Masteroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Maria Vy Nguyen Vu

Digitalisering av manuell oppfølging av sau på utmarksbeite

Masteroppgave i Datateknologi
Veileder: Svein-Olaf Hvasshovd
Juni 2022

NTNU
Kunnskap for en bedre verden

Maria Vy Nguyen Vu

Digitalisering av manuell oppfølging av sau på utmarksbeite

Masteroppgave i Datateknologi
Veileder: Svein-Olaf Hvasshovd
Juni 2022

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU
Norwegian University of
Science and Technology

Sammendrag

I følge Mattilsynet er det årlig rundt 2 millioner sauer på utmarksbeite på våren i Norge. I løpet av beitesesongen er det 125 000 sauer som dør hvert år, av årsaker som angrep fra rovdyr, sykdom og ulykker. For å unngå unødvendig tap av sau krever myndighetene at det drives jevnlig oppfølging av sauene i denne perioden. I følge norsk lov er bøndene pliktige til å se etter dyr på beite minst én gang i uken. Dette betyr at sauebonden må fysisk ut i terrenget hvor sauene befinner seg og se til at alt er som det skal. Dersom det skulle være noen sau som er syke, skadde eller døde må bonden iverksette tiltak for ivaretakelse av sauenes velferd.

På tilsynsturene til sauebonden krever også myndighetene dokumentasjon på tilstanden til sauebestanden. Denne dokumentasjonen blir sendt inn til myndighetene ved sesongens slutt. I dag blir den innsamlede informasjonen registrert manuelt med penn og papir, som fritekst uten en fastsatt struktur eller metode. Det er derfor et ønske om å lage et digitalt system som kan forenkle innsamlingsmetoden til bøndene på tilsynstur. Denne masteroppgaven presenterer designet, utviklingsprosessen, brukertestingen og evalueringen av et konsept av et system som kan forbedre og effektivisere dokumentasjonen av bøndenes tilsyn.

I evalueringen av systemet ble det foretatt syv brukertester på medstudenter og romkamerater med ulik grad av tilknytning til landbruk. Brukertestene gir en indikasjon på hvor brukervennlig systemet er, hvor godt det møter sauebøndenes behov i henhold til kravspesifikasjonen og om systemet potensielt kan erstatte eksisterende innsamlingsmetode av informasjon på tilsynsturer.

Resultatene fra brukertestene antyder at systemet per nå krever ytterligere forbedringer og videreutvikling før den er et fullverdig produkt med høy nytteverdi for sauebønder. Likevel, har konseptet til systemet vist seg å være godt motatt med positiv respons. Store deler av systemet ble opplevd som oversiktlig, strukturert og intuitivt. Registreringsprosessen av informasjon på tilsynstur var lettvint og enkel å følge, med et brukergrensesnitt som gjorde det effektivt å registrere detaljer.

Abstract

According to the Norwegian Food Safety Authority, there are around 2 million sheep grazing in the open in the spring in Norway every year. During the grazing season, 125,000 sheep die each year, due to causes such as attacks from predators, diseases and accidents. In order to avoid unnecessary loss of sheep, the authorities require that the sheep should be regularly monitored during this period. According to the Norwegian law, farmers are obliged to look after animals on pasture at least once a week. This means that the sheep farmer must physically go out into the terrain where the sheep are located and make sure that everything is as it should be. Should there be any sick, injured or dead sheep, then the farmer must implement measures to safeguard the welfare of the sheep.

On the inspection trips of the sheep farmer, the authorities also require documentation of the condition of the sheep population. This documentation is sent to the authorities at the end of the season. Today, the collected information is registered manually with pen and paper, as free text without a set structure or method. There is therefore a desire to create a digital system that can simplify the collection method for farmers on inspection trips. This master's thesis presents the design, development process, user testing and evaluation of a concept of a system that can improve and streamline the documentation of farmers' inspections.

In the evaluation of the system, seven user tests were performed on fellow students and roommates with varying degrees of association to agriculture. The user tests give an indication of how user-friendly the system is, how well it meets the needs of the sheep farmers in accordance with the requirements specification and whether the system can potentially replace the existing collection method of information on inspection trips.

The results from the user tests suggest that the system currently requires further improvements and further development before it is a full-fledged product with high utility value for sheep farmers. Nevertheless, the concept of the system has proven to be well received with a positive response. Large parts of the system were perceived as clear, structured and intuitive. The registration process of information on the inspection trip was easy and simple to follow, with a user interface that made it efficient to register details.

Forord

I løpet av dette prosjektet har det blitt utviklet et *Minimal Viable Product* i form av en mobilapplikasjon. Dersom det er ønskelig å teste ut applikasjonen er informasjon til tilgang gitt nedenfor. For å få en bedre forståelse for hvordan applikasjonen fungerer og skal brukes, er det anbefalt å lese gjennom hele masteroppgaven før applikasjonen skal testes.

For å kjøre mobilapplikasjonen lokalt på PCen kan den bli klonet og lastet ned fra *Github*:
https://github.com/Fordypningsprosjekt/frontend-app.git

Brukernavn: ola.nordmann@outlook.com

Passord: 123456

Det anbefales å lese gjennom denne guiden [79] for nærmere informasjon om hvordan utviklingsmiljøet skal settes opp og denne [78] for hvordan applikasjonen kan kjøres på en mobilenhet. Til informasjon vil noe av dataen i applikasjonen bli lagret i den tilknyttede skydatabasen. Hvis det er ønskelig å få tilgang til databasen til videre utvikling av applikasjonen kan det bli sendt en forespørsel til følgende e-postadresse: maris.vu@gmail.com

Jeg ønsker å rette en stor takk til min veileder, Professor Svein-Olaf Hvasshovd, som la grunnlaget for prosjektet og for god veiledning og gode råd gjennom hele perioden. Det ville ikke vært mulig å gjennomføre oppgaven uten ham.

Jeg ønsker også å takke alle medstudenter og romkamerater som lot meg utføre brukertester av applikasjonen på dem.

