til/slette gårder lokalt		
Created by:	Andreas & Lars	**Date created:**	03.02.2022
Primary actor:	Gjeter	**Secondary actors:**	Firebase Firestore
Description:	En bruker kan søke etter gårder i systemet og lagre dem for bruk under tilsyn. Lagrede gårder kan slettes lokalt.		
Triggers	En bruker benytter applikasjonen for første gang. En bruker ønsker å endre hvilke gårder de har lagret lokalt.		
Preconditions:	PRE-1: Brukeren er logget inn på mobilapplikasjonen. PRE-2: Minst én gård har blitt registrert inn i systemet fra webapplikasjonen.		
Postconditions:	POST-1: Brukeren har satt hvilke gårder som skal være tilgjengelig å markere saueflokker med under tilsyn.		
Normal flow:	**13.0 Legge til gårder lokalt** 1. Naviger til gård-siden. 2. Trykk på søkeknappen for å åpne søkemenyen. 2a. Skriv et gårdsnavn i søkefeltet for å søke etter tilgjengelige gårder. 2b. Trykk på et søketreff i listen for å markere det. 2b-a. Trykk på søketreffet igjen for å fjerne markeringen. 2b-b. Trykk på «Lagre»-knappen for å lagre den markerte gården lokalt. 3. Trykk på «Ferdig»-knappen for å gå tilbake til å se lokalt lagrede gårder.		
Alternative flows:	**13.1 Slette gårder lokalt** 1. Naviger til gård-siden. 2. Trykk på en lagret gård i listen for å markere den. 2a. Trykk på «slett»-knappen. 2b. Trykk på gården i listen igjen for å fjerne markeringen.		
Exceptions	13.0.E1 Ingen gårder finnes i systemet.		

Tabell 32: Bruksmønster 13 - Velge lokalt lagrede gårder.

10.4.1.2 Registrere død sau

ID and Name:	BM-14 - Registrere død sau		
Created by:	Andreas & Lars	**Date created:**	03.02.2022
Primary actor:	Gjeter	**Secondary actors:**	Firebase Firestore
Description:	En bruker kan registrere døde sauer som del av et tilsyn. En slik registrering kan inneholde bilder og en tekstbeskrivelse.		
Triggers	En bruker ønsker å registrere et funn av død sau.		
Preconditions:	PRE-1: Brukeren har startet et tilsyn.		
Postconditions:	POST-1: Brukeren har lagt til en observasjon av død sau i tilsynet.		
Normal flow:	**14.0 Registrere død sau** 1. Trykk på knappen for død sau. 2. Plasser trådkorset så det er plassert over ønskelig lokasjon på kartet. 3. Trykk «plasser død sau» for å bekrefte plasseringen. 4a. Trykk på «Last opp bilde» for å legge til et bilde lagret på mobilen. 4b. Trykk «Ta bilde» for å ta et nytt bilde med mobilens kamera. 4c. Trykk på tekstboksen merket med «Beskrivelse» for å skrive en beskrivende tekst om observasjonen. 5. Trykk på «Ferdig»-knappen for å ferdiggjøre registreringen av død sau.		
Alternative flows:	**14.1 Registrere død sau uten bilder** 1. Trykk på knappen for død sau. 2. Plasser trådkorset så det er plassert over ønskelig lokasjon på kartet. 3. Trykk «plasser død sau» for å bekrefte plasseringen. 5. Trykk på «Ferdig»-knappen. 6. Brukeren blir møtt av en anbefaling om å ha med bilder når død sau registreres. 6a. Trykk «Lagre uten bilder» for å ferdiggjøre registreringen. 6b. Trykk «OK» for å bli tatt tilbake til registreringen av død sau og legge til bilder.		
Exceptions	None		

Tabell 33: Bruksmønster 14 - Registrere død sau.

10.4.1.3 Registrere skadet sau

ID and Name:	BM-15 - Registrere skadet sau		
Created by:	Andreas & Lars	**Date created:**	03.02.2022
Primary actor:	Gjeter	**Secondary actors:**	Firebase Firestore
Description:	En bruker kan registrere antall skadet sau i en saueflokk som en del av bruksmønsteret for å registrere en saueflokk.		
Triggers	En bruker ønsker å registrere et funn av skadet sau i en saueflokk.		
Preconditions:	PRE-1: Brukeren har startet registrering av en saueflokk.		
Postconditions:	POST-1: Brukeren har registrert et gitt antall skadde sau i saueflokken.		
Normal flow:	**15.0 Registrere skadet sau** 1. Registrer andre eventuelle detaljer om saueflokken (sau og deres slips, antall lam, bilder). 2. Bruk pluss- og minusknappene for å inkrementere/dekrementere telleren for antall skadet sau til ønsket antall. 3. Trykk på «Ferdig»-knappen når alle registreringer for saueflokken er som ønsket.		
Exceptions	None		

Tabell 34: Bruksmønster 15 - Registrere skadet sau.

10.4.1.4 Registrere bilder av saueflokk

ID and Name:	BM-16 - Registrere bilder av saueflokk		
Created by:	Andreas & Lars	**Date created:**	03.02.2022
Primary actor:	Gjeter	**Secondary actors:**	Firebase Firestore
Description:	En bruker kan legge til bilder som en del av bruksmønsteret for å registrere en saueflokk.		
Triggers	En bruker ønsker å registrere bilder av en saueflokk.		
Preconditions:	PRE-1: Brukeren har startet registrering av en saueflokk.		
Postconditions:	POST-1: Brukeren har lagt til bilder i registreringen av saueflokken.		
Normal flow:	**16.0 Registrere bilder av saueflokk** 1. Registrer andre eventuelle detaljer om saueflokken (sau og deres slips, antall lam, antall skadde dyr). 2. Trykk på «Legg til bilder»-knappen. 3a. Trykk på «Last opp bilde» for å legge til et bilde lagret på mobilen. 3b. Trykk «Ta bilde» for å ta et nytt bilde med mobilens kamera. 4. Trykk på «Tilbake»-knappen når alle de ønskede bildene er lagt til for å navigere tilbake til den generelle skjermen for registrering av saueflokk.		
Exceptions	None		

Tabell 35: Bruksmønster 16 - Registrere bilder av saueflokk.

10.4.1.5 Registrere beskrivelse av tilsyn

ID and Name:	BM-17 - Registrere beskrivelse av tilsyn		
Created by:	Andreas & Lars	**Date created:**	03.02.2022
Primary actor:	Gjeter	**Secondary actors:**	Firebase Firestore
Description:	En bruker må legge til en tekstbeskrivelse av tilsynet de nettop har utført. Intensjonen er å beskrive ruten de har gått tekstlig, da dette skal vises i rapporten generert i bruksmønster 9.		
Triggers	En bruker har trykket på stopp-knappen for å avslutte tilsynet.		
Preconditions:	PRE-1: Brukeren har startet utføring av tilsyn i mobilapplikasjonen.		
Postconditions:	POST-1: Brukeren har lagt til en tekstbeskrivelse av tilsynet.		
Normal flow:	**17.0 Registrere beskrivelse av tilsyn** 1. Brukeren avslutter det aktive tilsynet ved å trykke på stoppknappen. 2. En dialogboks åpner seg hvor brukeren skal skrive inn tekstbeskrivelse av ruten de har gått. 3. Brukeren skriver inn en tekstbeskrivelse. 4. Brukeren trykker «Avslutt og lagre tilsyn» for å bekrefte avslutningen av tilsynet.		
Exceptions	None		

Tabell 36: Bruksmønster 17 - Registrere beskrivelse av tilsyn.

10.4.2 Funksjonelle krav

De funksjonelle kravene for den nye funksjonaliteten til mobilapplikasjonen følger samme struktur som i Seksjon 10.3.2 og kan bli funnet i Tabell 37 til Tabell 41.

ID	Beskrivelse	Prioritet (L/M/H)
FK27	Systemet skal tillate brukeren å søke etter registrerte gårder.	Høy
FK28	Systemet skal tillate brukeren å lagre en gård lokalt.	Høy
FK29	Systemet skal tillate brukeren å slette en lokalt lagret gård.	Medium

Tabell 37: Funksjonelle krav for bruksmønster i Tabell 32.

ID	Beskrivelse	Prioritet (L/M/H)
FK30	Systemet skal tillate brukeren å registrere døde sauer.	Høy
FK30.1	Systemet skal tillate brukeren å bruke en bildevelger-komponent til å legge til bilder i registreringen av død sau.	Høy
FK30.2	Systemet skal tillate brukeren å oppgi en tekstbeskrivelse for registreringen av død sau.	Høy
FK31	Systemet skal tillate brukeren å slette en observasjon av død sau.	Høy

Tabell 38: Funksjonelle krav for bruksmønster i Tabell 33.

ID	Beskrivelse	Prioritet (L/M/H)
FK32	Systemet skal tillate brukeren å oppgi et antall skadde sau i en saueflokk.	Høy
FK33	Systemet skal tillate brukeren å bruke en bildevelger-komponent til å legge til bilder av en saueflokk.	Høy

Tabell 39: Funksjonelle krav for bruksmønster i Tabell 34.

ID	Beskrivelse	Prioritet (L/M/H)
FK34	Systemet skal tillate brukeren å laste opp bilder lagret lokalt på mobilen.	Høy
FK35	Systemet skal tillate brukeren å ta bilder med mobilens kamera.	Medium
FK36	Systemet skal vise en liste over alle bildene som har blitt lagt til så langt.	Høy
FK37	Systemet skal tillate brukeren å fjerne et bilde fra listen av bilder.	Høy

Tabell 40: Funksjonelle krav for bildevelgeren som er en del av bruksmønster 33 og bruksmønster 35.

ID	Beskrivelse	Prioritet (L/M/H)
FK38	Systemet skal tillate brukeren å oppgi en beskrivelse av ruta for et utført tilsyn.	Høy

Tabell 41: Funksjonelle krav for bruksmønster i Tabell 36.

11 Programvarearkitektur

Programvarearkitekturen til et system bestemmes ut ifra de kravene som ble satt i starten av planleggingsfasen – som for dette prosjektet ble presentert i Seksjon 10. For at programvarearkitekturen skal utfylle disse kravene på best mulig måte så er det tre kategorier av valg som må tas, nemlig teknologi, arkitektoniske taktikker og arkitektoniske- og designmønster.

Valg av teknologi presenteres i Seksjon 11.1, arkitektoniske taktikker presenteres i Seksjon 11.2 og arkitektoniske- og designmønster presenteres i Seksjon 11.3. Disse valgene vil igjen føre til komponenter og tekniske begrensninger som må tas hensyn til under utviklingen, og som presenteres i Seksjon 11.5. For å enklere få en oversikt over den totale arkitekturen så er diagrammer av views-ene en viktig del av dokumentasjonen, og disse presenteres i Seksjon 11.4.

Applikasjonen vi lager i dette prosjektet er en webapplikasjon, i tillegg til en videreutvikling (eller videre bruk) av backend-systemet vårt i Firebase fra forarbeidet. Dermed er vi ganske begrenset når det kommer til valg av taktikker i Seksjon 11.2 og mønster i Seksjon 11.3 som vi kan bruke for å påvirke og oppfylle de ikke-funksjonelle kravene i Seksjon 10.3.4, ettersom Firebase allerede har valgt og implementert flere av disse taktikkene og designmønstrene selv uten at utvikleren trenger å tenke på, innføre eller i det hele tatt påvirke dem selv.

11.1 Valg av teknologi

I denne delseksjonen vil vi ta for oss den teknologien som systemet er bygd opp av, i tillegg til verktøy og annen teknologi som ble brukt til utvikling og planlegging av systemet.

Valget av teknologi var som sagt til dels allerede fastsatt før prosjektet startet ettersom backend-systemet Firebase ble valgt allerede tilbake i fordypningsprosjektet. I tillegg ble teknologien for mobilapplikasjonen som nevnt i Del III valgt på bakgrunn av at det ville vært fordelaktig å bruke en teknologi som var såpass kryssplatformkompatibel at vi ikke behøvde å lære to forskjellige teknologier for å lage mobilapplikasjonen og webapplikasjonen. Dermed var det veldig naturlig at React Native ble valgt for fordypningsprosjektet og at ReactJS skulle bli brukt for masteroppgaven, ettersom vi begge hadde erfaring med React.

11.1.1 Backend-system - Firebase

Firebase [54] er en såkalt Backend-as-a-Service platform [55] med i alt over elleve mikrotjenester, i tillegg til flere utvidelser som støtte for Algolia [56] og Stripe [57]. Enkelt oppsummert så er Firebase

> A toolset to "build, improve, and grow your app", and the tools it gives you cover a large portion of the services that developers would normally have to build themselves, but don't really want to build, because they'd rather be focusing on the app experience itself. This includes things like analytics, authentication, databases, configuration, file storage, push messaging, and the list goes on. The services are hosted in the cloud, and scale with little to no effort on the part of the developer. [58]

Som sagt tidligere ble Firebase valgt allerede tilbake i forarbeidet på grunn av hvordan det med ferdiglagde løsninger gjorde utviklingen kjappere og enklere. Dette gjorde at vi heller kunne fokusere på utviklingen av det brukeren ser, som er selve applikasjonen. Selv om Firebase tilbyr en rekke forskjellige tjenester er det i hovedsak bare to som ble brukt under forarbeidet: Firebase Authentication og Firebase Firestore. I tillegg til dem planlegger vi å bruke to til i dette prosjektet: Firebase Storage og Firebase Hosting.

11.1.1.1 Authentication

Firebase Authentication er tjenesten som sørger for autentisering og autorisering over alle tjenestene som brukerne kan interagere med. Gjennom konsollen på nettsiden til Firebase kan man sette opp en rekke «providers» som lar brukeren logge inn på flere forskjellige måter, som for eksempel gjennom Google og Facebook, eller gjennom mer tradisjonelle metoder som e-post eller telefon. For fordypningsprosjektet og masterprosjektet vil vi holde oss til e-post, ettersom det ikke er noe behov for andre innloggingsmetoder for målgruppen til systemet.

Måten Firebase Authentication kobler seg til de andre tjenestene er ved at man gjennom «Firebase Security Rules» kan skrive tillatelsesregler som bestemmer hvem som kan lese/skrive/slette/endre data, samt hva slags data som kan skrives hvor. Dette kan både være data i en database som Firebase Firestore eller data i form av filer i Firebase Storage. Et eksempel på reglene for Firebase Firestore kan sees i Listing 1, og der kan man se hvordan «auth» kan bli brukt gjennom «request.auth»-objektet som inneholder uid-en (den unike identifikatoren) til brukeren og token-objektet som inneholder informasjon som «email», «email_verified», «phone_number», «name», «sub» og eventuelt annen informasjon som har blitt lagret i det gjennom «provider»-en eller gjennom Firebase Admin SDK. Et annet eksempel på hvordan Firebase Authentication blir brukt i reglene for Firebase Storage kan sees i Listing 2. Her er det et egendefinert felt i token-objektet som blir brukt for å sjekke hvilke roller brukeren har.

11.1.1.2 Firestore

Firebase Firestore er tjenesten for en NoSQL dokumentbasert database. Dette vil si at all data er lagret i dokumenter og kolleksjoner, og at strukturen generelt ligner på et JSON-dokument [59].

En av grunnene til at man generelt velger en NoSQL-database over en SQL-database er at man har data som ikke har sterke relasjonelle bånd. NoSQL-databaser har også mange fordeler over SQL-databaser, som også kan være noen av grunnene til at man går for en NoSQL-database. I motsetning til tradisjonelle SQL-databaser er NoSQL-databaser ofte bygget for å enkelt kunne skalere i skytjenester. I tillegg tillater NoSQL-databaser store menger data å bli lagret på fleksible måter som gjør Agile utvikling og hurtig prototyping enklere ved at det er enkelt å endre datamodellen. SQL har derimot noen viktige funksjoner som støtte for store og komplekse transaksjoner, ACID-samsvar, og støtte for komplekse spørringer [60].

Ettersom kravene for denne applikasjonen ikke krevde store og komplekse transaksjoner eller spørringer, og dataen ikke hadde sterke behov for såkalte mange-til-mange relasjoner eller lignende datastrukturering som relasjonelle databaser gjerne er gode på, var Firebase Firestore et godt alternativ for valget av database med sin dokument/kolleksjon-baserte organisering. Sammen med alle de andre tjenestene til Firebase så gjør det Firebase Firestore til et godt valg for dette prosjektet.

Noe som skiller Firebase Firestore fra andre databasesystem er at ved første øyekast ser ut som om at brukeren har direkte tilgang til å lese/skrive/endre/slette data fra databasen, i motsetning til tradisjonelle systemer hvor brukeren må gå gjennom et API. Dette er fordi istedenfor et API hvor man validerer og transformerer dataen, har Firebase Firestore et system med sikkerhetsregler som forklart i Seksjon 11.1.1.1, hvor man setter opp regler for hvem som kan skrive/lese/slette/endre data i Firestore gjennom Firebase Security som sett i eksempelet i Listing 1. Dette gjør det som sagt mulig å sette restriksjoner på hvem som har tilgang hvor, men det gjør det også mulig å validere data og sørge for at all data følger datastrukturen som ble definert i reglene.

Ved hjelp av objektet «**request.resource.data**» (ny data som skal lagres) og objektet «**resource.data**» (eksisterende data i databasen) så kan man definere valideringsregler for struktur og innhold av ny og eksisterende data som må følges før den nye dataen skrives til databasen. Dermed kan man sjekke typen til felt i objekter, sjekke at objekter bare inneholder en forhåndsdefinert liste av felt eller at den må inneholde feltene i en forhåndsdefinert liste, sjekke lengden til en array, sjekke at en streng følger et regex-utrykk, og så videre. En full liste over hva man kan styre ved hjelp av reglene kan bli funnet her.

Listing 1: Firebase Security Rules for Firebase Firestore eksempel [61]

```
service cloud.firestore {
  match /databases/{database}/documents {
    // For attribute-based access control, check for an admin claim
    allow write: if request.auth.token.admin == true;
    allow read: true;

    // Alterntatively, for role-based access, assign specific roles to users
    match /some_collection/{document} {
     allow read: if request.auth.token.reader == "true";
     allow write: if request.auth.token.writer == "true";
   }
  }
}
```

11.1.1.3 Storage

Firebase Storage er tjenesten for å lagre brukergenerert innhold som bilder, videoer, PDF-filer, og så videre, og ligner til dels på S3-tjenesten til AWS [62]. Forskjellen er at Firebase Storage i realiteten er en wrapper rundt Google Cloud Storage som er Google Cloud sin S3-konkurrent.

Fordelen Firebase Storage gir over S3 og Google Cloud Storage er at wrapperen gir støtte for sikkerhetsregler på samme måte som for Firebase Firestore i Seksjon 11.1.1.2, men med noen mindre forskjeller. Disse forskjellene gir for eksempel muligheten for å bruke metadataen til filen i sikkerhetsreglene, å kun tillate et sett med forhåndsbestemte filtyper, eller å kun tillate filer under eller over en viss størrelse. En full liste over hva man kan styre ved hjelp av reglene kan bli funnet her.

Listing 2: Firebase Security Rules for Firebase Storage eksempel [61]

```
service firebase.storage {
  // Allow reads if the group ID in your token matches the file metadata's `owner`
      property
  // Allow writes if the group ID is in the user's custom token
  match /files/{groupId}/{fileName} {
    allow read: if resource.metadata.owner == request.auth.token.groupId;
    allow write: if request.auth.token.groupId == groupId;
  }
```

```
8 }
```

11.1.1.4 Hosting

Firebase Hosting er tjenesten for lagring av statiske filer. Det er i hovedsak tjenesten for hosting av nettsider. Dette er hvor den stabile versjonen av webapplikasjonen vil være tilgjengelig for brukere til et hvert øyeblikk. Det er mulig å koble til sitt eget domene som for eksempel `www.sauesporing.no`, men Firebase Hosting tilbyr også to gratisdomener som brukerne kan benytte. Disse er `www.<prosjektnavn>.web.app` og `www.<prosjektnavn>.firebaseapp.com`, hvor **<prosjektnavn>** er Firebase-prosjektnavnet.

11.1.1.5 Local Emulator Suite

Firebase Local Emulator Suite er et sett med avanserte verktøy for utviklere som vil bygge og teste apper som bruker Firebase-tjenester som Firestore og Authentication lokalt uten å påvirke produksjonstjenestene [63].

For at det skal være mulig å teste systemet uten å påvirke produksjonstjenestene når man jobber lokalt på ny funksjonalitet eller kjører tester gjennom Cypress, må man enten «mocke» («liksom»-data som kun benyttes når tester kjøres) Firebase-implementasjonen i applikasjonen, eller kjøre Firebase-tjenestene lokalt. Å mocke Firebase-implementasjonen kan være vanskelig og tar mye tid, derfor er det enklere å kjøre tjenestene lokalt og det er her emulatoren kommer inn i bildet. Firebase Local Emulator Suite gjør det enkelt å kjøre tjenestene lokalt samtidig som den også har noe funksjonalitet for feilsøking for når man implementerer ny funksjonalitet.

I dette prosjektet vil emulatoren i all hovedsak bli brukt for tester i Cypress ettersom det ikke gjør noe at produksjonstjenestene blir endret under utvikling, siden systemet ikke er satt i produksjon. Hvis produktet skulle blitt satt i produksjon med ekte brukere, kunne emulatoren også blitt brukt for lokal utvikling og feilsøking.

11.1.2 Frontend - ReactJS

Når det kom til valget av hvilken teknologi vi skulle bruke for selve webapplikasjonen stod vi ganske fritt til å velge det meste, ettersom et frontend rammeverk i liten grad har innvirkning på om vi kan oppfylle funksjonelle og ikke-funksjonelle krav. Dermed stod det valget mellom de mest kjente JavaScript-rammeverkene for frontend, som ifølge State of JS 2021 [64] er blant annet «Solid», «Svelte», «React», «Vue» og så videre. Disse brukes fordi å skrive i ren JavaScript (og eventuelt med jQuery) er veldig gammeldags, tregt og har stor fare for bugs. Det var heller ikke aktuelt å bruke «cutting-edge» teknologi som WebAssembly ettersom teknologien er langt fra moden nok til å bli brukt i en produksjonssetting.

Et viktig krav når det kom til hvilke av de mest kjente JavaScript-rammeverkene vi ønsket å bruke var at det skulle være modent og godt brukt, altså burde det gjerne være mer enn et par år gammelt. Da står man igjen med «React», «Svelte», «Vue», «Angular» og «Ember». Et annet viktig krav var at vi skulle være kjent med rammeverket fra før av, sånn at vi ikke måtte bruke unødvendig lang tid på å lære oss noe nytt helt fra bunnen av. Da stod vi igjen med «React», «Svelte» og «Vue».

React er et av de mest populære rammeverkene og har eksistert i over åtte år. Et av de store salgspunktene til React når det først kom ut var bruken av Virtual DOM, noe som gjorde UI-oppdateringer mye kjappere og mer effektivt enn det som tidligere var mulig [65]. I tillegg til dette så bruker React også composition for å lage gjenbrukbare komponenter, noe som gjør utviklingen både enklere og mer effektivt [66]. Det at React bruker en enveis dataflyt er noe mindre kjent, men det er enn så viktig ettersom det gjør kompleksiteten til systemet mye lavere og gjør det enklere å kontrollere en stor kodebase [67]. Som all

teknologi har React også noen ulemper; selv om Virtual DOM generelt er en fordel så er det enkelte tilfeller der det kan være en ulempe. Hvis man har tusener eller titusener av komponenter som skal oppdateres samtidig så vil det gå fra å være raskt til å være ekstremt tregt, eller rett og slett fryse hele applikasjonen fordi nettleseren blir oversvømt av små renders [68]. I tillegg så har React i likhet med mange systemspråk – som Java – et problem med at det kreves mye boilerplate-kode for å sette opp applikasjonen og for å lage komponenter. Det skal likevel sies at oppstarts-boilerplaten kan til en viss grad fjernes ved å bruke Create-React-App [69].

Svelte er en av de yngre blant de mest populære rammeverkene og har bare eksistert siden 2016, men tok ikke ordentlig av før rundt 2019. Det største salgspunktet til Svelte er hvor bra ytelsen og minneallokeringen er sammenlignet med andre store rammeverk, i tillegg til at bundlestørrelsen også generelt er mindre [70]. I likhet med React så bruker Svelte også composition [71] og nyter dermed de samme fordelene fra det som React. Men i motsetning til React kommer Svelte med mye av den nødvendige funksjonaliteten – State Management, Server Side Rendering (SSR), scoped styling, etc – innebygd i rammeverket [70] (og det kan dermed diskuteres om Svelte heller burde bli kalt et bibliotek enn et rammeverk). Det som derimot er negativt med Svelte er at det fortsatt er et relativt nytt rammeverk med et relativt lite økosystem og tilhengermasse, noe som gjør at det er vanskeligere å finne ferdige komponenter/bibliotek/løsninger. Det betyr også at det er mindre dokumentasjon og hjelp å finne på nett [70].

Vue har i likhet med React eksistert i over åtte år og ble utviklet som et lettvekt-alternativ til Angular – og det fikk de til, med en bundlestørrelse på bare 18KB [72]. I tillegg til dette ligner ellers Vue ganske mye på React med tanke på at Vue også bruker Virtual DOM, composition [72] og har et bra økosystem med tredjepartsbibliotek og bra mengder med dokumentasjon og informasjon på nett [73]. Det skal likevel sies at mye av informasjonen og dokumentasjonen på nett er på kinesisk ettersom Vue har blitt veldig poplært i Kina over årene [74]. I motsetning til React har derimot Vue toveis dataflyt, noe some gjør det enklere å gjøre ting som å oppdatere relaterte komponenter. Det kan derimot diskuteres om det fører til flere feil og høyere kompleksitet som øker vanskeligheten med å holde kontroll på store systemer [74].

Som vi kan se har både React, Svelte og Vue mange fordeler og noen ulemper hver for seg. Med mindre man har veldig restriktive krav (som for eksempel en applikasjon som «rendrer» tusenvis av komponenter og trenger høy ytelse) så har det relativt lite å si hvilket rammeverk man velger. For de fleste vil det beste valget være det rammeverket man er mest komfortable med å bruke og som passer best sammen med den teknologistakken man allerede har fra før av. Dermed endte vi med å velge React som rammeverket for webapplikasjonen i dette prosjektet ettersom vi begge har god erfaring med React, og det passer bra sammen med teknologien valgt for mobilapplikasjonen – React Native.

11.1.3 Kartteknologi

Use case #8 (Tabell 8) innebærer at det eksisterer en side med kartvisning. Dette utdypes videre i FK13 og FK14 (Tabell 20): data fra tilsynet skal tegnes over kartet, og observasjonene lagret under tilsynet på mobilapplikasjonen skal være klikkbare markører på webapplikasjonen. Dette forutsetter derfor at vi tar i bruk et kartbibliotek. Å lage et kart selv er helt utelukket, ettersom det ville krevd å samle inn geografisk informasjon for minimum hele Norge og deretter lage et interaktivt kart ut ifra dette, som åpenbart ville vært en enorm utfordring. Ved å ta i bruk et eksisterende kartbibliotek får vi tilgang til geografisk data og gjerne et medfølgende API for å eksempelvis illustrere ting på kartet eller styre kartvisningen programmatisk.

Ettersom kartet skal eksistere på en webapplikasjon, må kartbiblioteket vi tar i bruk være tilgjengelig for JavaScript. Det finnes flere kartbibliotek tilgjengelig for JavaScript, blant annet Mapbox GL JS[6], Leaflet[7], OpenLayers[8] og JavaScript-APIet til Google Maps[9].

[6] https://docs.mapbox.com/mapbox-gl-js/guides/
[7] https://leafletjs.com/
[8] https://openlayers.org/
[9] https://developers.google.com/maps/documentation/javascript

Det var viktig å velge et kartbibliotek som ikke krevde betaling for vårt nivå av belastning. Med belastning mener vi nettverkskall mot APIet til det gitte kartbiblioteket. For vårt bruk er det primært to typer nettverkskall som utføres: den initielle innlastingen av kartdata, og påfølgende kall for innhenting av kartdeler når brukeren beveger kartet (alle de nevnte kartbibliotekene sine kart er delt inn i deler (engelsk: *tiles*) som lastes inn dynamisk ettersom brukeren beveger kartet – dette er mindre ressurskrevende ettersom relativt lite kartdata behøver å eksistere i minnet på brukerens enhet av gangen).

Mapbox GL JS tillater opptil 50 000 initielle kartinnlastinger og uendelig mange påfølgende kartdel-innlastinger per måned før utvikleren må betale[10]. Google Maps tillater bruk for opptil 200 dollar verdt av kreditt i måneden før utvikleren må betale for å fortsette bruken av deres API, som tilsvarer 28 500 initielle kartinnlastinger og et uendelig antall kartdel-innlastinger deretter[11]. Selv om Mapbox har en mer generøs grense enn Google Maps sitt, er begge disse grensene mer enn høye nok for at vi aldri vil være i nærheten av å nå dem, grunnet det svært lave volumet av brukere på løsningen vår - i hovedsak er det kun vi utviklerne.

De to resterende bibliotekene, Leaflet og OpenLayers, er gratis og open-source løsninger og har derfor ingen begrensinger på kartinnlastinger.

En annen mulig begrensning på valg av kartbibliotek var at de faktisk måtte støtte de funksjonalitetene vi hadde for kartvisningen vår, som definert i FK13 og FK14 (Tabell 20). Disse kravene er som følger:

- Tegne punkt, linjer og egendefinerte symboler over kartet.
- Støtte for «pop-ups» som åpnes ved å trykke på markører på kartet. Disse må kunne vise egendefinert informasjon og helst egendefinert HTML-innhold, slik som tekst og knapper.

Ved å lese gjennom dokumentasjonen for alle disse kartbibliotekene fant vi ut at de alle støttet disse funksjonalitetene på en god måte. Vi stod dermed fritt til å velge hvilket som helst av disse kartbibliotekene. Vi valgte til slutt **Mapbox GL JS** som kartbiblioteket for dette prosjektet ettersom vi allerede hadde tatt i bruk Mapbox for kartet på mobilapplikasjonen fra forarbeidet, og vi var fornøyd med hvordan det fungerte og så ut. Å fortsette med Mapbox tillot oss å kunne bruke akkurat samme kartstil mellom både mobilapplikasjonen og webapplikasjonen, som gir en konsekvent visuell stil for systemet som en helhet.

11.1.4 Typescript

> TypeScript is a strongly typed programming language that builds on JavaScript, giving you better tooling at any scale. [75]

Typescript er i bunn og grunn et supersett av JavaScript som gir oss sikkerhetsnettet av å ha strengt typet språk, noe som ifølge en postmortem analyse kan stoppe opp til nesten 40 % av bugs [76]. I tillegg til dette har det også noe tilleggsfunksjonalitet, gjerne den samme funksjonaliteten som man vil finne i Babel [77] eller i ECMAScript proposal stage 0-4 [78].

Når det kommer til popularitet er Typescript også kongen av alle JavaScript- språk/kompilatorer, hvor 78 % av respondentene til 2020 State of JS [79] svarte at de bruker Typescript i dag.

Alt i alt så var det veldig naturlig å ta i bruk Typescript, både for å forhindre bugs, men også for å følge industristandarden rundt hvordan man skal utvikle webapplikasjoner i 2022.

[10] https://www.mapbox.com/pricing
[11] https://mapsplatform.google.com/pricing

11.1.5 Webpack

Webutvikling har gradvis blitt mer og mer komplekst over de siste 20 årene, og for å utvikle en webapplikasjon trengs det flere verktøy og prosesser for å få gjort ting på en effektiv måte.

Webpack er et allsidig og enkelt verktøy, men som samtidig kan konfigureres til å gjøre ekstremt komplekse prosesser som lar oss gjøre det meste som trengs innenfor moderne webutvikling. Dette kan være alt fra «module bundling», «hot module replacement», utviklingsserver, «source mapping» eller minifisering av kildekoden [80]. I dette prosjektet vil Webpack bli brukt med to forskjellige konfigurasjoner, én for utvikling og én for produksjon. I tillegg blir disse konfigurasjonene slått sammen med en felles konfigurasjon som begge bruker.

Den felles konfigurasjonen tar seg av innlastingen av forskjellige filer som CSS og bilder som .png- og .svg-filer ved hjelp av forskjellige «loaders». I tillegg til dette tar den seg også av tre forskjellige plugins: «HtmlWebpackPlugin» som tar seg av innlastingen av HTML index-filen, «WebpackManifestPlugin» som tar seg av genereringen av manifest-filen, og «Dotenv» som gjør det mulig å laste inn miljøvariabler gjennom en .env-fil.

Konfigurasjonen for utvikling tar seg av innlastingen av source maps for å gjøre utviklingen og feilsøking enklere, i tillegg til å kjøre en utviklingsserver for å teste koden på. Til slutt tar konfigurasjonen seg av «module bundling»; bundlingen av JavaScript- og TypeScript-filer (i tillegg til å kompilere TypeScript filene med TypeScript-kompilatoren) med en egendefinert transformator som hindrer stylingbibliotekets CSS-klassenavn fra å bli tilfeldig generert for å gjøre det enklere å utvikle og feilsøke.

Konfigurasjonen for produksjon tar seg av minifisering av koden, «module bundling» av JavaScript- og TypeScript-filer (i tillegg til å kompilere TypeScript-filene med TypeScript-kompilatoren), og til slutt generering av filene som er klare for å bli brukt i produksjon.

11.1.6 Testteknologi

For automatisert testing vil prosjektet bli satt opp med Cypress som teknologi for end-to-end testing. Cypress har også støtte for integrasjonstesting, men dette er ikke noe som vil bli brukt i dette prosjektet.

11.1.6.1 Cypress

Det finnes hovedsakelig to forskjellige rammeverk som blir brukt for end-to-end testing av webapplikasjoner: Selenium og Cypress [81]. Selenium, det mest brukte end-to-end rammeverket, er en industristandard som har eksistert i 20 år og fungerer på flere operativsystemer og kan krysstestе flere nettlesere som Chrome, Firefox, Safari, Edge, og så videre. Det er et veldig kraftfullt rammeverk og har en stor brukergruppe, men det har to store problem. Selenium var ikke originalt utviklet for å imøtekomme det høye nivået av asynkronitet i moderne nettsider, som fører til mye kode for å vente på at komponenter og lignende skal laste inn, noe som ofte kan føre til en lang rekke forskjellige unntak. Det største problemet med Selenium er likevel at testene er ikke-deterministiske og kan feile uten noen god grunn, noe som kan føre til at testene må kjøres flere ganger før alle blir godkjente. Cypress derimot har ingen av disse ulempene og mange av de samme fordelene, men kommer med sine egne ulemper. I motsetning til Selenium hvor flere språk kan bli brukt, må Cypress-tester skrives i JavaScript. Cypress er i tillegg mye yngre (2015) og har ikke den samme brukermassen som Selenium har, men det skal sies at Cypress har blitt mye mer populært i løpet av de siste årene og er definitivt en utforder til Selenium.

Alt i alt så valgte vi å ta i bruk Cypress på grunn av de klare fordelene som Cypress kan tilby. Dette kommer i tillegg til at fra personlig erfaring så vil Cypress gi oss like dekkende tester som Selenium, men med mindre problemer og utviklingstid. Dette tillater oss altså å skrive end-to-end tester i JavaScript

(eller TypeScript), noe som gjør det enkelt å gjøre overordnede tester som sjekker at funksjonaliteten fungerer som den skal fra en brukers perspektiv.

11.1.7 Verktøy

I tillegg til forskjellige rammeverk og bibliotek så er forskjellige verktøy også viktig teknologi i utviklingsprosessen. Dette kan være verktøy som gjør kodingen enklere, lagring og håndtering av kildekoden enklere, eller automatiserer deler av prosessen fra koden blir skrevet til koden skal settes i produksjon.

11.1.7.1 Intellij IDEA/Visual Studio Code (VSC)

For utviklingen av applikasjonen så har det ikke blitt satt noen begrensninger på hvilke Integrated Development Environment (IDE) som skal brukes. Dette er ettersom det generelt har veldig lite å si for det ferdige systemet, og spesielt lite for webapplikasjoner som ikke trenger flere konfigurasjoner for kompilering eller annen avansert funksjonalitet. Dermed vil forfatterne selv velge hvilket IDE de ønsker å bruke ut ifra deres personlige preferanser – hvorav Andreas ønsker å bruke Intellij IDEA [82] og Lars Erik ønsker å bruke VSC [83].

11.1.7.2 GitLab

For kildekontrollversjonering innenfor programvareutvikling er Git så og si det eneste systemet som blir brukt. Men når det kommer til hvilken plattform (som wrapper Git) man ønsker å bruke, finnes det flere forskjellige alternativer, der GitHub[12], Bitbucket[13] og GitLab[14] er noen av de mest brukte.

For dette prosjektet ble GitLab valgt til dels fordi NTNU hoster sin egen GitLab-instanse, men også ettersom i tillegg til implementasjonen av Git så har GitLab også en rekke tjenester som gjør Agile utvikling enklere. Dette inkluderer ting som Issues og Milestones som er fantastisk for å enklere holde oversikt over FK-er som må implementeres og hvor langt man har kommet i den aktive sprinten.

GitLab har også veldig god støtte for Continuous Integration (CI)/Continuous Deployment (CD) som vi går nærmere inn på i Seksjon 11.1.7.3

11.1.7.3 CI/CD - Gitlab

CI/CD er en metode for å introdusere automasjon i de forskjellige stegene i utviklingsprosessen og for å kontinuerlig levere nye versjoner av programvaren til kunden/brukerne [84].

For dette prosjektet vil Gitlab sitt CI/CD pipeline/system [85] bli brukt. Dette er fordi Gitlab tilbyr et av de beste CI/CD-systemene tilgjengelig og det er allerede koblet til Gitlabs Git-system, noe som gjør det veldig enkelt å koble sammen kildekoden og CI/CD-pipelinen.

For CI så vil det være to jobber i ett steg i pipelinen som kjører tester for å sjekke at systemet ikke har noen feil som testene klarer å oppdage. Steget «test» vil inneholde jobben «test-e2e» som kjører end-to-end testene i Cypress.

For CD så vil det være to jobber i to forskjellige steg i pipelinen som jobber sammen for å publisere den nye versjonen av programvaren til Firebase Hosting. Først vil jobben «build-web» kjøre i steget «build»

[12] https://github.com/
[13] https://bitbucket.org
[14] https://about.gitlab.com/

som bygger filene til webapplikasjonen gjennom «npm run build:production»-kommandoen. Disse filene blir deretter lagret som et artifact som tillater den neste jobben «deploy-web» i steget «deploy» å laste opp og publisere den nye versjonen av webapplikasjonen til Firebase Hosting ved hjelp av Firebase-tools.

Konfigurasjonen av CI/CD-pipelinen i Gitlab-repoet vårt vil kunne bli funnet i «.gitlab-ci.yml»-filen i roten av repoet [86].

11.2 Arkitektoniske taktikker

En arkitektonisk taktikk er en designbeslutning som påvirker et ikke-funksjonelt krav sitt «Response Measure» [87, kapittel 3.4]. Dette påvirker altså et systems respons på stimulus, noe som betyr at taktikker kan øke (eller senke) et systems oppnåelse av et ikke-funksjonelt krav som for eksempel usability.

På grunn av begrensningene forklart tidligere vil vi bare ta for oss de ikke-funksjonelle kravene som har arkitektoniske taktikker som vi har mulighet til å implementere selv i webapplikasjonen.

11.2.1 Usability

For å forbedre brukervennlighet har vi i hovedsak to taktikker vi kan ta i bruk, «Support User Initiative» og «Support System Initiative» [87, kapittel 13.2]. Disse kan igjen deles inn i flere undertaktikker, men vi vil i all hovedsak bruke alle sammen i sin helhet som beskrevet i Seksjon 11.2.1.1 og Seksjon 11.2.1.2.

11.2.1.1 Support User Initiative

Denne taktikken består av fire undertaktikker: «Cancel», «Pause/Resume», «Undo» og «Aggregate» [87, kapittel 13.2]. Ut ifra disse planlegger vi å ta i bruk «Cancel», «Undo» og «Pause/Resume».

For at en applikasjon skal ha god brukeropplevelse, er det viktig at den tillater brukeren å utforske applikasjonen uten å være redd for å gjøre feil som de ikke senere kan angre på. Det er derfor viktig at de kan «Cancel» eller «Undo» tidligere handlinger. Dette kommer i tillegg til at den tillater brukeren å kunne gjøre arbeidet effektivt ved å «Pause/Resume» oppgaver eller «Aggregate» over flere ting for å utføre en handling istedenfor flere. I denne applikasjonen vil det ikke finnes handlinger som det er mulig å aggregere over, og derfor vil «Aggregate» ikke bli tatt i bruk her.

11.2.1.2 Support System Initiative

Denne taktikken består av tre undertaktikker: «Maintain Task Model», «Maintain User Model» og «Maintain System Model» [87, kapittel 13.2]. Alle tre vil bli brukt implisitt i implementeringen av systemet.

Når brukeren benytter applikasjonen er det viktig at systemet holder styr på konteksten og forstår hvordan den skal oppføre seg for å maksimere brukeropplevelsen gjennom å assistere brukeren til å få utført det de ønsker så kjapt og enkelt som mulig. Denne konteksten får systemet gjennom å opprettholde en «Task Model», samtidig som den holder kontroll på brukerens kunnskap av systemet gjennom en «User Model». For å kunne assistere brukeren riktig opprettholder systemet også en modell av seg selv gjennom en «System Model».

11.2.2 Security

Når det kommer til sikkerhet er det meste av taktikker og arkitektoniske-/designmønster allerede implementert og håndtert av Firebase bak kulissene. Innenfor sikkerhetstaktikker finnes i hovedsak fire overordnede taktikker: «Detect Attacks», «Resist Attacks», «React to Attacks» og «Recover from Attacks» [87, kapittel 11.2]. Utav disse tar Firebase seg av det meste; de eneste taktikkene vi kan ta for oss er et par underkategorier innenfor «Resist Attacks».

11.2.2.1 Resist Attacks

Denne overordnede taktikken består av åtte underkategorier: «Identify Actors», «Authenticate Actors», «Authorize Actors», «Limit Access», «Limit Exposure», «Encrypt Data», «Separate Entities» og «Change Default Settings» [87, kapittel 11.2]. Av disse så har vi mulighet til å implementere «Authenticate Actors», «Authorize Actors», til en viss grad «Limit Access» og til dels «Change Default Settings».

Gjennom å sette opp Firebase Authentication og endre standardinstillingene til Firebase Firestore så vil vi kunne autorisere og autentisere brukere før de får lese data fra databasen. Samtidig hvis vi er forsiktige med hvordan vi skriver Firebase Firestore sine sikkerhetsregler, vil vi også kunne snevre inn tilgangen til dataen sånn at bare de som trenger tilgang til spesifikke dokumenter eller samlinger får det, noe som gjør det vanskeligere for hackere å lese/endre/slette data de ikke skal ha tilgang til.

11.2.3 Modifiability

For å forbedre modifiserbarheten så finnes det en rekke forskjellige taktikker rundt hvordan man skriver og organiserer koden sin, «Reduce Size of a Module», «Increase Cohesion», «Reduce Coupling» og «Defer Binding» [87, kapittel 8.2]. Av disse skal vi bruke alle utenom «Defer Binding» ettersom frontend/webapplikasjoner generelt sett ikke har variable verdier som man kan sette senere i livssyklusen til applikasjonen. Dermed får man ikke noe gevinst av å bruke taktikken «Defer Binding».

11.2.3.1 Reduce Size of a Module

Denne overordnede taktikken består bare av én taktikk, nemlig «Split Module» [87, kapittel 8.2]. Den går enkelt og greit ut på at vi splitter opp større moduler inn i flere mindre moduler, slik at ved endringer blir så lite som mulig av koden påvirket.

11.2.3.2 Increase Cohesion

Denne overordnede taktikken består også bare av én taktikk, nemlig «Increase Semantic Coherence» [87, kapittel 8.2] og ligner til dels på Seksjon 11.2.3.1 ved at hvis en modul har to eller flere ansvar så burde modulen splittes inn i flere moduler (nye eller eksisterende) slik at hver modul bare har ett ansvar. Dette hindrer at moduler blir overkompliserte og at logiske forandringer i systemet påvirker flere moduler.

11.2.3.3 Reduce Coupling

I motsetning til de tidligere overordnede taktikkene i Seksjon 11.2.3.2 og Seksjon 11.2.3.1 så har denne fem underordnede taktikker: «Encapsulate», «Use an Intermediary», «Restrict Dependencies», «Refactor», «Abstract Common Services» [87, kapittel 8.2]. Av disse underordnede taktikkene er det hovedsakelig bare «Refactor» og til en viss grad «Abstract Common Services» vi vil ta i bruk.

I motsetning til typiske systemapplikasjoner, er webapplikasjoner og gjerne de mest populære UI-bibliotekene bygd på en måte som gjør at det generelt allerede har lav coupling. Dette gjør at flere av taktikkene ikke gir mening for vår applikasjon. «Refactor» vil bli brukt i de tilfellene man ser at to eller flere moduler blir påvirket av en forandring, da refaktorer vi koden for å redusere couplingen så mye som mulig. I tillegg så vil «Abstract Common Services» til en viss grad bli brukt når vi abstraherer UI-komponenter og API-tjenester som å kunne lese eller skrive data til backend-systemet.

11.3 Arkitektoniske- og designmønster

Arkitektoniske mønster er sammensetninger av arkitektoniske elementer som er satt sammen for å løse et bestemt problem og som har blitt dokumentert og formidlet over en lengre tidsperiode. Altså er det en gruppe arkitektoniske elementer som sammen gir en pakke med strategier for å løse et problem som et system sliter med [87]. Designmønster derimot er mer generalisert og består av et navn, problem, løsning og konsekvenser (resultater og avveininger). Hvert designmønster fokuserer på et bestemt problem, og beskriver når mønsteret kan bli brukt, om det kan brukes med andre designbegrensninger og til slutt resultatene og avveiningene som kommer med mønsteret [88].

I denne delseksjonen vil vi gå gjennom hvilke arkitektoniske- og designmønster som vi har brukt i arkitekturen til applikasjonen for å løse noen av problemene og oppfylle noen av kravene systemet har. I tillegg forklares hvordan disse mønstrene løser problemene og hvilke begrensinger mønstrene legger på systemet.

11.3.1 Shared Data

Shared data er et designmønster beskrevet av Keeling i [89] og tilhører gruppen «Component & Connector». Mønsteret går ut på at man har flere komponenter eller systemer som aksesserer den samme dataen gjennom en felles database. Dermed skjer meste parten av interaksjonene mellom datakilden og komponentene, i motsetning til et eventbasert system hvor komponentene ville kommunisert seg imellom.

Fordelene med dette mønsteret er at man øker sikkerheten og personvern ved at man bare har én enkel kilde å beskytte. Det kan også øke skalerbarheten hvis databasen er satt opp riktig, men samtidig så får man ulempen med at man får et såkalt «single point of failure» som kan skade tilgjengeligheten og ytelsen. Men ettersom det er sikkerhet og personvern, og ikke tilgjengeligheten og ytelsen som er de ikke-funksjonelle kravene til dette systemet, passer det bra for det vi ønsker å oppnå.

I systemet vil Firebase Firestore være den felles databasen, mens mobilapplikasjonen og webapplikasjonen vil være komponentene.

11.3.2 Client-Server

Client-Server er et arkitektonisk mønster beskrevet av Bass et al. i [87] og går ut på at en server tilbyr en tjeneste for flere distribuerte brukere/klienter samtidig.

Det finnes flere fordeler som at koblingen mellom serveren og klienten blir etablert dynamisk; det er ingen kobling blant brukerne/klientene, og at det er enkelt å skalere opp. Mens ulempene er relativt minimale, som at kommunikasjonen skjer over et nettverk som kan føre til degradering av ytelse ved nettverksoverbelastning.

Dette er et veldig vanlig og kjent mønster som blir brukt av de aller fleste webservere på nettet for å tilby en nettside med informasjon eller tjenester for brukere. Det finnes enkelte P2P-nettverk som ZeroNet [90] hvor nettsider blir hostet gjennom P2P-forbindelser, men sammenlignet med Client-Server er dette

veldig uvanlig. For en vanlig webapplikasjon for en målgruppe som vi har så er P2P helt utelukket og Client-Server er det definitivt beste valget.

11.3.3 Multi-Tier

Multi-Tier er et designmønster beskrevet av Keeling i [89] og går ut på at systemer/applikasjoner er organisert inn i logiske grupper som deretter kan bli plassert i fysiske komponenter som en server eller en skyplattform. Altså kan man for eksempel ha selve webapplikasjonen i et nivå som kommuniserer med API-grensesnittet på et annet nivå, som igjen kommuniserer med en tjeneste i et annet nivå som til slutt kommuniserer med databasen i et siste nivå. Dette fører til lang rekke fordeler som økt sikkerhet, ytelse, tilgjengelighet og modifiserbarhet. Men det kan være vanskelig å håndheve nivåinndelingene når det vokser seg stort, samtidig kan for mange nivå skape problemer med ytelse og vedlikeholdbarhet.

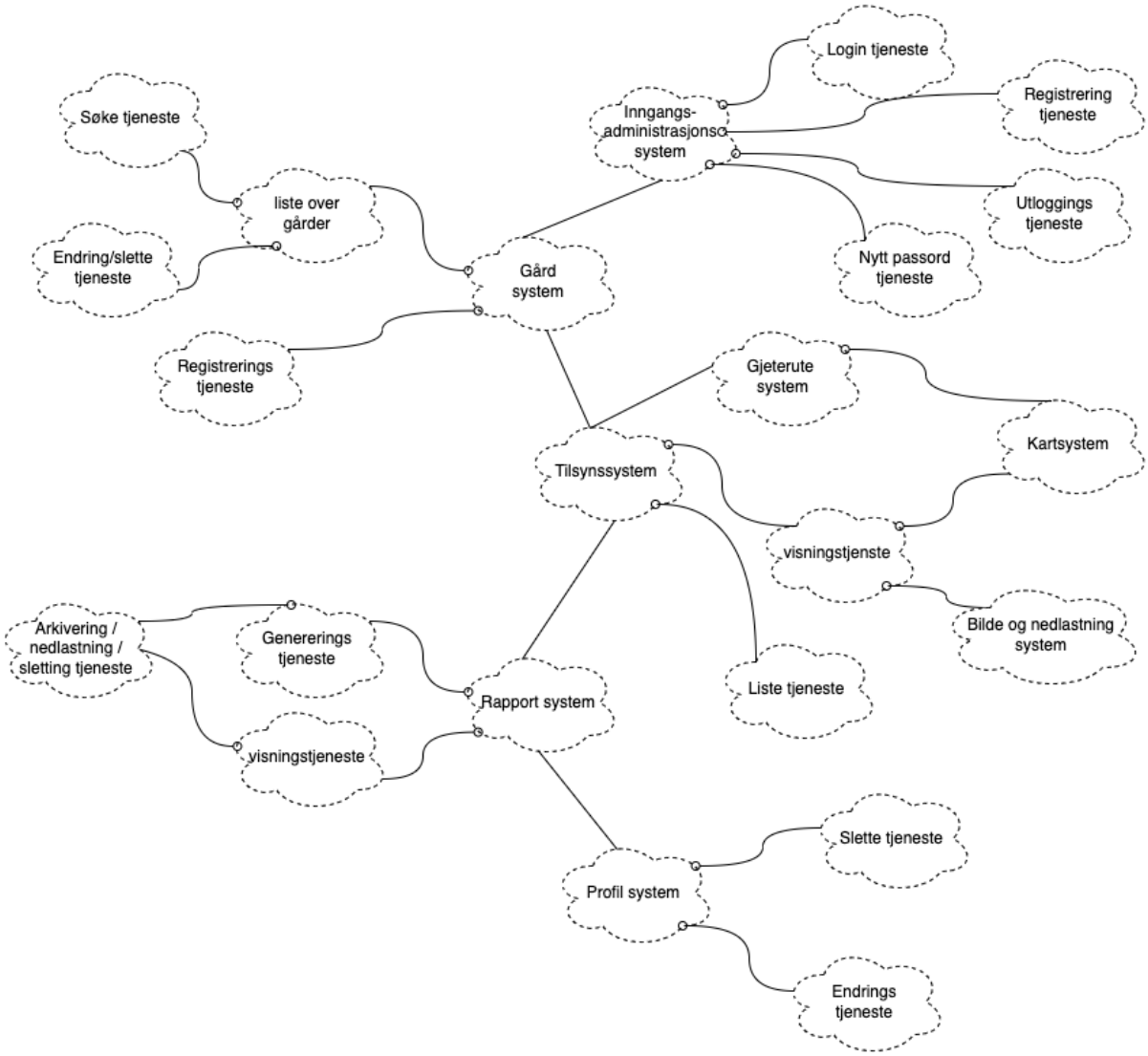
Det fulle systemet fra dette prosjektet vil følge Multi-Tier mønsteret med mobil- og webapplikasjonen i et nivå og deretter de forskjellige Firebase-tjenestene som trekker seg over flere nivå. En grafisk representasjon kan bli sett i Phsyical view-diagrammet i Seksjon 11.4.4. Ettersom systemet vårt er relativt lite så vil det få mange av fordelene fra mønsteret, uten de ulempene som kommer fra systemer med for mange nivå.

11.4 Arkitektoniske views (viewpoint)

Beskrivelsen av en arkitektur kan bli organisert rundt fire forskjellige views – logical view, development view, process view og physical view – i tillegg til bruksmønster eller scenarier som blir det femte viewet. Dette blir kalt 4+1 view-modellen av Kruchten [91]. I denne seksjonen vil vi ta for oss de fire viewene, men ikke bruksmønstre eller scenarier ettersom vi har bruksmønster i Seksjon 10.3.1. Det skal sies at disse bruksmønstrene er noe annerledes enn det Kruchten beskrev i [91], men vi ser på det vi har som tilfredsstillende for beskrivelsen av arkitekturen vår.

11.4.1 Logical view

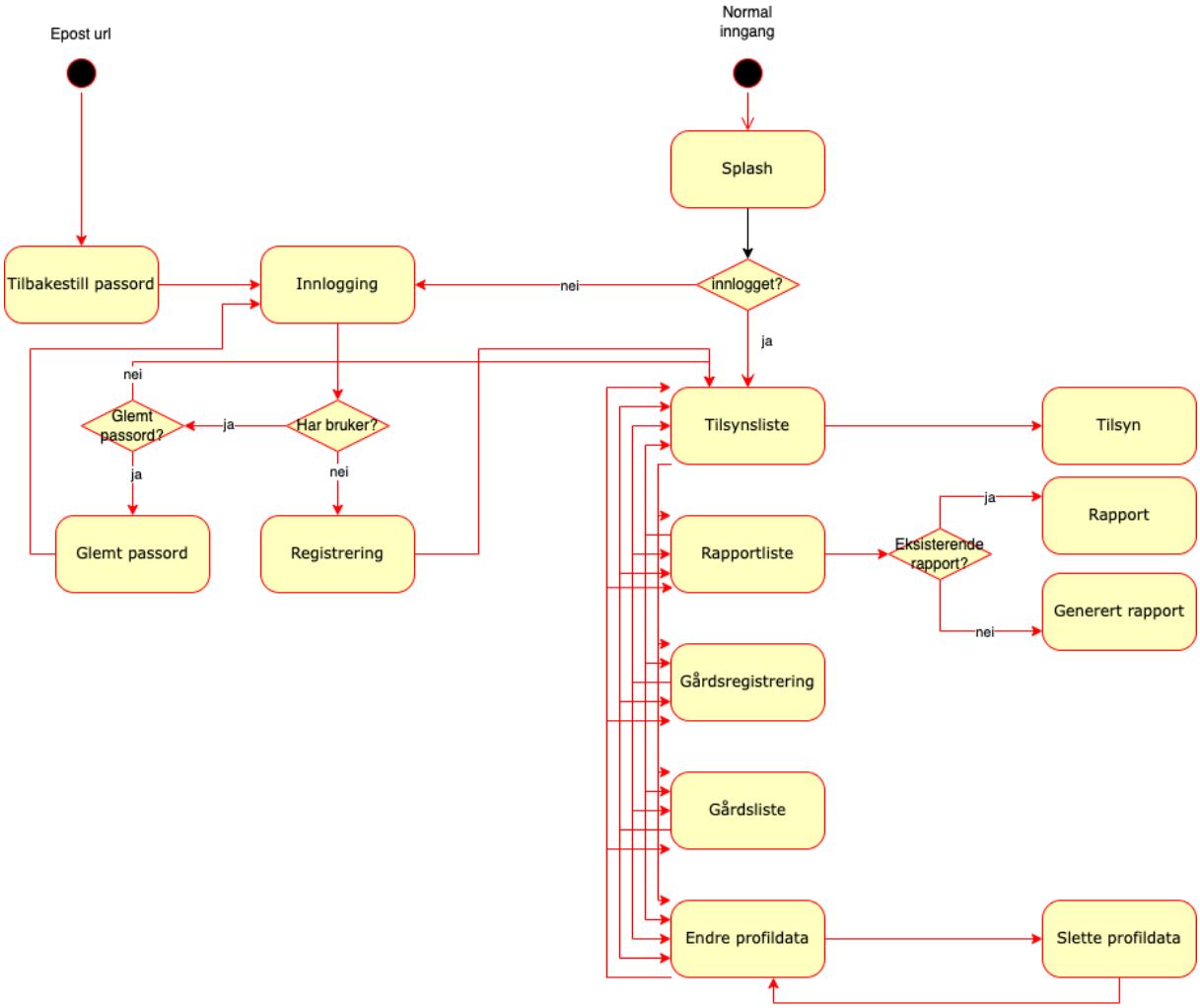
Den logiske arkitekturen støtter primært de funksjonelle kravene, altså hvilke tjenester den skal tilby brukerne [91]. I Figur 18 blir webapplikasjonens logiske arkitektur beskrevet i form av et logical view.



Figur 18: Logical view av systemet.

11.4.2 Process view

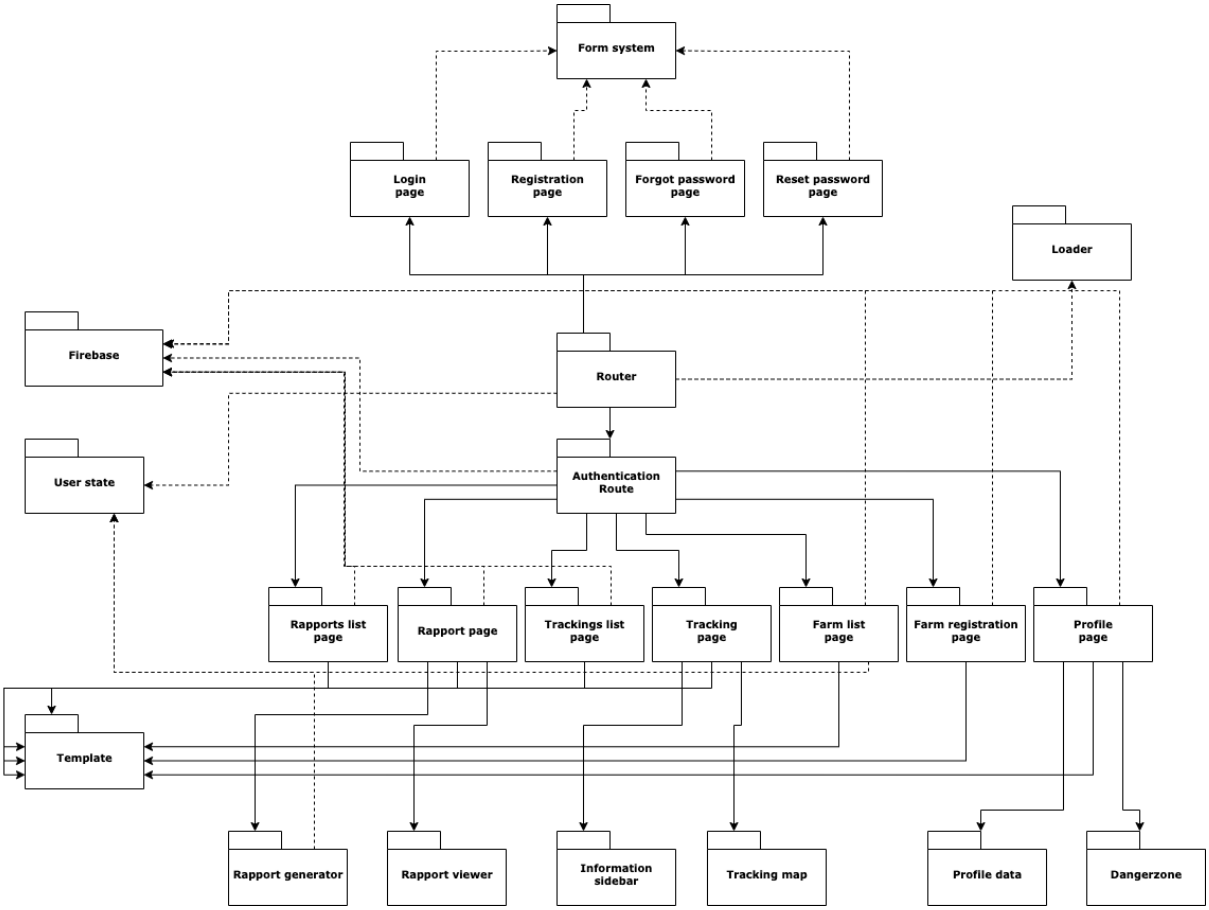
Prosess-arkitekturen støtter til grads de ikke-funksjonelle kravene og kan beskrives med forskjellige nivå av abstraksjon [91]. Dette er derimot mer relevant i avanserte systemer eller spill som har backend-systemer som jobber sammen, med flere tjenester som kan kommunisere eller kjøre parallelt. I en webapplikasjon er ikke dette like relevant. Dermed er vårt process view i Figur 19 mer som et process state-diagram som viser de forskjellige rutene i systemet enn et tradisjonelt process view.



Figur 19: Process view av webapplikasjonen i form av UML Activity Diagram-notasjon.

11.4.3 Development view

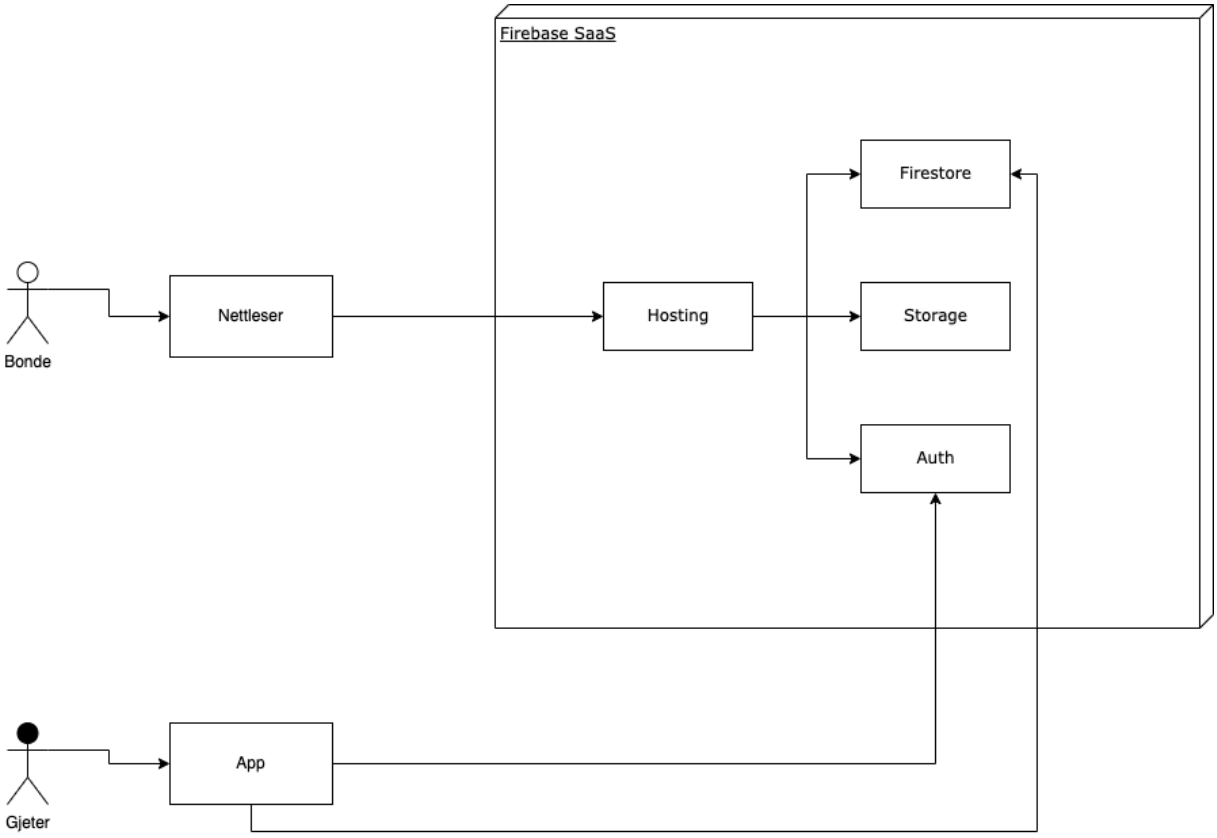
Development-arkitekturen fokuserer på hvordan programvaremoduler faktisk er organisert i utviklingsmiljøet [91]. Selv om dette er vanskelig og til dels helt umulig å forutsi før man faktisk starter å jobbe på utviklingen, kan fortsatt Development viewet i Figur 20 gi et godt overblikk over hvordan ting skal organiseres og hvilke av de større modulene som skal bli laget.



Figur 20: Development view av systemet.

11.4.4 Physical view

Den fysiske arkitekturen støtter primært de ikke-funksjonelle kravene, ettersom programvare kjører på fysiske servere, noder og over nettverk [91]. Det Physical viewet beskriver er hvor de forskjellige tjenestene kjører i den fysiske verden og hvordan de samhandler. Figur 21 beskriver dette prosjektets fysiske arkitektur, som er relativt simpel ettersom hele backend-systemet kjøres i Firebase som er en skybasert BaaS.



Figur 21: Physical view av systemet.

11.5 KOTB - Komponenter og tekniske begrensninger

Valget av teknologi har en enorm innvirkning på arkitekturen og hvilke ikke-funksjonelle krav man kan innfri til hvilken grad. Det fører også til teknologiske begrensninger som vil følge systemet og arkitekturen gjennom utviklingsprosessen. Denne delseksjonen vil gå gjennom de komponentene som ble valgt som legger til teknologiske begrensninger og hvordan dette legger begrensninger for hva som er mulig for systemet og arkitekturen å oppnå.

11.5.1 Firebase

Å bruke en Backend-as-a-Service plattform som Firebase vil føre til en rekke restriksjoner og tekniske begrensninger rundt både hva som er mulig å gjøre, men også hva som er mulig med tanke på utvidelse av funksjonalitet og individualisering av programvarearkitekturen for å tilpasse det til prosjektets krav.

En av de mest åpenbare restriksjonene ved å bruke Firebase er at man gir opp mye av kontrollen rundt hvilke arkitektoniske taktikker, arkitektoniske mønster og designmønster man kan bruke for å tilpasse arkitekturen til å fylle de ikke-funksjonelle kravene til prosjektet. Dermed må man enten akseptere at den graden av for eksempel ytelse, tilgjengelighet, modifiserbarhet og så videre som Firebase tilbyr er bra nok, eller så må man finne en annen løsning for backend-systemet.

11.5.1.1 Authentication

Firebase Authentication fungerer som forklart tidligere ved at man aktiverer en eller flere providers som Twitter, Google, e-post eller telefoninnlogging. Problemet er at hvis man velger de fleste av disse provideme så vil man være bundet til et relativt lite utvalg av informasjon som man kan lagre om brukeren. Ofte så lite som «email», «email_verified» og «nickname» («nickname» er ofte heller ikke påkrevd eller mulig å registrere ved registrering av en ny konto). Hvis man ønsker å lagre mer informasjon om en bruker, for eksempel fornavn og etternavn, må man enten lagre det i Firebase Firestore (noe som begrenser hvordan dataen kan brukes og endre) eller lage sin egen egendefinerte provider. Dette vil i så fall kreve en dedikert autentiseringsserver utenfor Firebase Authentication.

11.5.1.2 Firestore

Hvis man er vant med SQL-databaser så kan det komme som en stor overraskelse hvor begrenset Firebase Firestore sine spørringer er sammenlignet med relasjonelle databaser. En av Firebase Firestore sine største tekniske begrensninger er en veldig begrenset spørringsfunksjonalitet. Eksempler på disse begrensningene inkluderer ingen mulighet for aggregerte resultater, ikke mulighet for å blande «range» og «not equal»-operatorer i en sammensatt spørring, og ikke mulighet for å sortere en spørring som har en «likhets»-operator. En full liste over spørringsbegrensninger kan bli funnet i dokumentasjonen til Firebase Firestore i [92].

Som forklart i tidligere seksjoner så er Firebase Firestore en dokumentbasert database, og dette fører til begrensninger på hvordan du leser og sletter data. I motsetning til i en SQL-database hvor du kan utføre JOIN på flere tabeller sammen for å få ut all informasjonen du trenger, er det ikke like enkelt i en dokumentbasert database. Hvis du har et dokument som har én (eller flere) subkolleksjon (som igjen kan ha flere dokumenter og subkolleksjoner) og du vil ha all informasjonen assosiert med et dokument, må du manuelt hente ut alle dokumentene fra en subkolleksjon på hvert nivå rekursivt helt til du har hentet alle dokumentene fra alle subkolleksjonene. Det blir praktisk talt som å gjøre et dybde-først-søk (DFS) hvor du fortsetter til du har besøkt alle nodene i treet.

Testing er en veldig viktig del når det kommer til systemutvikling for å sørge for at brukerne ikke opplever feil når de bruker systemet. Dette er normalt ganske greit å få til gjennom integrasjonstester, end-to-end tester, enhetstester, og så videre. I Firebase derimot så var det lenge ikke mulig å teste ting som sikkerhetsreglene, Authentication providers, eller cloud functions uten å manuelt teste det opp mot Firebase-serverne. Men i 2020 [93] kom Firebase ut med Firebsae Local Emulator Suite som lar deg teste opp mot en lokal instans gjennom manuell testing, enhetstesting og integrasjonstesting. Det må derimot merkes at det fortsatt er i betatesting som betyr at funksjonaliteten kan endre seg på bakover-inkompatible måter. Den er heller ikke underlagt noen SLA eller deprecation-policy og har begrenset eller ingen støtte [94].

Selv om databaser generelt ikke burde lagre store mengder data i et felt eller et dokument kan det fortsatt være situasjoner hvor man ønsker å lagre store mengder data. Et eksempel er hvis man skulle ønske å midlertidig lagre bilder som base64-strenger før de blir prosessert av en ekstern tjeneste (slik som Firebase Cloud Functions) og lagret i en bedre egnet tjeneste slik som Firebase Storage. Firebase Firestore har en øvre grense på 1 MiB for dokumentstørrelse og 1 MiB - 89 B for en feltverdi [95]. PostgresSQL derimot har en øvre grense på 1 GB på feltstørrelse og 32 TB på relasjoner [96].

I tillegg til alle disse begrensningene så har Firebase også flere kvoter og grenser med tanke på feltnavn, antall lesinger/endringer/slettinger/skrivinger, tidsbegrensninger, indekser, og så videre. Dette kan man kan lese mer om i [95].

11.5.2 React

Fordelene og ulempene med React ble beskrevet i Seksjon 11.1.2, og flere av disse ulempene er tekniske begrensninger som pålegges applikasjonen ved bruk av React.

React har som sagt en stor bundle size sammenlignet med andre rammeverk som Svelte og Vue, og med tredjepartsbibliotek som Firebase kan React fort komme opp i flere 100 KB eller til og med over 1 MB. For de fleste moderne datamaskiner i land med moderne internettinfrastruktur så er ikke dette et problem, men det legger en teknisk begrensning på hvor liten man kan få webapplikasjonen. Dette kan bli et problem hvis produktets målgruppe blir utvidet til brukere med eldre datamaskiner og tregere internettforbindelser.

Det ble også nevnt at React bruker en Virtual DOM, noe som i noen sammenheng kan være en fordel, men det fører også med seg en ulempe. Ved re-rendering av flere tusen eller titusener av komponenter kan man få ytelsesproblemer. Altså vil bruk av React legge en viss begrensning på hva slags funksjonalitet man kan legge til – for eksempel et spill eller en graf med mange tusen noder vil ikke kunne implementeres uten store ytelsesproblemer.

I motsetning til både Svelte og Vue inneholder React et minimum av det som trengs for å bygge et UI og mangler dermed viktig funksjoner som ruting, CSS-håndtering, skjema-håndtering, og så videre. Det fører til at man enten må implementere funksjonene selv, noe som absolutt ikke er anbefalt, eller bruke tredjepartsbiblioteker. Å lage funksjonene selv vil gi minst begrensninger, men vil koste mye utviklings- og vedlikeholdstid. Å bruke tredjepartsbibliotek vil derimot legge begrensninger på sikkerhet og vedlikeholdbarhet ettersom man stoler på utenforstående når det kommer til sikkerhetsvurderinger og hvor raskt de kan oppdatere biblioteket når det er nødvendig.

11.5.3 Mapbox GL JS

Som diskutert i Seksjon 11.1.3 ble det undersøkt om de ulike kartbibliotekene som er tilgjengelig støttet de spesifikke funksjonene som var nødvendige for vårt bruksscenario. Ettersom alle bibliotekene vi fant støttet alle de påkrevde funksjonene, valgte vi Mapbox GL JS på grunn av tidligere erfaring med biblioteket i utviklingen av mobilapplikasjonen. Vi kunne derfor ikke identifisere noen spesifikke begrensinger når det kom til praktisk bruk av dette biblioteket.

Det skal likevel nevnes at bruk av Mapbox GL JS i bunn og grunn ikke er gratis, ettersom antall initielle kartinnlastinger per måned er begrenset til 50 000 i måneden. Om dette overskrides, vil hver initielle kartinnlasting etter dette koste en ørliten sum[15]. Ettersom denne grensen er svært høy i forhold til vår bruk, er dette en begrensing som ikke vil ha noen praktisk betydning for oss i løpet av utførelsen av dette prosjektet. Om vi hadde valgt en helt gratis løsning som for eksempel Leaflet, ville vi ikke hatt noen praktisk begrensning på antall gratis kartinnlastinger. Om systemet vårt skulle vært satt opp for reell bruk med et par titalls tusen brukere eller høyere, ville en migrering til en helt gratis løsning som Leaflet eller OpenLayers måtte vurderes.

[15] https://www.mapbox.com/pricing#maploads - se Mapbox GL JS.

Del V:

Design, utvikling og testing

Etter planleggingen er ferdig starter selve produktutviklingen, som består av designing, utvikling og testing. Her vil vi gå gjennom selve designprosessen og den endelige prototypen, implementasjonen av prototypen og til sist hvordan testingen av implementasjonen vil foregå.

12 Design og utforming

Design og utforming av et produkt eller applikasjon er en viktig del av prosessen for å sørge for at man får et resultat med god brukervennlighet og estetikk, i tillegg til at det gjør utviklingsfasen enklere og kortere ved at man ikke må tenkte på hvordan det man implementerer skal se ut. I denne seksjonen vil vi ta for oss prosessen rundt valg av designbibliotek, verktøy for prototyping, arbeidet med prototyping, og til slutt en presentasjon av prototypen for mobilapplikasjonen og webapplikasjonen.

12.1 Valg av designbibliotek

For å sikre et konsistent brukergrensesnitt valgte vi å følge et spesifikt designspråk og ta i bruk et komponentbibliotek.

Et designspråk kan tenkes på som en overordnet visuell profil kombinert med ulike regler og retningslinjer (*best practices*) om hvordan visuelle elementer skal utformes. For dette prosjektet valgte vi å ta i bruk Material Design [97], et designspråk utviklet av Google. Dette designspråket er blant annet tatt i bruk i det mobile operativsystemet Android. Dette designspråket ble valgt for å sørge for et sammenhengende design mellom mobilapplikasjon og webapplikasjonen, ettersom mobilapplikasjonen allerede tok i bruk Material Design.

Et komponentbibliotek er et sett med ferdiglagde, lett tilpasselige komponenter typisk implementert som nedlastbare moduler til programmeringsspråk som JavaScript. Skjermelementer på en webapplikasjon består av HTML-kode som beskriver innholdet og CSS-kode som beskriver utseendet til disse elementene. En komponent i et komponentbibliotek er i praksis ferdig laget HTML og CSS-kode, som da sparer utviklere fra å måtte implementere skjermelementer som for eksempel knapper eller søkefelt fra bunnen av. Dette gjøres fordi det ofte vil være en utfordring å gi disse et tilfredsstillende design som skalerer til alle skjermstørrelser. Ved å ta i bruk et komponentbibliotek kan utvikleren spare tid på å slippe å utforme alle skjermelementene, samtidig som de kan utforme grensesnitt som gir et inntrykk av høy kvalitet.

For dette prosjektet tok vi i bruk komponentbiblioteket MUI[16], som er basert på designspråket Material Design. Material Design som designspråk er kun et abstrakt sett med retningslinjer og «maler» for hvordan skjermelementer kan se ut, mens MUI er en konkret implementasjon av disse designprinsippene.

12.2 Verktøy for prototyping

Det nettbaserte verktøyet Figma[17] ble brukt for å skape prototypene. Figma er et gratis verktøy for å opprette vektorgrafikk og er ofte benyttet for å skape prototyper av brukergrensesnitt. En stor fordel med å bruke Figma er at det tillater flere brukere å jobbe innenfor samme arbeidsområde samtidig, noe som gjør det lett å samarbeide om prototypingsarbeid. Dette er i kontrast til bilderedigeringsprogrammer som for eksempel Adobe Photoshop[18] hvor programmet kun kjører lokalt på én PC og resultatene må eksporteres som bildefiler, deles over for eksempel e-post og så redigeres igjen avhengig av tilbakemeldingen.

12.3 Arbeid med prototyping

Arbeidet med prototyping begynte etter de ulike bruksmønstrene og funksjonelle kravene var kartlagt. Formålet med å utvikle en prototype var å utforme et grensesnitt som dekket alle de nødvendige funksjonalitetene og støttet de dokumenterte bruksmønstrene. En annen motivasjon for prototypingen var å

[16] https://mui.com/
[17] https://www.figma.com/
[18] https://www.adobe.com/products/photoshop.html

komme fram til et design på grensesnittet som vi mente var av høy brukskvalitet før selve utviklingen begynte, slik at vi kun ville behøve å fokusere på den kodemessige implementasjonen under utviklingsfasen – å redigere en prototype i Figma tar kortere tid enn å gjøre endringer i et ferdig utviklet grensesnitt. Når prototypen var ferdigstilt kunne vi gå direkte til utviklingen og ha et konkret bilde av hvordan sluttproduktet skulle se ut og virke i bruk.

Arbeidet med prototypen startet med å lage ulike skjermbilder som representerer visningene som burde være en del av løsningen, basert på de kravene vi fikk oppgitt av veileder Svein-Olaf Hvasshovd. Ettersom flere og flere funksjonelle krav ble innført etter hvert møte, jobbet vi iterativt med prototypen for å innføre nye visninger eller gjøre endringene på de eksisterende visningene slik at disse nye kravene ble imøtekommet. I flere av møtene ble visninger fra prototypen presentert for Svein-Olaf Hvasshovd og disse ble enten godkjent eller ga opphav til tilbakemeldinger om forbedringspotensiale som da ble implementert innen neste møte.

Først når hele prototypen ble godkjent og alle bruksmønstre og funksjonelle krav var dekket av de forskjellige visningene, valgte vi å gå videre til utviklingsfasen. Resultatet av arbeidet med prototypen presenteres nedenfor i neste delseksjon.

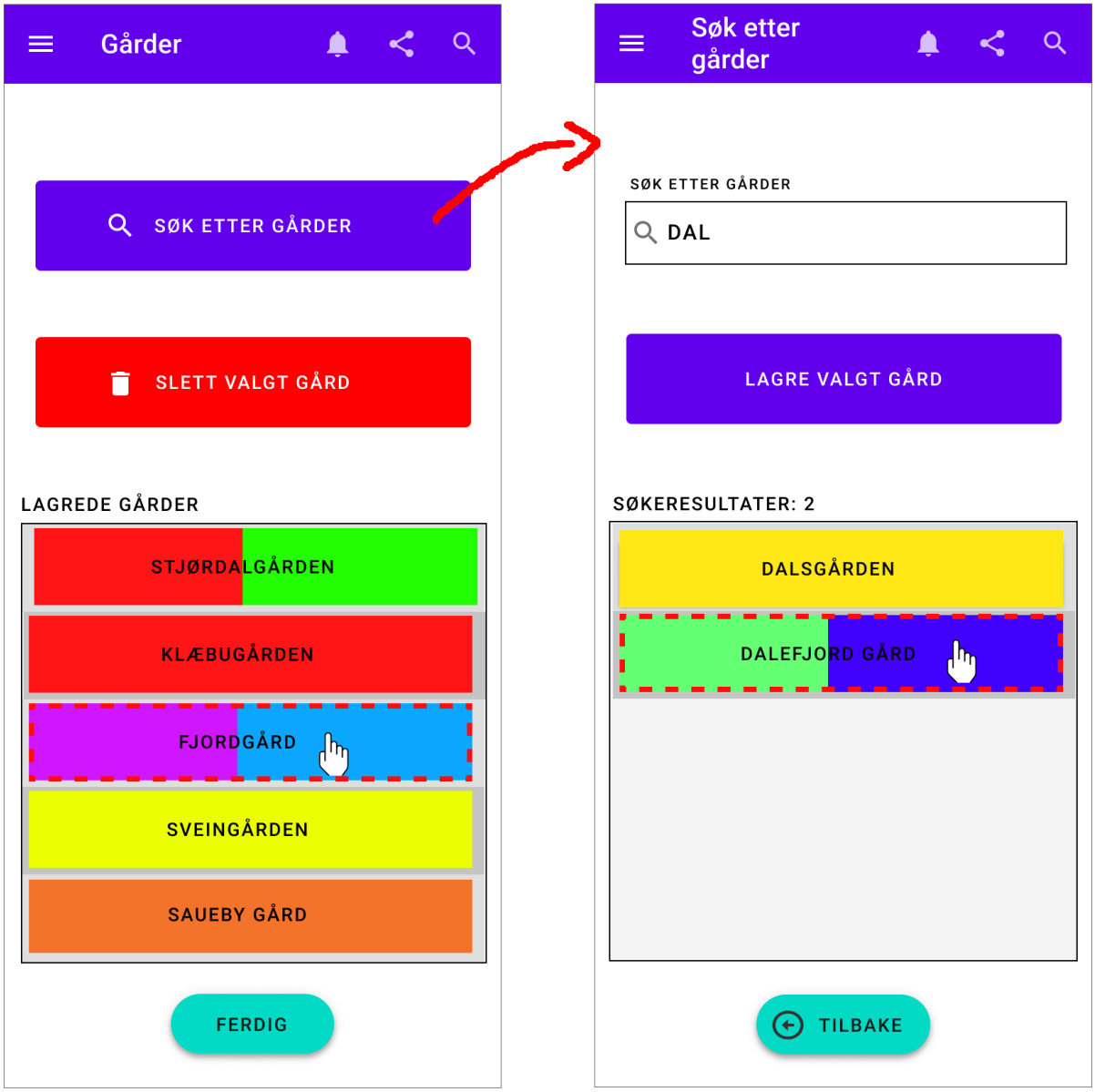
12.4 Presentasjon av prototype - mobilapplikasjon

Denne prototypen representerer hvordan de nye funksjonalitetene for mobilapplikasjonen, som er presentert i Seksjon 10.4, ble tiltenkt å representeres i grensesnittet. Kun de nye funksjonalitetene for dette prosjektet vil presenteres her, ettersom mobilapplikasjonen allerede ble gjennomgått i detalj i fordypningsprosjektet [1] og Del III av denne rapporten.

12.4.1 Lagring av gårder

Skjermbildene i Figur 22 representerer handlingene som skal kunne utføres i bruksmønster 32. Skjermbildet til venstre i Figur 22 viser hvilke gårder brukeren har lagret lokalt. Her har brukeren trykket på en spesifikk gård, og får derfor mulighet til å slette gården lokalt. En gård kan ha ensfargede eller tofargede øremerker. Om en gård er lagret lokalt vil man kunne markere at denne gårdens øremerkefarge(r) ble observert i en saueflokk under tilsyn.

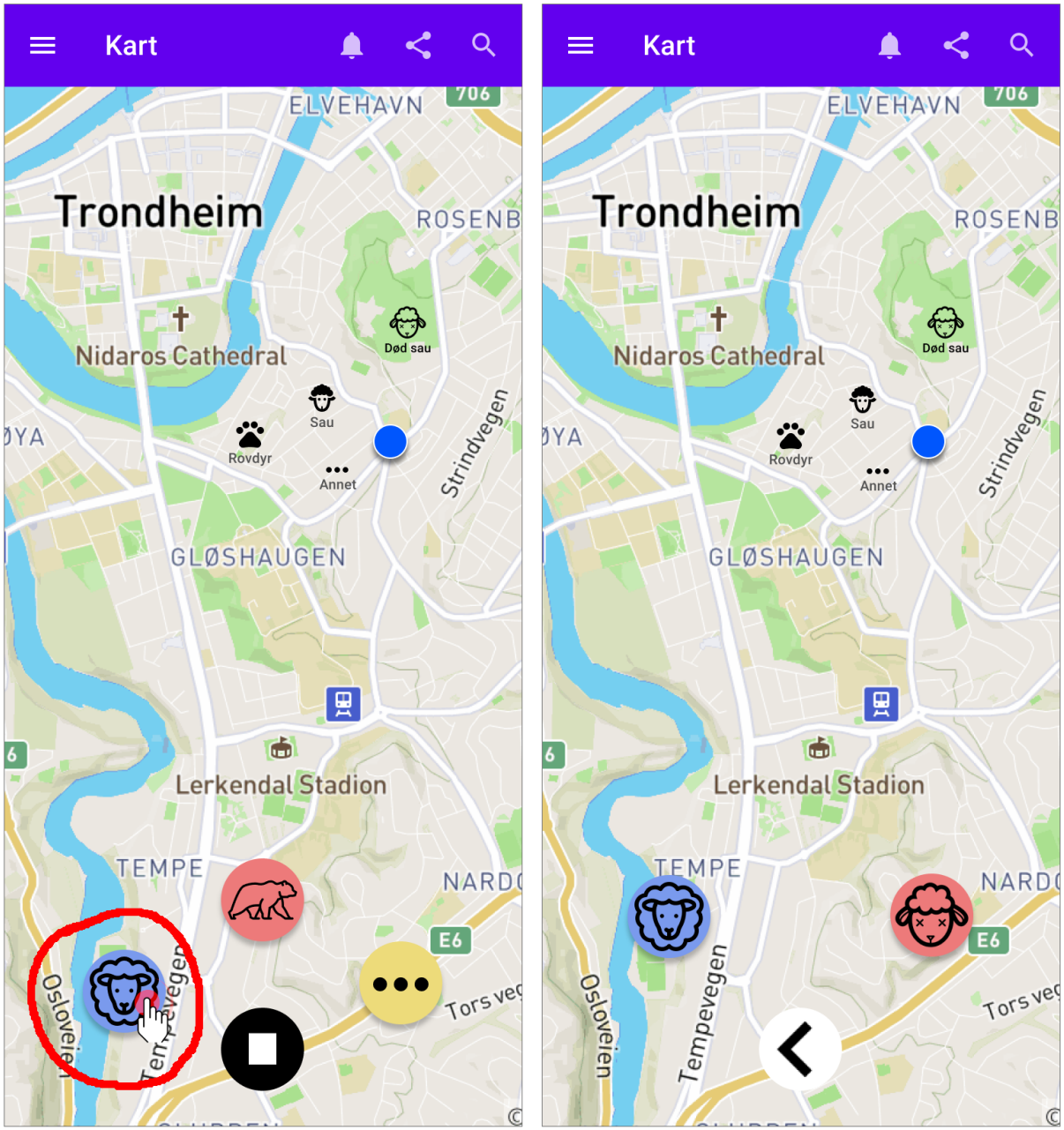
Øverst i det venstre skjermbildet i Figur 22 er det en knapp som navigerer brukeren til en skjerm der de kan søke etter gårder som har blitt registrert på webapplikasjonen. Denne skjermen vises på høyre side i Figur 22. Her har brukeren søkt etter gårdsnavnet «Dal» og fått opp to søkeresultater. Brukeren har trykket på det ene søkeresultatet og har muligheten til å lagre denne gården så den kommer opp lokalt. Om dette gjøres vil den da vises i listen over lokalt lagrede gårder vist i venstre skjermbilde.



Figur 22: Skjermbilder fra prototypen - Gårder

12.4.2 Ny flyt for registrering av saueflokk og død sau

Skjermbildene i Figur 23 viser en ny flyt som er relevant for registrering av saueflokker og døde sauer. Man trykker først på saueikonet, vist i venstre skjermbilde i Figur 23, for å åpne de mulige valgene for registreringer som er relevante for sau. De to mulighetene vises i høyre skjermbilde i Figur 23. Her har brukeren to valg - saueflokk, via det blå ikonet med bildet av en sau, og død sau, via det røde ikonet med bildet av en sau med utkryssede øyne. Nederst i det høyre skjermbildet er det en pil mot venstre som lar brukeren gå tilbake, slik at de igjen går til skjermen vist til venstre.



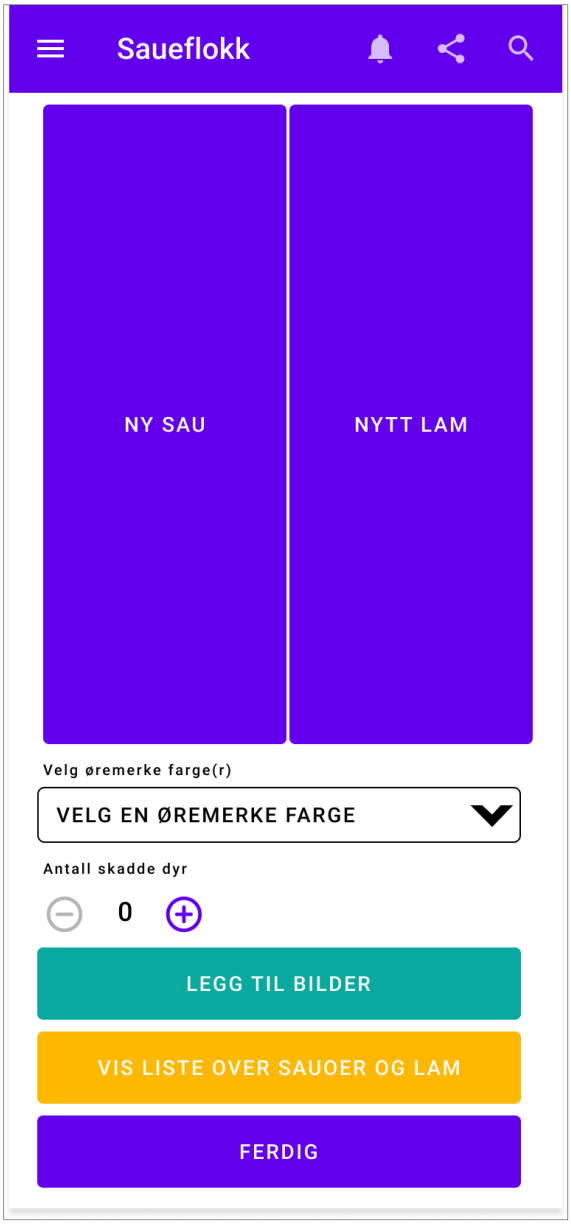
Figur 23: Skjermbilder fra prototypen - Navigere til registrering av saueflokk eller død sau

Tidligere var det slik at saue-knappen vist i venstre skjermbilde i Figur 23 tok brukeren direkte til registrering av en saueflokk. Nå er det altså slik at den åpner to valgmuligheter: saueflokk og død sau.

Det er to hovedårsaker til at «død sau» og «saueflokker» er to unike former for registrering og ikke samkjørt i en felles skjerm. Den viktigste årsaken er at vi ble informert av veileder Svein-Olaf Hvasshovd at døde sauer som regel oppdages enkeltvis, og det er sjeldent at det forekommer at flere døde sauer ligger sammen. Dermed er det ikke vits i å ha knapper for telling av antall døde dyr, slik som det er i registrering av saueflokker. Den andre grunnen er at saueflokker vil bevege på seg, mens døde sauer kun vil ligge i terrenget til noen flytter dem. Derfor er det ikke mulighet for at det iblant en flokk med sauer ligger en død sau, så det gir ikke mening å registrere en død sau innenfor en saueflokk. For å sørge for at grensesnittet og brukerflyten er oversiktlig for brukeren skiller vi derfor disse i to ulike skjermer.

12.4.3 Oppdatert skjerm for registrering av saueflokk

Figur 24 viser en ny versjon av skjermen for å registrere en saueflokk på mobilapplikasjonen. Her er det to nye komponenter sammenlignet med den gamle versjonen: En komponent der brukeren kan inkrementere eller dekrementere en telling over antall skadde dyr i flokken (en implementasjon av bruksmønster 10.4.1.3), og en knapp for å navigere til en ny skjerm der man kan legge til bilder av saueflokken (en del av implementasjonen av bruksmønster 10.4.1.4).

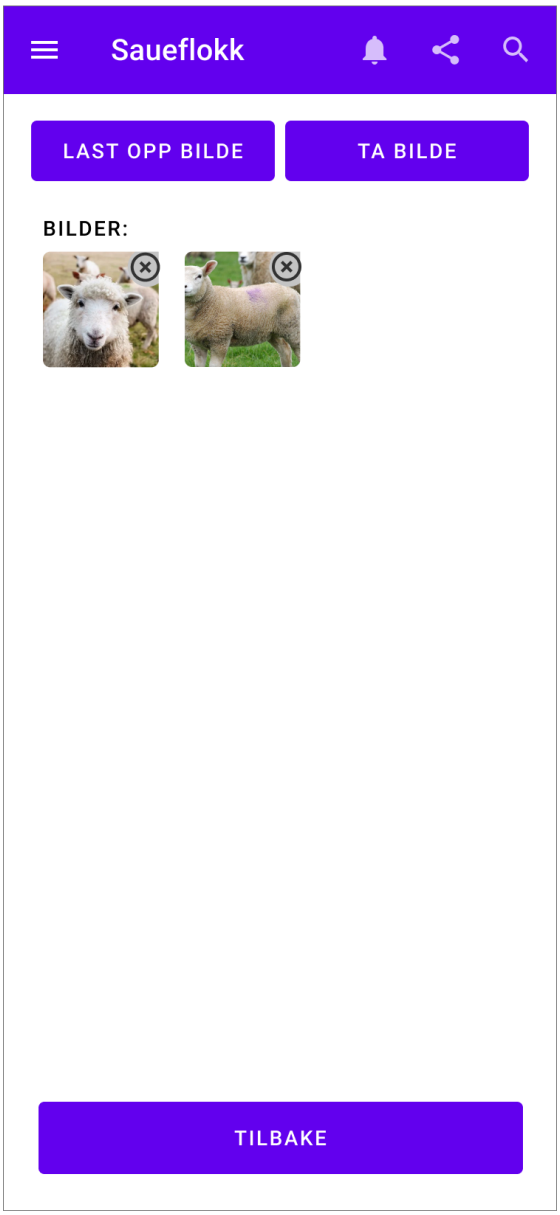


Figur 24: Skjermbilde fra prototypen - Oppdatert skjerm for registrering av saueflokk i mobilapplikasjonen

12.4.4 Registrering av bilder av saueflokk

Denne skjermen er tiltenkt å navigeres til via «Legg til bilder»-knappen sett i Figur 24. Skjermen vises i Figur 25 og er en implementasjon av bruksmønster 10.4.1.4. Å trykke på «Last opp bilde»- og «Ta

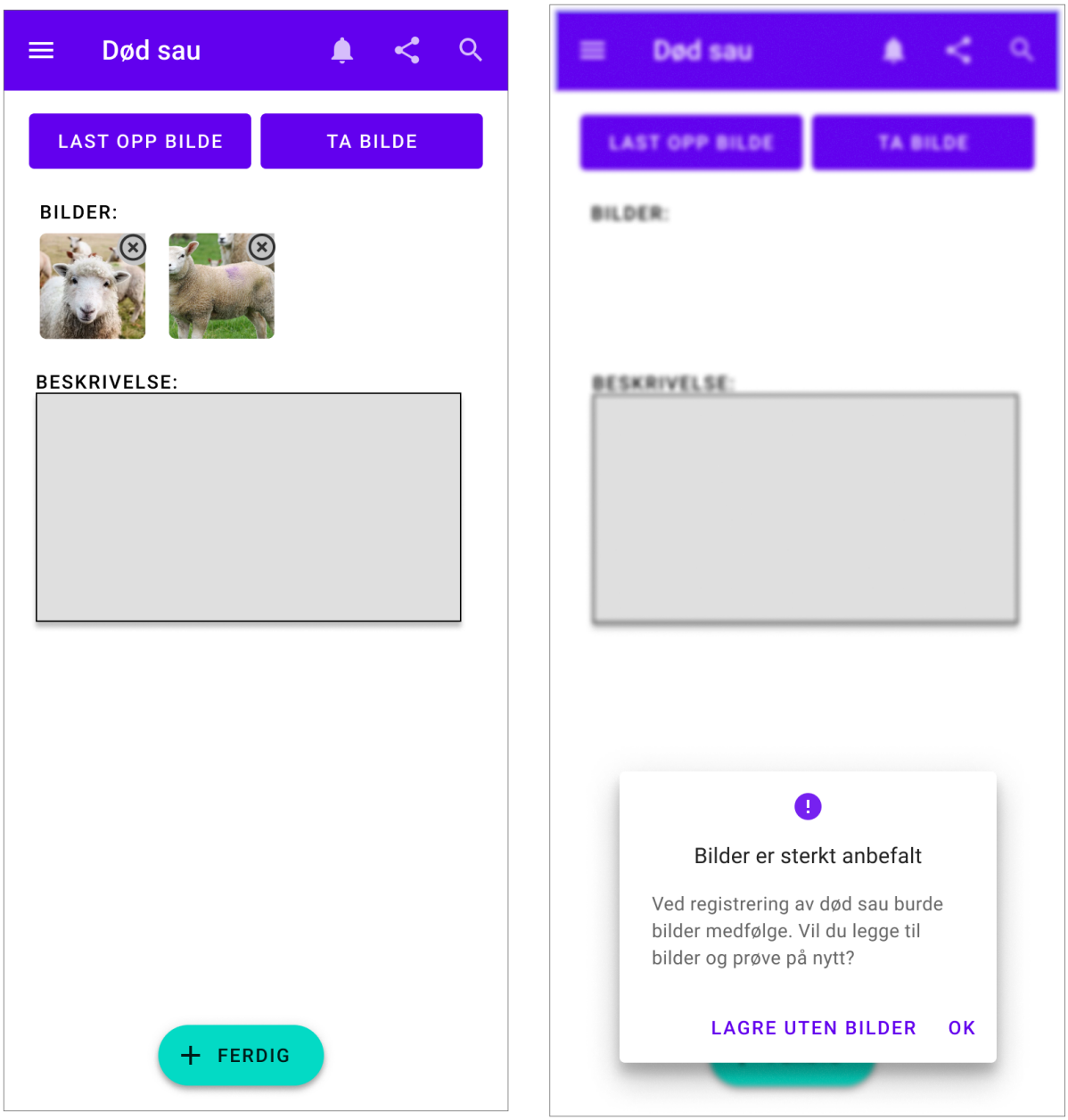
bilde»-knappene vil åpne egne menyer som er knyttet til telefonens operativsystem. Derfor har vi valgt å utelate disse menyene fra prototypen, ettersom de ikke er noe vi kan designe eller utvikle selv, men heller kommer fra mobiltelefonens operativsystem.