Til slutt ønsker jeg å rette en spesiell takk til min venn Casper Feng for å være en god sparringspartner i møte med tekniske utfordringer, og sist men ikke minst Haakon Solerød Tveiten for motivasjon gjennom hele denne perioden og for kreativ navngiving til applikasjonen.

Maria Vy Nguyen Vu
Trondheim, 5. juni 2022

Innhold

Figurer

Tabeller

Kapittel 1

Introduksjon

Denne masteroppgaven tar for seg hvordan manuell oppfølging av sau på utmarksbeite kan forbedres og effektiviseres ved hjelp av en mobilapplikasjon, *Bæædar*. Det blir forsket på brukergrensesnitt og om det har en innvirkning på registreringsmetoden av sau på beite. Vil et brukergrensesnitt som gjør det mulig å registrere informasjon med et tastetrykk effektivisere tilsynsturen? I tillegg til mobilapplikasjonen er det planlagt at en nettside skal bli utviklet. Nettsiden skal hente registrert informasjon fra tilsynsturer og lage en oversiktlig visualisering av det. Videre, skal denne informasjonen bli brukt til å generere en ferdigrapport som skal bli sendt til myndighetene i slutten av beitesesongen. Både mobilapplikasjonen og webapplikasjonen skal forenkle hverdagen til sauebøndene og rapporteringsprosessen til myndighetene. I denne masteroppgaven blir det presentert en løsning under utvikling og veien til valgt framgangsmåte, samt en detaljert plan for hvordan mobilapplikasjonen og nettsiden kan utvikles videre.

1.1 Motivasjon og bakgrunn

Dette masterprosjektet bygger videre på fordypningsprosjektet fra høsten 2021. På fordypningsprosjektet ble problemstilling og store deler av kravspesifikasjonen i Kapittel 3 utredet, samt håndskissene og *Figma*-skissene i Kapittel 5.1 utformet. I tillegg ble det gjort en vurdering av eksisterende systemer i Kapittel 2 og utviklingen av mobilapplikasjonen ble påbegynt.

I følge Mattilsynet er det rundt 2 millioner sauer på utmarksbeite i Norge hvert år [62]. I norsk lov står det at eier eller en annen med ansvar for dyrene er pliktig til å se etter dyr på utmarksbeite minst én gang i uken i løpet av beitesesongen [46]. Disse tilsynsturene gjennomføres av den enkelte sauebonde og informasjonen fra turene rapporteres deretter til myndighetene.

På tilsynsturene registrerer bøndene informasjon om sine observasjoner. Informasjonen som registreres er blant annet:

- Antall sau og lam
- Antall lam
- Antall brune, svarte og hvite/gråe sauer
- Eierskap (farge på øremerket)
- Farge på slips (antall lam en sau har)
- Antall skadde og døde sauer

Per nå blir disse observasjonene ført ned manuelt som fritekst uten å følge en fastsatt struktur eller metode, med penn og papir.

På bakgrunn av det ovennevnte har derfor Professor *Svein-Olaf Hvasshovd* utarbeidet et løsningsforslag om en mobilapplikasjon som kan forbedre dagens situasjon. Ettersom en saueflokk ikke nødvendigvis står i ro og som oftest er i bevegelse kan det være utfordrende å bytte mellom å observere og notere med penn og papir. En mobilapplikasjon kan derfor være løsningen på dette problemet. I denne masteroppgaven blir prosessen og arbeidet med å realisere et slikt konsept av en mobilapplikasjon presentert, samt en plan for hvordan en tilhørende nettside kan bli utviklet. Formålet med denne applikasjonen er å forenkle og effektivisere den eksisterende innsamlingsmetoden. Ved at registreringen følger en fastsatt struktur hvor observasjonene blant annet registreres i en bestemt rekkefølge, kan bonden få en bedre oversikt over hva som har blitt observert. Videre, blir ikke sauebondens posisjon og hvor det har blitt gått på tilsynsturen verken sporet eller loggført. Det er derfor et sterkt ønske om å kunne benytte seg av *Norgeskartet*, spore GPS-posisjon og hvor sauebonden har gått og loggføre hvor saueflokken har befunnet seg. Med tanke på at denne informasjonen senere skal rapporteres til myndighetene vil en slik standardisert registreringsmetode og loggført tur gjøre det lettere å automatisk generere en ferdigrapport som da kan bli sendt inn til myndighetene. Formålet med denne applikasjonen er å utvikle et produkt som har nytteverdi for både sauebøndene og myndighetene.

1.2 Mål og problemstilling

I dette kapittelet blir målet for prosjektet og problemstilling utledet. Problembeskrivelsen er tatt fra fordypningsprosjektet høsten 2021.

1.2.1 Overordnet mål

Det overordnede målet for prosjektet er å utvikle et konsept av et system som kan digitalisere og effektivisere oppfølgingen av sau på utmarksbeite. Målet har da blitt formulert på følgende måte:

Utvikle et konsept av et system som potensielt kan ha nytteverdi for både sauebønder og myndighetene. Konseptet skal kunne gjøre det enkelt og effektivt å registrere detaljert og strukturert informasjon om sau på utmarksbeite, samt ha mulighet for å ta i bruk interaktivt kart på tilsynstur.

1.2.2 Problemstilling

Problembeskrivelsen for prosjektet som Professor *Hvasshovd* har formulert er som følger:

"Sau slippes på vårparten ut på beite. Hele sommeren går sauene fritt rundt i terrenget. I denne perioden krever myndighetene at det drives jevnlig oppsyn med sauene. Bonden må da ut og gå i terrenget hvor sauene befinner seg for å sjekke at alt står bra til. I tilfelle han detekterer at noen sau er syke eller døde må han iverksette tiltak for ivaretakelse av sauenes velferd.

Fram til i dag har bøndene ført oversikt over hvilke oppsynsturer de har vært på, hvor de har gått og hva de har sett, på papir. Det er et ønske at rapporteringen av dette flyttes over til et databasert verktøy som bøndene kan ta med ut i felt på hver oppsynstur. Innholdet i rapporten og kartet over hvor bonden har gått, lastes så ned når han kommer hjem og en standardisert rapport produseres ved sesongens slutt og sendes myndighetene."