Figur 25: Skjermbilde fra prototypen - Skjerm for å registrere bilder av saueflokk i mobilapplikasjonen

12.4.5 Registrering av død sau

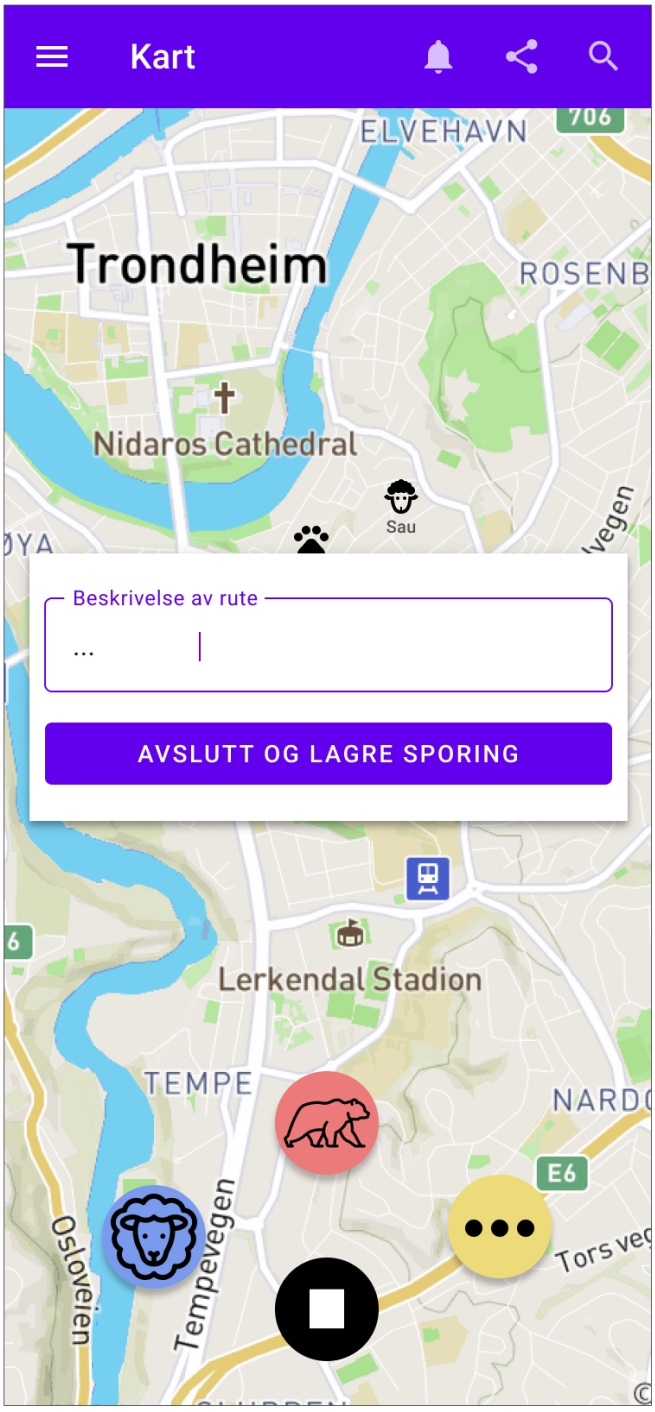
Skjermbildet til venstre i Figur 26 viser skjermen for å registrere død sau. Denne vises etter at observasjonen har blitt plassert på kartet, slik som alle andre observasjonstyper (plassering av observasjoner ble vist i presentasjonen av applikasjon i Figur 9). Dette er en implementasjon av bruksmønster 33. Her kan brukeren registrere bilder og en tekstbeskrivelse. Skjermbildet til høyre i Figur 26 viser en advarsel for når brukeren ikke har lagt til bilder i registreringen og trykker på «Ferdig»-knappen; brukeren blir da advart om at bilder burde medfølge, men brukeren får muligheten til å gå videre uten bilder.



Figur 26: Skjermbilder fra prototypen - Skjerm for å registrere død sau

12.4.6 Registrering av beskrivelse av tilsyn

Figur 27 viser dialogboksen for å skrive en beskrivelse av ruta som ble fulgt under tilsynet, og oppfyller kravene fra bruksmønster 36.



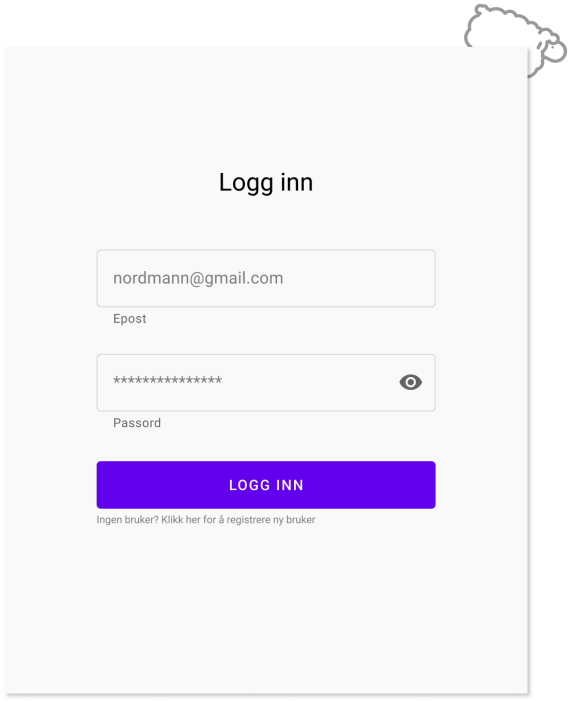
Figur 27: Skjermbilde fra prototypen - Dialogboks for å skrive en beskrivelse av ruta

12.5 Presentasjon av prototype - webapplikasjon

Prototypen, som ble laget i Figma, er en representasjon av hvordan grensesnittet til webapplikasjonen skal se ut. De ulike skjermene fra prototypen presenteres her.

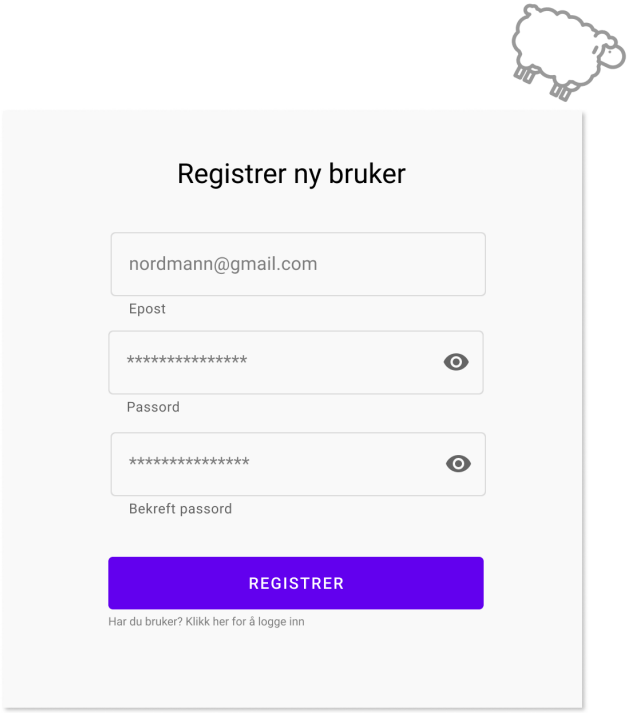
12.5.1 Innlogging og registrering

Figur 28 viser siden for innlogging, som oppnår funksjonaliteten påkrevd i FK1 (Tabell 13).



Figur 28: Skjermbilde fra prototypen - Innlogging

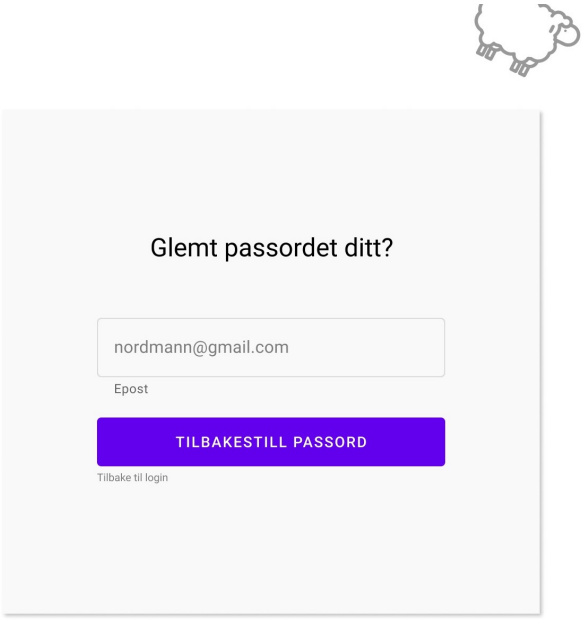
Figur 29 viser registreringssiden, som oppnår funksjonaliteten påkrevd i FK2 (Tabell 14).



Figur 29: Skjermbilde fra prototypen - Brukerregistrering

12.5.2 Glemt og tilbakestill passord

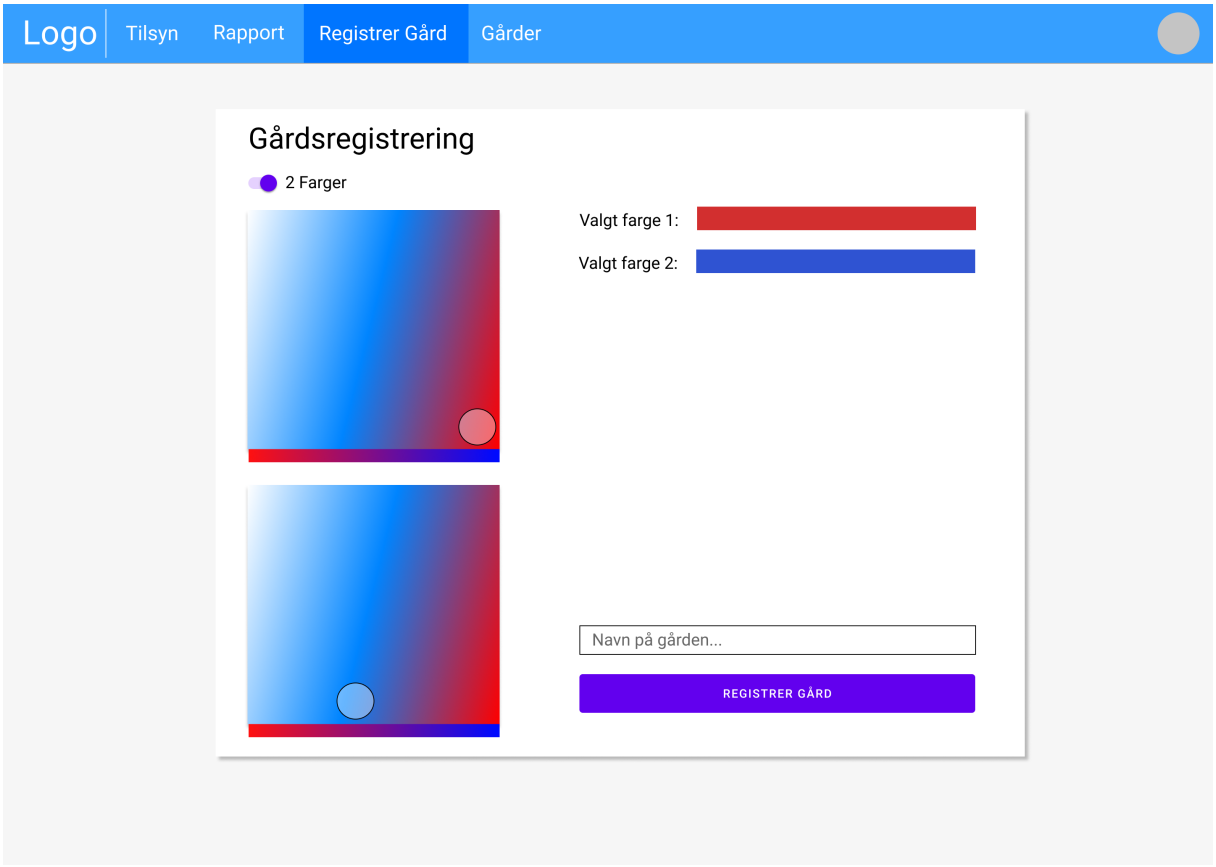
Figur 30 viser siden for glemt passord. Her skal brukeren kunne fylle inn e-postadressen sin hvis de ønsker å motta en e-post som tillater dem å tilbakestille passordet sitt i situasjoner hvor de har glemt det. Skjermen oppnår funksjonaliteten påkrevd i FK4 (Tabell 16). FK5 (Tabell 16) blir ikke utformet i designet ettersom Firebase har sitt eget brukergrensesnitt for å la brukeren tilbakestille passordet når man har klikket på lenken som blir sendt til brukerens e-post.



Figur 30: Skjermbilde fra prototypen - Glemt passord

12.5.3 Registrering av gårder

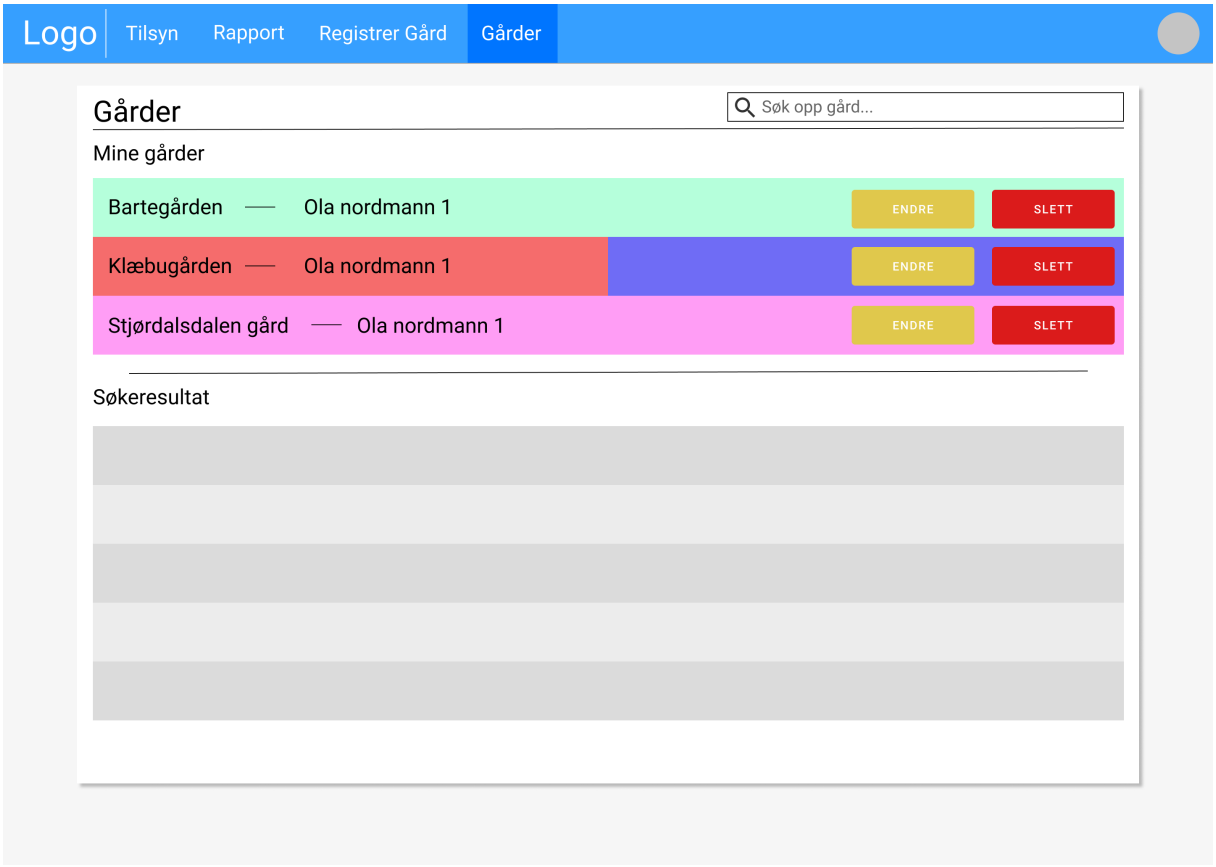
Figur 31 viser siden for å registrere gårder inn i systemet. Her skal fargen(e) på øremerket gården benytter registreres, slik at en bruker av mobilapplikasjonen kan knytte opp øremerkefargene de ser under tilsyn opp mot en gård. Skjermen oppnår funksjonaliteten påkrevd i FK6 (Tabell 17). Hver gård må ha et navn, og enten én eller to farger for øremerket. FK7 (Tabell 17) er et krav som må håndteres teknisk i implementasjonen og blir dermed ikke vist i designet.



Figur 31: Skjermbilde fra prototypen - Registrering av gård

12.5.4 Liste over gårder

Figur 32 viser oversikten over gårder som har blitt registrert i systemet. Brukerens egne gårder vises alltid først, deretter vises en liste over søkeresultater. Søkeresultatlisten starter alltid tom, men hvis brukeren skriver noe i søkefeltet og en match for dette søket finnes i systemet, vises de i søkeresultatlisten. Denne listen starter tom fordi vi vil unngå å vise en liste med alle gårdene i systemet til å begynne med, ettersom det kan ta tid å hente ut alle gårdene i systemet. Dette kunne resultert i dårlig responsivitet og dermed en verre brukeropplevelse. Dette oppnår funksjonalitetene påkrevd i FK8, FK9 (Tabell 18). «Endre»- og «Slett»-knappene på høyre side av hver av brukerens egne gårder tillater brukeren å endre farge eller navn på gården sin, eller å slette gården. Dette oppnår funksjonalitetene påkrevd i FK10 og FK11 (Tabell 19).

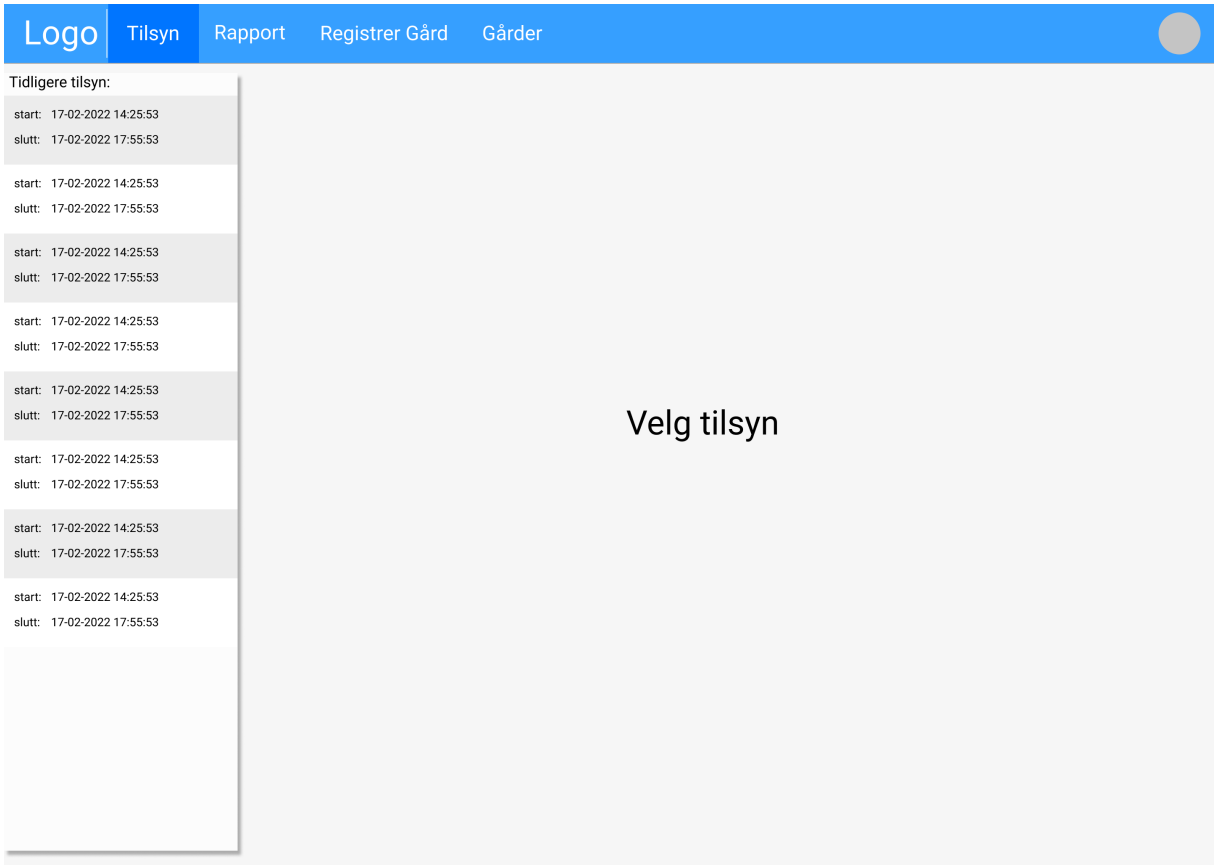


Figur 32: Skjermbilde fra prototypen - Oversikt over gårder i systemet

12.5.5 Liste over tilsyn

Figur 33 viser listen over utførte tilsyn som er relevante for den innloggede brukeren. Med dette menes de tilsynsrundene hvor gårder registrert av den aktive brukeren er relevante i tilsynet. Hver saueflokk registrert i et tilsyn har en liste med gårder som representerer hvilke ulike gårder sauene i saueflokken tilhører. Dermed er det slik at hvis et tilsyn er synlig i tilsynslisten på denne skjermen, betyr det at en saueflokk med sau fra brukerens egen gård/egne gårder ble observert i tilsynet. Tilsyn hvor den aktive brukerens gård(er) ikke er involvert, vil ikke være synlig i listen av tilsyn.

Skjermen vist i Figur 33 oppnår funksjonalitetene påkrevd i FK12 (Tabell 20). Ved å trykke på et spesifikt tilsyn i listen, vil brukeren presenteres med skjermen vist i Figur 34.



Figur 33: Skjermbilde fra prototypen - Liste over tilyn

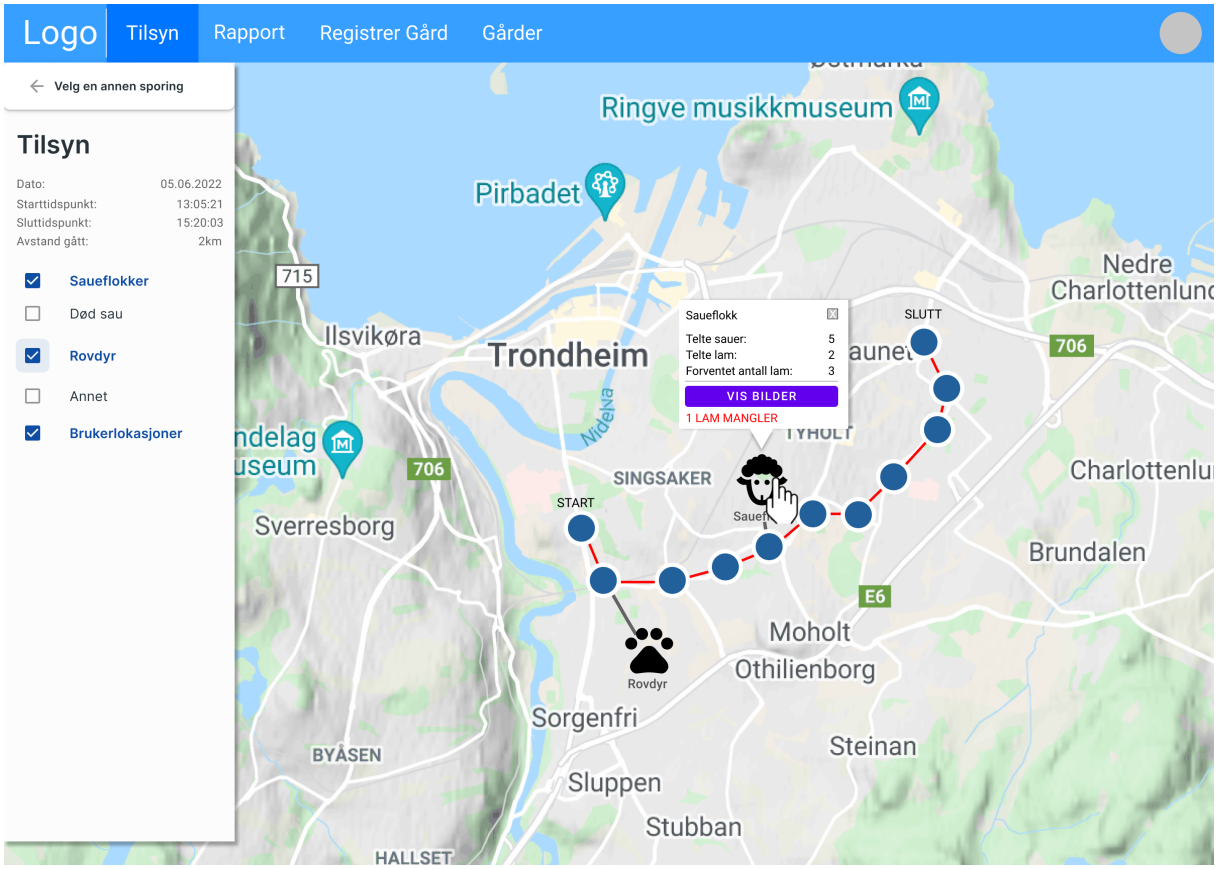
12.5.6 Visning av et tilsyn

Figur 34 viser skjermen for å vise detaljer fra et spesifikt tilsyn. På samme måte som i mobilapplikasjonen vil brukerlokasjoner og alle observasjoner vises som linjer og markører over kartet, og streker vil tegnes fra brukerens lokasjon på observasjonstidspunktet til der observasjonen ble plassert.

Forskjellig data om tilsynet, slik som start- og sluttidspunkt og antall kilometer gått vises i informasjonsboksen til venstre. Brukeren kan også skru av og på visning av de ulike typene markører, slik som rovdyr eller brukerlokasjoner. Dette tillater brukeren å utelate visse typer observasjoner for å lettere få en oversikt over det nøyaktig det de bryr seg om.

Ved å trykke på en markør for en observasjon vil ekstra informasjon om denne observasjonen vises. Informasjonen vil avhenge av hva slags type observasjon det dreier seg om. Om observasjonen har bilder, vil det også være en knapp for å vise bildene fra denne observasjonen. Dette vil åpne en bildeviser, som presenteres i Seksjon 12.5.7.

Skjermen i Figur 34 oppnår funksjonalitetene beskrevet i FK13 og FK14 (Tabell 20).



Figur 34: Skjermbilde fra prototypen - Visning av tilsyn

12.5.7 Visning av bilder fra tilsyn

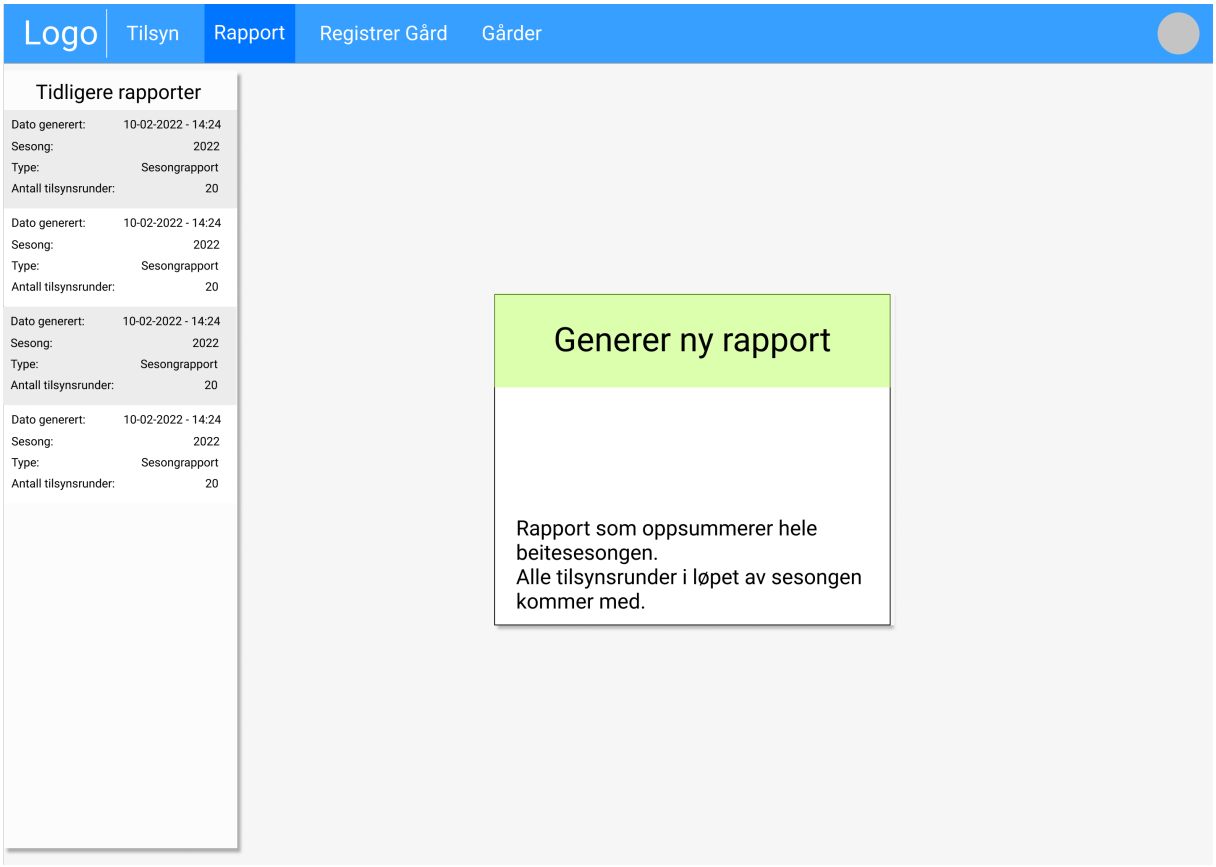
Skjermbildet i Figur 35 demonstrerer bildeviseren som åpnes når man trykker på «Vis bilder»-knappen for en observasjon sett i skjermbildet i Figur 34. Her vises ett bilde fra den gitte observasjonen av gangen, og brukeren kan navigere fram og tilbake gjennom bildene. Bildeviseren oppnår funksjonaliteten beskrevet i FK15 (Tabell 20).

Figur 35: Skjermbilde fra prototypen - Bilderviser

12.5.8 Rapporter

Figur 36 viser en skjerm på webapplikasjonen, med listen over tidligere genererte rapporter på venstre side. Rapportene er tiltenkt som PDF-filer, og tidligere genererte rapporter lagres og blir tilgjengelig slik at man slipper å genere en ny PDF-fil for tidligere sesonger på nytt om man har behov for å se dem. Dette oppnår funksjonalitet beskrevet i FK20 og FK21 (Tabell 22).

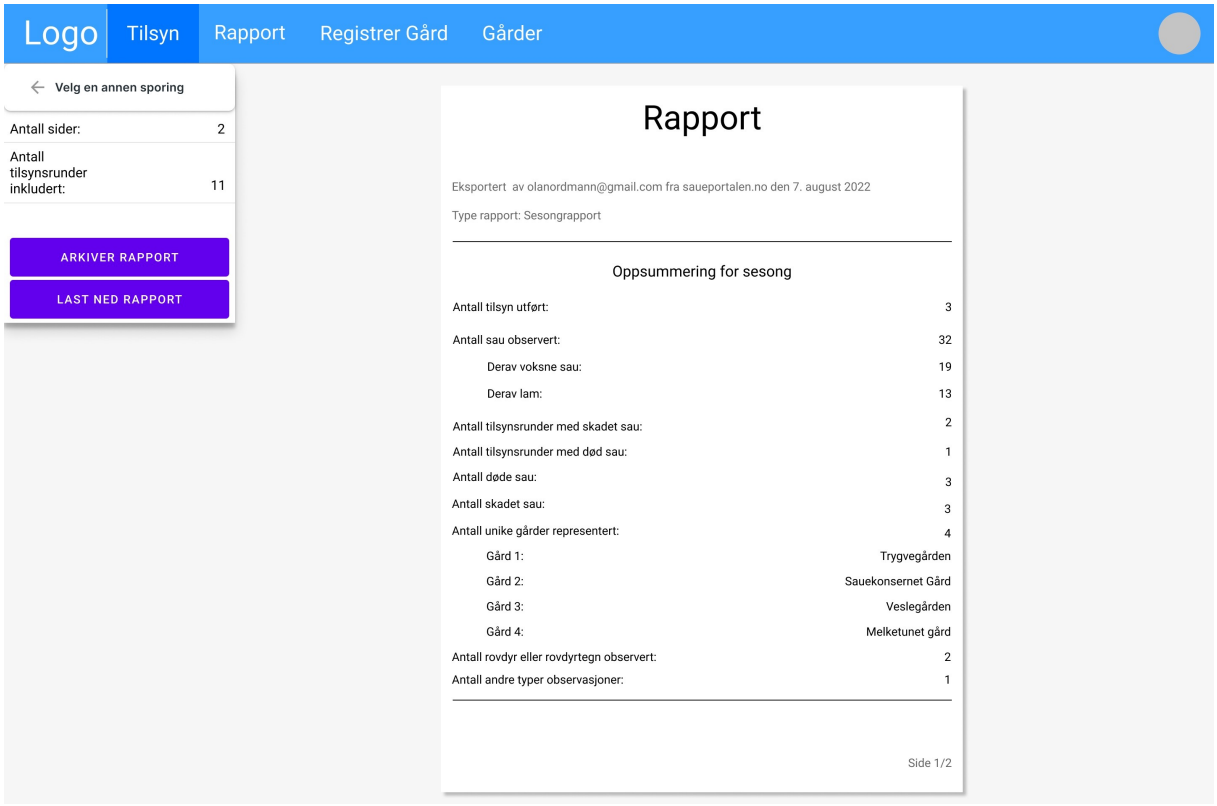
Det er også en knapp for å generere en ny rapport for årets beitesesong. Denne funksjonen skal benyttes ved endt beitesesong og oppnår funksjonaliteten beskrevet i FK17 (Tabell 22).



Figur 36: Skjermbilde fra prototypen - Liste av genererte rapporter

Figur 37 viser siden for å vise fram en generert sesongrapport. En knapp på venstre side tillater brukeren å laste ned rapporten som en PDF-fil og en annen knapp tillater brukeren å arkivere rapporten. Innholdet på første side i rapporten består av en oppsummering av tilsynene utført i løpet av beitesesongen. Side 2 av rapporten, vist i Figur 38, viser en tabell over tilsyn hvor hver rad oppsummerer et tilsyn.

Skjermene vist i Figur 37 og Figur 38 oppnår funksjonalitetene beskrevet i FK18 og FK19 (Tabell 21).



Figur 37: Skjermbilde fra prototypen - Visning av rapport

Rapport

Eksportert av olanordmann@gmail.com fra saueportalen.no den 6. mai 2022

Oversikt over utførte tilsyn

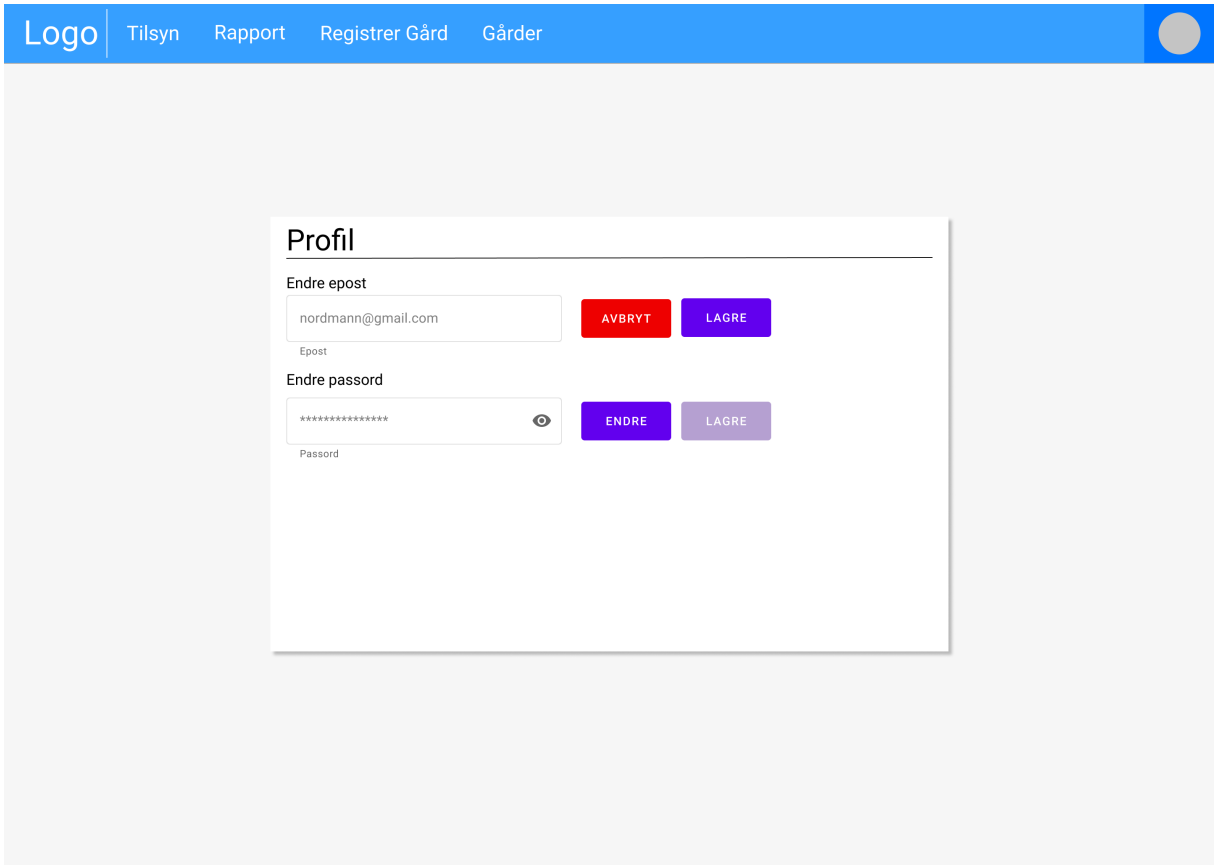
Dato	Beskrivelse av ruta	Antall sau	Antall lam	Døde sau	Skadde sau	Rovdyr
01.06.2022	Til hytta og tilbake	6	2	2	3	2
09.06.2022	Fra Bakketoppen til Geitfjellet	3	2	1	0	0
01.06.2022	Til hytta og tilbake	6	2	2	3	2
09.06.2022	Fra Bakketoppen til Geitfjellet	3	2	1	0	0
01.06.2022	Til hytta og tilbake	6	2	2	3	2
09.06.2022	Fra Bakketoppen til Geitfjellet	3	2	1	0	0
01.06.2022	Til hytta og tilbake	6	2	2	3	2
09.06.2022	Fra Bakketoppen til Geitfjellet	3	2	1	0	0
01.06.2022	Til hytta og tilbake	6	2	2	3	2
09.06.2022	Fra Bakketoppen til Geitfjellet	3	2	1	0	0

Side 2/2

Figur 38: Skjermbilde fra prototypen - Side 2 av rapporten

12.5.9 Profilside

Figur 39 viser profilsiden til brukeren. Her kan brukeren endre e-postadressen assosiert med brukerkontoen eller endre passord. Skjermen oppnår funksjonalitet beskrevet i FK23 (Tabell 23).



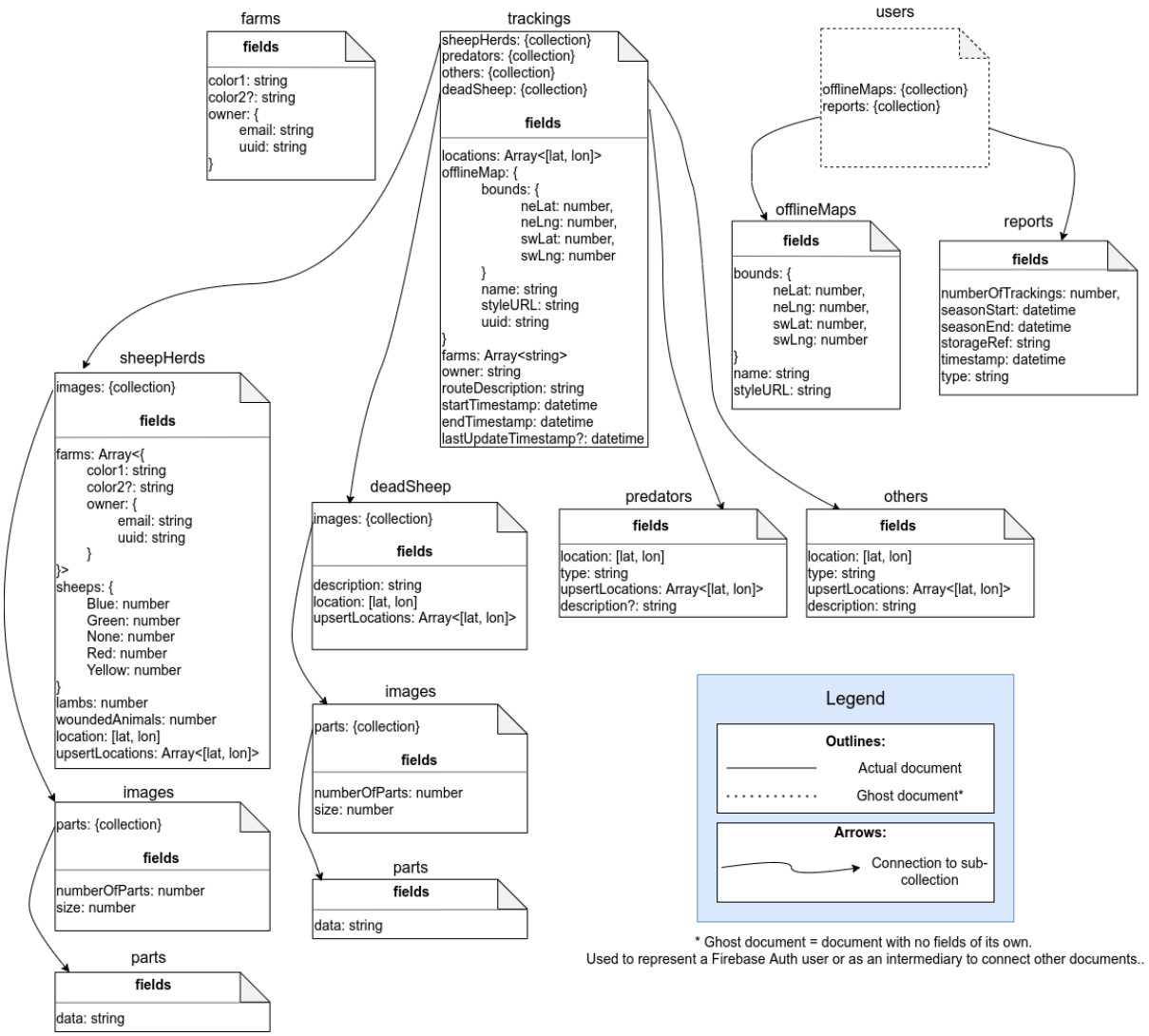
Figur 39: Skjermbilde fra prototypen - Profilside

13 Utvikling

I denne seksjonen vil vi ta for oss utviklingsfasen. Vi vil se på den endelige strukturen av databasen i Firebase Firestore, implementasjonen av sikkerhetskravene, den endelige strukturen til arkitekturen, implementasjonen av webapplikasjonen og hvordan den differensierer seg fra designet, og til slutt se på endringene som ble gjort etter brukertestingen i Seksjon 14.4.

13.1 Endelig Firebase Firestore-struktur

Figur 40 viser hvordan Firebase Firestore-strukturen ser ut etter alle funksjonalle krav er implementert. Ny data har blitt lagt til i forhold til den gamle strukturen vist i Figur 14 for å støtte ny funksjonalitet. I tillegg har noe data blitt gjort om på slik at den lagres på en mer kompakt måte. Flere av endringene forklares i underdelseksjonene under.



Figur 40: Oversikt over ny struktur til backend-data i Firebase Firestore. Figuren er på engelsk for å holde seg til en konkret konvensjon for å omtale datatyper («string», «datetime» og så videre).

13.1.1 Farms-felt i trackings-dokument

Trackings-dokumenter, som representerer en utført tilsynsrunde, har fått et *farms*-felt. Dette er en liste med navnene til gårdene (*farms*) som er representert i tilsynet. Den samme informasjonen eksisterer i sheepHerd-subkolleksjonen, som også har et *farms*-felt (men listen her består av farm-objekter med mange egenskaper, deriblant et *owner* (norsk: *eier*)-felt). Grunnen til at vi dupliserer denne informasjonen er for å redusere antall lesinger. Når tilsyn skal hentes ut på webapplikasjonen, ønsker vi å kun vise brukeren de tilsynene der den innloggede brukerens gårder er representert. Man slipper da å lese gjennom alle *trackings*-dokumentene sine *sheepHerds*-subkolleksjoner for å sjekke om den innloggede brukerens gårder er i listen. Istedenfor leser man kun hvert trackings-dokument, og trenger ikke gå dypere ned i kolleksjons-«treet». Dette sparer mange leseoperasjoner, som både forbedrer responstiden og gjør at vi bruker opp mindre av Firebase-ressursene vi har tilgjengelig.

13.1.2 Oppstykking av bildedata

Et bilde brukeren sender inn lagres i et felt som base64-streng. Felt i Firebase Firestore kan maksimalt være 1 MB i størrelse[19]. Dette er problematisk ved store bildestørrelser, ettersom det er vanlig at bilder fra mobilkameraer overgår denne filstørrelsen. Det er vanskelig å forutse nøyaktig hvor stort et bilde fra et mobilkamera kommer til å være, ettersom dette vil variere fra bilde til bilde, men ved uformell testing på en iPhone 12 og en Samung Galaxy S10e ligger bildestørrelsene på rundt 3-4 megabyte. Dette betyr at bildene fra disse mobilene allerede er 3-4 ganger større enn Firebase kan lagre i et enkelt felt. Ved introduksjonen av mer avanserte mobilkameraer i nye modeller og innstillinger for høyoppløste bilder er det til og med tenkelig at bildestørrelsen vil kunne overgå 10 MB. Dette gir opphav til et behov om å støtte bildestørrelser som overgår 1 MB. Hvert bilde er derfor et *images*-dokument med *parts*-subkolleksjon. Hvert *parts*-dokument inneholder den faktiske bildedataen som en base64-streng i data-feltet sitt. Om et enkelt bilde overgår 1 MB i størrelse vil bildedataen derfor være spredt over flere dokumenter som utgjør *parts*-subkolleksjonen til images-dokumentet. Alle parts-dokumentene sine data-felt leses, og innholdet av alle disse konkateneres til én string på frontend, som da utgjør hele bildet. Ved å splitte bildedataen i flere dokumenter får vi til å støtte lagring av bilder som overgår 1 MB i størrelse.

En annen begrensning Firebase setter, og som ikke lar seg unngås, er at en spørring til Firestore-databasen ikke kan returnere en respons som overgår 10 MB i størrelse[20]. Dette betyr at dataen om et tilsyn (bilder inkludert) ikke kan overgå 10 MB, fordi hvis det er tilfellet vil man ikke kunne hente ut all dataen på én gang. For å unngå å nærme oss denne grensen utfører vi også komprimering på bildefilene brukeren laster opp før de blir lagret i databasen.

13.1.3 Endring på struktur av lagring av sauer og lam

I den gamle Firebase Firestore-strukturen, vist i Figur 14, ble sau og lam lagret som egne subkolleksjoner i saueflokken. Dermed var hver sau og hvert lam sitt eget dokument. Lam (*lambs*)-dokumentet hadde ingen felt, og det var kun antallet dokumenter som var av interesse som data (ettersom dette er antallet lam i flokken). Dette kunne lett optimaliseres ved å heller inkludere det som et tall-felt i selve sheepHerds-dokumentet for å redusere antallet lese-operasjoner mot databasen. Dette er implementert i form av *lambs*-feltet på sheepHerds-dokumentet i den nye Firebase-strukturen.

Tellingen av antall sau (og forventede antall lam via slipsfargene) er endret til et felt (*sheeps*-feltet) i sheepHerd-dokumentet istedenfor en subkolleksjon, så leseoperasjoner spares her også. Dette er implementert i form av tellinger av forekomster av hver slipsfarge. Hvor mange forventede lam dette tilsvarer basert på slipsfargene gjøres på frontend (for eksempel vil to forekomster av blått slips bety to sauer og to forventede lam, ettersom et blått slips betyr at søyen har ett (1) lam).

13.2 Implementasjon av sikkerhetskrav

De ikke-funksjonelle sikkerhetskravene presentert i Seksjon 10.3.4.4 dreier seg om å begrense lese- og skrivetilgang til autentiserte brukere med korrekt lese- og skrivetilgang for den gitte dataen. Dette ble implementert gjennom sikkerhetsregler i Firebase Firestore. Et eksempel på hvordan disse sikkerhetsreglene kan se ut ble presentert i Seksjon 11.1.1.2. De konkrete reglene vi har implementert er som følger:

- Gårder (farms)
  - Alle gårder

[19] Firebase Firestore - Quotas and Limits - Collections, documents, and fields https://cloud.google.com/firestore/quotas#collections_documents_and_fields

[20] Firebase Firestore - Quotas and Limits - Writes and Transactions https://cloud.google.com/firestore/quotas#writes_and_transactions

        * **Lesing:** hvis brukeren er pålogget
    - Spesifikk gård
        * **Opprette ny:**
            · Eieren av den opprettede gården må samsvare med den autentiserte og innloggede brukeren
            · Alle datafelt i dokumentet må samsvare med forventede feltnavn og forventede verdier
        * **Sletting:** Brukeren som utfører slettingen må være autentisert, innlogget og være eieren av det gitte gård-objektet
        * **Oppdatering:**
            · Eieren av den oppdaterte gården må samsvare med den autentiserte og innloggede brukeren
            · Alle datafelt i dokumentet må samsvare med forventede feltnavn og forventede verdier
- Tilsyn (trackings)
    - Alle tilsyn
        * **Lesing:** hvis brukeren er pålogget
    - Spesifikt tilsyn[21]
        * **Opprette ny:**
            · Eieren av det opprettede tilsynet må samsvare med den autentiserte og innloggede brukeren
            · Alle datafelt i dokumentet må samsvare med forventede feltnavn og forventede verdier
        * **Sletting:** Brukeren som utfører slettingen må være autentisert, innlogget og være eieren av det gitte tilsyns-objektet
        * **Oppdatering:**
            · Eieren av det oppdaterte tilsynet må samsvare med den autentiserte og innloggede brukeren
            · Alle datafelt i dokumentet må samsvare med forventede feltnavn og forventede verdier
- Rapporter (reports)
    - Spesifikk rapport
        * **Lesing:** rapporten må tilhøre den innloggede brukeren
        * **Sletting:** rapporten må tilhøre den innloggede brukeren
        * **Opprette ny**: Alle datafelt i dokumentet må samsvare med forventede feltnavn og forventede verdier

Å sjekke at datafelt samsvarer med forventede feltnavn- og verdier sørger for å sikre at dataen er gyldig.

At alle tilsyn kan leses av hvilken som helst bruker kan muligens framstå som et personvernsproblem. Grunnen til at alle tilsyn kan leses av alle brukere er fordi systemet må gjøre spørringer for å hente ut relevante tilsyn til den innloggede brukeren på webapplikasjonen. Dette innebærer å lese gjennom alle tilsynene og finne ut hvilke av dem som inneholder den innloggede brukerens gårder. Det optimale ville vært å kun tillate lesing av kolleksjoner som har spesifikke feltverdier i kolleksjonen («Hvilke tilsyn inneholder saueflokker hvor min(e) gård(er) er representert? Returner kun disse»), men slike avanserte spørringer er ikke støttet av Firebase Firestore. Derfor må alle tilsyn hentes ut, for å så filtrere fram de relevante spørringene på klientsiden.

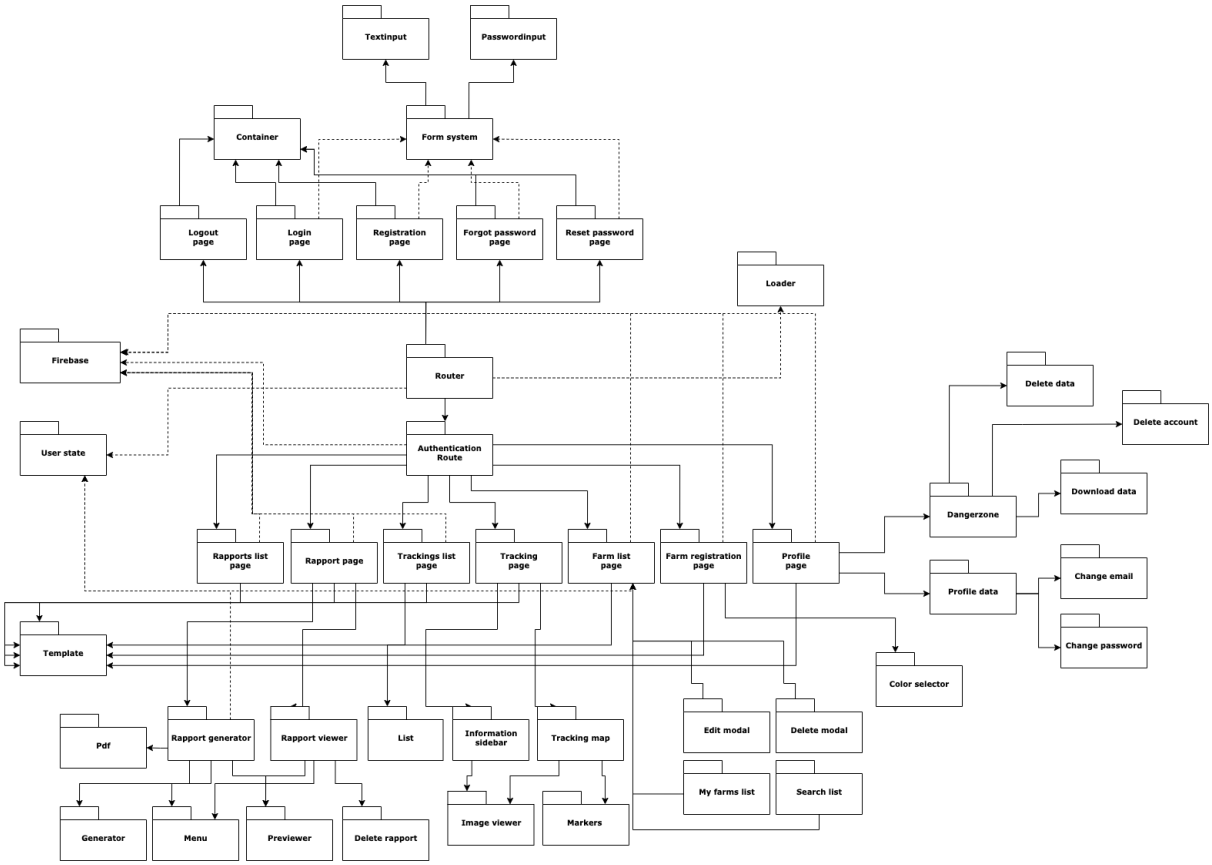
Ettersom alle tilsyn kan hentes ut av hvilken som helst innlogget bruker, har vi gjort grep for å sikre at personopplysninger ikke avsløres gjennom tilsynene lagret i systemet. Hvert tilsyn kobles til en bruker gjennom et «owner» (norsk: eier)-felt, som har verdi til brukeridentifikatoren til brukeren som gjorde tilsynet. Selve brukeridentifikatoren er kun en ID som er generert av Firebase Auth, og er av ubetydelig verdi for en hacker.

[21] Kun relevant for mobilapplikasjonen, ettersom tilsyn er read-only på webapplikasjonen

13.3 Endelig arkitektur

Den endelige arkitekturen har forandret seg ganske lite fra den originale framstillingen vist i Seksjon 11.4. Ettersom de funksjonelle kravene ikke har forandret seg siden planleggingen så er logical viewet uendret fra Seksjon 11.4.1. Det samme kan sies om physical viewet som er uendret fra Seksjon 11.4.4 ettersom backend-systemet fortsatt bruker de samme Firebase-tjenestene, og process viewet som også er uendret fra Seksjon 11.4.2, ettersom ingen endringer i de funksjonelle kravene betyr at process staten ikke har trengt å bli endret.

Development viewet derimot har hatt en del endringer, hovedsakelig i at det har blitt mer detaljert og utvidet med flere komponenter. Dette var egentlig helt som forventet ettersom det er umulig å lage et ferdig development view før utviklingen starter, men nå som det er ferdig kan man lage et mye mer detaljert og riktig bilde av hvordan programvaren er bygd opp. Det endelige development viewet kan bli sett i Figur 41 som viser den endelige strukturen til webapplikasjonen. Det skal nevnes at viewet ikke dekker alle komponentene i applikasjonen ettersom det ville blitt unødvendig kludret og komplekst, men det viser de viktigste strukturene og tjenestene som applikasjonen er bygget opp av.



Figur 41: Endelig development view av systemet

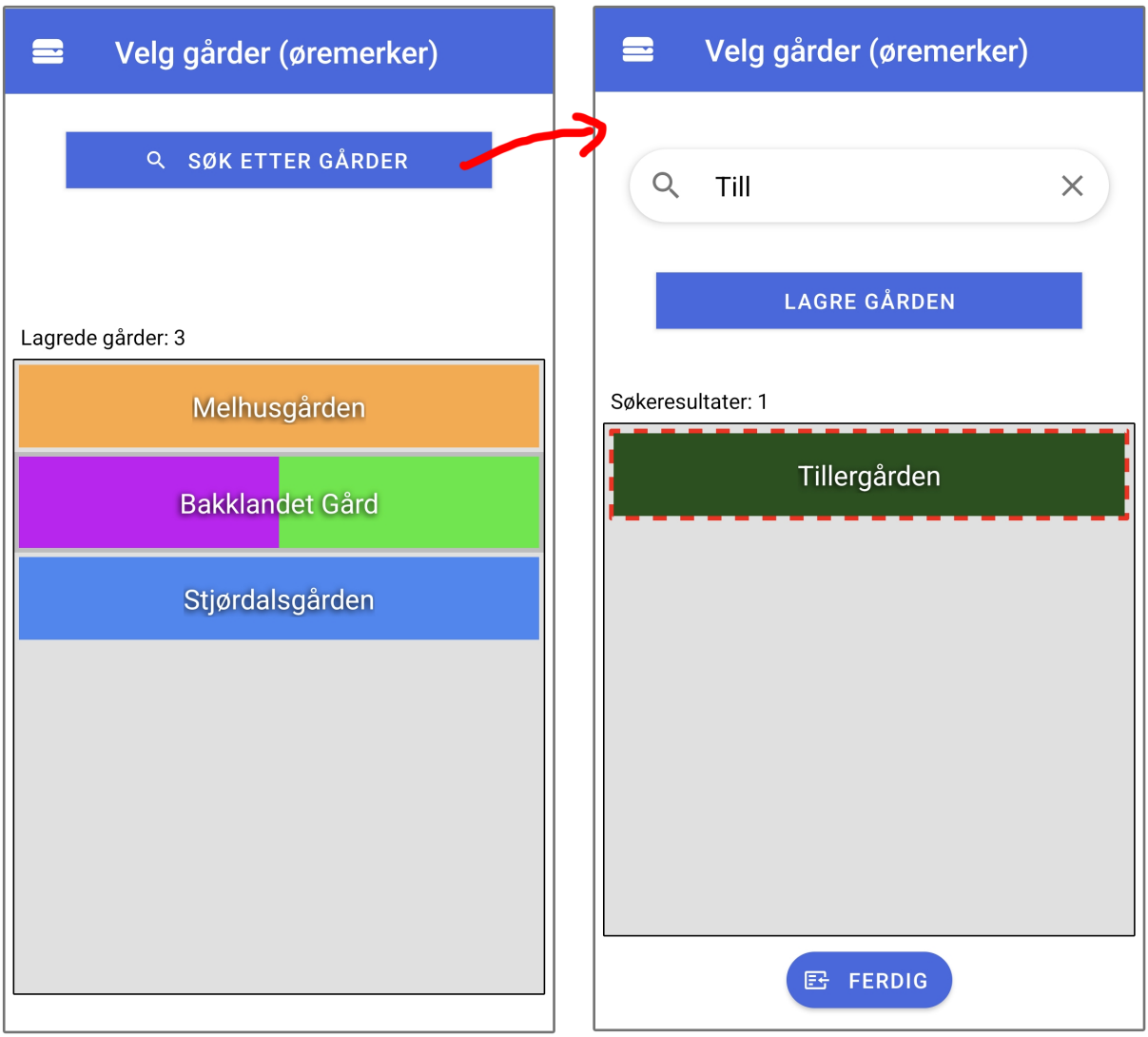
13.4 Presentasjon av implementasjon og differanse fra design

Her gjennomgår vi grensesnittet til systemet vi har utviklet og hvordan det eventuelt skiller seg fra prototypen presentert i Seksjon 12. Vi går gjennom mobilapplikasjonen og webapplikasjonen hver for seg.

13.4.1 Mobilapplikasjon

Her vises de konkrete endringene på grensesnittet til mobilapplikasjonen, som originalt ble prototypet i Figma og vises i Seksjon 12.4. Implementasjonen av disse følger prototypen i svært stor grad og det er ingen store forskjeller mellom prototype og implementasjon.

13.4.1.1 Velge lokalt lagrede gårder



Figur 42: Skjermbilder fra applikasjonen - Velge lokalt lagrede gårder

Skjermen til venstre i Figur 42 viser oversikten over hvilke gårder brukeren har lagret lokalt. Ved å trykke på «søk etter gårder»-knappen øverst på skjermen, tas brukerne til en skjerm hvor de kan søke etter gårder som har blitt registrert inn via webapplikasjonen (sett til høyre i Figur 42). Ved å trykke på en gård i søkeresultatlisten og trykke på «Lagre gården»-knappen vil gården så vises i listen over lagrede gårder.

Skjermene er implementert etter designet i Figur 22 og oppfyller FK27 og FK28 (Tabell 37). FK29 er

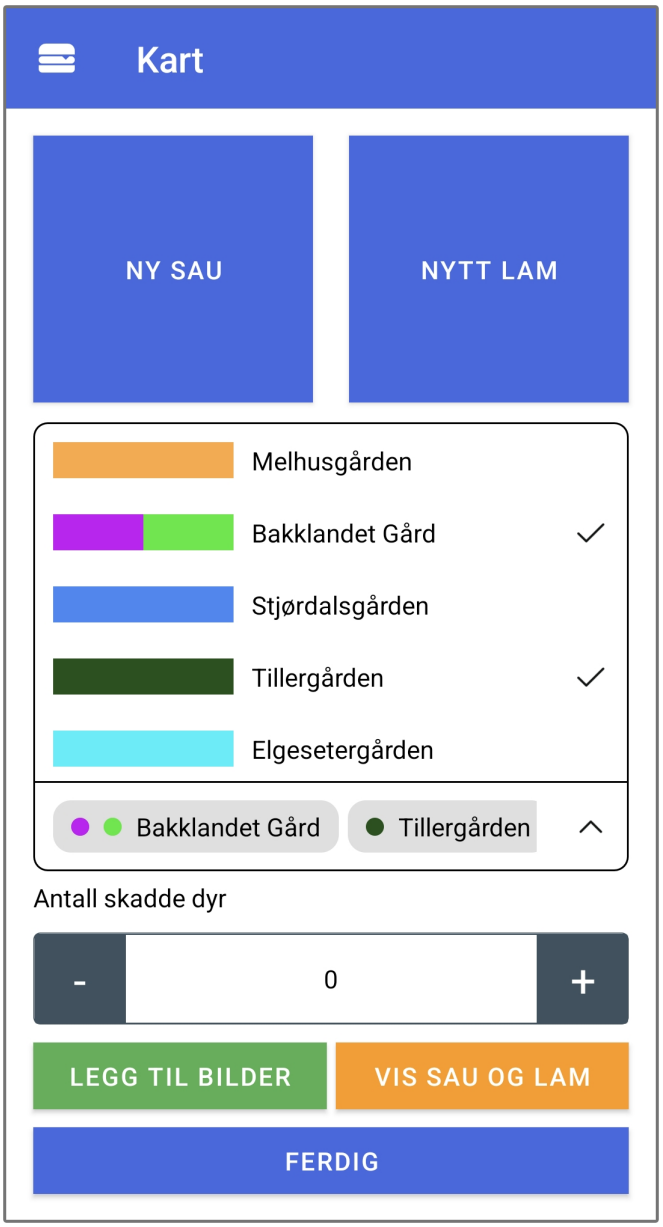
også implementert, men vises ikke i bildet; slette-knappen er synlig når en lokalt lagret gård er valgt ved å trykke på den.

13.4.1.2 Skjerm for registrering av død sau

Figur 43: Skjermbilde fra applikasjonen - Registrere død sau

Skjermen er implementert etter designet i Figur 26 og oppfyller FK30 (Tabell 38). FK31 er også implementert, men vises ikke i bildet (slette-knappen kommer opp når man redigerer en eksisterende observasjon av død sau).

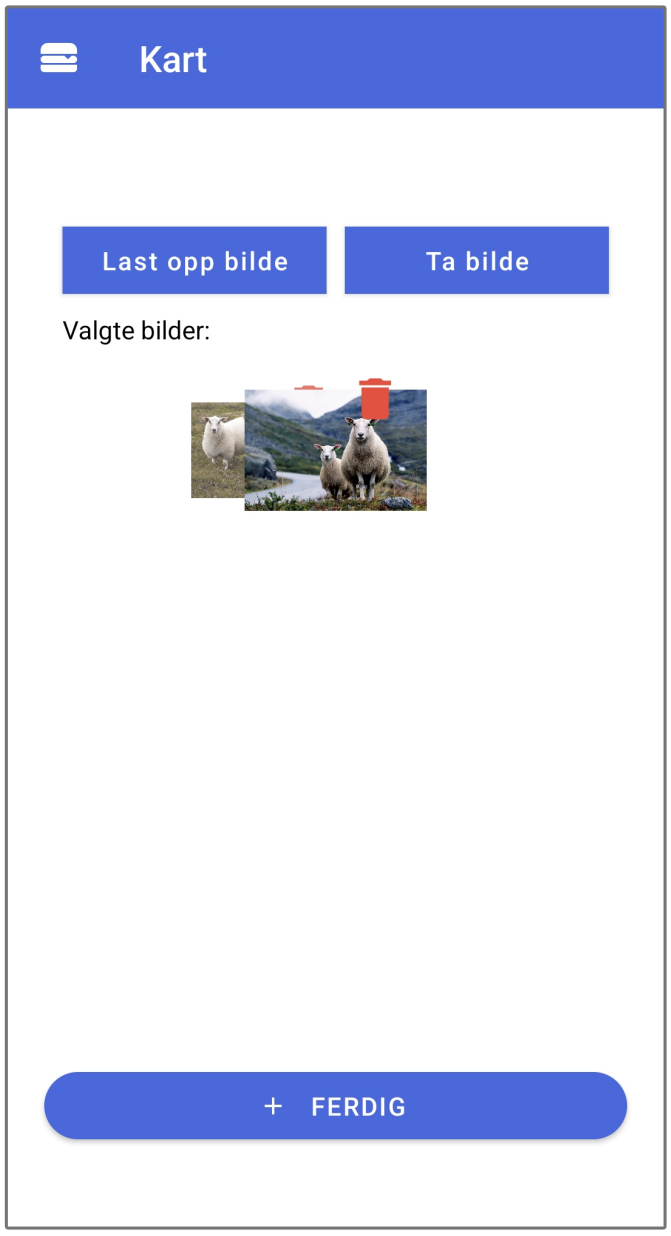
13.4.1.3 Oppdatert skjerm for registrering av saueflokk (registrere skadet sau)



Figur 44: Skjermbilde fra applikasjonen - Ny skjerm for registrering av saueflokk

Skjermen er implementert etter designet i Figur 24 og oppfyller FK32 (Tabell 39). Her kan brukeren trykke på pluss- og minusknappene for å henholdsvis inkrementere og dekrementere tellingen over skadde dyr.

13.4.1.4 Registrering av bilder av saueflokk

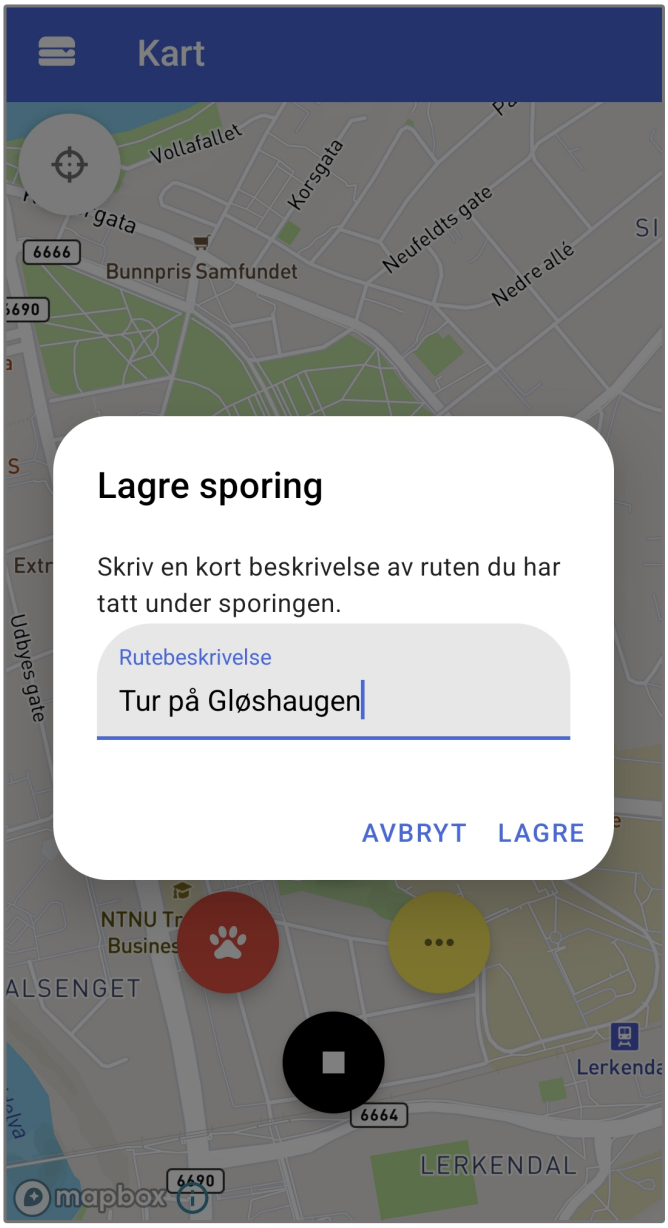


Figur 45: Skjermbilde fra applikasjonen - Registrering av bilder av saueflokk

Ved å trykke på «Legg til bilder»-knappen sett i Figur 44 kommer brukeren inn på skjermen vist i Figur 45. Her kan brukeren legge til bilder av saueflokken om ønskelig.

Skjermen er implementert etter designet i Figur 25 og oppfyller FK33 (Tabell 39).

13.4.1.5 Registrering av beskrivelse av tilsynsrute



Figur 46: Skjermbilde fra applikasjonen - Registrering av beskrivelse av tilsynsrute

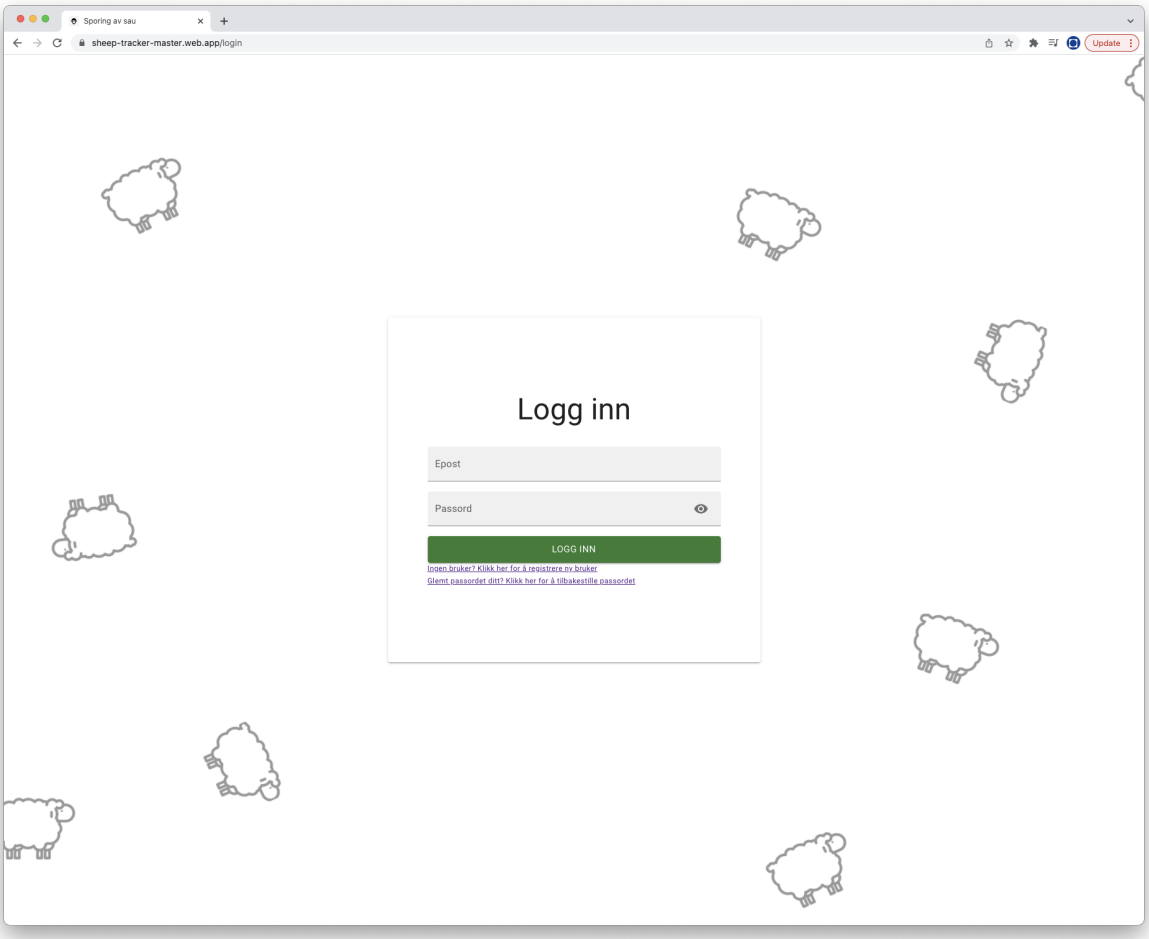
Skjermen er implementert etter designet i Figur 27 og oppfyller FK38 (Tabell 41).

13.4.2 Webapplikasjon

De ulike sidene fra webapplikasjonen presenteres her, og i tillegg blir differansen mellom designet og implementasjonen vist og forklart. Webapplikasjonen, som ble laget i React, er implementasjonen av prototypen som ble presentert i Seksjon 12.5.

13.4.2.1 Innlogging

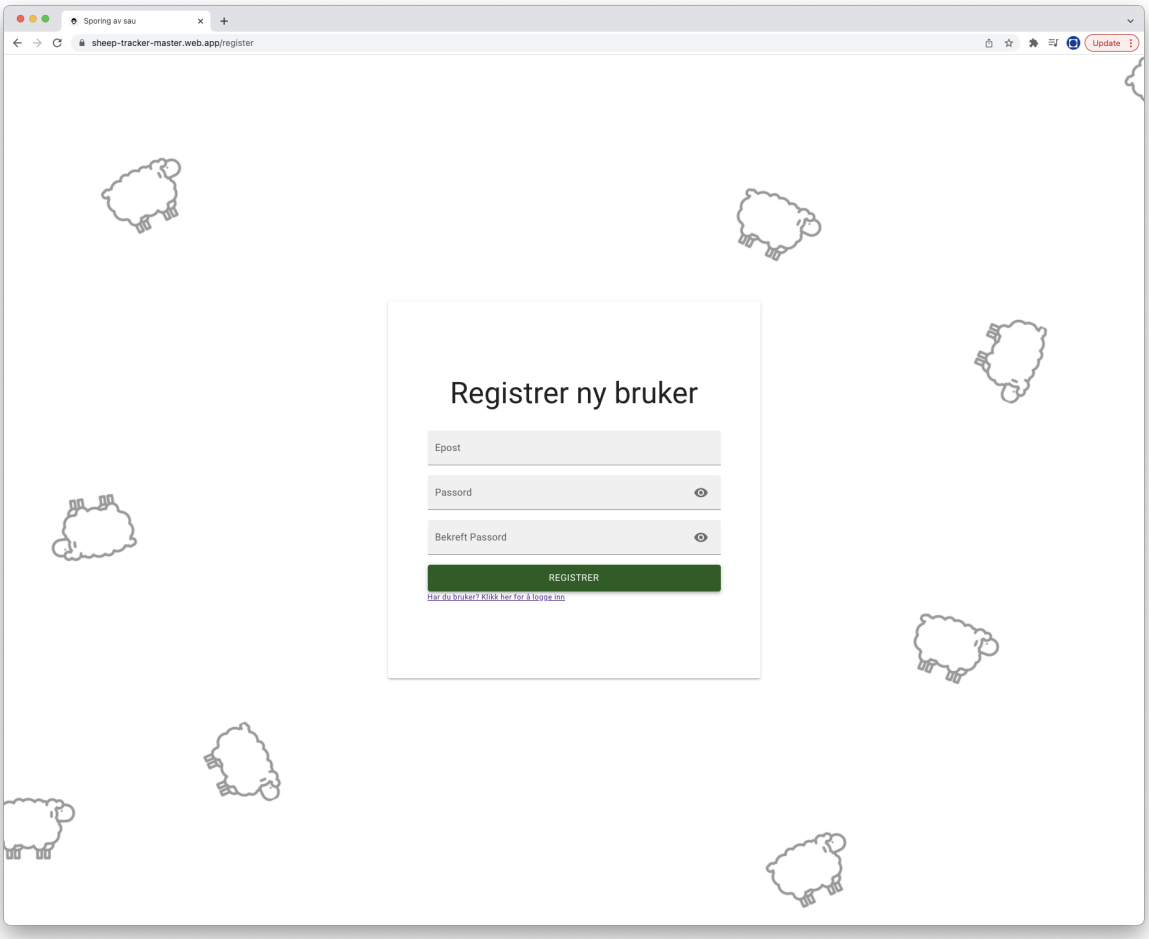
Figur 47 viser innloggingssiden som ble implementert etter designet vist i Seksjon 12.5.1, og oppfyller som sagt FK1 (Tabell 13). Det er noen mindre forskjeller mellom designet og implementasjonen, da hovedsakelig mellom fargen på «logg inn»-knappen og boksen som skiller innholdet og bakgrunnen på siden. Dette er i tillegg til at hjelpeteksten for tekstinput-feltet er flyttet fra under til inni feltet. Dette stammer hovedsakelig fra at designbiblioteket vi brukte hadde disse detaljene som sin default implementasjon og å endre på dette ikke ville gitt noe nytte for tiden det ville tatt.



Figur 47: Skjermbilde fra implementasjon - Innlogging

13.4.2.2 Registrering

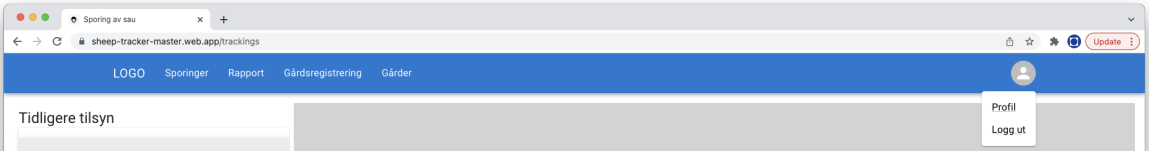
Figur 48 viser registreringssiden som ble implementert etter designet vist i Seksjon 12.5.1, og oppfyller som sagt FK2 (Tabell 14). Denne siden har de samme forskjellene mellom designet og implementasjonen som innloggingssiden på grunn av de samme årsakene.



Figur 48: Skjermbilde fra implementasjon - Brukerregistrering

13.4.2.3 Utlogging

Figur 49 viser implementasjonen av utloggingsfunksjonalitet, og oppfyller FK3 (Tabell 15). Det er som vist ingen egen side for utlogging, men heller en enkel knapp. Dette ble aldri lagt til i designet ettersom vi ikke tenkte på å legge den til med tanke på hvor simpelt det var.

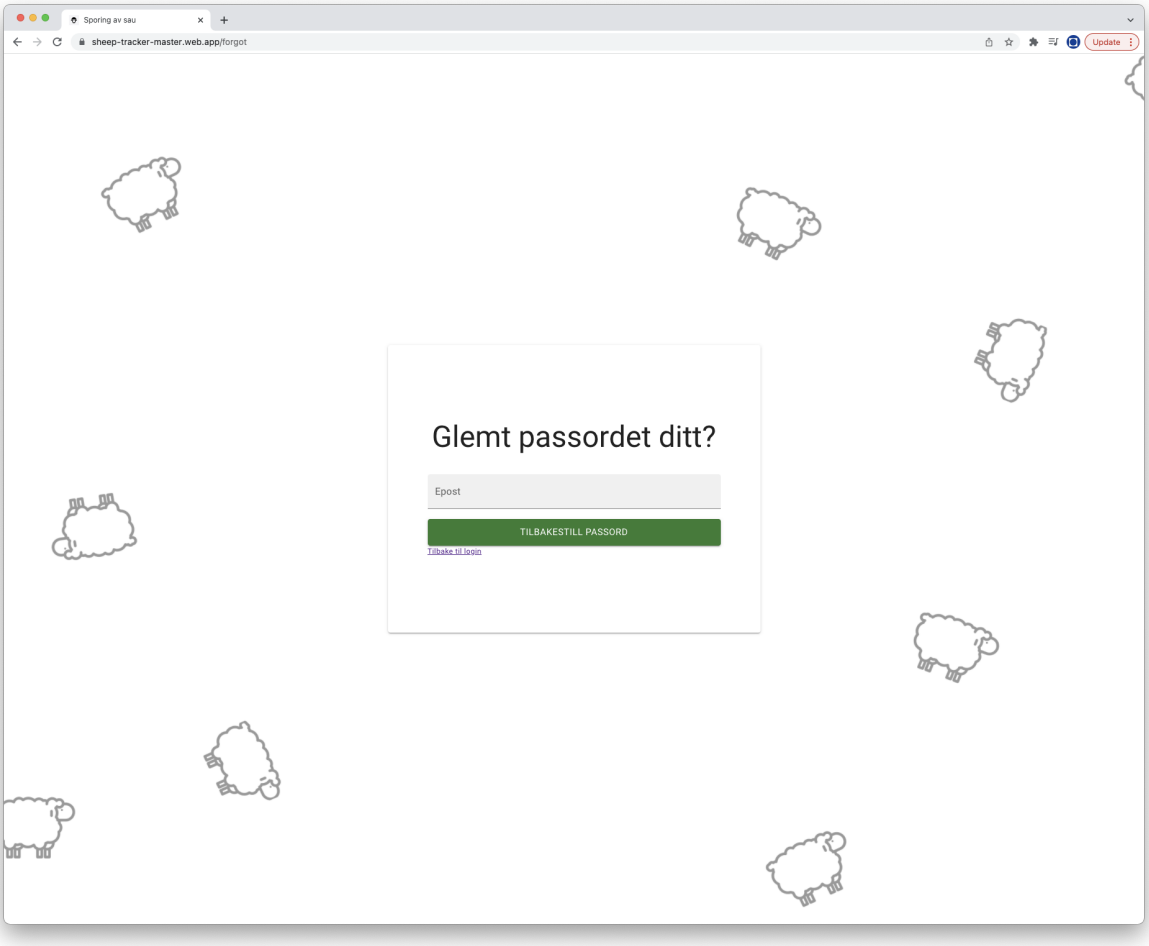


Figur 49: Skjermbilde fra implementasjon - Utlogging

13.4.2.4 Glemt passord

Figur 50 viser «glemt passord»-siden som ble implementert etter designet vist i Seksjon 12.5.2, og oppfyller FK4 (Tabell 16). Denne siden har de samme forskjellene mellom designet og implementasjonen som både

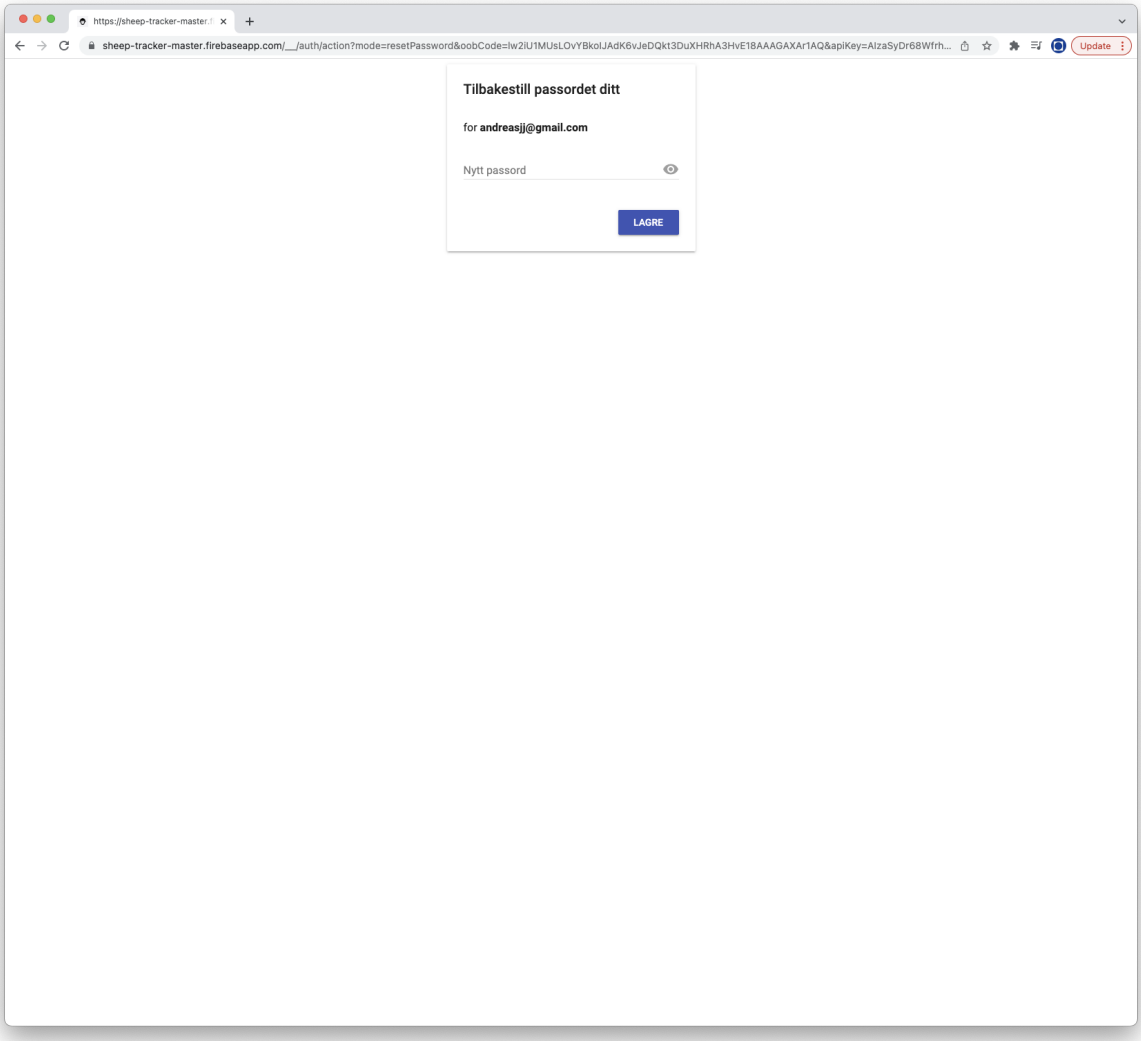
innlogging- og registreringssiden for de samme årsakene.



Figur 50: Skjermbilde fra implementasjon - Glemt passord

13.4.2.5 Tilbakestill passord

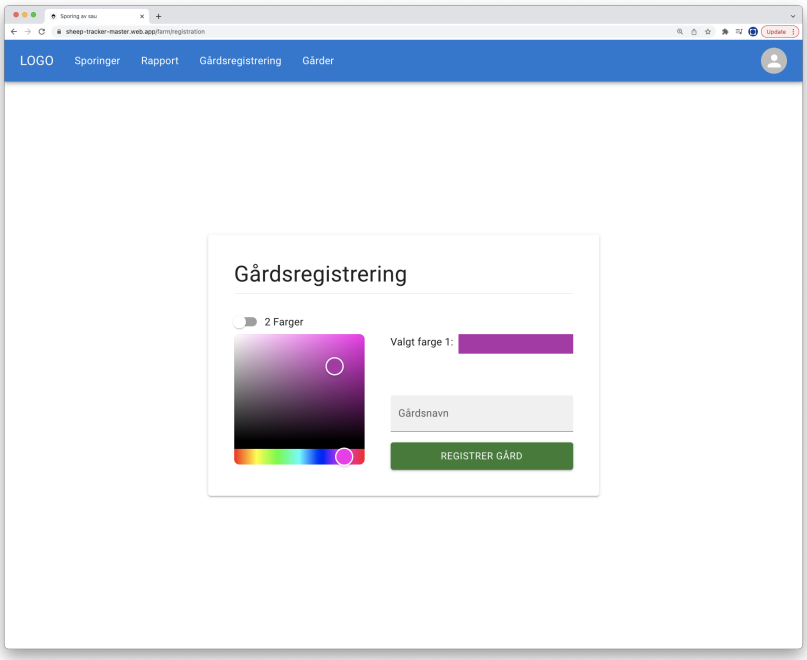
«Tilbakestill passord»-siden ble ikke implementert av oss som nevnt i Seksjon 12.5.2, men isteden ble Firebase sin egen UI-implementasjon for tilbakestilling av passord brukt. Dette oppfyller FK5 (Tabell 16).



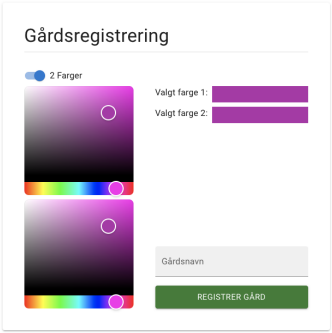
Figur 51: Skjermbilde fra implementasjon - Tilbakestill passord

13.4.2.6 Registrering av gårder

Figur 52 viser siden for gårdsregistrering som ble implementert etter designet vist i Seksjon 12.5.3. Forskjellene mellom designet og implementasjonen er relativt minimale utenom noen farge- og størrelses-endringer som kommer av at det blir brukt ferdiglagde komponenter fra design- og fargevelger-bibliotek. I tillegg viser Figur 52 en versjon der bare én farge er valgt, mens Figur 53 viser der to farger er valgt. Dette er mer i tråd med designet, men dette kommer som sagt i Seksjon 12.5.3 av at det ble sett på som unødvendig tidsbruk å designe begge versjonene når versjonen med én farge praktisk talt bare ville være én mindre fargevelger og noe endring av posisjon på elementer. Alt dette oppfyller FK6 (Tabell 17), i tillegg har implementasjonen oppfylt FK7 (Tabell 17) gjennom tekniske restriksjoner i backend-systemet som ikke tillater to gårder å ha samme navn.



Figur 52: Skjermbilde fra implementasjon - Registrering av gård



Figur 53: Skjermbilde fra implementasjon - Registrering med to farger

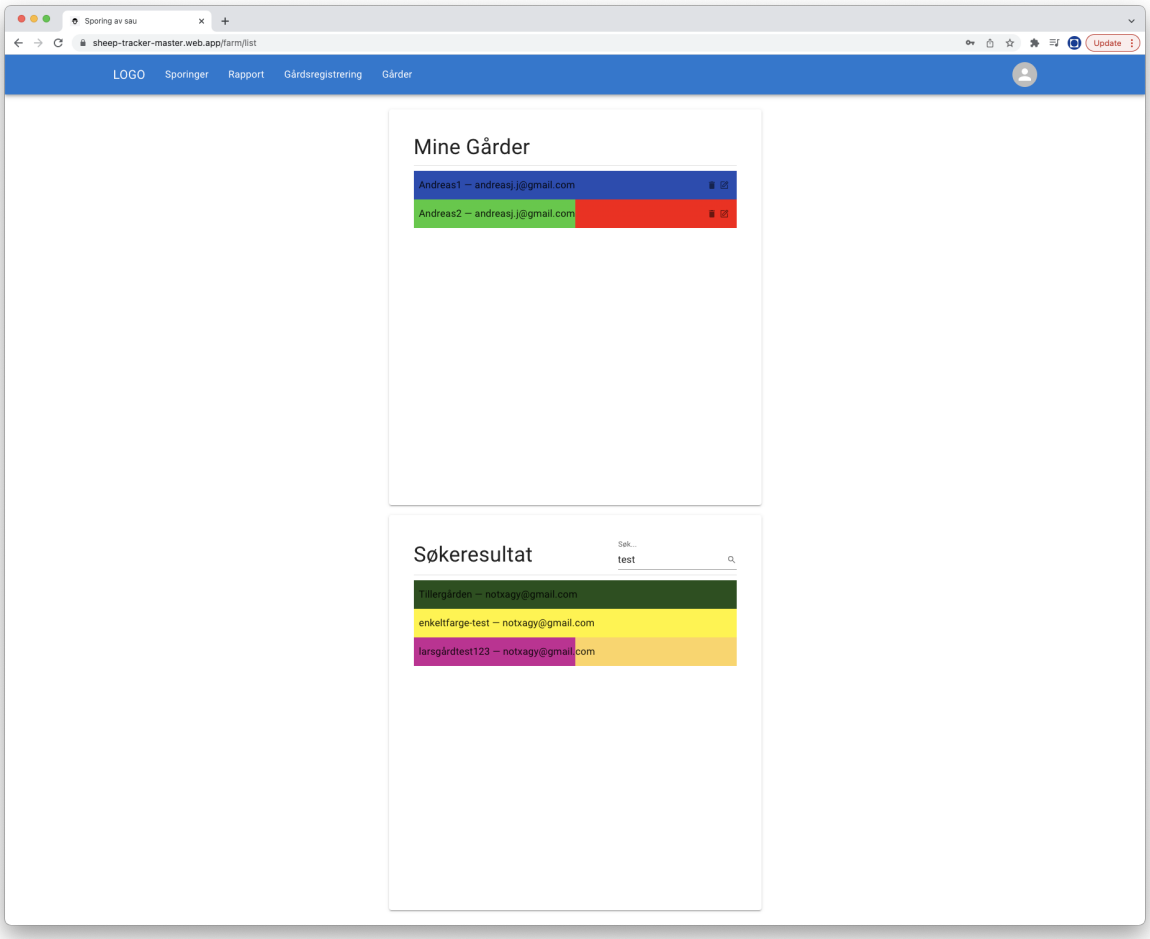
13.4.2.7 Liste over gårder

Figur 54 viser siden med liste over gårder som ble implementert etter designet vist i Seksjon 12.5.4, og oppfyller som sagt FK8 og FK9 (Tabell 18). På denne siden divergerte implementasjonen fra designet noe mer. Utenom de mindre endringene som tekst- og fargeendringer er det hovedsakelig to drastiske forskjeller.

For det første ble «Mine gårder» og «søkeresultat» skilt inn i to mer distinkte bokser istedenfor å skilles med en horisontal linje, i tillegg til at søkebaren ble flyttet fra å ligge ved «Gårder»-teksten til «Søkeresultat»-teksten. Dette ble gjort for å bedre følge gestaltprinsipper med å gruppere lignende elementer og skille forskjellige elementer [98].

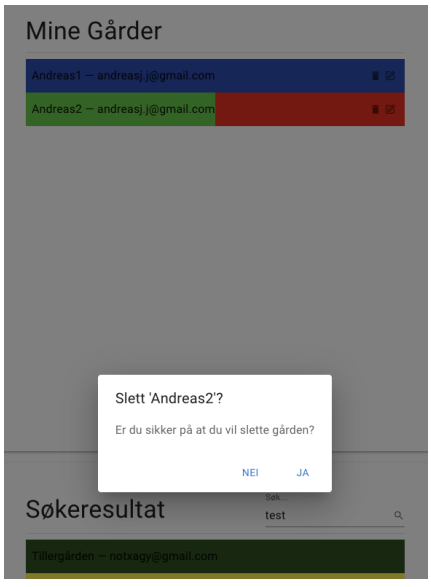
For det andre ble endre- og sletteknappene erstattet med mindre ikoner både fordi ikonene følger Recognition fra Jacob Nielsens 10 brukbarhets-heuristikker [99] og Consistency fra Don Normans designprinsipper [100] bedre ved at ikonene er kjente ikoner for endring og sletting fra andre kjente applikasjoner som brukerne har brukt eller bruker. En annen grunn er at de store endre- og sletteknappene ikke passet/så bra ut med listebiblioteket som ble brukt.

I tillegg til de nevnte endringene ble modaler for endring og sletting av gård også implementert som vist i Figur 55 og Figur 56, som oppfyller FK10 og FK11 (Tabell 19). Disse ble implementert uten å følge et design ettersom vi rett og slett glemte å lage et design for dem, og det ville føre til unødvendig tidsbruk uten noen særlige gevinster hvis vi man lagde et design under implementasjonsfasen.



Figur 54: Skjermbilde fra implementasjon - Liste over gårder

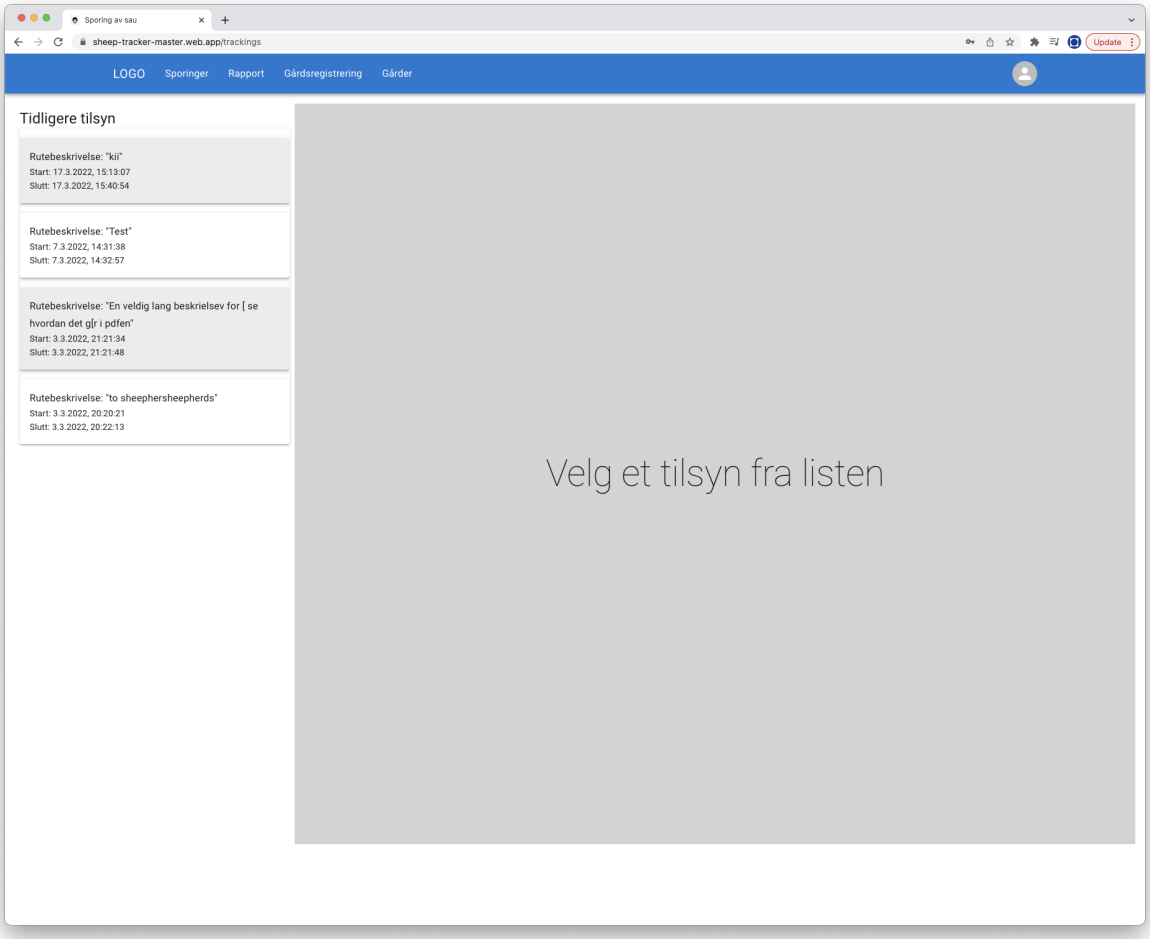
Figur 55: Skjermbilde fra implementasjon - Modal for endring av gård



Figur 56: Skjermbilde fra implementasjon - Modal for sletting av gård

13.4.2.8 Liste over tilsyn

Figur 57 viser siden med listen over tilsyn som ble implementert etter designet vist i Seksjon 12.5.5, og oppfyller FK12 (Tabell 20). Forskjellene mellom designet og implementasjonen er relativt minimale og handler i hovedsak om noen farge- og størrelsesendringer, men det generelle designet er fulgt under implementasjonen.