Problembeskrivelsen kan bli delt inn i tre forskningsspørsmål for å kunne evaluere det overordnede målet for prosjektet:

F1: Hvor godt dekker applikasjonen *Bæædar* sauebøndenes behov ved oppfølging av sau på utmarksbeite?
Ettersom *Bæædar* hovedsakelig skal være et verktøy for bøndene bør deres behov bli dekket i størst mulig grad.

F2: På hvilke måter er det bedre å bruke *Bæædar* til manuell oppfølging av sau på utmarksbeite sammenlignet med eksisterende innsamlingsmetode?
Formålet med å digitalisere oppfølgingsprosessen av sau er at det skal bli lettere og mer effektivt for bøndene å gjennomføre dem. Derfor er det logisk å sammenligne *Bæædar* med den eksisterende innsamlingsmetoden som blir tatt i bruk i dag.

F3: Hvor detaljert, strukturert og oversiktlig er informasjonen fra tilsynsturer samlet inn med *Bæædar* sammenlignet med eksisterende innsamlingsmetode?
En viktig egenskap ved *Bæædar* er at den innsamlede informasjonen skal være minst like presis og detaljert som ved bruk av eksisterende innsamlingsmetode. I tillegg vil strukturert data gjøre det enklere å genere en ferdigrapport senere som skal bli sendt til myndighetene.

Disse forskningsspørsmålene vil bli besvart i løpet av oppgaven.

Kapittel 2

Eksisterende systemer

Store deler av dette kapittelet er hentet fra fordypningsprosjektet høsten 2021. Det finnes flere teknologsike løsninger på markedet i dag, hvor hensikten er å forenkle tilsyn av sauer på utmarksbeite. Disse systemene er blant annet *Telespor*, *FindMy* og *Nofence*.

2.1 Telespor

Telespor er en bedrift som ble grunnlagt i 2004 og spesialiserer seg på elektronisk overvåkning av husdyr [84]. Et produkt de har er *radiobjella*. *Radiobjella* er et sporingsapparat som bruker *GNSS*-teknologi for å dele posisjonen til sauene slik at det er mulig å se hvor sauene befinner seg hen og hvor de har vært. Intervallet for hvor ofte posisjonen skal oppdateres kan bli satt etter eget ønske. Posisjonsdata sendes da til serveren, med *LTE-M* og *Narrowband IoT*-teknologi. Senderintervallet og alarmsensitiviteten er det mulig å endre på når som helst. I tillegg vil den hente de oppdaterte innstillingene man har valgt hver gang posisjonen sendes fra radiobjella. Deretter blir informasjonen sendt fra serveren til kundens PC eller mobilenhet.

Andre funksjoner som dette systemet har er for eksempel SMS varsling, hvor brukeren kan bli varslet dersom det har skjedd noe med radiobjella [84]. Radiobjella er også vanntett og tåler all slags vær, med en innebygd bevegelsessensor som utløses dersom

- Dyret ikke har beveget seg de siste 3 timene
- Dyret har vært på samme posisjon i en lengre periode
- Radiobjella ikke har klart å sende sin posisjon de siste 2 rapporteringene

Prisen på radiobjella er på kr 899 inkludert batteri, noe som tilsvarer opp mot halvparten av verdien for en sau.

2.2 FindMy

Findmy, tidligere *Findmysheep*, er en bedrift som ble startet opp av sauebønder [18]. Bedriften produserer og selger produkter til elektronisk sporing av dyr, mennesker og gjenstander. Disse produktene bruker satellitter til sporing, slik at det da ikke er nødvendig med mobildekning. De har et produkt, *E-bjella*, som har utskiftbare batterier med god levetid. Disse batteriene følger *MIL-STD-810G* standarden for utholdenhet [17]. Kommunikasjonen mellom E-bjella og bonden er via *Bluetooth*, slik at hele oppsettet av E-bjella kan gjøres på en applikasjon på mobilen eller nettbrettet. Det vil si at oppsett av meldingsplaner, oppdatering av programvare og lignende kan gjøres rett fra mobilen. I tillegg er det mulig å sette opp faste klokkeslett for når man skal få inn

posisjonsdata daglig. Videre, kan et internt *Geofence* bli lagt til i E-bjella med *Bluetooth*. Dersom dyret forlater eller kommer inn i et *Geofence*-område vil E-bjella automatisk varsle om det.

Andre funksjoner E-bjella har er blant annet stressvarsel. Når det oppstår uro i saueflokken vil produktet gi beskjed umiddelbart [17]. Prisen for en E-bjelle er på kr 1980 ekskludert mva per enhet, i tillegg kommer en årlig brukeravgift på kr 229 per bjelle.

2.3 Nofence

Bedriften *Nofence AS* ble stiftet i 2011 og tilbyr et digitalt gjerdesystem for husdyr [60]. Produktet deres er et system som har virtuell inngjerding av beitedyr, hvor brukeren selv kan tegne opp det virtuelle gjerdet som utmarksbeitet kan ligge innenfor [59]. Dette systemet kommer i form av en mobilapplikasjon, hvor bonden kan utforme beiteområdet med hensyn på terreng og omgivelser. På applikasjonen kan bonden følge med på flokken og holde oversikt over dyrene mens de er ute på beite.

Hvert dyr har på seg en solcelledrevet klave som kan gi fra seg et lydvarsel [59]. Hvis et dyr tråkker utenfor det virtuelle gjerdet, gir klaven fra seg et lydvarsel som tilsier at dyret skal snu. Dersom dyret ignorerer lydvarselet vil den oppleve et svakt strømstøt, slik at dyrene lærer seg å snu på lydvarselet. Skulle noe uforutsett oppstå, blir lyd og strømstøt frakoblet og bonden blir varslet på mobilen.

Dette digitale gjerdesystemet består av en mobilapplikasjon og en klave som kommuniserer med webportalen og applikasjonen over mobilnettet [59]. Klaven har *GPS* som gir bonden posisjonsdata til dyrene. Til tross for at gjerdefunksjonen bruker *GNSS*-posisjonering (*GPS* og *GLONASS*) for å fungere, krever det at bonden har mobildekning i hele eller store deler av det utvalgte beite dersom han/hun ønsker å gjøre endringer på inngjerdingen, samt få varsel fra klavene.

Helt siden 2017 har *Nofence* sitt gjerdesystem blitt brukt på geiter på utmarksbeite, mens på sau har det kun blitt forsket på [60]. Det var først i 2020 at Mattilsynet aksepterte bruken av *Nofence* på storfe og sau. I dag blir dette systemet brukt av 2 564 bønder, og 34 474 dyr går rundt med klaven. For å undersøke hvor mye det koster å bruke dette systemet har *Nofence* en priskalkulator på nettsiden sin [61].

2.4 Vurdering av eksisterende systemer

De tre ovennevnte systemene prøver på hver sin måte å forenkle oppfølging av dyr på beite. Både *Telespor*, *Findmy* og *Nofence* bruker en form for bjelle eller klave for å automatisk spore dyrene ved hjelp av posisjonsdata fra *GPS*.

Fordeler med disse systemene er at bøndene får en god oversikt over sauenes posisjon til enhver tid, eller til et ønsket tidspunkt. Alle systemene er utstyrt med en slags varsel-mekanisme dersom noe uforutsett skulle skje. I tillegg kan bøndene bruke posisjonsdata som samles inn til å analysere dyrenes bevegelsesmønstre, og identifisere hvilke beiteområder sauen vanligvis ferdes i. Med tanke på angrep fra rovdyr er det spesielt fordelaktig å finne ut av hvilke områder det gjelder. For eksempel med *Nofence* sitt gjerdesystem kan bøndene faktisk avgrense beiteområdet slik at sauene ikke befinner seg i faresonen.

Noen av ulempene med disse systemene er at de har funksjonaliteter som krever mobildekning. Dette er noe beiteområdene til sauene ikke nødvendigvis har. Som for eksempel kommunikasjonen mellom radiobjella og serveren i *Telespor* sitt system fungerer over mobilnett. Uten mobildekning vil bonden da ikke få tilsendt posisjonen til sauene. Med *Nofence* trenger bonden mobildekning i hele eller store deler av beiteområdet for å kunne endre på inngjerdingen og få varsler fra klavene. Derimot er ikke *Findmy* sin *E-bjelle* avhengig av mobildekning og skal fungere selv uten dekning, men er kostbart. Som nevnt, er prisen på en radiobjelle fra *Telespor* opp mot halvparten av verdien for én sau. Prisen på E-bjella til *FindMy* overgår til og med prisen på radiobjella, som koster over

dobbelt så mye som radiobjella til *Telespor*.

En ting disse systemene mangler er funksjonalitet for å registrere detaljert informasjon om observasjoner av sauer på tilsynstur. Det er for eksempel ingen funksjonalitet for å registrere antall sauer i en spesifikk farge, antall lam, farge på slipset og lignende. Disse systemene fungerer kun som hjelpemidler for å kunne utføre tilsynet mer effektivt, samt gjør det mulig for bøndene å dekke en større andel av saueflokken per oppsynstur. Det finnes ingen brukergrensesnitt for registrering av detaljert informasjon på tilsynstur eller ferdiggenerering av rapporter. Denne informasjonen må fortsatt bli notert manuelt og rapporten som skal bli sendt til myndighetene må skrives på egenhånd. Et brukergrensesnitt for registrering av informasjon på tilsynstur er noe den presenterte mobilapplikasjonen *Bæædar* i denne oppgaven kan tilby. I tillegg blir det presentert en plan for hvordan ferdiggenerering av rapporter kan bli satt opp på en fremtidig nettside.

Kapittel 3

Kravspesifikasjon

Utfra veiledningsmøter som ble gjennomført på høsten 2021 med Professor *Svein-Olaf Hvasshovd* har det blitt utledet et sett med funksjonelle og ikke-funksjonelle krav for mobilapplikasjonen og nettsiden. **Funksjonelle krav** er funksjonaliteter som systemet burde tilby. Det er krav ved responsmåten til systemet på brukerinput og hvordan systemet skal oppføre seg i ulike brukersituasjoner. **Ikke-funksjonelle krav** er funksjonaliteter som det forutsettes at systemet har, som blant annet pålitelighet og brukervennlighet. Hvert krav i kravspesifikasjon har blitt gitt en **id** og en **prioritet** som enten er **høy, middels eller lav**. Et krav med **høy** prioritet blir regnet som en svært nødvendig funksjonalitet ved systemet. **Middels** prioritet representerer et krav som burde bli implementert, men som ikke er like nødvendig. Til sist er krav med **lav** prioritet funksjonalitet som kan være nyttig for brukeren, men som strengt tatt ikke er nødvendig å implementere for at applikasjonen eller nettsiden skal fungere.

3.1 Kravspesifikasjon for mobilapplikasjon

Store deler av kravspesifikasjonen for mobilapplikasjonen er hentet fra fordypningsprosjektet høsten 2021. De funksjonelle og ikke-funksjonelle kravene til mobilapplikasjonen er vist på Tabell 3.1 og på Tabell 3.2. Ettersom det er forskjellig informasjon som skal bli registrert basert på avstanden mellom sauebonde og saueflokken, gir Kapittel 3.1.3 en nærmere beskrivelse om dette.

3.1.1 Funksjonelle krav

De funksjonelle kravene for mobilapplikasjonen er vist på Tabell 3.1.