Figur 57: Skjermbilde fra implementasjon - Liste over tilsyn

13.4.2.9 Tilsyn

Figur 58 viser tilsynssiden som ble implementert etter designet vist i Seksjon 12.5.6, og oppfyller FK13 og FK14 (Tabell 20). Her er designet og implementasjonen delt i to hoveddeler, kartet på høyre side og informasjons- og menybaren på venstre side. Kartet er implementert relativt nøyaktig etter designet utenom plasseringen av «vis bilder»-knappen og noe av teksten som er reformulert. Informasjons- og menybaren på venstre side derimot divergerer fra designet ganske mye. Det er hovedsakelig tre ting som skiller implementasjonen og designet av informasjons- og menybaren.

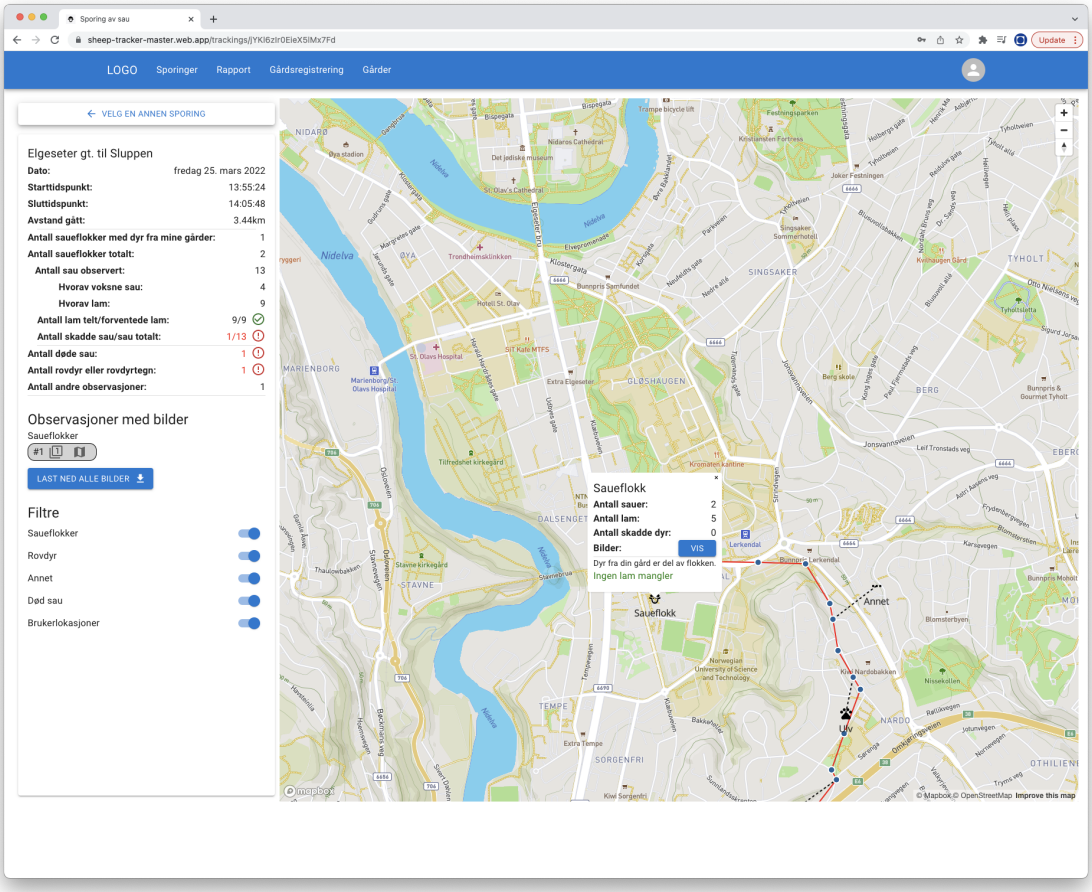
For det første inneholder implementasjonen av sidebaren mye mer informasjon enn designet, som for eksempel antall døde sau eller antall observerte lam og sau. Dette kommer av at under hurtigprototyping så kom det tilbakemeldinger som indikerte at vi trengte mer informasjon (og hva slags informasjon det

skulle være). Dette ble aldri endret i designet, men heller implementert direkte ettersom tilbakemeldingene kom under implementasjonsfasen.

For det andre ble det lagt inn en ny seksjon med en oversikt over saueflokker med bilder. Dette tillater brukeren å enkelt navigere til flokken på kartet automatisk ved å klikke på det lille kart-ikonet, eller å åpne bildeviseren for den saueflokken ved å klikke på bilde-ikonet. I tillegg til dette så er det også en «last ned alle bilder»-knapp som enkelt tillater brukeren å laste ned alle bildene fra tilsynet, noe som oppfyller andre del av FK16 (Tabell 20). Grunnen til at dette ikke var i designet kom av at vi under implementasjonsfasen følte at det ikke var en enkel måte å finne fram til spesifikke saueflokker for å se bildene deres, i tillegg til at vi skjønte at det å kunne laste ned alle bilder på en gang ville være nyttig funksjonalitet. Ettersom det skjedde midt under implementasjonsfasen ble det implementert direkte uten å lage et design ettersom vi følte det bare vil ta unødvendig mye tid for gevinsten man fikk ut av det.

For det tredje ble filtrene fra designet forandret fra avmerkingsbokser til såkalte *switcher* under implementasjonen. Dette kom av at vi synes switcher passet bedre til funksjonen knappene utfylte. En switch representerer tilstanden til noe som kan være av eller på, samtidig som at den indikerer for at tilstanden kan endres. Avmerkningsbokser derimot er mer typisk å finne på utfyllingsskjemaer og gir ikke brukereren like gode assosiasjoner til at noe kommer til å skje umiddelbart om de avmerkes.

Figur 59 viser bildevisning på tilsynssiden som ble implementert etter designet vist i Seksjon 12.5.7, og oppfyller FK15 og første del av FK16 (Tabell 20). Det er noen mindre forskjeller mellom designet og implementasjonen som menybaren på toppen som lar brukeren laste ned enkeltbilder eller zoome inn. Dette kommer av at designet var en rask kopi av bildevisningsbiblioteket vi hadde valgt ut på forhånd, dette bare for å se at det passet inn med resten av designet. Vi valgte å ikke bruke unødvendig tid på å legge til små detaljer i designet, ettersom utseendet uansett ble bestemt av bildevisningsbiblioteket.

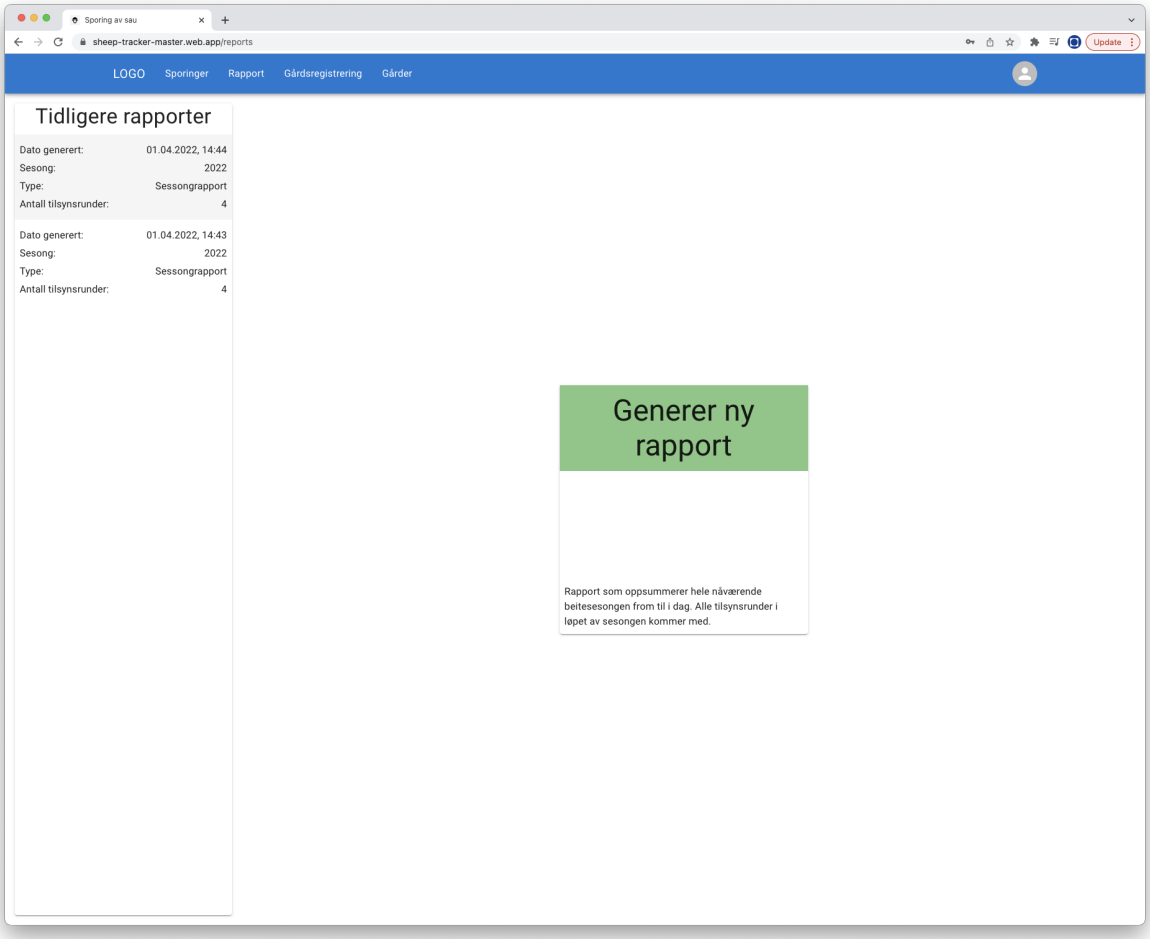


Figur 58: Skjermbilde fra implementasjon - Tilsyn

Figur 59: Skjermbilde fra implementasjon - Visning av bilder i tilsyn

13.4.2.10 Liste over rapporter

Figur 60 viser siden med liste over rapporter, som ble implementert etter designet vist i Seksjon 12.5.8, og oppfyller som sagt FK20 og FK21 (Tabell 22). «Generer ny rapport»-knappen oppfyller også FK17 (Tabell 22). Utenom noen mindre forskjeller i farge- og tekstvalg – som ble endret på grunn av designbibliotekets farger og grammatiske grunner – så ble designet fulgt under implementasjonen.



Figur 60: Skjermbilde fra implementasjon - Liste over rapporter

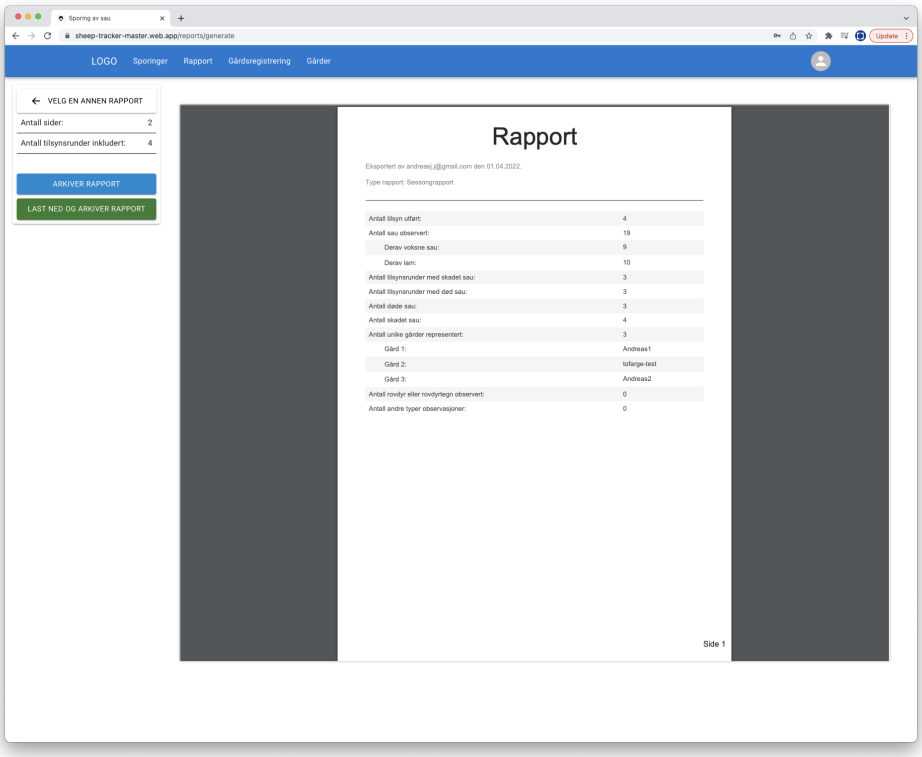
13.4.2.11 Rapport

Figur 61 og Figur 62 viser rapportsiden som ble implementert etter designet vist i Seksjon 12.5.8, og oppfyller som sagt FK18, FK19, FK21 og FK22 (Tabell 21). Det generelle oppsettet av siden følger designet ganske nøye, men det ble noen forandringer i rapportforhåndsvisningen og menyen.

Rapportforhåndsvisningen divergerer hovedsakelig fra designet i og med at rapporten her ligger på topp av en grå bakgrunn, dette kommer av at rapporten blir generert som en PDF og deretter vist i nettleseren sin PDF-leser (som man ikke kan style selv). Funksjonelt sett er det likevel ingen forskjell mellom implementasjonen og designet.

Menyen har noen mindre forskjeller når det kommer til fargevalget av knappene for å bedre differensiere

knappene og signalisere hvilken funksjon de har. I tillegg til dette så vil menyen vise forskjellige knapper avhengig av om man ser på en nylig generert rapport eller en arkivert rapport, hvor menyen for en nylig generert rapport kan bli sett i Figur 61 og menyen for en arkivert rapport kan bli sett i Figur 62. Dette kommer av at det ikke gir mening å vise en «arkiver»-knapp på en arkivert rapport eller en «slett»-knapp på en generert rapport, dermed ga det mening å vise forskjellige knapper. Grunnen til at det ikke ble vist i designet kom rett og slett fra at vi glemte å designe denne forskjellen under designfasen og det ga ikke mening å bruke tid på å designe det under implementasjonsfasen når vi allerede jobbet med implementere resten av siden.



Figur 61: Skjermbilde fra implementasjon - Generert rapport

Figur 62: Meny under visning av tidligere Generert rapport

Figur 63 og Figur 64 viser de to hoveddelene – oppsummering og oversikt over tilsyn – som en generert rapport består av (i PDF-format) som ble implementert etter designet vist i Seksjon 12.5.8.

Oppsummeringssiden følger designet ganske tett, med unntak av noen mindre estetiske forandringer som kom av at vi brukte et bibliotek som tok seg av genereringen av PDF-en, og å forandre mange av de mindre estetiske detaljene ville tatt unødvendig mye tid eller ikke vært mulig uten å generere PDF-en manuelt.

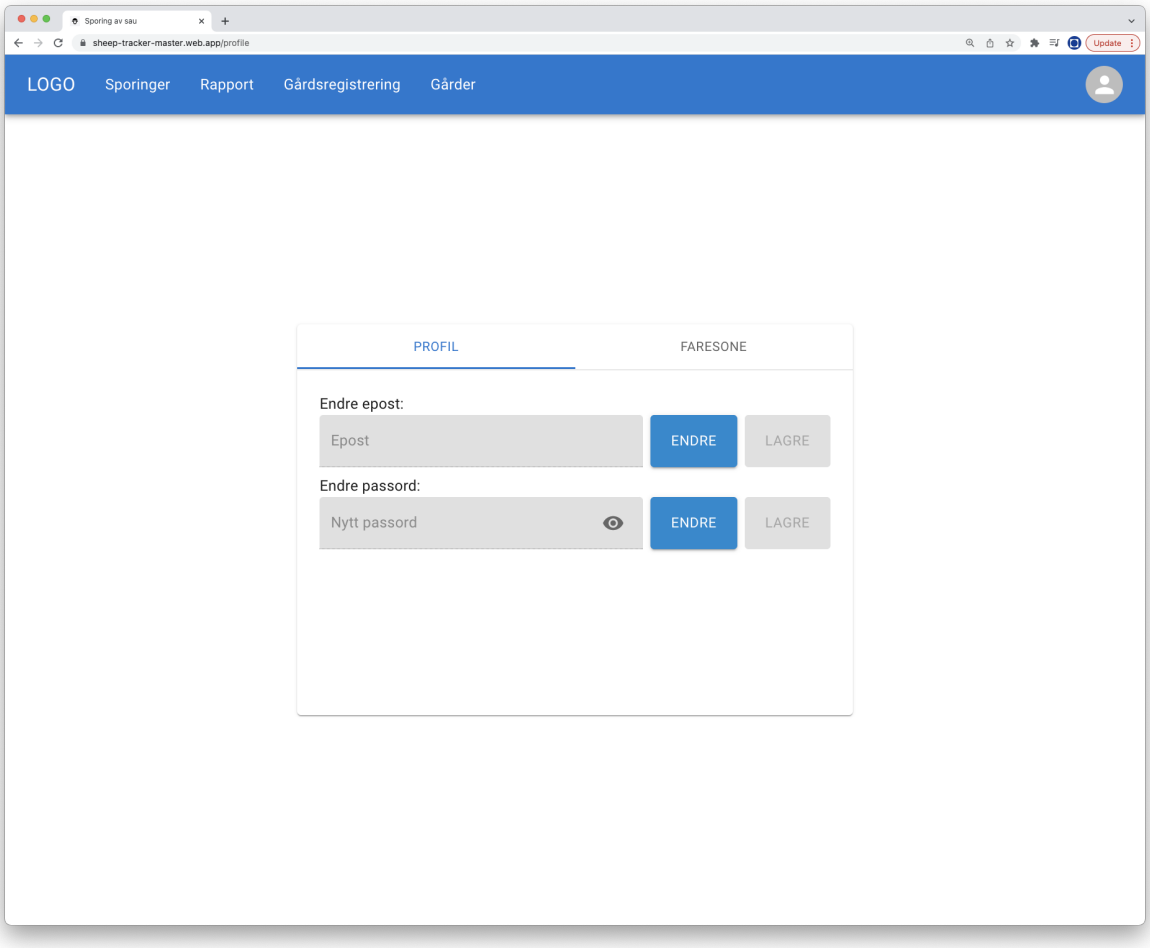
Oversiktssiden har også noen mindre estetiske forandringer av samme grunn som oppsummeringssiden, men i tillegg til dette ble det lagt til en ekstra kolonne i implementasjonen. «Bilde av ruta»-kolonnen ble lagt mot slutten av den hurtige prototypingen hvor det ble kjent at det var et krav (FK26 (Tabell 24)) at man burde kunne se et bilde over ruta gjeteren har gått under tilsynet når man leser rapporten. Grunnen til at det ikke ble vist i designet kom rett og slett fra at vi glemte å legge det til i designet ettersom vi var helt på slutten i designfasen og det ga ikke mening å bruke tid på å designe det under implementasjonsfasen når vi allerede jobbet med implementere resten av siden.

Figur 63: Skjermbilde fra implementasjon - Rapport PDF side 1

Figur 64: Skjermbilde fra implementasjon - Rapport PDF side 2

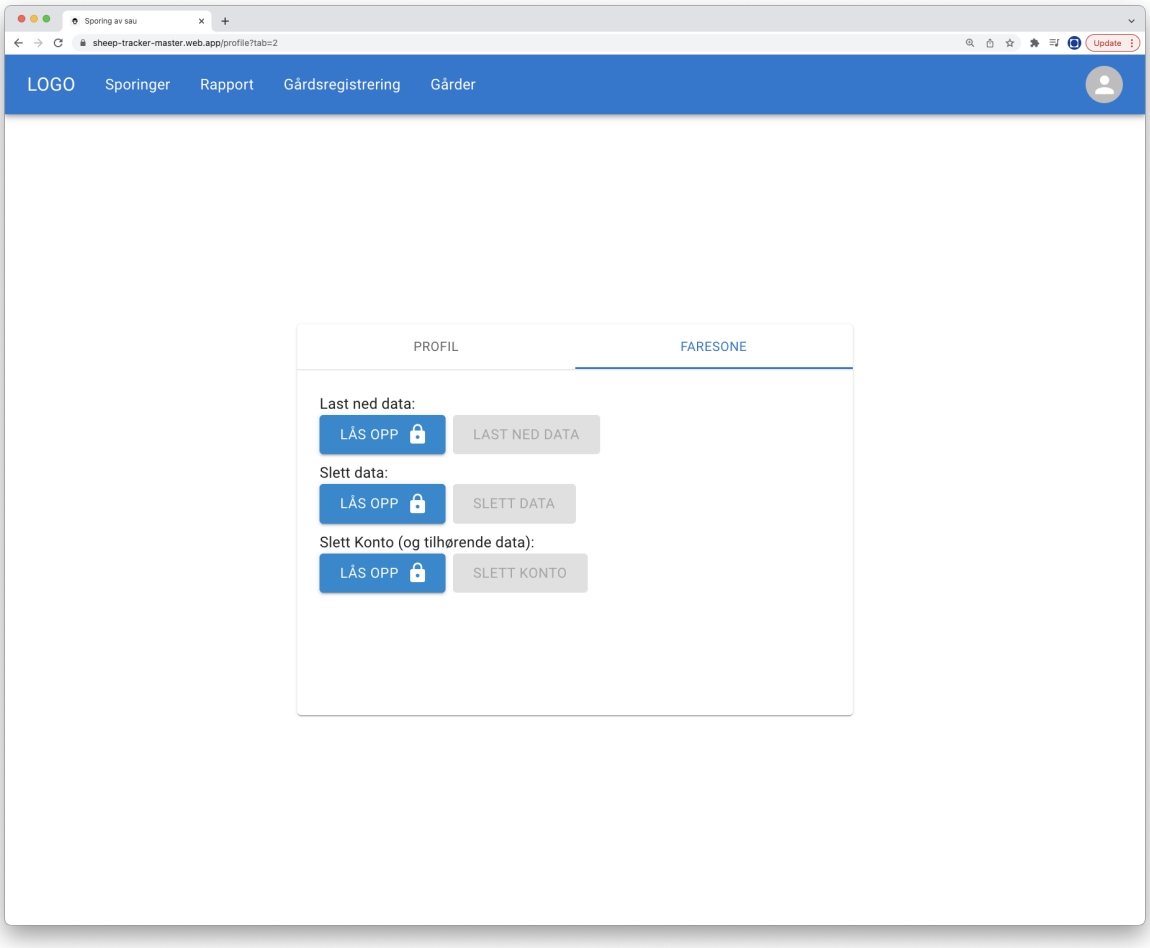
13.4.2.12 Profilside

Figur 65 viser profilsiden og mer spesifikt profildata tab-en på profilsiden som ble implementert etter designet vist i Seksjon 12.5.9, og oppfyller som sagt FK23 (Tabell 23). Rent funksjonelt er designet fulgt, men som på andre sider så er det blitt gjort mindre estetiske endringer som kom av at komponenter fra et UI-bibliotek ble brukt. I tillegg til de mindre endringene, ble overskriften endret til en tab-style meny for å tillate navigering mellom profildata tab-en og faresone tab-en.



Figur 65: Skjermbilde fra implementasjon - Profildata

Figur 66 viser implementasjonen av «faresone-siden», eller heller faresone tab-en på profilsiden, og oppfyller FK24 og FK25 (Tabell 23). Denne siden ble ikke lagt til i designet ettersom kravene rundt personvern ikke ble lagt til før slutten av den hurtige prototypingen og starten av implementasjonsfasen, og dermed glemte vi rett og slett å legge det til i designet. Som tidligere forklart rundt andre manglende designimplementasjoner så var funksjonene såpass simple at å lage et design før implementasjonen som allerede var startet i implementasjonsfasen ville tatt unødvendig mye tid for fordelene man får ut av det.



Figur 66: Skjermbilde fra implementasjon - Faresone

13.5 Endringer fra brukertesting

Seksjon 13.4 viser som sagt den ferdige implementasjonen før testingen i Seksjon 14.4, som bestod av to faser med brukertesting. Begge fasene med brukertesting førte med seg tilbakemeldinger om feil som måtte fikses, tekst som måtte endres og UI som burde endres for en bedre brukeropplevelse. Tilbakemeldingene fra første fase med brukertesting ble implementert som endringer og vil bli diskutert i Seksjon 13.5.1. Tilbakemeldingene fra andre fase med brukertesting derimot ble ikke implementert på grunn av manglende tid, men kan være mulige framtidige endringer som kan bli tatt med i videre arbeid og vil bli diskutert i Seksjon 13.5.2.

De tilbakemeldingene som ikke ble sett på som problematiske vil ikke bli diskutert her ettersom de ikke førte til endringer.

13.5.1 Implementerte endringer

Tilbakemeldingene fra første fase førte til fem endringer som ble implementert. To av endringene var relativt betydelige, mens resten av endringene var mindre finpuss.

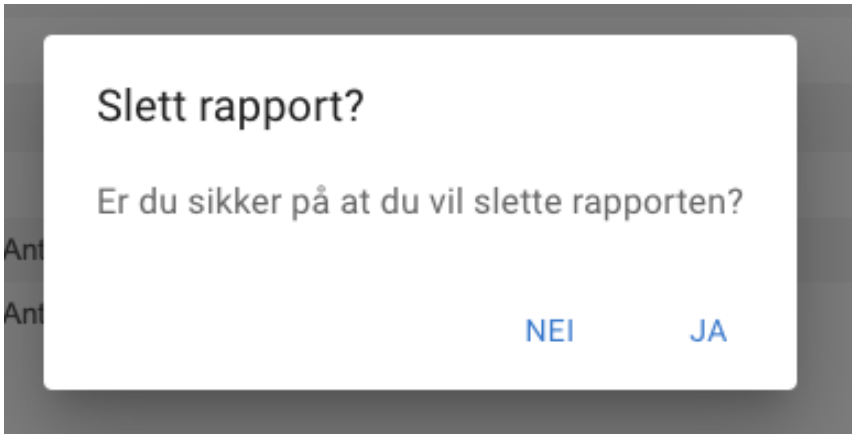
13.5.1.1 Filter for opptegning på kartet i tilsynsvisning

Det ble oppdaget at brukeren hadde noen mindre problemer med å kjapt skru av filtrene hvis han startet fra toppen og beveget seg nedever, Dette var fordi tooltipen originalt åpnet seg direkte under knappen som ble hovret over, som førte til at den neste filterknappen under ble tildekket. Dette ble enkelt fikset med å legge tooltipen på høyre siden istedenfor under, som vist i Figur 67.

Figur 67: Skjermbilde av endring av tooltips på tilsynssiden

13.5.1.2 Sletting av rapport

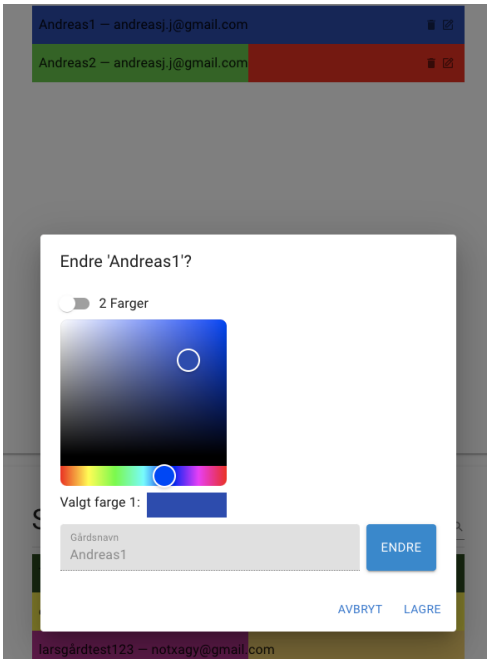
Denne endringen var ikke originalt et problem som brukerne hadde, men det ble oppdaget at sletting av rapport ikke hadde en «Er du sikker på om du vil slette»-modal. Dermed ble dette lagt til etter første fase med brukertesting som vist i Figur 68.



Figur 68: Skjermbilde av innføring av modal for å være sikker på at brukeren vil slette en rapport

13.5.1.3 Endre navn på gård

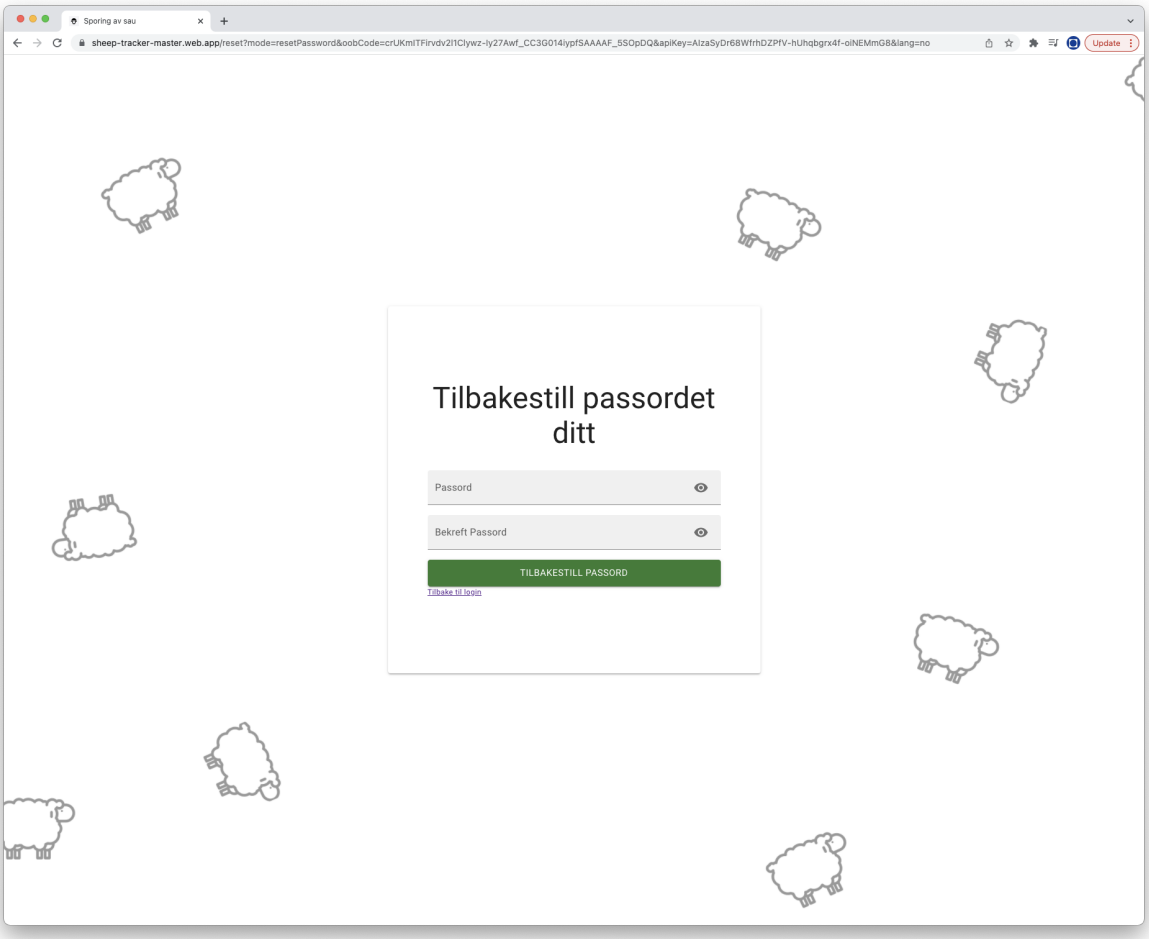
Endring av navn på gård var originalt relativt vanskelig for brukere å gjennomføre. Praktisk talt var dette ikke mulig å gjøre direkte, men heller indirekte gjennom å første slette gården for å deretter lage en ny gård med samme farge og nytt navn. For å fikse dette så la vi til et tekstinput-felt og en knapp (som gir deg muligheten til å endre navnet i tekstinput-feltet) i den originale modalen som lar deg endre farge på gården. Denne endringen kan bli sett i Figur 69 og kan bli sammenlignet med Figur 55 for å se forskjellen.



Figur 69: Skjermbilde av endring av modal for endring av gård

13.5.1.4 Tilbakestillingsside for passord

Det ble oppdaget at tilbakestilling av passord gjennom Firebase sin egen UI kunne føre til problemer hvis brukeren brukte et passord som ikke fulgte de originale passordkravene. Dette kom av at man ikke kan innføre krav om hvor sterkt et passord skal være gjennom Firebase sin UI, noe som førte til at brukere kunne lage passord som de ikke fikk lov til å logge inn med. Dermed endte vi opp med å måtte lage vår egen UI for tilbakestilling av passord som sett i Figur 70, hvor vi hadde mulighet for å innføre de samme kravene som registreringssiden har.



Figur 70: Skjermbilde fra implementasjon - Tilbakestill passord

13.5.1.5 Tekstendring

Det ble oppdaget at beskrivelsen av hvilke spesialtegn som var tillatt i passordet på registreringssiden var noe uklart ettersom «@$!%*?&» var omringet av anførselstegn (“), noe som kunne få brukeren til å tro at anførselstegnene var blant de tillatte spesialtegnene. Dette ble enkelt fikset ved å fjerne anførselstegnene fra strengen.

13.5.1.6 Feilfiksing

I tillegg til forbedringene som ble implementert i underkapitlene ovenfor, var det også en feil som førte til at én av funksjonene i applikasjonen rett og slett ikke fungerte, noe som gjorde at FK11 (Tabell 19) ikke ble oppfylt. Under brukertestingen så ble det oppdaget at å endre fra én til to farger på en gård ikke fungerte – senere viste det seg at å endre fargene uavhengig av hva forandringen var ikke fungerte på grunn av en feil i Firebase Security Rules. Dette ble fikset ved å gjøre noen endringer på sikkerhetsreglene.

13.5.2 Mulige framtidige endringer

Tilbakemeldingene fra andre fase førte til to mindre mulige endringer som kan bli implementert i framtiden i videre arbeid.

13.5.2.1 Visning av gjeterrute

Under andre fase av brukertestingen ble det oppdaget at hvis man prøver å se en gjeterrute uten å være logget inn får man en feilmelding som sier «Bilde finnes ikke». Dette kan gjøre brukeren forvirret ettersom feilen ikke er at bildet ikke finnes, men at brukeren ikke er logget inn. Dette kan enten fikses ved å gjøre om Firebase Security Rules til å tillate alle brukere (selv ikke-innloggede) å lese data om gjeterruter, eller ved å endre feilmeldingen til å be brukeren om å logge inn. Det andre alternativet er nok den beste mulige endringen ettersom dette ikke påvirker sikkerhetsimplementasjonen og sikkerhetsvurderingen.

13.5.2.2 Endre navn på gård

Endringene i Seksjon 13.5.1.3 førte til en bedre brukeropplevelse, men som nevnt i Seksjon 17 så førte den nylige innførte «Endre»-knappen til noe forvirring rundt hvilken knapp som lar deg endre navnet og hvilke knapp som lagret endringene på gården. En mulig endring for å fikse dette kan være å endre teksten fra «Endre» til et ikon av en lås for å indikere at knappen tillater deg å endre navn og ikke å lagre endringene på gården.

14 Testing

Testing er en viktig del av et programvareutviklingsprosjekt for å sikre seg om at alle krav er innfridd, at funksjonaliteten fungerer uten feil og at programvarearkitekturen fungerer best mulig. I denne seksjonen vil vi gå gjennom testplanen og de forskjellige testmetodene som vil bli brukt i prosjektet – end-to-end testing, akseptansetesting og brukertesting.

14.1 Testplan

En testplan er et detaljert skriv som beskriver en teststrategi, testobjekter, en tidsplan og estimering, testleveranser og eventuelle ressurskrav og ressursplanlegging. Kort fortalt skal en testplan hjelpe et team med å bestemme hvor mye innsats som trengs for å validere at et bestemt mål har blitt nådd og hva som eventuelt skal leveres etter at testingen er utført. Testplanen skal altså fungere som en blåkopi for testprosessen [101].

14.1.1 Omfang og identifikasjon av testtyper

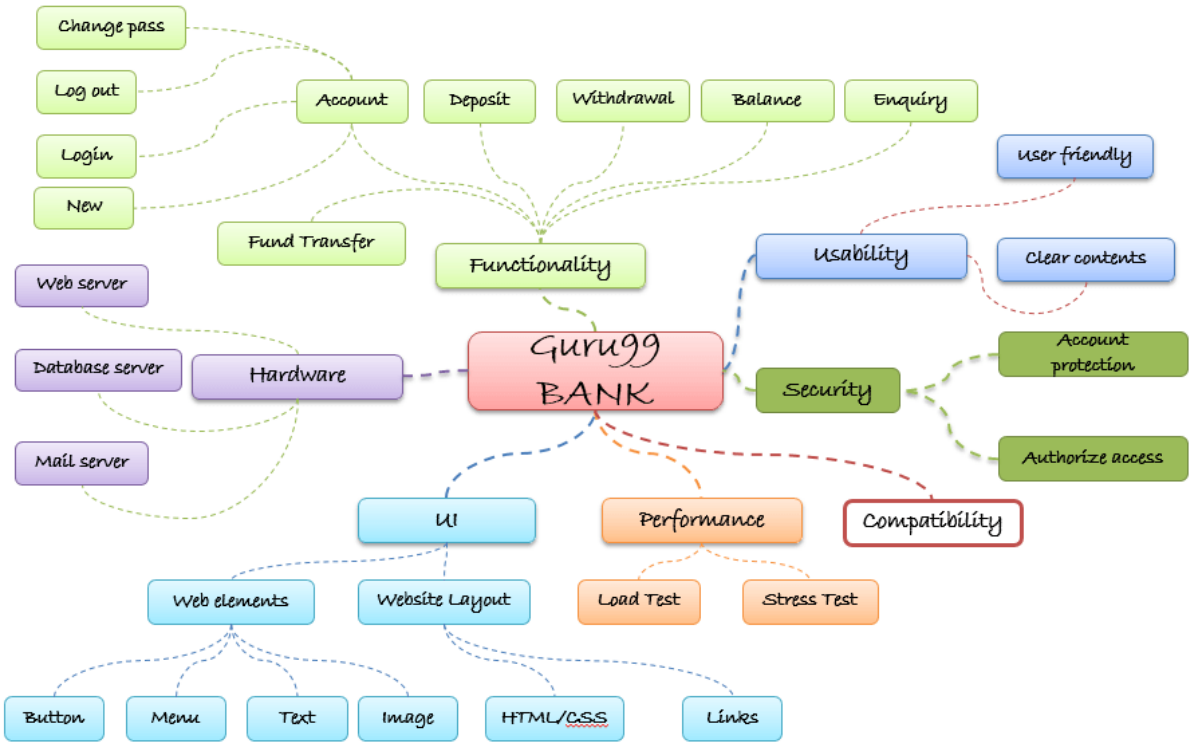
Ettersom dette prosjektet ligner mer på konseptutvikling eller prototypeutvikling enn produktutvikling, vil testingen være relativt begrenset. Koden trenger ikke å være 100 % feilfri, det viktigste er at den oppfyller de funksjonelle og til dels de ikke-funksjonelle kravene, i tillegg til at den får gode tilbakemeldinger fra brukergruppen. På grunn av dette vil de funksjonelle kravene bli testet gjennom end-to-end testing, akseptansetesting og brukertesting, og de ikke-funksjonelle kravene vil bli testet gjennom akseptansetesting så langt det lar seg gjøre. Det vil altså ikke bli utført enhetstesting, men koden har blitt tilrettelagt for at det skal være mulig å skrive enhetstester hvis det skulle bli relevant.

14.1.2 Risiko og problemhåndtering

I et produktutviklingsprosjekt som faktisk skal bli brukt i en produksjonssetting vil risikovurdering og problemhåndtering være en viktig del av testplanen for å håndtere eventuelle problemer som oppstår under testingen. Et eksempel kan være at et teammedlem mangler de nødvendige ferdighetene for å teste eller at man ikke får utført alle testene innenfor tidsfristen. Ettersom dette prosjektet ikke skal ut i produksjonssetting eller har et budsjett som må følges så er det minimalt med risiko og problemer som må håndteres. Det eneste problemet som kan oppstå er at det ikke er tid til å utføre alle testene før tidsfristen, og hvis det skulle skje vil følgende prioritering bli gjort med tanke på hvilke tester som blir utført først: Brukertesting → akseptansetesting → end-to-end testing.

14.1.3 Testmål

Når det kommer til å finne testmål så kan det ofte være en fordel å lage et såkalt Mind-map (norsk: *tankekart*) – som vist i Figur 71 – ved å analysere kravspesifikasjonen til programvaren.



Figur 71: Eksempel på mind-map. Kilde: [101]

For dette prosjektet vil det være unødvendig å bruke tid på den type analyse ettersom vi allerede har en klar oversikt over hva det er programvaren skal kunne utføre, nemlig de funksjonelle og ikke-funksjonelle kravene fra Seksjon 10. Testmålene vil altså være å teste de funksjonelle og ikke-funksjonelle kravene.

14.1.4 Testkriterier

Det finnes hovedsakelig to forskjellige typer testkriterier: suspensjonskriterier og utgangskriterier.

Suspensjonskriterier er kriterier som blir brukt for å bestemme at all testing skal suspenderes før man fortsetter testingen. I dette prosjektet er dette derimot ikke nødvendig på grunn av den begrensede funksjonaliteten og det faktum at det kun er to teammedlemmer på prosjektet. Ved store mangler eller feil skal testingen altså bli fullført og dokumentert sånn at de kan fikses for å oppfylle utgangskriteriene.

Utgangskriterier er kriterier som blir brukt for å bestemme at testing/testfasen er over. Normalt vil dette for eksempel kunne være at 80 % av alle test casene blir bestått, men ettersom vi hovedsakelig bruker akseptansetesting og brukertesting vil utgangskriteriene våre være noe annerledes. I tillegg til dette har vi såpass få og konkrete krav at utgangskriteriene våre burde være at alle krav er oppfylt og at tilbakemeldingene fra brukertestingen er at brukerne liker applikasjonen.

14.1.5 Tidsplan og estimering

Det kan være lurt å få et estimat på hvor mye arbeid eller hvor mange timer testingen kan ta, og hvem som skal utføre hvilke tester. En måte å utføre estimeringen på er å bryte testperioden ned i mindre oppgaver, tildele disse til forskjellige medlemmer/roller og deretter estimere hvor mange timer hver oppgave vil ta. For å lage en tidsplan ut ifra estimeringen er Gantt-diagram en god måte å få en god visualisering av hvem, når og hvor lang tid testingen vil ta.

For dette prosjektet så vil det bli gjort minimalt med estimering ettersom den mest sannsynlig vil være veldig unøyaktig, ettersom vi ikke har nok erfaringer med hvor lenge en testperiode kan vare, til at vi kan gjøre en god estimering. Dermed blir testingen utført i en mer smidig form uten faste estimater og oppgaveinndeling, men det har fortsatt blitt lagd en tidsplan for testingen hvor den generelle testperioden og tidsfristene er ført inn i Gantt-diagrammet for prosjektets tidsplan sett i Figur 16.

14.1.6 Testleveranser

Testleveranser kan bli delt inn i tre forskjellige kategorier: testleveranser for før test-fasen, testleveranser for under test-fasen og testleveranser for etter test-fasen.

Testleveranser for før test-fasen består hovedsakelig av testplaner, test cases, og testspesifikasjoner. Testleveranser for under test-fasen består av verktøy og data som trengs for å teste som test scripts, simulatorer, test data, error logs, skjema og så videre. Testleveranser for etter test-fasen er derimot dokumenter som testresultater/rapport, feilrapporter, skjemaer, og så videre.

For dette prosjektet så vil testleveransene for hver testmetode være følgende:

- End-to-end testing
  - Før test-fasen
    - Funksjonelle krav som Test Case dokument
  - Under test-fasen
    - Cypress test scripts
    - Cypress simulator
    - Test data
    - Logger
  - Etter test-fasen
    - Cypress testrapport
- Akseptansetesting
  - Før test-fasen

    - Funksjonelle krav som Test Case dokument
    - Test plan dokument
  - Under test-fasen
    - Kjørende versjon av applikasjonen
  - Etter test-fasen
    - Testresultat rapport

- Brukertesting
  - Før test-fasen
    - Testplan-dokument
    - Test design spesifikasjonsdokument
  - Under test-fasen
    - Kjørende versjon av applikasjonen
    - Papirdokumenter for å lage rapport for hånd
  - Etter test-fasen
    - Testrapport
    - Utfylte SUS-skjemaer
    - Brukerskapte oppsummeringer av tilsyn for hånd
    - Skjema med intervju-notater

14.2 End-to-end testing

End-to-end testing går ut på å teste systemet i sin helhet, og ligner på akseptansetesting, men automatisert med kode. Fordelen er at man får teste at systemet i sin helhet fungerer og hindrer feil hvor et sub-system krasjer og tar ned hele applikasjonen. I tillegg til dette finner man også bugs som hadde blitt oppdaget under testing som enhetstesting og integrasjonstesting. Dette er selvfølgelig ikke i samme grad som i de nevnte testtypene, men det er en fantastisk måte å oppdage feil uten å måtte investere tiden som trengs for enhets- og integrasjonstesting.

Som nevnt i Seksjon 11.1.6.1 vil Cypress bli brukt for end-to-end testing, og tester ble hovedsakelig skrevet under test-fasen av prosjektet. Resultatet av en enkel test vil være «Passed»/«Failed» og totalt sett vil testleveransen etter test-fasen være en testrapport hvor resultatet av hver test blir oppsummert, som nevnt i forrige delseksjon.

14.3 Akseptansetesting

For å sjekke at funksjonaliteten som er implementert oppfyller bruksmønsteret og de funksjonelle kravene vil akseptansetesting bli brukt. Denne testingen går ut på å manuelt bruke funksjonaliteten for det spesifikke bruksmønsteret og de tilhørende funksjonelle kravene og deretter å spørre seg selv om funksjonaliteten oppfyller dem. Resultatet av en enkel akseptansetest vil være godkjent/ikke-godkjent med en mulighet for å legge igjen en kommentar hvis resultatet er ikke-godkjent.

Generelt sett vil akseptansetesting bli utført på en kunde eller domeneekspert, men ettersom prosjektet ikke har en enkel kunde så vil akseptansetestingen bli utført på både oss – forfatterne/utviklerne – ettersom det var vi som lagde flere av kravene og bruksmønstrene, så vel som veilederen vår Svein-Olaf Hvasshovd, som kan sies å være en domeneekspert.

14.4 Brukertesting

I utførelsen av brukertesting lot vi utenforstående personer til prosjektet benytte systemet med hensikt om å samle data og tilbakemeldinger på brukskvaliteten til systemet. Hvordan selve testen av systemet ble utført beskrives i Seksjon 14.4.4.2. For å få bedre innsikt i hvordan vårt system sammenligner seg i bruk med dagens mest utbredte løsning, nemlig papirnotater, gjorde vi også brukertesting på en prosess der brukeren oppsummerer papirnotater for hånd. Dette vil utdypes om i Seksjon 14.4.4.1.

14.4.1 Brukskvalitet

ISO 9241-11 [102] definerer brukskvalitet (engelsk: *usability*) slik:

> **Usability:** extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.

Et viktig aspekt av denne definisjonen er at brukskvalitet ikke er en iboende egenskap til systemet, men at brukskvaliteten heller er avhengig av konteksten systemet brukes i. I selve definisjonen nevnes tre variabler som alle vil påvirke hvordan vi vurderer brukskvaliteten:

- Brukere (engelsk: *users*)
  - Hvem er brukerne?
- Mål (engelsk: *goals*)
  - Hva ønsker de å oppnå med systemet?
- Brukskontekst (engelsk: *context of use*)
  - Hvor skal systemet brukes?
  - I hvilken sammenheng brukes systemet?

Gitt dette var det viktig at systemets design, så vel som utføringen av brukertestene, passet overens med disse variablene. Vi gikk derfor inn for å kartlegge brukerne, målene og brukskonteksten til systemet:

- **Brukere:** Sauebønder og medlemmer av beitelag. Brukeren forventes å ha et helt alminnelig ferdighetsnivå når det kommer til IT-dyktighet, og brukeren er i stand til å navigere seg rundt i en nettleser.
- **Mål:**
  - **Primærmål:** Få oversikt over tilsyn som har blitt utført, på digitalt vis.
  - **Sekundærmål:** Registrere gårder slik at tilsynsansvarlig/gjeter kan finne dem i mobilapplikasjonen.
- **Brukskontekst:**
  - **Hvor skal systemet brukes:** På en stasjonær eller bærbar datamaskin, med systemet kjørende i en nettleser. Brukeren befinner seg innendørs og har tilgang til mus og tastatur.
  - **I hvilken sammenheng brukes systemet**: Før, under og (viktigst) etter endt beitesesong. Brukeren har god tid til å utføre oppgavene sine.

Disse faktorene var viktige å reflektere i utføringen av brukertestingen for å sikre at brukertestingen gjøres innenfor en kontekst som ligner så mye som mulig på den tiltenkte brukskonteksten til systemet. Som nevnt tidligere hadde vi ikke tilgang til et utvalg av sauebønder eller gjetere under prosjektutførelsen, og måtte derfor tilpasse testutføringen litt slik at brukertestingen faktisk lot seg utføre.

14.4.2 Hva brukertesting angår

Målet med brukertesting er å undersøke hvordan faktiske brukere benytter systemet. I boka *A Practical Guide to Usability Testing* [103] presenteres en kort oppsummering på hva brukertesting (engelsk: *usability testing*) går ut på:

> While there can be wide variations in where and how you conduct a usability test, every usability test shares these five characteristics:
>
> 1. The primary goal is to improve the usability of a product. For each test, you also have more specific goals and concerns that you articulate when planning the test.
> 2. The participants represent real users.
> 3. The participants do real tasks.
> 4. You observe and record what participants do and say.
> 5. You analyze the data, diagnose the real problems and recommend changes to fix those problems.

Brukertestene våre var planlagt og utført med utgangspunkt i disse fem prinsippene.

14.4.3 Vurdering av brukskvalitet: System Usability Scale

System Usability Scale (SUS) er et verktøy for å vurdere brukskvaliteten til et system eller en tjeneste. SUS ble først laget av John Brooke i 1986, og består av en undersøkelse på ti spørsmål som en bruker besvarer umiddelbart etter de har brukt systemet [104]. Hvert spørsmål besvares med en tallverdi mellom 1 og 5, der 1 representerer sterk uenighet og 5 representerer sterk enighet (en *Likert-skala* [105]). Summen av svarene transformeres slik at totalsummen representeres med en verdi mellom 0 og 100, hvor høyere verdier korrelerer med et system med bedre brukskvalitet.

I en større analyse av brukskvalitetsvurderinger av systemer hvor SUS ble tatt i bruk, ble kvaliteten til SUS som et verktøy for å bedømme brukskvalitet framhevet (sitat Bangor, Kortum & Miller (2008)):

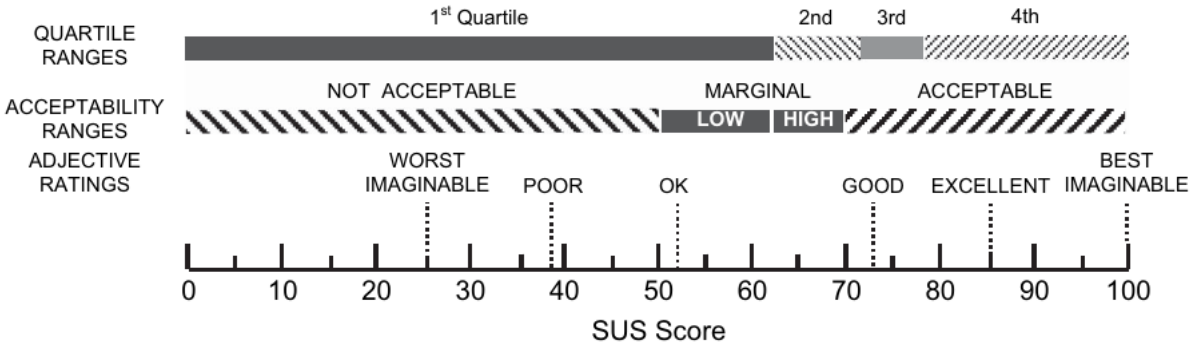
> Results from the analysis of this large number of SUS scores show that the SUS is a highly robust and versatile tool for usability professionals. [106]

En stor fordel med å ta i bruk SUS som verktøy for å vurdere brukskvalitet er at vi slipper å formulere spørsmålene eller drøfte hvordan brukskvaliteten kan kvantifiseres. SUS har allerede løst disse problemene med ferdigformulerte spørsmål og en skala på 0-100 man kan gå ut i fra.

Hver utførelse av en SUS-test resulterer i et enkelttall som representerer en sammensatt måling av brukskvaliteten til systemet, men Brookes artikkel hvor SUS presenteres [104] gir ingen konkrete forslag til hvordan denne tallverdien skal tolkes. Et system som får 0 i SUS-poengsum har den laveste mulige brukskvaliteten og et system med 100 i SUS-poengsum har den høyeste mulige brukskvaliteten, men det er rimelig å anta at alle reelle systemer vil ligge et sted imellom disse to ytterpunktene. Hvor går grensen mellom et system med lavt og høy brukskvalitet? Bangor, Kortum og Miller adresserte dette spørsmålet i deres analyse av SUS, og kom med en generell veiledning basert på deres erfaringer som lyder som følger:

> [...] products which are at least passable have SUS scores above 70, with better products scoring in the high 70s to upper 80s. Truly superior products score better than 90. Products with scores of less than 70 should be considered candidates for increased scrutiny and continued improvement and should be judged to be marginal at best. [106]

I samme undersøkelse av SUS presenterte Bangor, Kortum og Miller en inndeling av 0-100 skalaen av SUS-poengsummer inn i kvartiler, aksebilitetsområder (bestående av «*not acceptable*», «*low marginal*»; «*high marginal*» og «*acceptable*»), og adjektiver (bestående av «*worst imaginable*», «poor», «OK», «good», «excellent» og «*best imaginable*»).



Figur 72: Ulike inndelinger av hva en SUS-poengsum kan si om brukbarheten til et system. Kilde: Bangor, Kortum og Miller (2008) [106].

De ti spørsmålene i SUS ble originalt formulert på engelsk; vi tar i bruk en norskoversatt versjon av disse spørsmålene fra Dag Svanæs. Dette skjemaet vises i Vedlegg D. Spørsmålene som blir brukt i SUS, oversatt til norsk, er som følgende:

1. Jeg kunne tenke meg å bruke dette systemet ofte.
2. Jeg synes systemet var unødvendig komplisert.
3. Jeg synes systemet var lett å bruke.
4. Jeg tror jeg vil måtte trenge hjelp fra en person med teknisk kunnskap for å kunne bruke dette systemet.
5. Jeg syntes at de forskjellige delene av systemet hang godt sammen.
6. Jeg syntes det var for mye inkonsistens i systemet (Det virket «ulogisk»).
7. Jeg vil anta at folk flest kan lære seg dette systemet veldig raskt.
8. Jeg synes systemet var veldig vanskelig å bruke.
9. Jeg følte meg sikker da jeg brukte systemet.
10. Jeg trenger å lære meg mye før jeg kan komme i gang med å bruke dette systemet på egen hånd.

14.4.4 Forløp av brukertestene

I løpet av en testutførelse ble bruk av både papirskjemaer og systemet (webapplikasjonen) testet, og disse beskrives hver for seg i henholdsvis Seksjon 14.4.4.1 og Seksjon 14.4.4.2. Før utførelsen av hver test fikk brukeren litt grunnleggende informasjon om hvordan testen ville foregå:

- Du vil få oppgitt ulike oppgaver du må løse.
- Det er systemets brukskvalitet som bedømmes, ikke *din* evne til å løse oppgavene.
- Løs oppgavene slik du selv mener de burde løses. Om du står fast, kan du hoppe til neste oppgave. Testfasilitatoren kan ikke gi instrukser om hvordan oppgaven *burde* løses. Du sier i selv fra når du mener oppgaven er løst.
- Tenk gjerne høyt og si hva du gjør – da lærer vi mest om hvordan brukeren tenker.

Bruk av papirskjemaet ble testet først, og umiddelbart etterpå ble brukeren satt til å teste webapplikasjonen. Under alle testene hadde begge prosjektmedlemmene et ansvar; én person ga instruksjoner om hva brukeren skulle gjøre og besvarte eventuelle spørsmål brukeren hadde om testprosessen. Den andre personen satt bak brukeren for å observere og notere brukerens handlinger underveis. Etter testen av webapplikasjonen var fullført ble brukeren bedt om å fylle ut et SUS-skjema, og til sist ble et kort intervju med brukeren utført.

14.4.4.1 Brukertesting av papirbaserte notater

For å kunne sammenligne hvordan det er å jobbe med papirnotater i mot vårt digitale system, lagde vi en egen brukertest som gikk ut på å oppsummere notater gjort for hånd inn i et eget oppsummeringsskjema. Hver bruker fikk oppgitt et ferdigutfylt ark hvor notater fra fire ulike tilsyn hadde blitt tatt. Brukeren fikk oppgitt et annet skjema der de måtte oppsummere hva som ble sett i hvert tilsyn inn i en egen tabell. Til slutt måtte alle tallene legges sammen for å oppsummere tallene for hele sesongen.

Arket utfylt med notater fra tilsyn vises i Vedlegg E. Selve malen for arket er hentet fra første side av tilsynsskjemaet fra NSG, vist i Vedlegg B, og er derfor representativt for hvordan notater kan bli tatt ved utføring av tilsyn i dag. Notatene ble skrevet i blokkskrift for å forhindre at brukeren måtte bruke tid på å tyde håndskrift.

Skjemaet brukeren måtte benytte for å oppsummere tilsynsnotatene vises i Vedlegg F. Vi valgte å ikke benytte oppsummeringstabellene i tilsynskjemaene diskutert i forarbeidet (vist i Vedlegg B og Vedlegg C), fordi disse har flere felt å fylle ut som er helt unødvendige for vår test (for eksempel «dyr sluppet på beite»‘, «tap i %» og så videre). For å bedre kunne sammenligne papirbasert oppsummering av tilsyn mot vårt system som automatisk genererer rapporter, sørget vi for at oppsummeringsskjemaet brukeren måtte fylle ut minnet om oppsummeringen i rapportene generert av systemet vårt – som igjen er basert på hva veileder Svein-Olaf Hvasshovd mener er hensiktsmessig å ha med i en slik sesongoppsummerende rapport.

I tråd med forskningsprosessen i Seksjon 3.3 er **dokumenter**, i form av utfylte oppsummeringsskjemaer, dataen som blir generert fra dette steget i testingen.

14.4.4.2 Brukertesting av webapplikasjonen

Brukertesten av webapplikasjonen bestod av at brukerne måtte utføre ulike oppgaver som representerer realistiske brukerflyter innenfor systemet. Totalt var det 34 oppgaver brukerne måtte løse.

Oppgave ID	Beskrivelse
1	Som en uregistrert bruker, vil jeg opprette en konto.
2	Som en ny bruker, vil jeg registrere en gård som har dette navnet og denne fargen (viser bilde).
3	Jeg ønsker å endre passord på kontoen min.
4	Som en innlogget bruker, vil jeg se listen over mine gårder.
5	Jeg ønsker å finne ut om nabogården, som heter Melhusgården, finnes i systemet. Finn ut om denne gården eksisterer i systemet.
6	Jeg ønsker å logge ut.

Tabell 42: Brukertesting av webapplikasjon - oppgaver del 1. Her oppretter brukeren en ny brukerkonto og gjør noen handlinger mens de er innlogget.

Oppgave ID	Beskrivelse
7	Jeg ønsker å logge inn som <demobrukeren> (gi login-ark til bruker).
8	Jeg ønsker å se på tilsynene som har blitt utført.
9	Jeg ønsker å se på tilsynet med beskrivelsen «Tur i Midtbyen».
10	Jeg ønsker å vite om skadede sau ble oppdaget i tilsynet. I så fall, hvor mange var det?
11	Hvor mange voksne sau og hvor mange lam ble observert i tilsynet?
12	Jeg ønsker å kun se saueflokker på kartet.
13	For hver saueflokk på kartet ønsker jeg å vite om antall lam stemmer, eller om noen mangler.
14	Jeg ønsker å skru på visning av alt på kartet igjen.
15	Ble det oppdaget døde sau i løpet av tilsynet? I så fall hvor mange?
16	Du har hørt rykter om at det er mange rovdyrangrep rundt Nidarosdomen. Er det noen saueflokker i nærheten av Nidarosdomen som viser tegn på å ha blitt bitt?
17	Du vil laste ned bildet av denne sauen som har viser tegn på å ha blitt bitt.
18	Undersøk funnene av død sau, og for hver av dem se om du kan finne tegn på at rovdyr har vært skyldig, eller om det har vært en ulykke.
19	Jeg ønsker å laste ned alle bildene fra tilsynet til PCen min.
20	Jeg ønsker å se på et annet tilsyn, som har beskrivelsen «Tur gjennom Gløshaugen».
21	Nevn minst to ting i dette tilsynet som skiller seg fra normalen og derfor kan være av interesse for deg som sauebonde.
22	Hvor lenge varte dette tilsynet?

Tabell 43: Brukertesting av webapplikasjon - oppgaver del 2. Her benytter brukeren en allerede eksisterende brukerkonto som har blitt forberedt for å teste visning av tilsyn.

Oppgave ID	Beskrivelse
23	Beitesesongen går mot slutt, og jeg vil generere en rapport som jeg kan sende til relevante myndigheter. Generer en ny rapport.
24	Les rapporten. Hvor mange sauer gikk tapt i løpet av sesongen?
25	Hvor mange tilsyn ble utført i løpet av sesongen?
26	Jeg ønsker å laste ned rapporten til PCen min.
27	Jeg vil åpne PDFen jeg lastet ned.
28	I tilsynet med beskrivelsen «Elgeseter gt. til Sluppen» er jeg nysgjerrig på hvilken rute tilsynspersonen tok. Finn ut dette.
29	Jeg ønsker å slette en rapport som ble tidligere generert.
30	Gården som tilhører min bruker, med navnet «Støjrdalsgården», er stavet feil. Jeg ønsker å gjøre det så gården har riktig navn.
31	Jeg ønsker å logge ut.

Tabell 44: Brukertesting av webapplikasjon - oppgaver del 3. Her fortsetter brukeren å benytte den forberedte brukerkontoen, og tester funksjonalitet knyttet til rapporter.

Oppgave ID	Beskrivelse
32	Jeg vil inn på brukeren jeg først opprettet. Men først må jeg nullstille passordet, fordi jeg har glemt passordet.
33	Det viser seg at gården jeg har lagt til i systemet, har feil farger oppgitt. Den ble lagt til med én farge, men skal egentlig være tofarget. Her er de korrekte fargene for denne gården (hold opp bilde). Gjør nødvendig korrigering på denne gården.
34	Jeg ønsker å slette denne kontoen og all dataen assosiert med den fra systemet.

Tabell 45: Brukertesting av webapplikasjon - oppgaver del 4. Her logger brukeren tilbake inn på brukerkontoen de selv opprettet og tester diverse funksjonalitet.

I sammenheng med Seksjon 3.3 er **observasjoner** dataen som blir generert fra denne delen av brukertestingen. Dette er altså notater om hva slags handlinger brukerne tok for å utføre oppgavene og hvorfor brukerne eventuelt slet med å utføre visse oppgaver.

14.4.4.3 SUS-skjema

Umiddelbart etter brukeren hadde fullført testoppgavene på webapplikasjonen, fylte de ut et SUS-skjema (vist i Vedlegg D) for hvordan de opplevde systemet.

Vi tok ikke i bruk et SUS-skjema for vurdering av brukskvaliteten når det kom til papirnotatene. I hovedsak er dette fordi vi var mest nysgjerrige på prosessen og hvordan brukeren opplevde bruken av papirnotater, mer enn vi var interessert i å måle opplevd brukskvalitet av papirnotater. En annen grunn til at vi ikke målte brukskvaliteten til bruk av papirnotater er at spørsmålene i SUS (for eksempel spørsmål 4 «*Jeg tror jeg vil måtte trenge hjelp fra en person med teknisk kunnskap for å kunne bruke dette systemet*») ikke virker hensiktsmessige å spørre når brukeren har gjort noe så enkelt som å skrive på et ark. I vår mening er oppsummeringsskjemaet på papir ikke tilstrekkelig komplisert nok til at en SUS-undersøkelse er nødvendig. For det tredje er det uansett ikke bruk av papirnotater som er hovedfokuset for dette prosjektet, og vi er ikke ute etter å forbedre prosessen rundt å benytte papirnotater. Eventuelle muligheter for forbedringspotensiale innenfor bruk av papirskjemaer, oppdaget via bruk av SUS, vil ikke være av verdi for oss.

Intensjonen med SUS er at brukeren skal svare ærlig med sine egne tanker om systemet. Det første spørsmålet i SUS, som sett i Seksjon 14.4.3 («Jeg kunne tenke meg å bruke dette systemet ofte.»), krevde likevel kontekstualisering fra vår side som testfasilitatorer. Ettersom ingen av testbrukerene våre

var sauebønder, er det realistisk sett null sannsynlighet for at de ville brukt et system for sauebønder i framtiden. For å unngå at dette spørsmålet dro ned SUS-poengsummen på alle testene (noe som ville ha deflatert SUS-pengsummene) instruerte vi brukerne om å forestille seg at de var sauebønder, og at alternativet til vårt system er å ta i bruk papirbaserte notater, slik som de fikk opplevd i begynnelsen av testen.

I kontekst av Seksjon 3.3 var dataen som ble generert fra dette steget i brukertestingen utfylte spørreskjemaer (engelsk: *questionnaires*).

14.4.4.4 Intervju

På slutten av hver brukertest ble det utført et *semi-strukturert* intervju [5, side 187] av brukeren. Et semi-strukturert intervju innebærer at intervjuet kan ta utgangspunkt i visse spørsmål, men at det ikke strengt tatt behøver å følge noen gitt struktur eller faste spørsmål.

Hvert intervju tok utgangspunkt i følgende spørsmål (med mulighet for at vi kom med tilleggsspørsmål basert på intervjuobjektets svar):

- Hva foretrakk du av å arbeide med papirnotater for å lage en oppsummeringsrapport, mot å jobbe med genererte PDF-rapporter? Forklar hvorfor.
- Hva likte du mest med systemet (webapplikasjonen)?
- Hva likte du minst med systemet (webapplikasjonen)?
- Har du noen øvrige tanker om systemet (webapplikasjonen)?

I sammenheng med Seksjon 3.3 var dataen som ble generert herfra svar på intervjuspørsmål.

14.4.5 Bearbeiding av resultater

Her går vi gjennom hva vi mente ville være viktig data å se på fra brukertestene. Selve resultatene fra brukertestene presenteres i Seksjon 17.

14.4.5.1 Utfylt papirskjema

For hver test tok vi tiden på hvor lang tid det tok å fylle ut oppsummeringsskjemaet. Snittverdien på hvor lang tid det tok for hver bruker å fylle ut arket var av interesse for oss, da dette ville gi en pekepinn på hvor effektivt (eller ineffektivt) bruk av dagens typisk papirbaserte løsning er. For hver test ble det også vurdert hvorvidt oppsummeringen brukeren skrev var «korrekt» med tanke på tellinger av sau, døde dyr og så videre i forhold til fasit.

14.4.5.2 Webapplikasjon-testoppgaver

Ettersom vi hadde en observatør som observerte testbrukeren mens de utførte oppgavene, fikk vi notert hva brukeren gjorde for å fullføre alle oppgavene. For hver oppgave ble måten brukeren løste den på notert, ettersom mange av oppgavene kunne fullføres på flere måter. Hvis brukeren tenkte høyt mens de utførte en oppgave ble hva brukeren sa notert. Alle notater i denne delen av brukertesten ble gjort i et regneark. Et eksempel på hvordan notatene så ut vises i Figur 73.

Oppgave nr #	Oppgavetekst	Notat
1	Som en uregistrert bruker, vil jeg opprette konto	Passordet hans var ikke langt nok. Deretter så manglet passordet hans tegn og tall osv.
2	Som en ny bruker, vil jeg registrere en gård som har dette navnet og denne fargen (viser bilde).	OK ("tror den er registrert")
3	Jeg ønsker å endre passord på kontoen min.	Prøvde først å klikke text input feltet en gang før han klikket på endre knappen. fikk melding om at passordet ikke var sterkt nok "quotes var ikke et tegn jeg kunne bruke som spesial tegn i passordet"
4	Som en innlogget bruker, vil jeg se listen over mine gårder.	OK
5	Jeg ønsker å finne ut om nabogården, som heter Melhusgården, finnes i systemet. Finn ut om denne gården eksisterer i systemet.	OK
6	Jeg ønsker å logge ut.	OK

Figur 73: Eksempel på hvordan notater ble tatt fra brukertesting av webapplikasjonen.

Når alle brukertestene var fullført, ble eventuelle avdekkede problemer med brukskvaliteten notert i en egen tabell.

14.4.5.3 SUS-skjema

For å få riktig SUS-poengsum måtte totalsummen av svarene først transformeres slik at man får en totalsum mellom 0-100. Her fulgte vi formelen som ble oppgitt i Brookes originale artikkel om SUS [104]:

> To calculate the SUS score, first sum the score contributions from each item. Each item's score contribution will range from 0 to 4. For items 1,3,5,7,and 9 the score contribution is the scale position minus 1. For items 2,4,6,8 and 10, the contribution is 5 minus the scale position. Multiply the sum of the scores by 2.5 to obtain the overall value of SU.
>
> SUS scores have a range of 0 to 100.

For hver fase med brukertesting så vi på snittverdien av SUS-poengsummen. Som diskutert i ikke-funksjonelt krav U2 (Tabell 28) hadde vi et mål på minimum 80 i snitt for SUS-poengsummen.

I tillegg til snittsummen av SUS-undersøkelsen som en helhet, valgte vi å også se på snittverdien til hvert enkelt spørsmål i SUS-undersøkelsen. Hensikten med dette var å forsøke å avdekke om det var noen spesielle aspekter av systemet som var spesielt problematiske og dermed bidro til å redusere SUS-poengsummen betydelig mye.

14.4.5.4 Intervjusvar

Intervjuene var semi-strukturerte, og vi var derfor mest interessert i å notere ned interessante innspil i form av sitater fra brukerne. Den ustrukturerte formen på intervjusvar betydde at vi ikke hadde noen konkret strategi for å behandle slik data på utenom å plukke ut de svarene som kom opp ofte eller vi syntes var mest interessante.

Del VI:

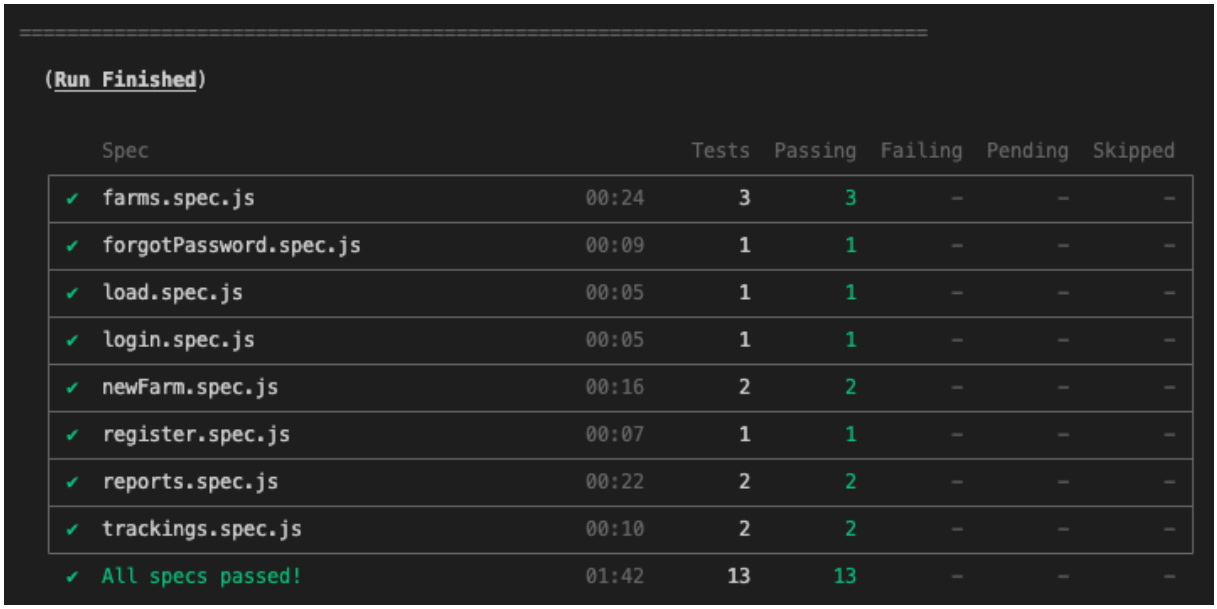
Resultater

Resultatene av testingen – end-to-end testing, akseptansetesting og brukertesting – vil bli presentert i denne delen. I tillegg til dette vil sikkerhetsvurderingen, som er en sikkerhetsanalyse av systemet, bli presentert her.

15 Resultater fra end-to-end testing

Resultatet av end-to-end testing med Cypress er i sin helhet veldig simpelt. Som beskrevet i testplanen består resultatet av en testrapport – denne testrapporten kan enten bli skrevet ut i konsollen eller skrevet til en fil etter at testene er ferdig. Denne rapporten oppsummerer simpelthen hvilke tester som kjørte med eller uten feil, noe som gjør det enkelt å oppdage hvilken funksjonalitet i applikasjonen som fungerer eller ikke fungerer.

Alt i alt så ble det skrevet åtte testspesifikasjoner med totalt 13 tester som tester funksjonaliteten til applikasjonen. Den endelige testrapporten fra end-to-end testingen etter at utviklingen var ferdig kan bli sett i Figur 74 og viser at all funksjonaliteten i applikasjonen fungerer som den skal.



(Run Finished)

	Spec		Tests	Passing	Failing	Pending	Skipped
✔	farms.spec.js	00:24	3	3	–	–	–
✔	forgotPassword.spec.js	00:09	1	1	–	–	–
✔	load.spec.js	00:05	1	1	–	–	–
✔	login.spec.js	00:05	1	1	–	–	–
✔	newFarm.spec.js	00:16	2	2	–	–	–
✔	register.spec.js	00:07	1	1	–	–	–
✔	reports.spec.js	00:22	2	2	–	–	–
✔	trackings.spec.js	00:10	2	2	–	–	–
✔	All specs passed!	01:42	13	13	–	–	–

Figur 74: Bilde av testrapporten skrevet ut i konsollen.

16 Resultater fra akseptansetesting

Akseptansetestingen ble utført på veilederen og forfatterne selv og består av de funksjonelle kravene i Seksjon 10.3.2 og de arkitektoniske betydelige kravene i Seksjon 10.3.3. De ikke-funksjonelle kravene har ikke blitt tatt med i akseptansetestingen ettersom de ikke er enkle å bekrefte ved at enkeltpersoner ser over og bruker applikasjonen.

Totalt sett endte vi opp med 29 tester – 26 tester av de funksjonelle kravene og 3 tester av de arkitektoniske betydelige kravene. Testene ble utført gjennom å sjekke at hvert krav ble oppfylt etter beskrivelsen til kravet. I motsetning til brukertestingen ble disse testene gjort kontinuerlig gjennom utviklingsfasen etter hvert som funksjonaliteten ble implementert, under veiledningsmøter mellom forfatterne og veilederen. Ved slutten av utviklingsfasen endte alle testene opp med å bli godkjent.

17 Resultater fra brukertesting

I denne seksjonen presenteres resultatene fra de to fasene med brukertesting. Mellom første og andre fase ble forbedringer gjort på produktet basert på tilbakemeldinger og observasjoner gjort under første fase av brukertestingen, med formål om å forbedre produktets brukskvalitet.

17.1 Resultater fra første fase brukertesting

Første fase med testing ble utført i slutten av mars. Etter første fase ble ulike problemer som ble oppdaget under testingen løst. Hvordan produktet ble endret på som et resultat av tilbakemeldingene og observasjonene fra denne fasen brukertesting, ble presentert i Seksjon 13.5.

17.1.1 Resultater fra test av bruk av papirskjemaer

Snittvarigheten på hvor lang tid det tok å fylle ut papirskjemaet var 5 minutter og 10 sekunder. Ettersom det var fire tilsyn som skulle oppsummeres, betyr dette at det tok mer enn ett minutt for hvert tilsyn som måtte oppsummeres.

I én av testene ble det gjort regnefeil i tellingen av totalt antall sau. Denne brukeren gjorde oppsummeringen av alle enkelttilsynene riktig, men totalsummen de kom fram til for antall sau stemte ikke.

17.1.2 Resultater fra test av bruk av webapplikasjonen

De oppgavene som førte til problemer, forvirring eller ikke lot seg løse av brukerne under brukertesting av webapplikasjonen presenteres her.

17.1.2.1 Bruker skjønner ikke umiddelbart at man må trykke på «endre»-knappen

Oppgave ID	Oppgavebeskrivelse
3	Jeg ønsker å endre passord på kontoen min.

Tabell 46: Brukertest av webapplikasjon - oppgave 3

Ettersom passordendring er noe man vil forhindre brukeren å gjøre med et uhell, må man først trykke på «Endre»-knappen for at inputfeltet der man skriver nytt passord skal aktiveres. Før dette gjøres er inputfeltet synlig, men deaktivert (engelsk: *disabled*). Brukeren forsøkte å bruke inputfeltet flere ganger før de skjønte at de måtte bruke «Endre»-knappen først.

17.1.2.2 Problem med tooltip

Oppgave ID	Oppgavebeskrivelse
12	Jeg ønsker å kun se saueflokker på kartet.

Tabell 47: Brukertest av webapplikasjon - oppgave 12

«Tooltips» kommer fram når brukeren holder musepekeren over filter-knappene for å skru av/på visning av elementer på kartet. Dette er forklarende tekst som sier hva knappen gjør. Disse tooltipene dekker over filterknappen under. Om brukeren går inn for å skru av visning av alt og beveger seg i rekkefølge fra topp til bunn, vil tooltipet som kommer fram stadig blokkere filterknappen ett nivå under.

17.1.2.3 Bruker skjønte ikke at de kunne klikke på markører på kartet

Oppgave ID	Oppgavebeskrivelse
13	For hver saueflokk på kartet ønsker jeg å vite om antall lam stemmer, eller om noen mangler.

Tabell 48: Brukertest av webapplikasjon - oppgave 13

Intensjonen med denne oppgaven var å få brukeren til å klikke på saueflokk-markørene på kartet, ettersom dette lar brukeren se informasjon om hver enkelt saueflokk. Én bruker brukte mye tid til å forsøke å finne denne informasjonen i sidebaren, men etter en stund innså de at de kunne klikke på markørene på kartet.

17.1.2.4 Å endre navn på gårder er en forvirrende prosess

Oppgave ID	Oppgavebeskrivelse
30	Gården som tilhører min bruker, med navnet «Støjrdalsgården», er stavet feil. Jeg ønsker å gjøre det så gården har riktig navn.

Tabell 49: Brukertest av webapplikasjon - oppgave 30

Det eksisterte ingen god flyt for å endre navnet på en gård. Hver gård har gårdsnavnet som unik identifikator, og det var derfor ikke mulig å direkte redigere navnet på en gård (dette var en teknisk begrensning). Istedenfor måtte brukeren slette gården og legge den til på nytt med riktig navn. Dette er en ikke-intuitiv flyt.

17.1.2.5 Krav for endring av passord samsvarer ikke med krav for registrering/innlogging

Oppgave ID	Oppgavebeskrivelse
32	Jeg vil inn på brukeren jeg først opprettet. Men først må nullstille passordet, fordi jeg har glemt passordet.

Tabell 50: Brukertest av webapplikasjon - oppgave 32

Når en bruker oppretter en ny konto er det visse krav for passordet når det kommer til lengde, påkrevde tegn og så videre. Når brukeren var på siden for å tilbakestille passordet var ikke disse kravene implementert, som betyr at det var tillatt å endre til et passord som ikke er gyldig - og brukeren blir derfor sperret fra å logge inn med det nye passordet. Dette ble forårsaket av at å nullstille passordet i Firebase skjer gjennom en side som er forhåndsgenerert av Firebase og lenkes til brukeren via en e-post. Denne siden har ingen begrensinger på hvilke tegn som er tillatt for det nye passordet.

17.1.2.6 Tegnkrav for passord er forvirrende

Dette brukbarhetsproblemet oppstod også i oppgave 32, som vist i Tabell 50. Til forskjell fra brukbarhetsproblemet diskutert i Seksjon 17.1.2.5, var ikke problemet her opplevd ved nullstilling av passordet, men når brukeren skulle logge inn. Brukeren ga tilbakemelding på at de syntes kravene for hvilke spesialtegn som var tillatt i passordet var forvirrende.

Grunnen til dette var at tegnene som var tillatt var omringet av anførselstegn (« »). Dette fikk brukeren til å tro at anførselstegnet var ett av de tillatte tegnene.

17.1.2.7 Redigering av gårdsfarge fungerte ikke

Oppgave ID	Oppgavebeskrivelse
33	Det viser seg at gården jeg har lagt til i systemet, har feil farger oppgitt. Den ble lagt til med én farge, men skal egentlig være tofarget. Her er de korrekte fargene for denne gården (hold opp bilde). Gjør nødvendig korrigering på denne gården.

Tabell 51: Brukertest av webapplikasjon - oppgave 33

Brukeren fikk i oppgave å redigere fargen på gården så den gikk fra å ha én spesifikk farge til å ha to farger. Det var en teknisk feil i systemet som gjorde at den gamle fargen ble bevart, slik at brukeren satt fast når de forsøkte å endre fargene. Dette scenariet, med feil når man går fra én farge på en gård til to farger, klarte vi ikke avdekke tidligere i vår egen testing.

17.1.3 Resultater fra SUS-skjema

Spm. nr	Spørsmål	Snittverdi av svar
1	Jeg kunne tenke meg å bruke dette systemet ofte.	4,5
2	Jeg synes systemet var unødvendig komplisert.	2
3	Jeg synes systemet var lett å bruke.	4,5
4	Jeg tror jeg vil måtte trenge hjelp fra en person med teknisk kunnskap for å kunne bruke dette systemet.	2
5	Jeg syntes at de forskjellige delene av systemet hang godt sammen.	4,5
6	Jeg syntes det var for mye inkonsistens i systemet. (Det virket «ulogisk»)	2
7	Jeg vil anta at folk flest kan lære seg dette systemet veldig raskt.	4,5
8	Jeg synes systemet var veldig vanskelig å bruke	1,5
9	Jeg følte meg sikker da jeg brukte systemet.	4,5
10	Jeg trenger å lære meg mye før jeg kan komme i gang med å bruke dette systemet på egen hånd.	2

Tabell 52: Oppsummering av resultater fra SUS-skjemaene utfylt etter hver brukertest i fase én av brukertesting. For svarene vil tallverdien 1 representere «sterkt uenig», og tallverdien 5 representere «sterkt enig».

Snittet for totalscoren til SUS-undersøkelsen var **82,5**.

17.1.4 Svar fra intervju etter bruk av webapplikasjonen

Noe som kom tydelig fram i intervjuene var at brukerne syntes å jobbe med digitalt genererte rapporter var en mye bedre opplevelse enn å jobbe med papirnotater. De la vekt på automatikken i å generere PDFer i systemet og syntes arbeid med papirnotater var ineffektivt og ikke spesielt oversiktlig. Én bruker nevnte at de mislikte å jobbe med matte (addisjon) for å regne ut totalsummer av ulike antall observasjoner.

Når det kommer til positive tanker brukerne hadde om systemet pekte de på at de likte flyten i grensesnittet - brukerne mente i stor grad at skjermelementer var logisk plassert og at informasjon var lett å finne fram til.

Den viktigste negative tilbakemeldingen brukerne hadde når det kom til systemet var knyttet til passord. Flere av brukbarhetsproblemene var knyttet til å lage et nytt passord. Én bruker var mindre positiv til at tilgang til viss informasjon krevde interaksjon med kartet – de sa de var mer vant til at kartet kan navigeres rundt på, men at ting på kartet ikke kan klikkes på for å avsløre mer informasjon.

17.2 Resultater fra andre fase brukertesting

Andre fase med testing ble utført i midten av april og produktet som da ble testet var en forbedret versjon basert på tilbakemeldingene fra første fase med testing – i tråd med vårt fokus på en iterativ prosess som beskrevet i Seksjon 9.

17.2.1 Resultater fra test av bruk av papirskjemaer

Snittvarigheten for å fylle ut papirskjemaet var 5 minutter og 30 sekunder. I likhet med første fase testing betyr det at brukerne brukte over ett minutt per tilsyn som skulle oppsummeres. I denne fasen av testing gjorde ingen av brukerne regnefeil.

17.2.2 Resultater fra test av bruk av webapplikasjonen

I denne fasen av brukertesting var det nesten ingen konkrete problemer med grensesnittet som enten stoppet brukerflyten eller forhindret testbrukerne fra å utføre en testoppgave. Det eneste som ble observert av problemer var at én bruker, under utførelse av testoppgave 33 (Tabell 53), skulle bekrefte fargeendringen på gården. Istedenfor å trykke på «Lagre»-knappen for å bekrefte endringen, trykket de instinktivt på «Endre»-knappen istedenfor. Denne knappens funksjon er å gjøre tekstfeltet for å endre navn på gården brukbart, ettersom før denne knappen trykkes er feltet skrudd av (engelsk: *disabled*) og tar ikke imot tastetrykk fra brukeren. Problemet illustreres i Figur 75.

Tabell 53: Brukertest av webapplikasjon - oppgave 33

Oppgave ID	Beskrivelse
33	Det viser seg at gården jeg har lagt til i systemet, har feil farger oppgitt. Den ble lagt til med én farge, men skal egentlig være tofarget. Her er de korrekte fargene for denne gården (hold opp bilde). Gjør nødvendig korrigering på denne gården.

.

Figur 75: Bilde av skjermen for å redigere en eksisterende gård - Knappen markert med «1» brukes for å låse opp skrivefeltet for å endre navn på gården. Knappen markert med «2» brukes for å bekrefte og lagre endringene som er gjort på gården.

«Endre»-knappen er større enn de andre knappene og har blå bakgrunn, som kan gi brukeren assosiasjoner til å «sende inn» (engelsk: *submit*). Dermed er det sannsynlig at denne stjal oppmerksomheten til brukeren og fikk dem til å trykke på feil knapp.

17.2.3 Resultater fra SUS-skjema

Spm. nr.	Spørsmål	Snittverdi av svar
1	Jeg kunne tenke meg å bruke dette systemet ofte.	5
2	Jeg synes systemet var unødvendig komplisert.	1.5
3	Jeg synes systemet var lett å bruke.	4,5
4	Jeg tror jeg vil måtte trenge hjelp fra en person med teknisk kunnskap for å kunne bruke dette systemet.	1
5	Jeg syntes at de forskjellige delene av systemet hang godt sammen.	5
6	Jeg syntes det var for mye inkonsistens i systemet. (Det virket «ulogisk»)	1
7	Jeg vil anta at folk flest kan lære seg dette systemet veldig raskt.	4,5
8	Jeg synes systemet var veldig vanskelig å bruke	1
9	Jeg følte meg sikker da jeg brukte systemet.	5
10	Jeg trenger å lære meg mye før jeg kan komme i gang med å bruke dette systemet på egen hånd.	1

Tabell 54: Oppsummering av resultater fra SUS-skjemaene utfylt etter hver brukertest i fase to av brukertesting. For svarene vil tallverdien 1 representere «sterkt uenig», og tallverdien 5 representere «sterkt enig».

Snittet for totalscoren til SUS-undersøkelsen i fase to av testingen var **96,25**.

17.2.4 Svar fra intervju etter bruk av webapplikasjonen

Intervjuene utført i andre fase av brukertesting ga mange av de samme svarene som i første fase. Også i denne fasen av testing svarte testbrukerne at de synes bruken av digitale rapporter var til å foretrekke over papirbaserte notater, og brukerne trakk fram utseendet og brukerflyten på grensesnittet som positive aspekter.

Ved spørsmål om negative aspekter ved systemet eller aspekter som ble likt minst, kom testbrukerne ikke med noen konkrete svar.

18 Sikkerhetsvurdering

Sikkerheten til digitale systemer er et stort tema og det finnes svært mange potensielle sikkerhetshull systemer kan være utsatt for. Ettersom systemet vårt ikke skal settes ut i produksjon er det begrenset hvor langt vi vil gå i å analysere alle disse mulige sikkerhetshullene i en sikkerhetsvurdering. I denne seksjonen har vi valgt å bruke OWASP Top 10-listen for 2021 som grunnlag for vår sikkerhetsvurdering.

Open Web Application Security Project (OWASP) er en stiftelse med formål om å tilby ressurser for å fremme applikasjonssikkerhet [107]. OWASP gir blant annet ut «OWASP Top Ten», som er en liste over de ti sikkerhetshullene som de anser som de mest kritiske for webapplikasjoner. OWASP Top Ten-listen oppdateres årlig, og den nyeste utgaven er OWASP Top Ten 2021.

> The OWASP Top 10 is a standard awareness document for developers and web application security. It represents a broad consensus about the most critical security risks to web applications. [108]

Punktene i topp ti-listen er kategorier av sikkerhetshull, og innenfor hver kategori er det mange ulike sikkerhetsproblemer som alene eller sammensatt kan forårsake det øvrige sikkerhetshullet for den kategorien. I denne sikkerhetsvurderingen vil vi se på hver av de ti kategoriene i topp ti-listen og vurdere hvorvidt vårt system kan være utsatt for disse problemene. Ettersom en kategori i OWASPs topp ti-liste kan ha svært mange mulige årsaker, har vi ikke mulighet til å analysere om hver eneste potensielle årsak til sikkerhetshullet kan være tilstede i vår kodebase.

OWASP Top Ten 2021 består av følgende kategorier [108]:

1. A01:2021-Broken Access Control
2. A02:2021-Cryptographic Failures
3. A03:2021-Injection
4. A04:2021-Insecure Design
5. A05:2021-Security Misconfiguration
6. A06:2021-Vulnerable and Outdated Components
7. A07:2021-Identification and Authentication Failures
8. A08:2021-Software and Data Integrity Failures
9. A09:2021-Security Logging and Monitoring Failures
10. A10:2021-Server-Side Request Forgery

18.1 A01:2021-Broken Access Control

> Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification, or destruction of all data or performing a business function outside the user's limits. [109]

Som diskutert i Seksjon 13.2 krever skrive- og slettehandlinger på data at brukeren er autentisert gjennom Firebase Authentication og at brukeren har eierskap over dataen det gjelder. Firebase Authentication tar selv hånd om å sikre at brukerens innlogging ikke har gått «ut på dato» og dette er ikke noe vi som utviklere behøver å ta for oss. Som også forklart i Seksjon 13.2 er en ulempe med å bruke Firebase Firestore som database at den ikke støtter svært avanserte spørringer, som betyr at vi må gjøre alle tilsynene leselige av alle innloggede brukere (det betyr likevel ikke at en bruker på webapplikasjonen kan se alle andres tilsyn, men om en teknisk kompetent aktør går inn for å forsøke å hente ut disse er det ingen sikkerhetsregel som forhindrer det). Et tilsyn har en referanse til bruker-IDen til brukeren som utførte tilsynet, men det er ikke mulig å benytte denne IDen til å uthente mer informasjon om brukeren.

18.2 A02:2021-Cryptographic Failures

> The first thing is to determine the protection needs of data in transit and at rest. For example, passwords, credit card numbers, health records, personal information, and business secrets require extra protection, mainly if that data falls under privacy laws, e.g., EU's General Data Protection Regulation (GDPR), or regulations, e.g., financial data protection such as PCI Data Security Standard (PCI DSS). [110]

Med «data in transit» menes data som overføres mellom klient (i vårt tilfelle webapplikasjonen kjørende på brukerens maskin) og tjener (i vårt tilfelle Firebase). Kommunikasjon mellom disse er krypert via Transport Layer Security (TLS)-protokollen ettersom Firebase Hosting automatisk ordner et TLS-sertifikat og formidler kommunikasjon mellom tjener og klient over TSL, uten krav om oppsett fra vår side. Dermed er det ikke mulig for utenforstående å lese dataen som overføres, såkalt *eavesdropping*, ettersom den er kryptert.

Med «data at rest» menes data som er på tjenersiden, i vårt tilfelle i Firebase Firestore. Denne dataen er kryptert av Firebase uten at dette er noe vi må konfigurere selv:

> Firestore automatically encrypts all data before it is written to disk. There is no setup or configuration required and no need to modify the way you access the service. The data is automatically and transparently decrypted when read by an authorized user. [111]

Kontodetaljer (brukernavn og passord) håndteres av Firebase Authentication og er ikke data vi selv har tilgang til. Å sikre innloggingsdataen til brukerkontoer er derfor Firebase Authentication sitt ansvar, og det er ikke flere grep vi kan ta for å sikre denne informasjonen.

18.3 A03:2021-Injection

Vi benytter oss av Firebase Firestore som database, og denne er ikke utsatt for SQL-injeksjon ettersom det er en NOSQL-database og «spørringer» som kan modifiseres av brukerinput er altså ute av bildet. Injeksjonsangrep på operativsystem-nivå (omtalt som *command injection* [112]) er Firebase sitt ansvar ettersom vi ikke har noen form for kontroll over maskinvaren eller mer spesifikt operativsystemet de kjører.

Cross-site scripting (XSS)-angrep [113] er vi beskyttet mot ettersom vi bruker React uten å bruke dets XSS-utsatte funksjoner slik som *dangerouslySetInnerHTML* eller å endre DOM-en direkte via å modifisere *document*-objektet eller et *Ref*-objekt. React skal i stor grad være beskyttet mot slike angrep så lenge man ikke bruke slike «farlige» mekanismer.

18.4 A04:2021-Insecure Design

Denne kategorien er ment til å kritisere webapplikasjoner som ikke har sikkerhetsmekanismer eller sikkerhetsprinsipper integrert i arkitekturen. Dette er i motsetning til de andre kategoriene, som dreier seg om usikker implementasjon.

> Insecure design is a broad category representing different weaknesses, expressed as “missing or ineffective control design.” Insecure design is not the source for all other Top 10 risk categories. There is a difference between insecure design and insecure implementation. We differentiate between design flaws and implementation defects for a reason, they have different root causes and remediation. A secure design can still have implementation defects leading to vulnerabilities that may be exploited. An insecure design cannot be fixed by a perfect implementation as by definition, needed security controls were never created to defend against specific attacks. [114]

Et viktig prinsipp innenfor Insecure Design-kategorien er at systemets sikkerhet burde tilpasses nivået av «business risk» knyttet til systemet. For eksempel vil noens personlige blogg ha mindre krav til sikkerhet enn en nettbank, ettersom konsekvensene av for eksempel datalekkasje eller kontoinnbrudd er mye større

for nettbanken. Risiko kan måles i for eksempel tap av tillit og omdømme eller bøter for lekkasje av kundeopplysninger.

Ettersom vi benytter oss av diverse Firebase-produkter for backend-systemet vårt, tar dette hånd om mange sikkerhetsmekanismer for oss, slik som brukerautentisering og kryptering av lagret data som nevnt tidligere. Som diskutert i Seksjon 13.2 har vi også sikkerhetsregler som forhindrer brukere fra å redigere eller slette dokumenter de ikke har eierskap over. Til sist må det nevnes at systemet ikke skal tas i bruk i produksjon, og dette setter en begrensing for hvor kritisk det hadde vært om eventuelle sikkerhetshull ble oppdaget og utnyttet. Med dette tatt i betrakning synes vi at sikkerhet har blitt med i designet av systemet i tilstrekkelig grad.

18.5 A05:2021-Security Misconfiguration

Denne kategorien dreier seg om diverse feilkonfigurasjoner på back-end som åpner for sikkerhetshull. Dette kan for eksempel være at webserveren er kjørende med serverens «default» kontoer skrudd på, slik at angripere kan komme seg inn via disse kontoene med kjente brukernavn og passord. Andre problemer kan være at programvare, som for eksempel en database, er satt på «default» innstillinger som ofte ikke er trygge for et system som kjøres i produksjon.

Firebase krever ikke noen spesiell konfigurasjon utenom sikkerhetsregler for lesing og skriving til databasen. Firebase er sikret «out of the box», og vi er derfor ikke utsatt for denne kategorien av sikkerhetshull.

18.6 A06:2021-Vulnerable and Outdated Components

Denne kategorien dreier seg om tredjepartskomponenter i systemet. I vårt system er Mapbox (kart) og React (front-end rammeverk) eksempler på slike komponenter. Eldre komponenter kan være mer utsatt for sikkerhetshull fordi angripere har hatt god tid til å utforske de og spre kunnskap om hullene, og det er derfor viktig å benytte siste versjonen av tredjepartskomponenter for å holde tritt med de siste sikkerhetsoppdateringene. Det er spesielt utdaterte tredjepartskomponenter i backend-systemer som åpner muligheten for sikkerhetshull. Ettersom vi bruker Firebase som et backend-system, er det Firebase sitt ansvar å holde programvaren de benytter seg av oppdatert. Komponentene vi bruker på frontend dreier seg stort sett om det visuelle grensesnittet og det er begrenset hva slags sikkerhetstrussel de utgjør hvis de er utdaterte. I hovedsak er det kun XSS-angrep disse komponentene kan være utsatt for. Ettersom komponentene vi bruker har åpen kildekode anser vi det som sannsynlig at en slik trussel ikke er til stede i komponentene vi bruker, da et slikt tilfelle eventuelt hadde blitt varslet om blant vedlikeholderne av disse bibliotekene. Et unntak er Firebase-biblioteket for JavaScript som knytter webapplikasjonen vår opp mot Firebase-backenden – denne burde holdes oppdatert ettersom den knytter webapplikasjonen til diverse Firebase-tjenester. For å ivareta god sikkerhet er det derfor viktig å sørge for at denne komponenten av systemet holdes oppdatert.

18.7 A07:2021-Identification and Authentication Failures

Denne kategorien tar for seg diverse sikkerhetshull knyttet til autentisering og hvordan brukerøkter håndteres etter innlogging. Mulige problemer innenfor denne kategorien er for eksempel:

- Systemet tillater korte eller svake passord
- Systemet lagrer passord som ren tekst
- Systemet tillater «brute force»-angrep der svært mange påloggingsforsøk blir gjort i løpet av kort tid i håp om at brukernavn og passord eventuelt samsvarer med en eksisterende konto

- Systemet logger ikke brukeren ut automatisk etter en gitt tidsperiode

Vi benytter oss av Firebase Authentication for brukerautentisering og for å håndtere innloggede økter. Firebase Authentication er en helhetlig løsning for brukerhåndtering og -autentisering, og tar hånd om alle de overnevnte problemene selv. Firebase Authentication setter blant annet krav på passordstyrke ved bruk av e-post+passord-autentisering. Ved for mange feilede påloggingsforsøk blir brukeren midlertidig låst ute fra innloggingsfunksjonen, og brukeren blir også automatisk logget ut etter en gitt tidsperiode.