FK-ID	Beskrivelse	Prioritet
FK-01	Sauebonden skal kunne se et kart over hele Norge	Høy
FK-02	Sauebonden skal kunne klippe ut en bit av norgeskartet og laste det ned på applikasjonen	Høy
FK-03	Sauebonden skal kunne se sin egen posisjon på kartet	Høy
FK-04	Sauebonden skal kunne merke av på kartet hvor han/hun har sett saueflokken	Middels
FK-05	Sauebonden skal kunne se streker på kartet som viser hvor han/hun har gått	Høy
FK-06	Sauebonden skal kunne registrere antall voksne og lam han/hun observerer	Høy
FK-07	Sauebonden skal kunne registrere antall lam han/hun observerer	Høy
FK-08	Sauebonden skal kunne registrere antall sau i ulike farger han/hun observerer	Høy
FK-09	Sauebonden skal kunne registrere antall sau etter fargen på slipset sauene han/hun observerer	Høy
FK-10	Sauebonden skal kunne registrere hvilke farger sauene har på øremerkene sine	Middels
FK-11	Sauebonden skal kunne zoome inn på den delen av kartet han/hun vil klippe ut	Middels
FK-12	Sauebonden skal kunne se hvilke områder han/hun har lastet ned fra før av	Høy
FK-13	Sauebonden skal kunne velge blant områder som har blitt lastet ned fra før av	Høy
FK-14	Sauebonden skal kunne velge dato og tidspunkt for tilsynsturen	Middels
FK-15	Sauebonden skal kunne se sine tidligere tilsynsturer	Middels
FK-16	Sauebonden skal kunne logge seg inn på applikasjonen	Høy
FK-17	Sauebonden skal kunne registrere forskjellige detaljer som han/hun observerer basert på avstanden til saueflokken	Høy
FK-18	Sauebonden skal kunne registrere antall skadde sauer han/hun observerer	Høy
FK-19	Sauebonden skal kunne registrere antall døde sauer han/hun observerer	Høy
FK-20	Sauebonden skal kunne ta bilde av skadde sauer og døde sauer han/hun observerer	Lav
FK-21	Sauebonden skal kunne se en oversikt over hva som har blitt registrert underveis på tilsynsturen	Middels
FK-22	Sauebonden skal kunne avslutte tilsynsturen når han/hun er ferdig	Middels
FK-23	Sauebonden skal kunne opprette en bruker på applikasjonen	Høy

Tabell 3.1: Funksjonelle krav for mobilapplikasjon, hentet fra fordypningsprosjektet høsten 2021

FK-01: Sauebonden skal kunne se et kart over hele Norge
En del av essensen til mobilapplikasjonen er å kunne se et kart over hele Norge for å kunne velge ut et område for tilsynsturen.

FK-02: Sauebonden skal kunne klippe ut en bit av norgeskartet og laste det ned på applikasjonen
En stor del av motivasjonen for applikasjonen er å kunne markere det området tilsynsturen skal gå i og laste det ned på applikasjonen for å bruke det på turen.

FK05: Sauebonden skal kunne se streker på kartet som viser hvor han/hun har gått
Det er sterkt ønske om å loggføre hvor sauene har befunnet seg under tilsynsturen og hvor sauebonden har gått. Dette ønsket kan visualiseres ved at det er streker på kartet som tilsier hvor han/hun har gått.

FK-06 / FK-07 / FK-08 / FK-09 / FK-10 / FK-18 / FK-19
Informasjonen som registreres på tilsynsturene er de nevnte kravene. Av den grunn er det essensielt at mobilapplikasjonen innehar funksjonalitet for å registrere disse detaljene.

FK-12: Sauebonden skal kunne se hvilke områder han/hun har lastet ned fra før av / FK-13: Sauebonden skal kunne velge blant områder som har blitt lastet ned fra før av
Det skal være mulig for sauebonden å laste ned kart over flere områder, og se en liste over disse områdene når han/hun skal velge ut ett kart blant disse for å bruke på tilsynsturen. Det vil si at hvert element i den listen som blir vist skal være knyttet til hvert sitt unike kart.

FK-16: Sauebonden skal kunne logge seg inn på applikasjonen / FK-23: Sauebonden skal kunne opprette en bruker på applikasjonen
Ettersom alt av informasjon fra tilsynsturer skal være lagret på en spesifikk bruker og kun den brukeren skal ha tilgang til denne informasjonen, burde det være funksjonalitet for at sauebonden skal kunne både opprette en bruker og logge seg inn på applikasjonen.

3.1.2 Ikke-funksjonelle krav

De ikke-funksjonelle kravene for mobilapplikasjonen blir vist på Tabell 3.2

NFK-ID	Beskrivelse	Kategori	Prioritet
NFK-01	Mobilapplikasjonen skal kunne fungere uten mobildekning	Tilgjengelighet	Høy
NFK-02	Mobilapplikasjonen skal kunne fungere på både *iOS* og *Android*	Brukervennlighet	Middels
NFK-03	Mobilapplikasjonen skal være intuitiv nok til at det tar mindre enn 10 minutter på å lære seg hvordan man bruker det	Brukervennlighet	Høy
NFK-04	Registrering av observasjoner skal ikke ta mer enn 3 minutter	Brukervennlighet	Middels
NFK-05	Registrerte data skal synkroniseres automatisk til databasen etter hver tilsynstur med en gang applikasjonen er koblet til internett igjen	Pålitelighet	Høy

Tabell 3.2: Ikke-funksjonelle krav for mobilapplikasjon, hentet fra fordypningsprosjektet høsten 2021

NFK-01: Mobilapplikasjonen skal kunne fungere uten mobildekning
Det er viktig at mobilapplikasjonen fungerer uten mobildekning ettersom det ikke nødvendigvis er mobildekning ute på beiteområdet.

NFK-03: Mobilapplikasjonen skal være intuitiv nok til at det tar mindre enn 10 minutter på å lære seg hvordan man bruker det
Mobilapplikasjonen skal være enkel og intuitiv å bruke for at det skal virke mer appellerende å bruke enn penn og papir. Dersom det tar for lang tid å lære seg hvordan man bruker applikasjonen mister den attraksjonen sin.

NFK-05: Registrerte data skal synkroniseres automatisk til databasen etter hver tilsynstur med en gang applikasjonen er koblet til internett igjen
For å ikke miste informasjonen som har blitt registrert på tilsynsturen uten internett er det viktig at data synkroniseres automatisk til databasen etter hver tur når applikasjonen er tilkoblet internett igjen.