Firebase Authentication er en moden og trygg løsning for brukerautentisering, og vi konkluderer derfor med at vi ikke er utsatt for sikkerhetshull innenfor denne kategorien.

18.8 A08:2021-Software and Data Integrity Failures

> Software and data integrity failures relate to code and infrastructure that does not protect against integrity violations. An example of this is where an application relies upon plugins, libraries, or modules from untrusted sources, repositories, and content delivery networks (CDNs). [115]

Dette sikkerhetsproblemet oppstår med kode som kommer fra eksterne kilder, for eksempel JavaScript-biblioteker. En hacker kan for eksempel ta over et publisert JavaScript-bibliotek og legge til skadelig kode i en oppdatering. Alle bibliotek som igjen er avhengige av dette biblioteket vil bli påvirket av en slik oppdatering hvis de velger å oppgradere til nyeste versjon av biblioteket. Et slikt scenario utspilte seg med biblioteket 'coa' publisert på npm [116], som er den mest populære pakkebehandleren (engelsk: *package manager*) for JavaScript.

Fordi vi har en Firebase-backend behøver vi kun å tenke på slike sikkerhetsproblemer for frontend-koden, som består av JavaScript-pakker og -biblioteker installert via npm. Noen måter å beskytte seg mot slike «kaprede» pakker er å:

- utsette oppdateringer; unngå å installere oppdateringer som akkurat ble publisert
- bruke de mest populære bibliotekene tilgjengelig, som da har flere «øyne» til å vurdere om de kan ha tatt inn skadelig kode
- begrense bruken av eksterne pakker og biblioteker
- selv lese gjennom kildekoden til pakken som ble oppdatert for å vurdere om det ble innført skadelig kode

Ettersom pakkene våre for det meste kun er for grensesnittet til webapplikasjonen, som for eksempel fargevelgeren vist i Figur 52, er det ikke noen sikkerhetsmessig ulempe å la være å oppdatere disse pakkene. Ved å la være å oppdatere pakker og biblioteker unødvendig vil man i stor grad unngå dette sikkerhetsproblemet.

18.9 A09:2021-Security Logging and Monitoring Failures

> [...] this category is to help detect, escalate, and respond to active breaches. Without logging and monitoring, breaches cannot be detected. [117]

Å loggføre handlinger i systemet er et viktig sikkerhetstiltak. Det tilrettelegger for at de som vedlikeholder systemet har oversikt over hvordan systemet brukes, og det kan gi nødvendig informasjon om at et cyberangrep tar sted eller annen uønsket aktivitet foregår.

Firebase produserer ingen hendelseslogg. Det er mulig å bruke *Firebase Cloud Functions* til å selv sette opp logging på spesifikke handlinger, for eksempel innlogging eller når nye tilsyn registreres. En stor ulempe er at bruken av Firebase Cloud Functions ikke er tilgjengelig når man tar i bruk en kostnadsfri versjon av Firebase[22]. For å få tilgang til Cloud Functions må man gå over til en såkalt «pay-as-you-go»-plan. Fordi vi ikke er villige til å registrere et kredittkort og bruke penger på Firebase-tjenestene, har vi ikke tilgang til Cloud Functions. Vi har derfor ikke mulighet for å sette opp logging, som er en ulempe fra et sikkerhetsperspektiv. Fordi systemet ikke settes i faktisk produksjon er ikke mangelen på logging problematisk for oss, men om systemet skulle blitt brukt i produksjon ville det vært nødvendig å sette opp en form for logging.

18.10 A10:2021-Server-Side Request Forgery

> SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL. [118]

Webapplikasjonen tar hverken imot eller behandler brukerinnsendte URL-er, så vårt system er ikke utsatt for denne typen angrep.

18.11 Konklusjon av sikkerhetsvurdering

Alt i alt er webapplikasjonen i en god tilstand når det kommer til sikkerhet. Dataintegritet er sikret gjennom sikkerhetsregler for Firebase Firestore-databasen, og brukerkontoene er beskyttet gjennom at de er bygget på Firebase Authentication-løsningen som er en «ferdig» løsning for å håndtere brukerkontoer. Å benytte Firebase som backend-system gir mange «gratis» fordeler når det kommer til sikkerhet, ettersom det stort sett er forhåndskonfigurert for å være trygt, og man slipper derfor å jobbe for å forhindre de mest kritiske og utbredte sikkerhetshullene en webapplikasjon kan ha. Den største ulempen er mangelen på logging, som vil kreve å oppgradere til en betalt Firebase-plan som krever registrering av et kredittkort. En form for logging burde implementeres hvis systemet skal settes i produksjon.

[22] Firebase Pricing https://firebase.google.com/pricing#authentication

Del VII:

Diskusjon

I denne delen vil vi diskutere og evaluere diverse aspekter rundt prosessen og produktet. Vi vil også besvare forskningsspørsmålene for prosjektet.

19 Diskusjon rundt brukertestresultater

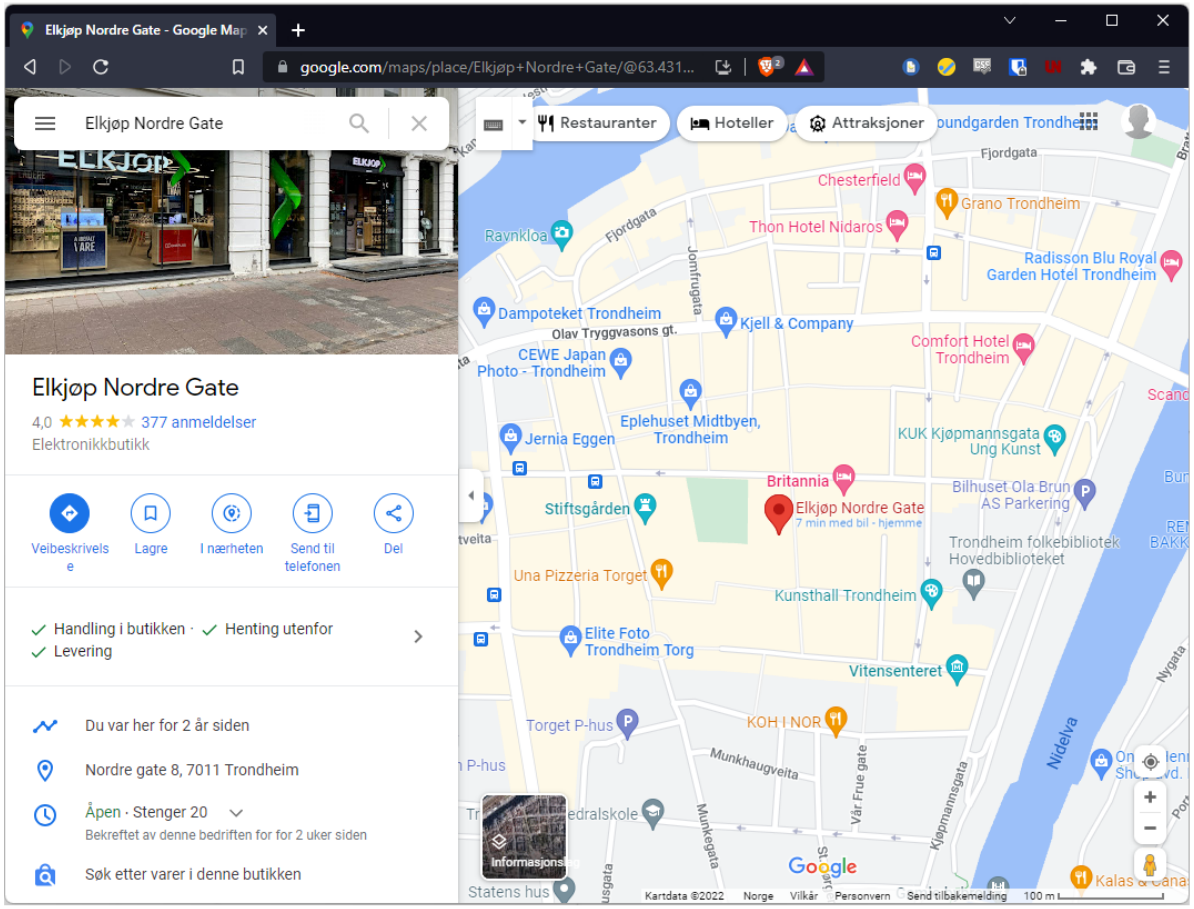
I denne seksjonen diskuterer vi funnene fra første og andre fase med brukertesting.

19.1 Resultater fra første fase testing

Ved utfylling av oppsummeringsskjema for hånd på papir var snittvarigheten på over ett minutt per tilsyn som måtte oppsummeres. I tilsynsskjemaet som brukeren måtte oppsummere var det fire tilsyn, som er svært lite og et langt lavere antall enn hvor mange tilsyn som faktisk utføres i løpet av en beitesesong. Dette betyr at i et mer realistisk scenario med flere utførte tilsyn vil tiden det tar å lage en oppsummering for hånd være lenger. Det skal også nevnes at det var relativt få punkter å oppsummere per tilsyn i vårt eksempelskjema med notater (vist i Vedlegg E), ettersom alt ble skrevet med stor blokkskrift. I et faktisk bruksscenario av et papirskjema er det mulig at langt flere notater tas per tilsyn, som vil øke tiden det tar å oppsummere tilsynet.

Selv om det var svært få tilsyn å oppsummere, var det fortsatt slik at én testbruker gjorde en regnefeil i skjemaet sitt, slik at totalsummen for sau ikke stemte. Det er tenkelig at med flere tilsyn som må oppsummeres vil sannsynligheten for å gjøre regnefeil øke. Dette avslører en svakhet ved bruk av papirskjemaer. Hvor stor betydning en regnefeil eventuelt vil få for den enkelte sauebonden eller lokale myndigheter som mottar skjemaet har vi ikke mulighet til å spå. Det at regnefeil så lett kan oppstå er uansett en åpenbar svakhet med bruk av papirskjemaer.

Én testbruker ga under intervjuet tilbakemelding på at de ikke er vant til å finne fram til informasjon ved å klikke seg fram i kartet. Dette er et punkt som vi som utviklere noterte, men valgte å ikke endre på. Prosjektmedlemmene er begge vant til å interagere med kart på Google Maps og å trykke på interaktive markører for å vise tilleggsinformasjon, som for å for eksempel se åpnings- og stengetiden til en butikk ved å trykke på markøren på kartet. Vi vil derfor argumentere for at slik interaksjon med et kart er langt fra uvanlig. Et eksempel på hvordan Google Maps presenterer informasjon vises i Figur 76.



Figur 76: Bilde av Google Maps - Sidebaren viser informasjon om det brukeren har klikket på.

Alternativet til klikkbare markører hadde vært at markørene ikke var klikkbare, og at all informasjonen heller måtte vises som en del av sidebaren. Dette hadde antagelig gjort grensesnittet rotete og uoversiktlig ettersom det er en del informasjon som kan vises fra hver enkel markør (enkeltinformasjon om en saueflokk, et rovdyr, en død sau eller en «annet»-registrering). Denne informasjonen er i hovedsak for rapportene som genereres og ikke ment til å ta all oppmerksomheten til brukeren når de ser gjennom et tilsyn på webapplikasjonen – brukeren må heller navigere seg fram til informasjonen ved å selv trykke på markørene for å vise informasjonen. Vi valgte derfor å se bort i fra denne tilbakemeldingen med det første. Om andre brukere kom med lignende synspunkter i framtidig testing, ville vi tatt en titt på hvordan det eventuelt kunne blitt håndtert.

Problemet beskrevet i Seksjon 17.1.2.1 ble ikke korrigert i endringene som ble gjort etter første brukertest. Dette var fordi vi var usikre på om dette egentlig var et reellt problem, og ville ha mer innsikt i om andre framtidige testbrukere også kom til å støte på dette problemet. Vi lot derfor denne knappen være og ventet på resultater fra andre fase brukertesting for å se om problemet også oppstod der (noe det ikke gjorde).

19.2 Resultater fra andre fase testing

Med en økning fra 82,5 i gjennomsnittlig SUS-poengsum i første fase til 96,25 i andre fase testing, er det tydelig at den opplevede brukskvaliteten har økt som et resultat av endringene gjort etter tilbakemeldingene fra første fase testing.

Fraværet av tekniske og brukerflyt-relaterte problemer i andre fase sammenlignet med første betydde at brukerne ikke hadde noen konkrete negative aspekter å trekke fram i intervjuene. Dette er sannsynligvis fordi mulige gjenstående problemer i så fall vil være knyttet til hva slags brukskvalitet produktet har innenfor faktisk sauegjeting og tilsyn under beitesesongen, som vi ikke får testet under utførelsen av prosjektet. Om beitesesongen hadde startet og vi hadde hatt et utvalg av sauebønder å teste produktet på, kunne dette vært en god kilde til tilbakemeldinger som går mer i dybden.

20 Evaluering av løsningen

Her vil vi avgjøre hvorvidt de ulike kravene som ble presentert i Seksjon 10 har blitt tilfredsstilt. Disse kravene består av de arkitektoniske betydelige kravene (Seksjon 20.1), bruksmønstre og funksjonelle krav (Seksjon 20.2) og til sist de ikke-funksjonelle kravene (Seksjon 20.3). I Seksjon 20.4 diskuterer vi hva slags effekt bruk av konkrete krav for systemet har hatt på utviklingen.

20.1 Arkitektoniske betydelige krav

I denne underseksjonen gjennomgår vi de arkitektoniske betydelige kravene (ABK-er) som ble presentert i Seksjon 10.3.3.

20.1.1 Skal være enkel å bruke på datamaskiner

Løsningen vår er en webapplikasjon utviklet i React. Dette betyr at løsningen vil kjøre på enhver enhet gitt at den har en nettleser som kan kjøre en moderne versjon av JavaScript. I praksis betyr dette at webapplikasjonen vil kjøre på alle moderne nettlesere. Vi kan derfor konkludere med at ABK-en oppgitt i Seksjon 10.3.3.1 er innfridd.

20.1.2 Interoperabilitet med eksisterende applikasjoner

ABK-en oppgitt i Seksjon 10.3.3.2 er bedre konkretisert gjennom det ikke-funksjonelle kravet om interoperabilitet, som blir presentert i Tabell 26. Som vil bli presentert i Seksjon 20.3.2 er det altså slik at dette ikke-funksjonelle kravet er oppnådd, og vi kan konkludere med at denne ABK-en derfor er tatt hensyn til.

20.1.3 Data skal ha god integritet, konfidensialitet og tilgjengelighetsbeskyttelse

ABK-en gitt i Seksjon 10.3.3.3 innebærer at dataen i systemet skal ha god integritet, konfidensialitet og tilgjengelighetsbeskyttelse.

Dataintegritet oppnås i form av sikkerhetsregler i Firebase Firestore, som ble presentert i Seksjon 13.2. Disse sikrer blant annet at dataen som lagres må bestå av riktige feltnavn og forventede dataverdier. Datakorrupsjon- og tap forhindres gjennom at Firebase replikerer dataen over flere datasentre:

> Cloud Firestore brings you the best of Google Cloud's powerful infrastructure: **automatic multi-region data replication**, strong consistency guarantees, atomic batch operations, and real transaction support [...] [119].

Tilgjengelighetsbeskyttelse (engelsk: *availability*) håndteres av Firebase gjennom replikering av lagret data og at de som nevnt tar i bruk flere datasentre, som sørger for å forhindre nedetid om for eksempel ett av datasentrene opplever et strømbrudd.

Den største svakheten til systemet er konfidensialiteten rundt tilsynsdata, ettersom alle innloggede brukere i følge sikkerhetsreglene kan lese alle tilsynsdataene fra alle brukere. Årsaken til dette ble forklart i Seksjon 13.2. Hva dette innebærer for personvernet, samt en potensiell løsning, vil diskuteres videre i Seksjon 21.2.2. Ved å sikre at brukerdata ikke er tilgjengelig i selve tilsynet minimerer vi graden av hvorvidt dette egentlig er et personvernsproblem, men det hadde likevel vært optimalt å kun vise de tilsynene hvor brukerens gård(er) er representert og ikke gi noen form for lesetilgang til resten av tilsynene. Hvilke grep en 'angriper' må ta for å lese data fra tilsyn som ikke omhandler deres gårder er vi ikke kjent med, ettersom å inspisere Firebase-dataen brukerklienten mottar med verktøy tilgjengelig i nettleseren (gjennom å inspisere nettverksaktiviteten) kun avslører obfuskert JavaScript-kode som vil kreve egne verktøy og ferdigheter for å analysere videre. Likevel, ettersom brukeren har lesetilgang til denne koden, kan vi anta at det gitt nok innsats vil være mulig for en bruker å lese denne dataen.

Vi har gjort det vi kan for å sikre konfidensialiteten rundt dataen i systemet og mener den er tilfredsstillende, gitt at dette ikke er tiltenkt som et produksjonsklart system. Om systemet skulle settes i produksjon ville dette konfidensialitetsaspektet med tilsynsdata vært det viktigste å få forbedret før systemet settes i bruk.

20.2 Bruksmønstre og funksjonelle krav

Gjennom akseptansetesting har vi bekreftet at alle bruksmønstre og funksjonelle krav introdusert i henholdsvis Seksjon 10.3.1 og Seksjon 10.3.2 er innfridd gjennom løsningen vår. I tillegg er brukertestoppgavene knyttet til webapplikasjonen, presentert i Seksjon 14.4.4.2, dekkende for alle bruksmønstrene. Dermed har vi på to forskjellige måter fått bekreftet at alle bruksmønstrene, og de funksjonelle kravene ekstrahert fra dem, er implementert.

20.3 Ikke-funksjonelle krav

Vi vil her gå gjennom de ikke-funksjonelle kravene presentert i Seksjon 10.3.4 og hvorvidt de er tilfredsstilt.

20.3.1 Functional Suitability

Functional suitability-scenariet oppgitt i Tabell 25 går ut på at geografisk plassert data, slik som observasjoner, skal representeres med en presisjon på minst 30 meter. Dette er for å kunne garantere at observasjoner som blir registrert lar seg finne igjen i etterkant, uten at brukeren må lete i et for stort område.

Lokasjonen til observasjoner registrert via mobilapplikasjonen er lagret i form av henholdsvis lengde- og breddegrader oppgitt som desimaltall. Vi lagrer disse koordinatene med en presisjon på syv desimaler. Ved bruk av fem desimaler oppnår man en presisjon med usikkerhet på maksimalt cirka 56 cm i nord-sør-retning (lengdegrader) og 26 cm i øst-vest-retning (breddegrader) [120]. Ettersom vi overgår dette nivået av nøyaktighet ved bruk av syv desimaler, kan vi konkludere med at vi har tilfredsstilt det ikke-funksjonelle kravet på minst 30m nøyaktighet.

20.3.2 Interoperability

Det ikke-funksjonelle kravet C1 om interoperabilitet (engelsk: *interoperability*) oppgitt i Tabell 26 er oppnådd ettersom backend-systemet er felles for både mobil- og webapplikasjonen. De leser og skriver mot samme database og benytter seg av samme autentiseringssystem. Lese- og skriverettigheter samt dataintegritet er gitt gjennom sikkerhetsregler som er felles for begge delsystemene. Brukerkontoer er også felles for begge. Ettersom vi ser på mobil- og webapplikasjonen som to aspekter av et felles system, og har utviklet webapplikasjonen med denne intensjonen, har vi oppfylt dette ikke-funksjonelle kravet.

20.3.3 Usability

De ikke-funksjonelle kravene om brukskvalitet (engelsk: *usability*) er oppgitt i Tabell 27 og Tabell 28.

Gjennom observasjon under brukertesting var det ingen brukere som brukte mer enn 30 sekunder på å utføre en spesifikk oppgave. Vi konkluderer derfor med at det ikke-funksjonelle kravet U1 gitt i Tabell 27 er oppnådd.

I siste fase av brukertestingen var den gjennomsnittlige SUS-poengsummen på 96,25 som vist i Seksjon 17.2.3. Vi kan derfor konkludere med at også det ikke-funksjonelle kravet U2 gitt i Tabell 28 er oppfylt.

20.3.4 Security

De ikke-funksjonelle kravene for sikkerhet er oppgitt i Tabell 29 og Tabell 30.

Krav S1 vist i Tabell 29 er praktisk talt garantert oppnådd ettersom vi bruker Firebase Authentication som autentiseringstjeneste, og denne driftes av Firebase. Ettersom Firebase Authentication er en populær og moden autentiseringstjeneste er det ikke tenkelig at en aktør klarer å bryte seg gjennom denne - i det minste ikke i løpet av én time.

Innfrielse av krav S2 gitt i Tabell 30 er gjort gjennom implementasjon av sikkerhetsregler i Firebase Firestore, som beskrevet i Seksjon 13.2.

20.3.5 Modifiability

Kravet om *modifiability* oppgitt i Tabell 31 kan være vanskelig å vurdere objektivt, ettersom det vil avhenge av hva slags ny funksjonalitet som skal implementeres. I løpet av vår smidige utviklingsprosess var all ny funksjonalitet implementert mindre enn to uker etter arbeidet med den gitte funksjonaliteten var påbegynt, noe som veier opp mot at dette kravet er innfridd. Systemet skal derfor være *modifiserbart* i tilfredsstillende grad.

20.4 Diskusjon rundt krav

Noe vi lærte fra gjennomføringen av fordypningsprosjektet, som diskutert i Seksjon 8.1, var ulempene med manglende dokumentasjon rundt tekniske krav og kriterier for å oppfylle disse. Det at vi ikke hadde dokumentert kravene godt nok førte til at visse krav ikke ble fulgt opp ordentlig og dermed ble glemt i løpet av prosjektforløpet. Resultatet av dette var et produkt som manglet funksjoner som egentlig var presentert som krav.

For å unngå å potensielt overse krav i dette prosjektet gikk vi inn for å gjennomføre en svært grundig prosess rundt kravene. Dette bestod av elisitering, dokumentering av krav gjennom en standardisert notasjon, oversikt over implementasjonsstatus gjennom bruk av kanban, og til sist verifisering av kravoppfyllelse gjennom ulike former for testing (akseptansetesting, end-to-end testing og brukertesting).

Som vist i de forrige delseksjonene har vi oppfylt alle de planlagte kravene for systemet, og mener at prosessen vår rundt hvordan vi forholdt oss til kravene utvilsomt har hjulpet med å oppnå dette målet. Det er spesielt fokuset på å standardisere notasjonen av krav som har gjort kravene mye lettere å følge opp gjennom utviklingsprosessen. Dette er fordi det har gjort det mye lettere å utvikle selve funksjonaliteten ettersom notasjonen i stor grad la en mal for nøyaktig hva som måtte utvikles. Notasjonen for funksjonelle krav har vært spesielt nyttig her, da den også dekker unntaksscenarioer som utvikleren kanskje ikke hadde tenkt på. Å være nøye med krav er absolutt noe vi vil legge vekt på i vår egen framtid som utviklere, og er noe vi hadde anbefalt til alle som skal gjennom et lignende prosjekt der de selv er involvert i planleggingen av utviklingen.

21 Diskusjon rundt teknisk implementasjon

Utviklingen av selve webapplikasjonen i React med MUI som UI-bibliotek gikk veldig bra uten noen spesielle utfordringer eller begrensninger for hverken prototypen eller selve produktet. I kombinasjon med TypeScript så var det enkelt og kjapt å lage en webapplikasjon med nye funksjoner eller forandringer etter hvert som prototypen ble endret. I tillegg til dette gjorde Mapbox det veldig enkelt å vise tilsyn på et interaktivt kart, noe som ville vært veldig vanskelig å lage fra bunnen av selv. Webpack var også et viktig verktøy for å håndtere alle de andre prosessene som er nødvendige for å lage en webapplikasjon, og gjorde resten av utviklingsprosessen veldig mye enklere.

For selve testingen ble som sagt Cypress brukt, og sammen med Firebase Local Emulator Suite var det veldig enkelt å automatisere testingen av de funksjonelle kravene. Etter at selve testoppsettet var satt opp, tok det minimalt med tid å skrive tester.

GitLabs Git- og CI/CD-system har også fungert bra for versjonkontroll av kildekode og pipelines for å automatisk kjøre tester og publisere nye versjoner av webapplikasjonen. Det eneste negative var at NTNUs instanse av GitLab hadde problemer med sin GitLab Runner som kjørte Docker Dind, noe som gjorde at vi ikke kunne bygge vårt eget Docker image for Cypress-testingen. Dette ført til at vi måtte installere visse dependencies for Cypress hver gang testene skulle kjøre, som gjorde testingen i pipelinen noe tregere enn det ellers ville vært. Dette førte likevel ikke til noen reelle problemer eller begrensninger.

Firebase har vært en fantastisk tjeneste som har gjort det kjempeenkelt å sette opp og bruke et authentiseringssystem, hostingsystem, storagesystem og til dels databasesystem, som alle har kjempegod samhandling og interoperabilitet. Firebase har praktisk talt fjernet mesteparten av arbeid som trengs for å sette opp et backend-system, noe som ga oss langt mer tid til å fokusere på selve webapplikasjonen og brukertesting. Selv om bruken av Firebase generelt har vært positivt, har det også ført til visse begrensninger som burde løses før systemet blir tatt i bruk av bønder og gjetere i en produksjonssetting. Disse begrensingene omhandler hovedsakelig begrensninger i bruk av gratisversjonen til Firebase, begrensninger i Firebase Firestore sine funksjonaliteter for databasespørringer, begrensede sikkerhetsmuligheter og begrensninger i Firebase Hosting sin støtte for offline-lagring. Disse begrensningene blir utdypet i underkapitlene nedenfor.

21.1 Bruk av Cloud Functions

En av de største begrensningene med Firebase når man bruker gratisversjonen («Spark Plan») er at man ikke har tilgang til Firebase Cloud Functions, som er Firebase sin serverless NodeJS-tjeneste. Dette er en tjeneste som gjør det mulig å gjøre flere ting enklere, mer effektivt og tryggere. I dette prosjektet ville

bruk av Cloud Functions kunne ført til flere forbedringer som både ville gjort enkelte implementasjoner enklere, i tillegg til å gjøre datastrukturen og sikkerheten rundt dataen bedre.

En av forbedringene man kunne gjort ved å bruke Cloud Functions er å lage endepunkt for å øke sikkerheten rundt hvem som kan lese data ved å lage et endepunkt som er åpent for alle, men som kun deler ut data slik som start- og sluttdata og selve gjeterruten for et tilsyn. Dette kunne blitt gjort ettersom Cloud Functions kan lese all data uavhengig av Firebase Security Rules og kan deretter filtrere ut den dataen man vil at kalleren av endepunktet skal få. En annen forbedring kan være å automatisk oppdatere tilsyn ved endring av gårdsnavn, istedenfor at gamle tilsyn tilknyttet det tidligere navnet blir utilgjengelig for bonden inntil eieren av tilsynet manuelt oppdaterer alle saueflokkene i tilsynene.

I tillegg til å være en serverless NodeJS-tjeneste, har Cloud Functions en tilleggsfunksjonalitet som er koblet direkte til Firestore (og andre tjenester som Authentication): Cloud Firestore Triggers. Dette er i bunn og grunn en Cloud Function som blir utløst av at et Create/Update/Delete/Write-event inntreffer på et dokument eller kolleksjon som er spesifisert i funksjonen.

Cloud Firestore Triggers er funksjonalitet som kunne forbedret koden på to forskjellige områder. For det første kunne det forbedret sletting av data slik som tilsyn og saueflokker. Måten sletting fungerer på i Firestore er at man ikke bare kan slette et dokument for å slette all data tilhørende det dokumentet, det er også nødvendig å slette alle subkolleksjoner og dokumenter i subkolleksjonene. All denne slettingen skjer nå manuelt på brukerens applikasjon (både mobilapplikasjon og webapplikasjon) og ved store mengder data kan det både ta tid og ressurser på brukerens maskin. En forbedring hadde vært å ha en trigger på tilsyn (og andre dokumenter som har subkolleksjoner, slik som døde sau og saueflokker) som automatisk slettet all tilhørende data når brukeren slettet et dokument. Dette ville drastisk redusert tiden og ressursene på brukerens maskin. For det andre kunne Cloud Firestore Triggers forbedret prosessen rundt hvordan listen av gårder observert i et tilsyn blir generert. Måten det fungerer nå er at før et tilsyn blir lagret i Firestore, blir alle unike gårdsnavn fra alle saueflokkene i tilsynet samlet til en liste og lagt inn under feltet «farms» i selve tilsynsdokumentet (dette ettersom det ikke er mulig å gjøre en databasespørring hvor vi sjekker «farms» feltet i alle «sheepHerd»-dokumentene som tilhører tilsynsdokumentet på én gang). Dette gjør det teknisk sett mulig for en bruker som aksesserer Firebase API-et direkte å legge inn en tom liste, eller at det eventuelt skulle komme en feil som gjorde at en tom liste ble opprettet. Dette ville ført til at tilsynet ble ubrukelig ettersom et tilsyn uten gårder ikke blir vist til noen brukere, som forklart mer utdypende i Seksjon 21.2.1. Dette kunne blitt løst ved å ha en trigger som automatisk lagde denne listen over gårder og oppdaterte tilsynet etter at et tilsyn ble laget eller oppdatert i databasen.

21.2 Begrensninger rundt Firestore

Det at Firebase Firestore er en NoSQL dokumentbasert database uten et API-lag mellom brukeren og databasen, har mange fordeler som vi har nytt godt av under utviklingen av systemet. Likevel har det også gitt oppstand til noen ulemper rundt personvernet, samt at det har ført til spørsmål rundt om dataen vi har i systemet egentlig egner seg for et databasesystem som ikke er relasjonelt, spesielt når Firebase Firestore har begrenset med databasespørringsstøtte.

21.2.1 Datarelasjoner

Valget rundt SQL- vs. NoSQL-database er et viktig valg å ta før man velger databasesystem. Som forklart i Seksjon 11.1.1.2 så endte vi med Firebase Firestore, som er en NoSQL-database. Dette har generelt fungert bra for det meste av systemet, men det har ført til et par utfordringer.

I motsetning til tradisjonelle databasesystem hvor man betaler for antall sekunder/minutter/timer og lagringsplass brukt, betaler man for antall lesinger og skrivinger av dokumenter i Firebase Firestore. Selv om dette ikke nødvendigvis blir et problem med hvor billig prisingen av lesinger/skrivinger er, kan det komme uforutsette store regninger hvis applikasjonen blir skalert opp med mange aktive brukere og

eventuell ny funksjonalitet med data som har en tendens til å deles opp i mange ulike dokumenter. Sånn som datastrukturen er nå er det hovedsakelig tilsyn som er delt opp i flest dokumenter; med ett for hvert rovdyr, annet, død sau, saueflokk, bilde og «parts» (deler av bildet). Hvis man har flere store tilsyn med masse dokumenter, kan man fort ende opp med mange lesinger avhengig av hvor mange aktive brukere man har og hvordan brukerne benytter systemet.

Det at Firestore er en ikke-relasjonell database kan gi utfordringer hvis deler av dataen har tendens til å ha relasjoner. Dette kan sees i tilsynsdelen av datastrukturen til systemet, hvor et tilsyn eies av brukeren som har vært på gjetetur, men kan sees av brukere som eier minst én av gårdene som ligger i «farms» feltet til tilsynsdokumentet. Altså har vi en eier-relasjon som er «1-1» og «0-n»-relasjoner til brukere som skal kunne se tilsynet, ettersom minst én sau fra deres gård har blitt sett på tilsynet. Dette gir en edge-case utfordring hvor hvis en gjeter er ute på tilsyn for gård x,y,z; men ikke finner noen sauer (kanskje alle har blitt drept av rovdyr), vil ikke saubøndene fra gård x,y,z ha mulighet til å se tilsynet. Dette er ettersom «farms»-feltet i tilsynsdokumentet er tomt. Dette kan derimot bli løst relativt enkelt ved å endre mobilapplikasjonen til å tvinge gjeteren til å velge hvilke gårder de går på tilsyn for ved oppstart eller slutt, som deretter blir lagt inn i «farms» feltet. Dette løser ikke relasjonsproblemet i seg selv, men fikser problemet med at tilsyn ikke blir tilgjengelig for bøndene som gjeteren har gått på tilsyn for.

Dette er to problemer som er direkte konsekvenser av hvordan dataen vår er strukturert og lagret, altså er det vanskelig å komme seg unna problemene uten å endre datastrukturen. Å endre datastrukturen kan derimot være vanskelig med hvordan Firebase Firestore – og NoSQL-databaser generelt – fungerer. Det som gjør det spesielt vanskelig er at selv om det finnes løsninger for å lagre «0-n»-relasjoner i NoSQL-databaser som forklart av MongoDB [121], så er det spesielt vanskelig å hente ut dataen på grunn av Firebase Firestore sin begrensede databasespørringsstøtte.

21.2.2 Personvern

Måten Firebase Firestore og systemet er satt opp åpner opp for at alle som er logget inn har mulighet til å lese alle tilsynsdokumenter og -kolleksjoner som finnes i databasen, som forklart i Seksjon 13.2. Selv om dette ikke nødvendigvis er farlig, så er det ikke en optimal løsning. Grunnen til at Firebase Security Rules er satt opp på denne måten er på grunn av måten et tilsyn er koblet til en bruker. Dette fungerer ved at man sjekker om minst én av brukerens gårder finnes i listen over gårder i tilsynsdokumentet. Dette skaper et problem på grunn av begrensningene til Firebase Security Rules, som nemlig ikke har mulighet til å gjøre databasespørringer for så å bruke resultatet fra en databasespørringen til å sjekke om det samstemmer med en oppsatt regel (som i dette tilfelle ville vært å sjekke om resultatet hadde et overlapp med listen over gårder i tilsynet). Dette gjør det altså umulig å sette lese-rettigheter til bare bønder som har en tilknytting til tilsynet, noe som fører til at man må tillatte alle innloggede brukere å kunne lese alle tilsyn. Dette er både et resultat av Firebase sine strenge begrensninger rundt Firebase Security Rules, men også måten Firebase Firestore er satt opp uten et API-lag som skiller brukeren og databasen.

En potensiell løsning utenom å endre databasetjeneste til et system som støtter relasjoner bedre (som diskutert i Seksjon 21.2.1) er å bruke Cloud Functions. Ved å sette lese-rettighetene til å bare gi tillatelse til eieren av tilsynet, for deretter å lage et Cloud Function-endepunkt hvor man bruker Firebase Admin, kan man tette tilgangen. Ved å bruke Firebase Admin så kan man omgå Firebase Security Rules og kjøre en databasespørring som henter ut alle tilsyn som er koblet til en bruker, for så å sende dem tilbake til brukeren. Det eneste problemet med dette er at man mister fordelene man får av å være koblet direkte til Firebase Firestore gjennom Firebase SDK-en, slik som å kunne lytte etter sanntidsoppdateringer eller støtten for lokal caching og offline-støtte.

21.3 Begrensninger rundt bildelagring

Som beskrevet i Seksjon 13.1.2, så er en av de største begrensningene i applikasjonen (som brukeren blir direkte påvirket av) at Firebase Storage ikke støtter lagring av filer når man ikke har internett-tilgang. Bilder er derfor lagret direkte i Firebase Firestore som base64-kodede strenger. Dette fører til at brukeren ikke kan laste opp mer enn 10 MB verdt av bilder (etter kompresjon), noe som kan være problematisk hvis brukeren har flere saueflokker og døde sau som de vil ta bilde av med et moderne smarttelefonkamera.

Hvordan dette skulle blitt løst er noe usikkert ettersom det ikke virker som om at Firebase Storage har noen planer om å endre offline-støtten. Dermed må man eventuelt ta i bruk noe mer egenlagde løsninger for å komme seg rundt denne begrensningen. Den mest opplagte løsningen vil være å legge inn en sjekk i mobilapplikasjonen som kjører kontinuerlig hvert x antall sekunder/minutter, som sjekker om internett-tilgangen er tilbake og om tilsynet har blitt lagret i Firestore (altså ikke lokalt). Deretter lagrer den bildet i Firebase Storage, og oppdaterer tilsynet med lenken til bildet i Firebase Storage. Dette vil komme seg rundt 10 MB-grensen, men vil ha andre problemer som hvordan man skal håndtere endringer av tilsynet før bildet har blitt lastet opp til Firebase Storage, eller hvordan man skal håndtere en eventuell feil som fører til at bildet/bildene ikke blir lastet opp. Dette er problemstillinger man må ta stilling til ved en eventuell implementasjon av en løsning i videre arbeid, før applikasjonen eventuelt skulle blitt tatt i bruk av bønder og gjetere i en produksjonsetting.

21.4 Evaluering av den tekniske implementasjonen

Den tekniske implementasjonen har alt i alt fungert bra for behovene våre under prototypeutviklingen, men som diskutert ovenfor så har Firebase noen begrensninger som kan være utfordrende hvis systemet skulle blitt brukt av ekte brukere i en produksjonssetting. Mange av disse begrensningene og utfordringene kunne derimot blitt løst enten ved bruk av Cloud Functions eller ved egenlagde løsninger. Det kunne muligens vært fordelaktig å bruke en egenlagd backend-løsning, men det ville krevd mye mer tid og arbeid både for selve utviklingen, men også for å oppfylle ikke-funksjonelle krav og ABK-er for både mobil- og webapplikasjonen. Man kan altså konkludere med at for en prototypeutivkling med begrenset tid – som dette prosjektet – så fungerte Firebase optimalt, men man burde ta en vurdering på om man ville fortsatt med det, eller lagd et nytt backend-system hvis man skulle lansert systemet som et ekte produkt.

22 Evaluering av prosessen

I denne seksjonen evaluerer vi prosessen som ble fulgt under utviklingen av webapplikasjonen og videre-utviklingen av resten av systemet.

Som beskrevet i Seksjon 9 tok vi i bruk en Agile metodikk med Kanban som implementasjonsrammeverket for å få struktur i utviklingen, men samtidig ha mye smidighet rundt utførelsen. Dette viste seg å være det perfekte valget for oss ettersom utviklingen gikk veldig bra, uten noen spesielle problemer eller tidsfrister som ikke ble møtt. Valget om å gå for det rammeverket med mest mulig frihet og minst mulig faste frister viste seg å være et bra valg, ettersom det ikke bare var en veldig naturlig måte å jobbe på som studenter, men det lot oss også jobbe på en veldig smidig måte. Dette gjorde at vi veldig effektivt fikk løst oppgaver uten problemer ettersom vi hadde veldig god og kontinuerlig kommunikasjon – som var mulig og veldig naturlig når vi bare var to personer på prosjektet. Det skal derimot sies at vi ikke alltid fulgte kanban-boardet hele tiden, og fra tid til annen glemte vi å legge inn oppgaver, eller glemte å flytte en oppgave fra en kolonne til en annen. Dette ville kunne ført til problemer i et større prosjekt med flere gruppemedlemmer, men på grunn av at vi som sagt bare var to personer – med god og kontinuerlig kommunikasjon – så var dette aldri et problem, ettersom vi var veldig godt oppdatert på hva den andre jobbet på.

På grunn av metodikken og rammeverket som ble valgt var tidsfrister og leveranser veldig flyttende uten faste rammer, men det ble fortsatt laget et Gantt-diagram som beskrevet i Seksjon 9.2. Denne tidsplanen var mer et estimat og et ønsket utfall av utviklingsperioden enn faste frister, men ettersom utviklingen gikk såpass bra uten utfordringer så ble disse fristene fint overholdt uten problemer. Enkelte hendelser ble flyttet med et par dager her og der, men overordnet så ble ting ferdig til riktig tid og vi fikk ingen dominoeffekter hvor en rekke ting måtte bli flyttet framover eller bakover. Det skal derimot sies at hvis vi hadde hatt problemer med å bli ferdig til de forskjellige fristene, hadde ikke dette vært et problem på grunn av den Agile prosessen. Hvis en frist ikke ble fulgt og det hadde ført til en dominoeffekt av andre frister som heller ikke ville blitt fulgt, så kunne vi relativt enkelt bare endre fristene på en måte som passet oss og prosjektmålet bra.

Alt i alt så skjedde ingenting uforventet i prosessen og man kan si at prosessen som ble planlagt på starten av prosjektet var en suksess, noe som førte til at prototypen ble ferdig i god tid før brukertestingen. Dette førte igjen til at prosjektet ble ferdig i god tid før prosjektfristen.

23 Evaluering mot forskningsspørsmål

I denne seksjonen evaluerer vi systemet mot forskningsspørsmålene introdusert i Seksjon 3.1 i lys av resultatene fra brukertestingen.

23.1 FS1: Kan vårt digitale system effektivisere å oppsummere beitesesongen sammenlignet med bruk av papirbaserte notater?

Bruk av vårt digitale system har økt effektivitet som en av de store potensielle fordelene. Ettersom en oppsummerende sesongrapport burde produseres på slutten av en beitesesong, kan mindre tidsbruk for å produsere en rapport være en motiverende faktor for å ta i bruk vår digitale løsning eller en lignende løsning. Det er derfor viktig å verifisere om prosessen faktisk har blitt effektivisert. For å gjøre effektiviteten kvantifiserbar legger vi tre ting i begrepet 'effektivitet':

- **Tidsbruk:** hvor lang tid det tar å utføre oppgaven.
- **Kompleksitet:** Hvor mange individuelle steg det tar å utføre oppgaven og hvor utfordrende det er å gjøre det, enten fysisk eller mentalt.
- **Kvalitet:** Hvorvidt kvaliteten på det produserte resultatet blir forbedret. Om en endring blir implementert slik at de to andre faktorene forblir det samme, mens kvaliteten øker, kan man si at effektiviteten har blitt forbedret.

Vi innfører alle disse som faktorer ettersom å kun måle én av dem kan føre til en misvisende konklusjon. Det ville for eksempel være mulig å minimere tidsbruk ved å kutte mange steg som vanligvis måtte gjøres i prosessen, men kvaliteten på det endelige resultatet vil sannsynligvis lide på grunn av dette.

Resultatene av utfylling av papirskjemaer i utføringen av brukertestene, vist i Seksjon 17, viser en svært stor forbedring når det kommer til tidsbruk ved bruk av digitale rapporter, framfor at brukeren fyller ut disse selv på papir. I alle brukertestene tok det brukerne i snitt over ett minutt å oppsummere ett tilsyn inn i oppsummeringsskjemaet. Til sammenligning er prosessen av å lage en digital rapport nærmest umiddelbar – brukeren trykker på knappen for å generere en ny rapport, og så kan de lese eller laste ned rapporten. I et reelt brukscenario der en hel sesong skal oppsummeres, og minst ett tilsyn har blitt utført i uka i tråd med loven, kan prosessen med å oppsummere alle tilsynene være svært tidskrevende hvis det gjøres for hånd. Vårt digitale system gjør denne oppsummering nærmest umiddelbart, praktisk talt uavhengig av hvor mange tilsyn som har blitt utført.

Overgangen til digitale rapporter innebærer også en økning i kvalitet. At én av brukerne gjorde en regnefeil ved bruk av papirskjema og at det dermed ble oppsummert feil antall sau i rapporten, viser potensialet for feilaktige oppsummeringer når dette gjøres for hånd. Med bruk av vårt digitale system faller dette problemet bort, ettersom oppsummering skjer automatisk og dens gyldighet er verifisert gjennom kodetester. En subjektiv fordel med våre digitale rapporter er at de er lette å tyde og har et ryddig oppsett. Siden de benytter seg av digital representert tekst gir de ikke rom til potensielle problemer som håndskrift kan gi oppstand til – med noens håndskrift kan for eksempel tallene «1» og «7» være lett å forvirre med hverandre. Dette er ikke tilfellet med vårt system.

Kompleksiteten har naturligvis også gått ned som et resultat av vårt digitale system for å generere rapporter. Brukeren behøver kun å trykke på en knapp for å generere rapporten, og resten skjer av seg selv. Å lage en oppsummeringsrapport for beitesesongen for hånd krever å tolke og så legge sammen ulike tellinger fra hvert eneste tilsyn utført i løpet av sesongen, som er en mye større mengde arbeid.

Ettersom alle de tre aspektene – tidsbruk, kvalitet og kompleksitet – har forbedret seg, kan vi trygt konkludere med at vårt digitale system har økt effektiviteten med tanke på å oppsummere beitesesongen.

23.2 FS2: Oppleves vårt digitale system for rapporter som mer tilfredsstillende for brukeren enn dagens papirbaserte løsning?

At vårt digitale system er mer effektivt, er ikke nødvendigvis alene en god nok grunn til at sauebønder burde gå over til å benytte det. Det er viktig at brukerne faktisk opplever at systemet har en god bruksopplevelse, og at den er bedre enn eller i det minste på samme nivå med den eksisterende papirbaserte løsningen.

Intervjuene gjort i sammenheng med brukertestene (Seksjon 17.1.4 og Seksjon 17.2.4) ga svar på akkurat dette: alle brukerne foretrakk det digitale systemet. Spesielt det at man slapp å regne seg fram til svarene for oppsummeringsskjemaet ble trukket fram av flere av testpersonene. Det var flere testbrukere som syntes at bruk av papirskjemaet var tungvint og uoversiktlig.

Det at den gjennomsnittlige SUS-poengsummen var svært høy – 96,25 – i andre fase av testingen, er også et tegn på at brukervennligheten til systemet er høyt og at brukerflyten tillater brukeren å utføre oppgavene sine uten nevneverdige hindringer.

23.3 FS3: Hvilke fordeler og ulemper tilbyr vårt digitale system en sauebonde framfor dagens papirbaserte løsning?

Utenom fordelen med rapporter nevnt i de forrige forskningsspørsmålene, skal vi her sammenligne hva vårt digitale system tilbyr brukeren av funksjonalitet sammenlignet med bruk av notater gjort for hånd.

En stor fordel med systemet er det legger til rette for deling av tilsynsdetaljer innenfor et beitelag. Så lenge alle gårdene i beitelaget blir registrert inn via webapplikasjonen, og de som utfører tilsynsturene har alle gårdene i beitelaget lagt til lokalt på sin enhet, vil alle medlemmene av beitelaget kunne bruke webapplikasjonen til å se tilsyn som har blitt gjort så lenge deres gårder er representert i tilsynet. Dette fritar medlemmene av beitelaget fra å manuelt dele notater fra utførte tilsyn mellom hverandre.

En annen fordel med bruk av systemet er at man ikke risikerer å miste tilsynsnotater. Det er mange mulige måter notater på papir kan gå tapt – de kan ende opp i en papirbunke og bli kastet, de kan bli sølt væske på og dermed bli uleselige eller de kan bli tapt i en brann. I kontrast til dette er digitale notater fra systemet lagret i skyen og ivaretatt av Firebase. Ettersom ukentlige tilsyn er påkrevd av loven, er disse tilsynsnotatene viktig materiale å ta vare på.

Enda en fordel med systemet er at det gjør det lett å holde styr på hvor observasjoner ble gjort og nøyaktig

hvor gjeteren gikk, ettersom observasjonene plasseres på et kart av brukeren og brukerlokasjonene lagres automatisk i bakgrunnen. Ved bruk av notater på papirskjema finnes det ingen bestemt måte å logge plasseringen til en observasjon. Gjeteren kan for eksempel ta med seg GPS og skrive koordinatene inn på papirskjemaet, eller de kan gi en tekstbeskrivelse av hvor de befinner seg. Et problem med å oppgi tekstlige beskrivelser av et spesifikt geografisk punkt er at det kan være vanskelig å finne konkrete landemerker ute i beitemarka. Å skrive av GPS-koordinater er tidskrevende, og den som leser notatene må selv slå opp koordinatene før de kan få en forståelse for hvor dette egentlig er.

Det at systemet bruker forhåndsdefinerte kategorier av observasjoner og forhåndsdefinerte datafelt for hver observasjon, er både en fordel og en ulempe. Det gir en god brukerflyt for gjeteren når de bruker mobilapplikasjonen, ettersom de ulike observasjonstypene er lette å navigere til og inneholder felt for all informasjonen veileder Svein-Olaf Hvasshovd mener er nødvendig for den observasjonstypen. 'Annet'-kategorien gir rom til å registrere ting som faller utenfor de vanlige observasjonene (saueflokker, døde dyr eller rovdyr). Siden det er et forhåndsbestemt antall observasjonstyper i et tilsyn, er det også mulig for systemet å lage en oversiktlig rapport som bruker data fra alle tilsynene. Ulempen med at systemet vårt følger et gitt skjema (engelsk: *schema*) for data fra tilsyn er at å innføre nye typer observasjoner krever at systemet utvides for å støtte dem.

Som eksempel kan det være et bondelag som bestemmer seg for at alle som utfører tilsyn skal oppgi informasjon om hvordan de vurder farenivået i området (med tanke på dyrenes sikkerhet), på en skala fra 1 til 10, basert på hvordan de tolker forholdene i området og nylige rovdyrobservasjoner. Med papirnotater er dette lett å innføre, ettersom det ikke finnes noen praktisk begrensing på hva som kan noteres på papir. Med vårt system finnes det ikke noen konkret flyt for å gjøre dette, og det finnes flere mulige løsninger på hvordan dette kunne innarbeides i systemet. Som eksempel kunne noen argumentert for at dette kunne oppgis i tekstboksen for å oppgi en beskrivelse av ruta (vist i Figur 46). Andre brukere kunne valgt å registrere dette som en 'Annen'-kategori registrering, der brukeren oppgir en tekstbeskrivelse som forklarer farenivået. Problemet med å registrere det som en 'Annet'-observajon er at den må plasseres på kartet, selv om dette ikke gir helt mening for slike farenivå-registreringer ettersom de burde være gjeldende for hele beiteområdet. At brukerne kunne valgt å registrere dette på forskjellige måter er problematisk, ettersom det kan gjøre det vanskelig å finne fram til den relevante informasjonen når tilsynet inspiseres på webapplikasjonen. Dette eksemplifiserer hvordan en 'fast' datastruktur kan være problematisk om det plutselig skulle oppstå krav til registreringer som ikke var tiltenkt i designet av systemet.

24 Øvrig diskusjon

I denne seksjonen vil vi se på hvorvidt prosjektets formål er oppnådd, samt diskutere hva de viktigste stegene for prosjektet i framtiden kunne vært.

24.1 Evaluering mot prosjektets formål

Formålet med masterprosjektet var å utvikle en ny applikasjon som skal presentere data registrert via mobilapplikasjonen fra fordypningsprosjektet. Bruk av denne applikasjonen skulle deretter sammenlignes med bruk av notater på papir, som er dagens mest utbredte løsning. Sammenligningen tok utgangspunkt i de tre forskningsspørsmålene, som ble introdusert i Seksjon 3.1.