3.1.3 Andre krav

Kravene i dette delkapittelet inngår i **FK-17**. En ting Professor *Hvasshovd* la vekt på var at forskjellige ting skulle registreres basert på avstanden fra gjeter til saueflokken. Dersom avstanden er **mindre enn 30m** til saueflokken, skal følgende ting registreres:

- Antall sau og lam
- Antall lam
- Antall brune, svarte og hvite/gråe sauer
- Farge på slips (antall lam en sau har)
- Eierskap (farge på øremerket)
- Antall skadde sauer
- Antall døde sauer

Dersom sauebonden er **mer enn 30m** unna saueflokken, skal disse detaljene registreres:

- Antall sau og lam
- Antall lam
- Antall brune, svarte og hvite/gråe sauer

3.2 Kravspesifikasjon for nettside

De funksjonelle og ikke-funksjonelle kravene til nettsiden er vist på Tabell 3.3 og på Tabell 3.4. Nettsidens hovedfunksjon er å komplimentere mobilapplikasjonen. Nettsiden henter ut innsamlet data som har blitt registrert på tilsynsturene fra mobilapplikasjonen og gir sauebonden en oversikt over denne informasjonen. På denne måten kan sauebonden se en eventuell utvikling eller nedgang i sauebestanden for sesongen. I tillegg skal nettsiden bruke data fra mobilapplikasjonen til å generere en rapport som skal sendes inn til myndighetene ved sesongens slutt.

3.2.1 Funksjonelle krav

De funksjonelle kravene for nettsiden blir vist på Tabell 3.3.

FK-ID	Beskrivelse	Prioritet
FK-01	Sauebonden skal kunne logge seg inn på nettsiden for å få tilgang	Høy
FK-02	Sauebonden skal kunne opprette en ny tilsynsgruppe	Høy
FK-03	Sauebonden skal kunne invitere andre brukere til sin tilsynsgruppe	Høy
FK-04	Sauebonden skal kunne bli med i tilsynsgrupper opprettet av andre brukere	Høy
FK-05	Sauebonden skal kunne se rapporter fra sine egne tilsynsturer	Høy
FK-06	Sauebonden skal kunne se rapporter fra brukere i samme tilsynsgruppe	Middels
FK-07	Sauebonden skal kunne se informasjon fra tilsynsturene visualisert på et kart	Høy
FK-08	Sauebonden skal kunne filtrere og sortere på informasjonen fra tilsynsrapporter som vises basert på observasjonsdetaljer	Lav
FK-09	Sauebonden skal kunne se ferdiggenererte rapporter med data fra sine egne tilsynsturer	Høy
FK-10	Sauebonden skal kunne velge mellom forskjellige visualiseringer på kartet	Lav
FK-11	Sauebonden skal kunne se en oppsummering av hver tilsynstur gjennomført i sesongen i den ferdiggenererte rapporten	Høy
FK-12	Sauebonden skal kunne se registrert, detaljert informasjon om hver observasjon fra tilsynsturer i den ferdiggenererte rapporten	Høy
FK-13	Sauebonden skal kunne laste ned den ferdiggenererte rapporten som en PDF-fil	Høy

Tabell 3.3: Funksjonelle krav for nettside

FK-01: Sauebonden skal kunne logge seg inn på nettsiden for å få tilgang
Ettersom nettsiden skal vise informasjon fra tilsynsturer til en spesifikk bruker burde det være innlogging for å bevare personvern.

FK-02 / FK-03 / FK-04
Det kan være flere gjetere som gjennomfører tilsyn av sau i samme område. Derfor burde det være funksjonalitet for å kunne opprette en tilsynsgruppe, legge til andre brukere og bli med i en gruppe selv.

FK-05: Sauebonden skal kunne se rapporter fra sine egne tilsynsturer
En av hovedgrunnene til å endre på eksisterende innsamlingsmetode er at informasjonen fra tilsynsturer skal ha en fastsatt struktur på rapportene. Dermed bør det være mulig for sauebonden å se disse rapportene på nettsiden for å få en oversikt av gjennomførte tilsynsturer.

FK-09 / FK-11 / FK-12 / FK-13
En hovedfunksjon nettsiden har er å generere ferdigrapporter som inneholder informasjon fra tilsynsturer gjennomført i løpet av beitesesongen slik at sauebonden slipper å skrive disse rapportene på egenhånd. Deretter skal det kun være nødvendig for sauebonden å laste ned disse rapportene som PDF og sende det til myndighetene.

3.2.2 Ikke-funksjonelle krav

De ikke-funksjonelle kravene blir vist på Tabell 3.4

NFK-ID	Beskrivelse	Kategori	Prioritet
NFK-01	Nettsiden skal ha en gjennomsnittlig responstid på 3 sekunder	Ytelse	Høy
NFK-02	Nettsiden skal en responstid på maksimalt 5 sekunder	Ytelse	Middels
NFK-03	Nettsiden skal være intuitiv nok til at det tar mindre enn 5 minutter på å lære seg hvordan det fungerer	Brukervennlighet	Høy
NFK-04	Brukere av nettsiden skal autentiseres for å få tilgang	Sikkerhet	Høy

Tabell 3.4: Ikke-funksjonelle krav for nettside

NFK-01: Nettsiden skal ha en gjennomsnittlig responstid på 3 sekunder
Nettsiden burde ikke bruke for lang tid på å respondere på brukerinteraksjonen. Da vil brukeropplevelsen synke.

NFK-03: Nettsiden skal være intuitiv nok til at det tar mindre enn 5 minutter på å lære seg hvordan det fungerer
Dersom det tar for lang tid å finne ut av hvordan nettsiden fungerer blir det tungvint for brukere å interagere med den.

NFK-04: Brukere av nettsiden skal autentiseres for å få tilgang
For å bevare personvern og sørge for at ikke hvilken som helst bruker har tilgang til nettsiden bør alle brukere autentiseres ved innlogging.

Kapittel 4

Arkitektur

I dette kapittelet blir programvarearkitekturen til mobilapplikasjonen *Bæædar* og nettsiden presentert. Det blir først gitt en oversikt over hvordan manuell oppfølging av sau på utmarksbeite foregår **nå (AS-IS)**, og hvordan det kommer til **å bli** med mobilapplikasjonen *Bæædar* **(TO-BE)**. Deretter blir den overordnede arkitekturen til mobilapplikasjonen og den planlagte arkitekturen til nettsiden presentert. I tillegg blir det gitt en oversikt over strukturen i databasen, systemets datastruktur, komponentstruktur og mappestruktur.

4.1 AS-IS

Denne delen er hentet fra fordypningsprosjektet høsten 2021. Sånn som det er **nå**, fungerer prosessen med tilsynstur på følgende måte:

- Sauebonden bestemmer seg for et område han/hun skal dekke på tilsynsturen på et fysisk kart
- Sauebonden noterer ned observasjonene sine underveis på turen med penn og papir
- Sauebonden avslutter turen og tar vare på papiret med informasjonen fra tilsynsturen
- Ved sesongens slutt skriver sauebonden en rapport med all informasjonen fra tilsynsturene og sender det inn til myndighetene

Det er verken struktur på informasjonen som blir notert i løpet av turen eller en oversikt av hvilken informasjon som har blitt registrert og hvor det har blitt gått på turen. I tillegg må disse notatene bli tatt vare på og renskrives i slutten av beitesesongen til rapporten som skal bli sendt til myndighetene. Av den grunn er det et behov for å erstatte den eksisterende innsamlingsmetoden med et system som kan forbedre prosessen med tilsyn av sau.

4.2 TO-BE

Denne delen er hentet fra fordypningsprosjektet høsten 2021. Hvordan prosessen med tilsynstur skal **bli**, er:

- Sauebonden åpner mobilapplikasjonen og laster ned det ønskede området for tilsynsturen utfra norgeskartet
- Sauebonden starter en tilsynstur med det valgte kartet

- Sauebonden ser sin egen nåværende posisjon og markerer på kartet hvor saueflokken har blitt sett

- Sauebonden velger en av funksjonalitetene basert på avstanden til saueflokken:

  - mer enn 30 m
  - under 30 m

  Informasjonen som skal bli registrert kommer i en bestemt rekkefølge og bonden må følge flyten

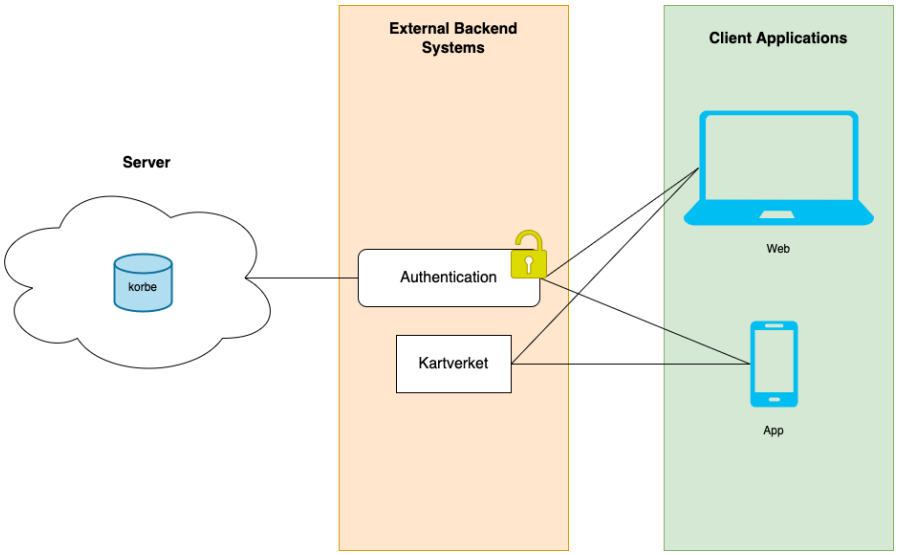
- Sauebonden avslutter turen og observasjonene blir lagret i databasen.

- Nettsiden henter informasjon om tilsynsturene som har blitt gjennomført og genererer automatisk en ferdigrapport som skal bli sendt inn til myndighetene.

4.3 Overordnet arkitektur

Bædar skal bestå av tre distinkte systemer: mobilapplikajsonen, nettsiden og serveren. Systemet har en *klient-server*-arkitektur, hvor mobilapplikasjonen og nettsiden kommuniserer med en felles server. Grunnen til at denne arkitekturen ble valgt er at nettsiden skal inneholde data fra mobilapplikasjonen. Hver bruker av mobilapplikasjonen skal kunne logge seg inn på nettsiden og se den informasjonen som har blitt registrert på mobilapplikasjonen. Ved å ha en felles server vil det da bli enklere å hente ut denne dataen. I tillegg hadde Professor *Hvasshovd* tenkt at serveren skulle være skybasert, slik at sauebøndene kunne ha enkel tilgang til relevant data og registrere informasjon fra hver sin enhet. *Cloud Firestore* ble dermed valgt, en skybasert database av *Firestore* som håndterer alt av lagring og synkronisering av data [20]. Mobilapplikasjonen og nettsiden kan kommunisere direkte med databasen og sende en forespørsel, og databasen svarer på forespørselen. *Cloud Firestore* har også *offline*-støtte, slik at data som blir registrert når brukeren er frakoblet internett blir *cachet*. Når brukeren da blir tilkoblet internett igjen synkroniserer *Firestore* denne dataen automatisk til databasen. Denne funksjonaliteten ved *Cloud Firestore* oppfyller **NFK-01** og **NFK-05** som står beskrevet i Kapittel 3 på Tabell 3.2. *Klient-server*-arkitekturen til systemet blir vist på Figur 4.1, med tredjepartssystemer som i dette tilfellet er *Firebase Authentication* og *Kartverket* sitt API. *Firebase Authentication* er et tredjepartssystem som *Firebase* tilbyr for å håndtere autentisering og innlogging av brukere til applikasjonen. *Kartverket* sitt API er et åpent API som kan brukes for å vise frem norgeskartet, og blir brukt sammen med *OpenLayers* i denne applikasjonen. Mer om *Firebase Autentication* og hvordan *Kartverket* sitt API og *OpenLayers* fungerer sammen kan leses i Kapittel 5.2.2.