Gjennom denne rapporten har vi vist prosessen og det endelige resultatet av applikasjonsutviklingen. Som diskutert i de tidligere seksjonene ble de tekniske kravene for løsningen oppfylt, samt at vi har besvart forskningsspørsmålene i Seksjon 23. Vi kan derfor konkludere med at prosjektets mål er oppnådd.

24.2 Diskusjon rundt løsning

Med kravene oppfylt og prosjektmålene nådd er vi alt i alt svært fornøyde med hva vi har fått til i løpet av prosjektforløpet. Vi har en del tanker om hvordan prosjektets vei videre kunne sett ut om vi ikke hatt noen konkret tidsbegrensning, som vil diskuteres i delseksjonene under.

24.2.1 Testing på bønder under beitesesong

Som nevnt i prosjektbeskrivelsen (Seksjon 1) pågikk dette masterprosjektet utenfor tidsrommet til beitesesongen, som varer fra rundt mai til september. Dermed var det ikke mulighet for å teste løsningen innenfor et realistisk bruksscenario.

Av det vi har avdekket i våre brukertester framstår det ferdige systemet som meget brukervennlig. Det som likevel gjenstår er å se hvordan løsningen faktisk fungerer når det tas i bruk 'på ekte', som innebærer at løsningen benyttes av gjetere på utmarksbeite og sauebønder. De eventuelle uforutsette problemene som kan oppstå ved bruk av systemet vil altså kreve virkelighetsnær bruk av den tiltenkte brukergruppen for å avsløres. Fordi systemet skal være et verktøy som kan brukes fra tidspunktet første tilsyn skal utføres og helt til beitesesongen er avsluttet, er det viktig at en eventuell tredje fase brukertesting er langvarig og følger testbrukerne fra beitesesongens start til slutt.

Basert på tilbakemeldingene fra en ny fase brukertesting kan det være behov for mindre eller større endringer på systemet. Resultatet av dette kan variere fra mindre endringer på grensesnittet til å omgjøre hele aspekter av systemet. En omfattende felttest av løsningen er uansett viktig å få utført for å kunne anse hvor godt det tilfredsstiller bønders og gjeteres behov over en langtidsperiode.

24.2.2 Gårder kunne vært implementert annerledes

Erfaringene våre med systemet gjennom design, implementasjon, brukertesting og egen bruk har gitt oss visse tanker om hvordan systemet kunne fungert annerledes, hvis vi kunne designet et lignende system med etterpåklokskapen fra dette prosjektet. En tanke vi har fått med oss ut ifra dette er hvordan konseptet med 'gårder' fungerer. Slik systemet fungerer nå må gårder registreres enkeltvis på webapplikasjonen. Brukere av mobilapplikasjonen må deretter enkeltvis søke etter de gårdene som er relevante for der de skal utføre tilsyn, og lagre disse lokalt.

Vi tror at denne funksjonaliteten kunne blitt erstattet med beitelag, der et beitelag består av flere gårder. Dette hadde fungert ved at beitelaget registreres på webapplikasjonen, og alle gårdene som er med i beitelaget registreres som en del av beitelaget. Deretter behøver brukerne av mobilapplikasjonen kun å søke etter et spesifikt beitelag istedenfor å måtte individuelt lagre alle enkeltgårdene lokalt. Mobilbrukeren ville da fått opp alle de mulige øremerkene fra beitelagets gårder som valgmuligheter ved registrering av saueflokker.

Med denne funksjonaliteten på plass kunne det likevel vært nødvendig å fortsatt beholde muligheten for å registrere enkeltgårder uten beitelag slik som i dag, da det ikke er garantert at alle gårder tilhører et beitelag.

24.2.3 Merkevarebygging

En observant leser vil ha oppdaget at webapplikasjonen og systemet i sin helhet ikke har blitt navngitt, og at bildene av webapplikasjonen i Seksjon 13.4 viser at «header»-en ikke faktisk har en logo av merkevarenavnet, men i stedet teksten «LOGO». Dette kommer av at merkevarebygging ikke er spesielt viktig under konsept- og prototypeutvikling, og dermed brukte vi ikke tid på å komme på et navn for

webapplikasjonen og systemet. Hvis prosjektet skulle blitt videreført og satt i produksjon så vil det være nødvendig å lage en merkevare, som blant annet betyr at et navn for applikasjonene og systemet må bli laget. I tillegg må en logo designes og legges til i webapplikasjonen.

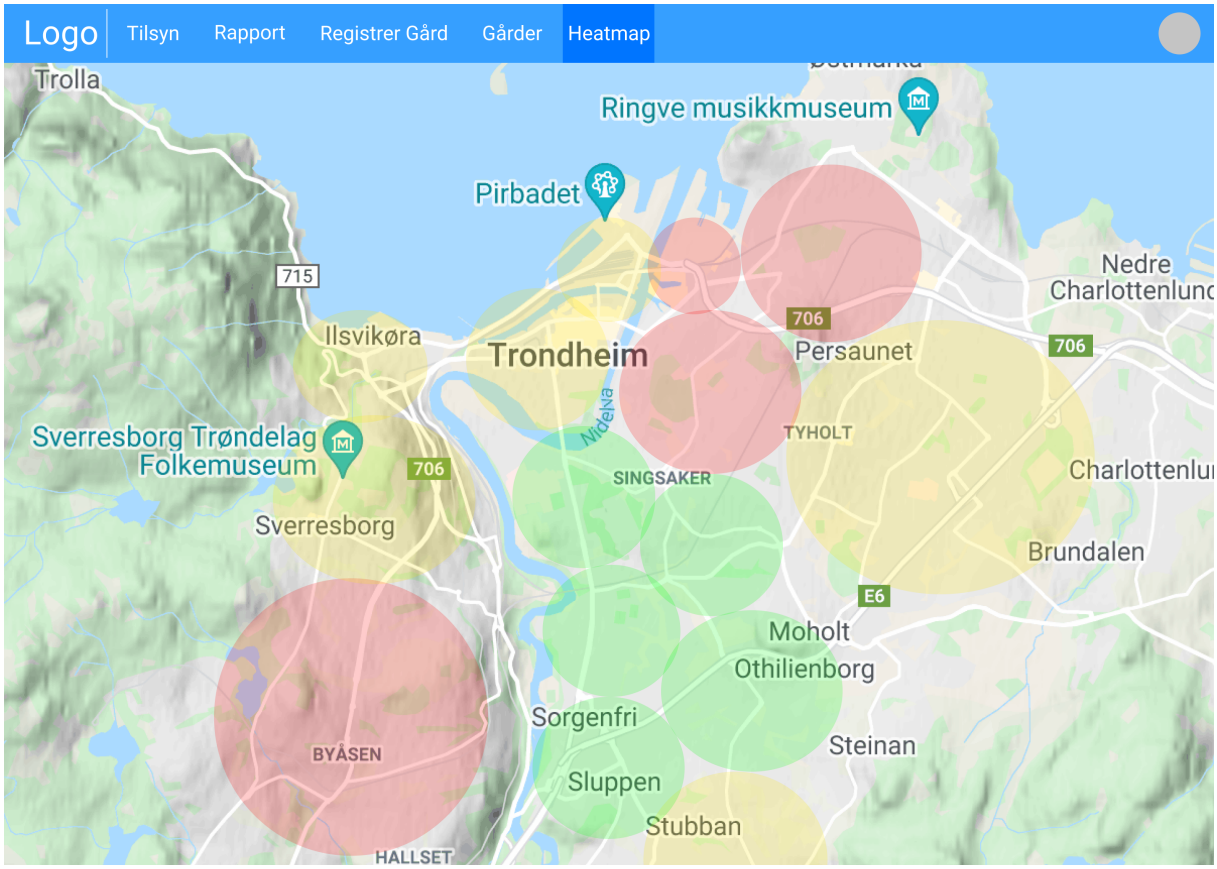
24.2.4 Idéer til tilleggsfunksjonalitet

Under prosjektforløpet kom vi på visse idéer til funksjonaliteter for webapplikasjonen som kunne vært nyttige for en sauebonde. Dette var idéer vi kom på sent i prosjektforløpet, og ikke ble innført som funksjonelle krav for prosjektet ettersom de ville kreve mer tid å designe og implementere enn vi hadde til rådighet. Disse idéene vil diskuteres i delseksjonene under.

24.2.4.1 «Heatmap»

Et «heatmap» eller «varmekart» er en måte å grafisk framstille data på, der områder på figuren framstilles med ulik farge avhengig av et eller annet mål på 'intensitet'. Dette kan for eksempel være et kart som framstiller befolkningstettheten i et land, der tettbefolkede områder er farget rødt, og lavtbefolkede områder er farget blått. Hvilken farge et område på kartet har vil altså bestemmes av 'intensiteten' til dataen, som i dette tilfellet ville vært befolkningstettheten, der høyere befolkningstetthet gir utslag i en varmere farge.

Idéen vår om å inkludere et heatmap i systemet går ut på å implementere dette som en egen side på webapplikasjonen. Et eksempel på hvordan et heatmap kan se ut i vår løsning vises i Figur 77. Heatmapet gir informasjon om hvilke områder som nylig har blitt gjort tilsyn på, basert på dataen fra tilsynene. Områder som nylig har blitt dekket av tilsyn er farget grønt. Områder som det ikke nylig har blitt utført tilsyn på er farget gule, mens områder som det er lenge siden det har blitt utført tilsyn på er farget rødt. Ufargede områder kan antas å aldri ha blitt dekket av et tilsyn (for eksempel hvis de er utenfor beiteområdet). Et slikt heatmap kan hjelpe sauebønder å ta avgjørelser om hvor de og gjeterne burde utføre tilsyn, ettersom det kan være mer gunstig å utføre tilsyn i områder som ikke nylig har blitt sett på.



Figur 77: Et eksempel på hvordan et heatmap kan se ut i løsningen vår.

24.2.4.2 Tidslinje av tilsyn

Denne idéen er en evolusjon av tilsynssiden i webapplikasjonen, som ble presentert i Seksjon 13.4.2.9. Istedenfor å kun se ett tilsyn av gangen, er tilsyns-tidslinjen tiltenkt som en funksjon som lar brukeren dra en tidslinje fram og tilbake som da dynamisk vil vise det tilsynet som tilsvarer med datoen brukeren har valgt i tidslinjen. Dette skal tillate brukeren å fort kunne «bla gjennom» tilsyn, istedenfor å måtte se listen av tilsyn, velge og inspisere ett spesifikt tilsyn, for å så gå tilbake til listen av tilsyn og velge et annet tilsyn. Fordelen med tidslinje-idéen er i hovedsak at det lar brukeren se hvordan saueflokkene har beveget seg over tid ved å hurtig kunne «scrolle» gjennom tidslinjen. Dette kan være en fordel under sankingsarbeidet, da sauebonden kan bygge opp en forståelse for hvor sauene befinner seg på slutten av beitesesongen ved å observere hvordan de har beveget seg over lengre tid.

24.2.4.3 Trusselbilde

Som nevnt i Seksjon 4.1.1 er kravet om ett tilsyn i uka et basiskrav for områder «uten særskilt risiko». Om farenivået ansees som høyere vil dette kreve hyppigere utføring av tilsyn. En trusselbilde-funksjonalitet for systemet er tiltenkt som en egen side på webapplikasjonen som i stor grad ligner på heatmap-idéen. Forskjellen her blir at områdene på heatmapet fargelegges basert på rovdyr- og død sau-observasjonene som er registrert. Flertallige nylige observasjoner av rovdyr eller død sau i et område vil føre til at området kategoriseres som høyrisiko og farges på kartet deretter. På samme måte vil områder uten observasjoner av rovdyr eller død sau over en viss tidsperiode da kategoriseres som lavrisikoområder og fargelegges deretter. Dette hadde gitt brukeren en visuell indikator på hvilke områder tilsyn burde utføres hyppigere.

24.2.4.4 Integrering av sporingsbjeller

I redegjørelsen for eksisterende løsninger ble det vist at Telespor (Seksjon 5.2.2) – et system for elektronisk overvåkning av dyr på beite – tilbyr funksjonalitet for å koble deres radiobjeller til en tredjeparts mobilapplikasjon slik at dyrenes lokasjon vises i appen. En lignende funksjonalitet kunne ha stor nytteverdi for vårt system, i form av at sporingsbjeller fra en ekstern leverandør kunne ha delt sin lokasjonsdata med vårt system. Dette kunne ha blitt integrert inn i både mobil- og webapplikasjon ved at disse sporingsbjellene vises som ikoner på kartet. Bruk av sporingsbjeller tilfredsstiller ikke behovet for ukentlige tilsyn, men de kan hjelpe bonden/gjeteren å lokalisere noen av flokkene som er på beite.

Del VIII:

Videre arbeid og konklusjon

I denne delen vil vi ta for oss forslag for videre arbeid ved en eventuell videreføring av prosjektet, i tillegg til konklusjonen for prosjektet.

25 Videre arbeid

Denne masteroppgaven hadde en frist for innlevering 13. juni. Dette førte til at flere ting som brukertesting på bønder, utvikling av tilleggsfunksjonalitet, fiksing av tekniske utfordringer og fiksing av funn fra brukertest 2, gjenstår som framtidig arbeid. Denne seksjonen presenterer en plan for hvordan vi hadde gått videre med prosjektet om vi hadde hatt mer tid til rådighet.

25.1 Testing på riktig målgruppe

Som diskutert i Seksjon 24.2.1 ble systemet aldri testet på den riktige målgruppen – sauebønder og gjetere. Selv om systemet fikk gode tilbakemeldinger på brukertestingen, er det fortsatt stor usikkerhet rundt om den riktige målgruppen også synes systemet er enkelt å bruke, og ikke minst om det faktisk løser problemene deres på en god og intuitiv måte. Testing på sauebønder og gjetere er dermed noe av det første man burde gjøre ved eventuell videreutvikling av systemet.

25.2 Fiksing av tekniske utfordringer

Før man går videre med eventuell videreutvikling av systemet er det viktig å løse eller i det minste gjøre en grundig uttredelse av de tekniske utfordringene som systemet har. De tekniske utfordringene som preger systemet ved prosjektets slutt har blitt grundig gjennomgått i Seksjon 21 og oppramses i listen nedenfor – med referanse til seksjonene hvor de ble gjennomgått.

- Forbedring av systemet ved bruk av Cloud Functions, gjennomgått i Seksjon 21.1.
- Optimalisering av datastruktur for å minimere kostnader, gjennomgått i Seksjon 21.2.1.
- Løse problemet rundt tilsynenes relasjoner, gjennomgått i Seksjon 21.2.1.
- Forbedre personvernet, gjennomgått i Seksjon 21.2.2.
- Endre bildelagring for å tillate tilsyn å ha mer enn 10 MB data, gjennomgått i Seksjon 21.3.

25.3 Ny funksjonalitet

På grunn av den begrensede tidsperioden vi hadde for å fullføre prosjektet var det noe tilleggsfunksjonalitet vi ikke hadde tid til å inkludere, som diskutert i Seksjon 24.2.4. Dette er funksjonalitet som vi sammen med veileder har kommet fram til at muligens kan ha nytteverdi for sauebønder og gjetere, og kan bli funnet i listen nedenfor – med referanse til seksjonene som utreder for dem. Dermed vil dette være funksjonalitet som det er naturlig å legge til ved videreutvikling av systemet, etter at de tekniske utfordringene har blitt utbedret.

- «Heatmap», diskutert i Seksjon 24.2.4.1.
- Tidslinje av tilsyn, diskutert i Seksjon 24.2.4.2.
- Trusselbilde, diskutert i Seksjon 24.2.4.3.
- Integrering av sporingsbjeller, diskutert i Seksjon 24.2.4.4.

26 Konklusjon

For å kunne produsere en obligatorisk oppsummeringsrapport for beitesesongen, er det viktig at bønder og gjetere tar notater under de påkrevde ukentlige tilsynene i beitemarka. Vi har tidligere, gjennom fordypningsprosjektet, utviklet en mobilapplikasjon som digitaliserer prosessen med å ta slike notater. Målet for dette prosjektet var å utvikle en webapplikasjon som presenterer og oppsummerer de digitale tilsynsnotatene, og å sammenligne bruken av et slikt system med bruk av de mer tradisjonelle papirnotatene.

Resultatet av utviklingen vår er et system som tilfredsstiller alle de planlagte kravene, og som er preget av høy brukskvalitet verifisert gjennom to faser med brukertesting. Gjennom bruk av varierte datagenereringsmetoder har vi besvart forskningsspørsmålene for prosjektet. Vi har avdekket at systemet vi utviklet effektiviserer arbeidet med å oppsummere tilsynsnotater og fasiliterer for lettere deling av tilsynsdata innenfor et beitelag.

For å kunne vurdere systemets nytteverdi innenfor et virkelig bruksscenario vil det kreves videre testing på bønder og gjetere gjennom en hel beitesesong. Dette vil kunne avdekke eventuelle problemer systemet har som ikke lot seg avsløre i brukertestene, som kun simulerte det tiltenkte bruksmiljøet. Det gjenstår også et par mindre tekniske utfordringer som burde adresseres hvis systemet skulle bli satt i produksjon. I tillegg er det mulig å utvide systemet med tilleggsfunksjonalitet som trolig kan øke nytteverdien til systemet videre.

Referanser

[1] A. J. Jonassen og L. E. Vassbotn, «Fordypningsprosjekt: Manuell Sanking av Sau», 2021.

[2] Statsforvalteren i Oslo og Viken. «Når rovvilt tar beitedyr», adresse: https://www.statsforvalteren.no/oslo-og-viken/miljo-og-klima/rovvilt/nar-rovvilt-tar-beitedyr/ (sjekket 2. feb. 2022).

[3] Mattilsynet. «Om Mattilsynet», adresse: https://www.mattilsynet.no/om_mattilsynet/ (sjekket 2. feb. 2022).

[4] Miljødirektoratet. «Dokumentere rovviltskade på husdyr og tamrein», adresse: https://www.miljodirektoratet.no/ansvarsomrader/tilsyn-naturoppsyn/naturoppsyn-og-kontroll#Dokumentere_rovviltskade_p_husdyr_og_tamrein_5 (sjekket 2. feb. 2022).

[5] Briony J. Oates, *Researching Information Systems and Computing*. London: SAGE Publications Ltd., 2006.

[6] A. Blix og O. Vangen, «Sau», 2021. Adresse: https://snl.no/sau (sjekket 30. jan. 2022).

[7] J. R. E. Johanssen og K. M. Sørheim, «Atferd og velferd hos sau», 2021. Adresse: https://www.agropub.no/fagartikler/atferd-og-velferd-hos-sau (sjekket 30. jan. 2022).

[8] O. Vangen, «Småfe», 2021. Adresse: https://snl.no/sm%C3%A5fe (sjekket 30. jan. 2022).

[9] Lovdata. «Forskrift om velferd for småfe», adresse: https://lovdata.no/forskrift/2005-02-18-160/ (sjekket 2. feb. 2021).

[10] Statistisk Sentralbyrå (SSB). «12660: Husdyr på utmarksbeite (K) 1995 - 2020», adresse: https://www.ssb.no/statbank/table/12660/ (sjekket 28. jan. 2022).

[11] O. M. Harstad, «Beite», 2021. Adresse: https://snl.no/beite (sjekket 30. jan. 2022).

[12] Lovdata. «Forskrift om erstatning når husdyr blir drept eller skadet av rovvilt», adresse: https://lovdata.no/dokument/SF/forskrift/2014-05-30-677 (sjekket 2. feb. 2022).

[13] Rovbase. (2021). «Erstatning for sau», adresse: https://www.rovbase.no/erstatning/sau (sjekket 31. jan. 2022).

[14] Statens Naturoppsyn. (2011). «Drept av rovvilt?», adresse: https://www.miljodirektoratet.no/globalassets/publikasjoner/m1319/m1319.pdf (sjekket 14. feb. 2022).

[15] Statensforvalteren i Innlandet. (2021). «Rovvilt og beitesesongen 2021», adresse: https://www.statsforvalteren.no/contentassets/1950101037d64328acdde67b89565cc8/informasjonsbrosjyre-rovvilt-og-beitesesong-2021-a5_web.pdf (sjekket 13. feb. 2022).

[16] Mattilsynet. (2020). «Mattilsynets årsrapport for 2020», adresse: https://www.mattilsynet.no/om_mattilsynet/mattilsynets_aarsrapport_for_2020.42901/binary/Mattilsynets%20%C3%A5rsrapport%20for%202020 (sjekket 4. feb. 2022).

[17] Dyrebeskyttelsen. «Tap av sau på beite», adresse: https://www.dyrebeskyttelsen.no/tap-sau-pa-beite/ (sjekket 2. feb. 2022).

[18] Lovdata. «Forskrift om merking, registrering og rapportering av småfe». Merk (19. mai 2022): Loven har siden blitt opphevet som følge av *Forskrift om endring i forskrifter om dyrehelse for landdyr mv. ved fastsetting av nye dyrehelseforskrifter* (https://lovdata.no/dokument/LTI/forskrift/2022-04-06-635). Den opphevede loven kan nå bli funnet på følgende lenke: https://lovdata.no/dokument/SFO/forskrift/2005-11-30-1356, adresse: https://lovdata.no/dokument/SF/forskrift/2005-11-30-1356 (sjekket 2. feb. 2022).

[19] Nortura. «Elektronisk merking av småfe (RFID)», adresse: https://medlem.nortura.no/smaafe/RFID/elektronisk-merking/ (sjekket 2. feb. 2022).

[20] Nortura. «Rasjonalisering av produksjon og omsetting av sau og lam ved hjelp av elektroniske hjelpemidler»,
adresse: https://medlem.nortura.no/smaafe/RFID/rasjonalisering/ (sjekket 18. mai 2022).

[21] OS ID AS. «Combi 3000 Små øremerke»,
adresse: https://www.osid.no/product/combi-3000-sma-oremerke/ (sjekket 18. mai 2022).

[22] Norsk Sau og Geit (NSG). «Bjelleslips - kodemerking for lammetall på beite»,
adresse: https://www.nsg.no/beitebruk/utmarksbeite/merking-av-smafe/bjelleslips/ (sjekket 2. feb. 2022).

[23] Norsk Sau og Geit (NSG) Oppland. «Slips og fargekoder»,
adresse: https://www.nsg.no/oppland/nyheter/slips-og-fargekoder (sjekket 3. feb. 2022).

[24] Rennebu Kommune. (2021). «Beite og gjerding»,
adresse: https://www.rennebu.kommune.no/innhold/landbruk-nering/beite-og-gjerding/ (sjekket 3. feb. 2022).

[25] Bondeboka. «Sanking og beitelag», adresse:
https://www.bondeboka.no/sau-kjot-mjolk/sanking-beitelag-og-produksjonstilskot/ (sjekket 3. feb. 2022).

[26] Miljødirektoratet. (2021). «Kvote for betinget skadefelling av gaupe og brunbjørn 2021/2022»,
adresse: https://www.miljovedtak.no/Registrering/LastNedFil?FilId=5fe29801-97e8-4d12-f48b-08d9240223b4 (sjekket 16. feb. 2022).

[27] Lovdata. «Forskrift om tilskudd til forebyggende tiltak mot rovviltskader og konfliktdempende tiltak»,
adresse: https://lovdata.no/dokument/SF/forskrift/2013-01-01-3 (sjekket 16. feb. 2021).

[28] Findmy AS. «FindMy - Produkt»,
adresse: https://www.findmy.no/nb/produkt (sjekket 28. jan. 2022).

[29] Findmy AS. «FindMy - Funksjoner»,
adresse: https://www.findmy.no/nb/funksjoner (sjekket 28. jan. 2022).

[30] Findmy AS. «FindMy - Bestill», adresse: https://www.findmy.no/shop (sjekket 28. jan. 2022).

[31] Telespor AS. «Telespor - Om Oss»,
adresse: https://nettbutikk.telespor.no/pages/om-oss (sjekket 28. jan. 2022).

[32] Telespor AS. «Telespor - Brukkerstøtte»,
adresse: https://telespor.no/brukerstotte/ (sjekket 28. jan. 2022).

[33] Telespor AS. «Telespor - Produkt»,
adresse: https://telespor.no/produkt/ (sjekket 28. jan. 2022).

[34] Telespor AS. «Telespor - Nettbutikk»,
adresse: https://nettbutikk.telespor.no/categories/radiobjella (sjekket 28. jan. 2022).

[35] A. S. Haugset og G. Nossum,
«Erfaringer med bruk av elektronisk overvåkningsutstyr på sau i 2010», 2010. adresse: https://www.bondelaget.no/getfile.php/13117498-1310042486/MMA/Bilder%20fylker/Nord%20-%20Tr%C3%B8ndelag/Dokumenter/Radibojeller%20notat%202010.pdf.

[36] Norsk Sau og Geit (NSG). «Verdisatser for småfe»,
adresse: https://www.nsg.no/om-nsg/okonomi/verdisatser/ (sjekket 28. jan. 2022).

[37] Fant AS. «BeiteSnap», adresse: https://www.beitesnap.no/ (sjekket 4. mai 2022).

[38] Fant AS. «BeiteSnap brukermanual»,
adresse: https://www.beitesnap.no/download/228/ (sjekket 4. mai 2022).

[39] Firebase. (2021). «Cloud Firestore Data Model»,
adresse: https://firebase.google.com/docs/firestore/data-model (sjekket 7. feb. 2022).

[40] Firebase. (2022). «Non-existent ancestor documents»,
adresse: https://firebase.google.com/docs/firestore/using-console?authuser=1#non-existent_ancestor_documents (sjekket 7. feb. 2022).

[41] Workfront. «Waterfall Methodology», adresse: https://www.workfront.com/project-management/methodologies/waterfall (sjekket 18. mar. 2022).

[42] Meredith Courtemanche. «What is DevOps? The ultimate guide», adresse: https://www.techtarget.com/searchitoperations/definition/DevOps (sjekket 18. mar. 2022).

[43] Priya Pedamkar. «Iterative Methodology», adresse: https://www.educba.com/iterative-methodology/ (sjekket 18. mar. 2022).

[44] Agilealliance. «What is Agile?», adresse: https://www.agilealliance.org/agile101/ (sjekket 18. mar. 2022).

[45] Atlassian. «What is Kanban?», adresse: https://www.atlassian.com/agile/kanban (sjekket 19. mar. 2022).

[46] Phoenixnap. «What is SDLC? Phases of Software Development, Models, & Best Practices», adresse: https://phoenixnap.com/blog/software-development-life-cycle (sjekket 19. mar. 2022).

[47] Agilealliance. «Extreme Programming (XP)», adresse: https://www.agilealliance.org/glossary/xp/#q=~(infinite~false~filters~(postType~(~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'xp))~searchTerm~'~sort~false~sortDirection~'asc~page~1) (sjekket 19. mar. 2022).

[48] Lauren Durdan. «When is it appropriate to choose the Kanban method», adresse: https://djaa.com/when-is-it-appropriate-to-choose-the-kanban-method/ (sjekket 17. apr. 2022).

[49] K. E. Wiegers og J. Beatty, «Use cases and usage scenarios», i *Software requirements*, Third Edition. Microsoft Press, 2013, s. 149–157.

[50] P. Rose, B. Balasubramaniam, J. Cleland-Huang, R. Wieringa, M. Daneva og S. Ghaisas, «Identifying Architecturally Significant Functional Requirements», mai 2015. DOI: 10.1109/TwinPeaks.2015.9.

[51] The International Organization for Standardization. (2011). «ISO/IEC 25010», adresse: https://www.iso.org/standard/35733.html (sjekket 8. feb. 2022).

[52] W3. (2009). «Task Model», adresse: https://www.w3.org/2005/Incubator/model-based-ui/wiki/Task_Model#Further_Readings (sjekket 8. feb. 2022).

[53] Datatilsynet. (2019). «Personvernprinsippene», adresse: https://www.datatilsynet.no/rettigheter-og-plikter/personvernprinsippene/ (sjekket 8. feb. 2022).

[54] Google. «Firebase», adresse: https://firebase.google.com/ (sjekket 15. feb. 2022).

[55] Educative.io. «What is Firebase?», adresse: https://www.educative.io/edpresso/what-is-firebase (sjekket 13. feb. 2022).

[56] Agolia. «The best Search and Discovery Platform for your business», adresse: https://www.algolia.com/ (sjekket 18. feb. 2022).

[57] Stripe. «Payments infrastructure for the internet», adresse: https://stripe.com/en-no (sjekket 18. feb. 2022).

[58] Doug Stevenson. «What is Firebase? The complete story, abridged.», adresse: https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0 (sjekket 15. feb. 2022).

[59] MongoDB. «What is a Document Database?», adresse: https://www.mongodb.com/document-databases (sjekket 15. feb. 2022).

[60] Integrant. «When to Use SQL vs. NoSQL», adresse: https://integrant.com/blog/when-to-use-sql-vs-nosql/ (sjekket 17. feb. 2022).

[61] Google. «Custom-claim attributes and roles», adresse: https://firebase.google.com/docs/rules/basics#custom-claim_attributes_and_roles (sjekket 15. feb. 2022).

[62] AWS. «Amazon S3 Object storage built to retrieve any amount of data from anywhere», adresse: https://aws.amazon.com/s3/ (sjekket 30. mai 2022).

[63] Firebase. «Introduction to Firebase Local Emulator Suite», adresse: https://firebase.google.com/docs/emulator-suite (sjekket 16. apr. 2022).

[64] Sacha Greif and Raphaël Benitte. «Front-end Frameworks», adresse: https://2021.stateofjs.com/en-US/libraries/front-end-frameworks (sjekket 16. feb. 2022).

[65] Mosh Hamedani. «React Virtual DOM Explained in Simple English», adresse: https://programmingwithmosh.com/react/react-virtual-dom-explained/ (sjekket 17. feb. 2022).

[66] React. «Composition», adresse: https://reactjs.org/docs/design-principles.html#composition (sjekket 18. feb. 2022).

[67] Flavio Copes. «Unidirectional Data Flow in React», adresse: https://flaviocopes.com/react-unidirectional-data-flow/ (sjekket 17. feb. 2022).

[68] Rich Harris. «Virtual DOM is pure overhead», adresse: https://svelte.dev/blog/virtual-dom-is-pure-overhead (sjekket 17. feb. 2022).

[69] Nathan Sebhastian. «The React Scripts Start Command – Create-React-App NPM scripts explained», adresse: https://www.freecodecamp.org/news/create-react-app-npm-scripts-explained/ (sjekket 17. feb. 2022).

[70] Sodeeq Elusoji. «Should you use Svelte in production?», adresse: https://blog.logrocket.com/should-you-use-svelte-in-production/ (sjekket 17. feb. 2022).

[71] Oleksandr Demian. «Composition in Svelte 3: slots», adresse: https://dev.to/9zemian5/composition-in-svelte-3-slots-82d (sjekket 17. feb. 2022).

[72] Rakesh Patel. «What is Vue.js? The Pros and Cons of Vue.js Framework», adresse: https://www.spaceo.ca/what-is-vue-js-and-its-pros-and-cons/ (sjekket 17. feb. 2022).

[73] ThemeSelection. «Vue Ecosystem», adresse: https://medium.com/js-dojo/vue-ecosystem-979773a9bf54 (sjekket 17. feb. 2022).

[74] Altexsoft. «The Good and the Bad of Vue.js Framework Programming», adresse: https://www.altexsoft.com/blog/engineering/pros-and-cons-of-vue-js/ (sjekket 17. feb. 2022).

[75] Microsoft. «TypeScript is JavaScript with syntax for types», adresse: https://www.typescriptlang.org/ (sjekket 16. feb. 2022).

[76] Z. Gao, C. Bird og E. T. Barr, «To Type or Not to Type: Quantifying Detectable Bugs in JavaScript», i *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, 2017, s. 758–769. DOI: 10.1109/ICSE.2017.75.

[77] Babel. «Babel is a JavaScript compiler.», adresse: https://babeljs.io/ (sjekket 16. feb. 2022).

[78] Ecma International, Technical Committee 39. «The TC39 Process», adresse: https://tc39.es/process-document/ (sjekket 16. feb. 2022).

[79] Sacha Greif and Raphaël Benitte. «JavaScript Flavors», adresse: https://2020.stateofjs.com/en-US/technologies/javascript-flavors/ (sjekket 16. feb. 2022).

[80] SurviveJS. «What is Webpack», adresse: https://survivejs.com/webpack/what-is-webpack/ (sjekket 16. apr. 2022).

[81] E. Kinsbruner. «Cypress vs. Selenium: What's the Right Cross-Browser Testing Solution for You?», adresse: https://www.perfecto.io/blog/cypress-vs-selenium-whats-right-cross-browser-testing-solution-you (sjekket 15. mai 2022).

[82] Jetbrains. «Intellij IDEA», adresse: https://www.jetbrains.com/idea/ (sjekket 16. feb. 2022).

[83] Microsoft. «Visual Studio Code», adresse: https://www.jetbrains.com/idea/ (sjekket 16. feb. 2022).

[84] Redhat. «What is CI/CD?», adresse: https://www.redhat.com/en/topics/devops/what-is-ci-cd (sjekket 16. apr. 2022).

[85] Gitlab. «GitLab CI/CD», adresse: https://docs.gitlab.com/ee/ci/ (sjekket 16. apr. 2022).

[86] Andreas Jensen Jonassen and Lars Erik Vassbotn. «Master-of-the-sheeps», adresse: https://gitlab.stud.idi.ntnu.no/andreajj/master-of-the-sheeps (sjekket 16. apr. 2022).

[87] L. Bass, P. Clements og R. Kazman, *Software architecture in practice*. Addison-Wesley, 2022.

[88] E. Gamma, R. Helm og R. Johnson, *Design patterns elements of Reusable Object Oriented Software*. Addison Wesley, 1998.

[89] M. Keeling, *Design it! From programmer to software architect*. The Pragmatic Bookshelf, 2017.

[90] Zeronet. «Open, free and uncensorable websites, using Bitcoin cryptography and BitTorrent network», adresse: https://zeronet.io/ (sjekket 18. feb. 2022).

[91] P. Kruchten, «Architecture blueprints—the "4+1" View model of software architecture», *Tutorial proceedings on TRI-Ada '91 Ada's role in global markets: solutions for a changing complex world - TRI-Ada '95*, nov. 1995. DOI: 10.1145/216591.216611.

[92] Firebase. «Perform simple and compound queries in Cloud Firestore», adresse: https://firebase.google.com/docs/firestore/query-data/queries (sjekket 19. feb. 2022).

[93] Yuchen Shi and Tyler Crowe. «Say hello to the helpful Firebase Emulator - a local first UI to boost your productivity», adresse: https://firebase.googleblog.com/2020/05/local-firebase-emulator-ui.html (sjekket 19. feb. 2022).

[94] Firebase. «Set up the Local Emulator Suite», adresse: https://firebase.google.com/docs/rules/emulator-setup (sjekket 16. mai 2022).

[95] Firebase. «Quotas and limits», adresse: https://cloud.google.com/firestore/quotas (sjekket 19. feb. 2022).

[96] PostgresSQL. «Appendix K. PostgreSQL Limits», adresse: https://www.postgresql.org/docs/12/limits.html (sjekket 19. feb. 2022).

[97] Material Design. «Material Design», adresse: https://material.io/ (sjekket 10. feb. 2022).

[98] D. Todorovic. «Gestalt principles», adresse: http://www.scholarpedia.org/article/Gestalt_principles (sjekket 16. mai 2022).

[99] Jakob Nielsen. «10 Usability Heuristics for User Interface Design», adresse: https://www.nngroup.com/articles/ten-usability-heuristics/ (sjekket 4. mai 2022).

[100] Edpresso Team. «What are Norman's design principles?», adresse: https://www.educative.io/edpresso/what-are-normans-design-principles (sjekket 4. mai 2022).

[101] Thomas Hamilton. «TEST PLAN: What is, How to Create (with Example)», adresse: https://www.guru99.com/what-everybody-ought-to-know-about-test-planing.html (sjekket 20. mar. 2022).

[102] International Organization for Standardization. (2018). «Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts», adresse: https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-2:v1:en (sjekket 1. apr. 2022).

[103] J. S. Dumas og J. C. Redish, *A Practical Guide to Usability Testing*. Intellect Books, 1999.

[104] John Brooke, «SUS - A quick and dirty usability scale», 1995.

[105] U. Malt og S. Grønmo. (2020). «Likert-skala», adresse: https://snl.no/Likert-skala (sjekket 30. mar. 2022).

[106] A. Bangor, P. T. Kortum og J. T. Miller, «An Empirical Evaluation of the System Usability Scale», *Intl. Journal of Human-Computer Interaction*, s. 574–594, 2008.

[107] Open Web Application Security Project. «OWASP», adresse: https://owasp.org/ (sjekket 22. apr. 2022).

[108] Open Web Application Security Project. «OWASP Top Ten», adresse: https://owasp.org/www-project-top-ten/ (sjekket 22. apr. 2022).

[109] Open Web Application Security Project. «A01:2021 – Broken Access Control», adresse: https://owasp.org/Top10/A01_2021-Broken_Access_Control/ (sjekket 22. apr. 2022).

[110] Open Web Application Security Project. «A02:2021 – Cryptographic Failures», adresse: https://owasp.org/Top10/A02_2021-Cryptographic_Failures/ (sjekket 22. apr. 2022).

[111] Google Cloud. «Server-side encryption», adresse: https://cloud.google.com/firestore/docs/server-side-encryption (sjekket 22. apr. 2022).

[112] Open Web Application Security Project. «Command Injection», adresse: https://owasp.org/www-community/attacks/Command_Injection (sjekket 22. apr. 2022).

[113] Open Web Application Security Project. «Cross Site Scripting (XSS)», adresse: https://owasp.org/www-community/attacks/xss/ (sjekket 5. mai 2022).

[114] Open Web Application Security Project. «A04:2021 – Insecure Design», adresse: https://owasp.org/Top10/A04_2021-Insecure_Design/ (sjekket 24. apr. 2022).

[115] Open Web Application Security Project. «A08:2021 – Software and Data Integrity Failures», adresse: https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/ (sjekket 24. apr. 2022).

[116] GitHub Advisory Database. «Embedded malware in coa», adresse: https://github.com/advisories/GHSA-73qr-pfmq-6rp8 (sjekket 24. apr. 2022).

[117] Open Web Application Security Project. «A09:2021 – Security Logging and Monitoring Failures», adresse: https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/ (sjekket 24. apr. 2022).

[118] Open Web Application Security Project. «A10:2021 – Server-Side Request Forgery (SSRF)», adresse: https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/ (sjekket 24. apr. 2022).

[119] Firebase. «Cloud Firestore - Key capabilities», adresse: https://firebase.google.com/docs/firestore#key_capabilities (sjekket 8. mai 2022).

[120] Norsk Folkehjelp. «Geografisk Navigasjon», adresse: https://folkehjelp.no/forstehjelp-og-redningstjeneste/ressurser-for-frivillige/navigasjonsutstyr-i-s%C3%B8k-og-redning (sjekket 8. mai 2022).

[121] MongoBD. «Model One-to-Many Relationships with Document References», adresse: https://www.mongodb.com/docs/manual/tutorial/model-referenced-one-to-many-relationships-between-documents/ (sjekket 3. mai 2022).

[122] Norsk Sau og Geit. «Rapportskjema – Tilsyn på utmarksbeite», adresse: https://www.nsg.no/getfile.php/134652-1283259043/_NSG-PDF-filer/Beitebruk/Organisert%20beitebruk/Rapportskjema.PDF (sjekket 11. feb. 2022).

[123] Statsforvalteren.no. (2018). «Rapportskjema for tilsyn av dyr på utmarksbeite 2018», adresse: https://www.statsforvalteren.no/contentassets/1da77da59c9d454686db3a94cde84273/rapportskjema-for-tilsyn-av-dyr-pa-utmarksbeite-2018.pdf (sjekket 11. feb. 2022).

Del IX:

Vedlegg

Tillegg A Møte 3. februar 2022

Møte med Svein-Olaf

3. februar 2022

Notater:

- Hva er riktig statistikk for normaltap i 2021?
    - Rovbase sier ca. 2000 sau og 9000 lam.
    - Mattilsynet sier totalt 102k tap i 2019 i deres årsrapport.
    - **Svar:** rovbase sier om innmeldte normaltap.
        - "100k" er ESTIMERTE normaltap.
- Vet du om noen områder hvor de ikke tar i bruk NSG sin standardisering av slipsfarger? Eller er NSG sin standard brukt over hele landet?
    - **Svar:** Hele Norge bruker nå NSG.
- Kan øremerker på sauen ha todelte farger? Hvordan fungerer to farger?
    - Evt. én gård har to unike farger som representerer den?
    - To farger ved siden av hverandre som til sammen identifiserer gården?
    - **Svar:** to farger på samme øremerke.
        - En farge er mest vanlig.
- Mattilsynets krav om øremerking: ett elektronisk og ett visuelt. Det visuelle er det samme som vi ser på i dette prosjektet, sant?
    - **Svar:** elektronisk er ikke noe vi bruker, et system som har rekkevidde på 0.5m og brukes når sauen kommer tilbake hjem, typ om de skal slaktes eller tas vare på over vinteren.
- **Svar:** Trenger ikke være noen instruks på hva man skal ta bilde av angående død sau.
- Stemmer det at det ikke finnes noen konkrete krav til hva man skal skrive ned under tilsynsrunde med penn og papir? Skjemaene har bare fritekst-boks, virker det som.
    - Har hørt at dette er krav:
        - Dato
        - Totalt antall sau og lam
        - Tilfeller av død eller skadede dyr
    - **Svar:**
        - Send inn en rapport om når du var ute og gikk, hva du så, og hvor du så det, døde og skadde dyr; farge, voksne dyr eller lam.
            - Dette er godkjent av relevante etater (fylkeskommunal etat).

Skadd sau kan være en del av flokk.

- Du kommer som regel ikke nærme nok til det skadde dyret.
- Men bilde skal være mulighet.
- Kan også ha bilde selv uten skadet sau, uvanlige ting i flokken etc.
- Død sau skjerm:
    - Burde ikke stoppe brukeren, men hvertfall en advarsel.

Død sau:

- Ta bilde av øremerke (få med teksten som står skrevet på den)

Tillegg B Tilsynsskjema fra NSG[122]

Organisert beitebruk

RAPPORTSKJEMA – TILSYN PÅ UTMARKSBEITE

Dette skjemaet kan nyttast for å sikre beitelaget ein dokumentasjon på utført tilsyn med dyra i utmarka gjennom beitesesongen, slik det er sett krav om i *Forskrift om tilskott til organisert beitebruk.* Det er lagt opp til at kvar tur skal noterast fortløpande på dette skjemaet. Av aktuelle opplysningar kan nemnast daude og skadde dyr, årsak, rovvilt, laushundar, generell uro, søyer som manglar lam, beiteforhold og anna. Kartfesting av observasjonar som tillegg til dette skjemaet kan vera aktuelt. Etter endt beitesesong skal oppsummeringsskjemaet på neste side (baksida) fyllast ut og sendast leiaren i laget.

Beitelag: ……. Beiteår: ……

Tilsynsperson: ……..

Dato:	Rute/område:
Observasjonar:	
Dato:	Rute/område:
Observasjonar:	
Dato:	Rute/område:
Observasjonar:	
Dato:	Rute/område:
Observasjonar:	

Dato:	Rute/område:
Observasjonar:	
Dato:	Rute/område:
Observasjonar:	
Dato:	Rute/område:
Observasjonar:	
Dato:	Rute/område:
Observasjonar:	

Oppsummering av beitesesongen - informasjon om dyretal, tap og eigeninnsats, inkludert km køyring

	Sau	Lam	Sau + lam	Storfe	Geit/kje
Dyr sleppt på utmarksbeite					
Dyr tapt på utmarksbeite					
Tap i % (dyr tapt/dyr sleptx100)					
Eigeninnsats, dagar:	Tilsyn	Sanking	Anna arbeid	Samla	Køyring, km

Tillegg C Tilsynsskjema fra Statsforvalteren[123]

RAPPORTSKJEMA FOR TILSYN AV DYR PÅ UTMARKSBEITE 2018

Namn: _____

Organisasjonsnummer: _____

Tabellen **Oppsummering av beitesesongen** (under) skal etter beitesesongen fyllast ut og sendast til leiaren i beitelaget. Tapstal som skal rapporterast er berre tap på utmarksbeite. Tap inne og på vår- og haustbeite på innmark skal ikkje takast med i denne statistikken. For å få best mogleg oversyn over tapsårsaker er det viktig at kvar bonde i si rapportering til beitelaget om hausten melder inn ei så fullstendig opplisting av sikre og trulege tapsårsaker for sin buskap som mogeleg. Styret skal deretter rapportere inn ei tilsvarande oversikt til Fylkesmannen for laget saman med laget sin søknad om driftsmidlar

Tilsynslogg (sjå baksida) kan nyttast for å sikre dokumentasjon på utført tilsyn med dyra i utmarka gjennom beitesesongen. Kvar tur skal noterast fortløpande i tabellen. Aktuelle opplysningar kan t.d. vera: daude og skadde dyr, tapsårsaker, observasjon av rovvilt, sauer som manglar lam, generell uro i beitet, beiteforhold med meir.

Dyr sleppt/ sanka på *utmarksbeite*	**Sau**		**Lam**		**Geit**		**Storfe**	
	Sleppt	Sanka	Sleppt	Sanka	Sleppt	Sanka	Sleppt	Sanka

Tilsyn i dagsverk per veke (gjennomsnittleg):	

Sikre/dokumenterte tapsårsaker :								
Flått	Alveld	Rev	Konge-ørn	Freda rovdyr	Ulukker	Svake dyr	Flugemakk	Anna
Trulege tapsårsaker :								
Flått	Alveld	Rev	Konge-ørn	Freda rovdyr	Ulukker	Svake dyr	Flugemakk	Anna
Kommentarar:								

Tilsynslogg for: …… **Beiteår:** ……..

Dato:	**Rute/område:**
Observasjonar:	
Dato:	**Rute/område:**
Observasjonar:	
Dato:	**Rute/område:**
Observasjonar:	
Dato:	**Rute/område:**
Observasjonar:	
Dato:	**Rute/område:**
Observasjonar:	
Dato:	**Rute/område:**
Observasjonar:	
Dato:	**Rute/område:**
Observasjonar:	
Dato:	**Rute/område:**
Observasjonar:	

Tillegg D System Usability Scale - Spørsmålsskjema på norsk

Noen spørsmål om systemet du har brukt.

Vennligst sett kryss i kun en rute pr. spørsmål.

	Sterkt uenig				Sterkt enig
1. Jeg kunne tenke meg å bruke dette systemet ofte.	1	2	3	4	5
2. Jeg synes systemet var unødvendig komplisert.	1	2	3	4	5
3. Jeg synes systemet var lett å bruke.	1	2	3	4	5
4. Jeg tror jeg vil måtte trenge hjelp fra en person med teknisk kunnskap for å kunne bruke dette systemet.	1	2	3	4	5
5. Jeg syntes at de forskjellige delene av systemet hang godt sammen.	1	2	3	4	5
6. Jeg syntes det var for mye inkonsistens i systemet. (Det virket "ulogisk")	1	2	3	4	5
7. Jeg vil anta at folk flest kan lære seg dette systemet veldig raskt.	1	2	3	4	5
8. Jeg synes systemet var veldig vanskelig å bruke	1	2	3	4	5
9. Jeg følte meg sikker da jeg brukte systemet.	1	2	3	4	5
10. Jeg trenger å lære meg mye før jeg kan komme i gang med å bruke dette systemet på egen hånd.	1	2	3	4	5

|

SUS
Norsk versjon ved Dag Svanæs
NTNU 2006

Tillegg E Materiale for brukertesting - ark med notater fra tilsyn

Organisert beitebruk

RAPPORTSKJEMA – TILSYN PÅ UTMARKSBEITE

Dette skjemaet kan nyttast for å sikre beitelaget ein dokumentasjon på utført tilsyn med dyra i utmarka gjennom beitesesongen, slik det er sett krav om i *Forskrift om tilskott til organisert beitebruk*. Det er lagt opp til at kvar tur skal noterast fortløpande på dette skjemaet. Av aktuelle opplysningar kan nemnast daude og skadde dyr, årsak, rovvilt, laushundar, generell uro, søyer som manglar lam, beiteforhold og anna. Kartfesting av observasjonar som tillegg til dette skjemaet kan vera aktuelt. Etter endt beitesesong skal oppsummeringsskjemaet på neste side (baksida) fyllast ut og sendast leiaren i laget.

Beitelag: TRONDHEIMS LAGET **Beiteår:** 2021

Tilsynsperson: JOTUN GÅRDSEN

Dato:	Rute/område:
20.5	LADESTIEN

Observasjonar:
- FLOKK 1: 6 SAU, 3 LAM.
- FOTSPOR FRA ULV.
- FLOKK 2: 5 SAU, 1 LAM. 1 HAR SÅR.

Dato:	Rute/område:
2.6	BYÅSEN

Observasjonar:
- 2 DØDE SAU.
- FLOKK 1: 4 SAU, 1 LAM.
- GJØRMETE TERRENG.
- FLOKK 2: 3 SAU, 0 LAM.

Dato:	Rute/område:
15.6	TILLER

Observasjonar:
- SÅ EN BJØRN.
- FLOKK 1: 5 SAU, 2 LAM. 1 HAR BRUKKET BEN.

Dato:	Rute/område:
5.7	VIKÅSEN

Observasjonar:
- DØD SAU LANGS VEIEN. PÅKJØRT?
- FLOKK 1: 6 SAU, 3 LAM.
- ELLERS ALT OK.

Tillegg F Materiale for brukertesting - oppsummeringsskjema for tilsynsnotater

Oppsummering av tilsyn i løpet av sesongen

Navn: _____ Dato: _____

Oppsummering av hvert tilsyn:

Dato:	Rute/område:	Antall sau:	Antall lam:	Antall skadde dyr:	Antall døde dyr:	Antall rovdyr/ rovdyrtegn:

Oppsummering for sesongen:

Antall tilsyn utført: _____

Totalt antall sau: _____

Totalt antall lam: _____

Totalt antall skadde dyr: _____

Totalt antall døde dyr: _____

Totalt antall rovdyr/rovdyrtegn: _____