Fordeler med en *klient-server*-arkitektur er blant annet at all data samles og lagres på samme sted. Ettersom den tilknyttede nettsiden skal hente ut informasjon og detaljer fra tilsynsturer registrert på mobilapplikasjonen senere, forenkler en slik arkitektur dataflyten og datatilgang. Nye klienter kan også bli lagt til underveis uten problemer ettersom etterspørselen øker [33]. På denne måten er denne modellen lett å utvide og skalerer dermed også godt. Flere klienter kan dele på de samme ressursene gitt av serveren. Med *Cloud Firestore* kan flere klienter kommunisere og serveren håndterer lagring og lesing av data. Det gjør også utvikling og endring av brukergrensesnitt enklere når klient og server er separert og uavhengig av hverandre. I tillegg blir informasjon som registreres på tilsynstur uten internett lagret lokalt og synkronisert automatisk med en gang mobilapplikasjonen er koblet til internett igjen.



Figur 4.1: Arkitektur for *Bæædar*

Ulemper med å bruke en *klient-server*-arkitektur, spesielt med *Cloud Firestore*, er at tjenestene koster penger [20]. Med *Cloud Firestore* har man et begrenset antall gratis lesninger av og skrivinger til databasen før man må betale for utvidet tilgang. I tillegg er det et begrenset antall brukere man kan autentisere. Dersom systemet skal benyttes av et ubegrenset antall brukere, er det en stor mulighet for at det kan bli dyrt etter hvert som tilgangen må utvides.

En annen ulempe med denne modellen er at det er nødvendig med en del vedlikehold for å holde serveren oppe og organisert [45]. Siden alle klientene er knyttet til en felles server og henter data fra samme sted, kan det bli kritisk dersom serveren skulle krasje. En slik hendelse kan blokkere flere klienter fra å hente og skrive data, og potensielt være til hinder for sauebønder som er avhengig av serveren for å kunne avgi en rapport til myndighetene. Dette er en såkalt *single point of failure*, som vil si at dersom serveren går ned vil hele systemet stoppe opp [45].

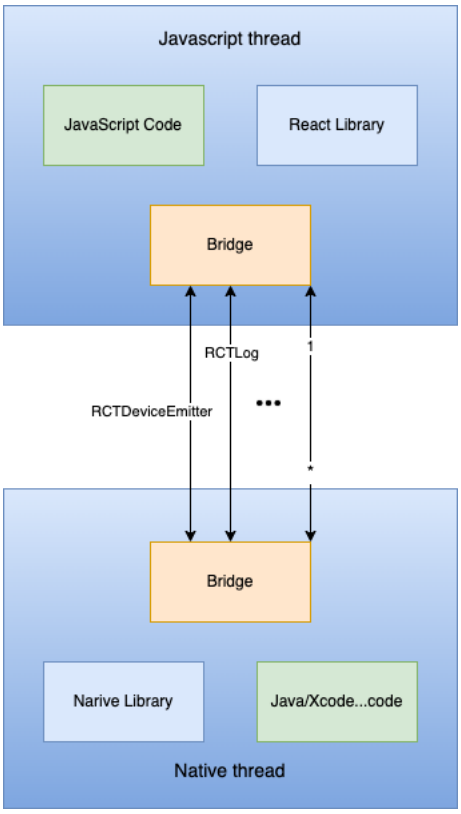
4.3.1 Overordnet arkitektur mobilapplikasjon

Mobilapplikasjonen til *Bæædar* er programmert i *React Native* med *TypeScript*.

React Native

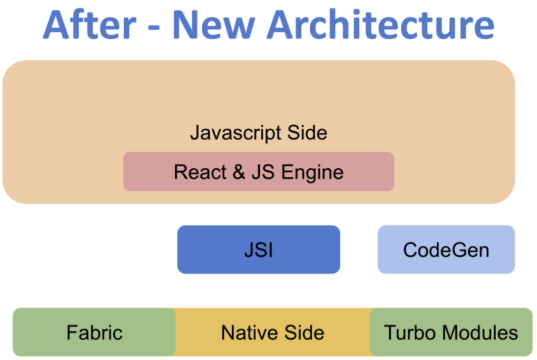
React Native er et *cross-platform*-rammeverk som gjør det mulig å utvikle en webapplikasjon som fungerer på både *Android*- og *iOS*-enheter, ved bruk av kun én kodebase. Dette *UI*-rammeverket innehar *React* sitt konsept om gjenbrukbare komponenter, samt *Native*-moduler som forbedrer ytelsen. Mer informasjon om *React Native* kan leses i Kapittel 5.2.2.

Nylig har *React Native* fått et nytt *rendering*-system som tar i bruk *Fabric*-arkitektur [76]. *Fabric* bygger om *rendering*-laget til *React Native*. Tidligere har alle *UI*-operasjoner blitt håndtert av en slags bro, som vist på Figur 4.2. Den tidligere arkitekturen bestod av *JavaScript*-tråden og *Native*-tråden som var satt sammen av broen. Trådene var ikke "bevisste" på hverandre og trengte denne broen for kommunikasjon. Den nye implementasjonen av *Fabric* derimot, gjør det mulig for *UI-manager* å opprette et *shadow*-tre direkte i *C++*, noe som gjør prosessen smidigere ved å redusere antall steg som må kjøres [69].



Figur 4.2: Tidligere arkitektur for *React Native*

Med *Fabric* vil den nye arkitekturen til *React Native* se slik ut som vist på Figur 4.3.



Figur 4.3: Ny arkitektur for *React Native* hentet fra [69]

1. JSI:
Erstatter broen fra tidligere arkitektur. *JSI* gir API til *JS Runtime*-motor og gjør *JavaScript* direkte bevisst på *Native*-funksjoner og objekter. *JSI* synkroniserer kall fra *JavaScript*-tråden til *Native*-tråden, *renderer* raskt ved å bruke et direkte kall til *UI Main*-tråden og deler data mellom trådene.

2. Fabric:
Erstatter *UI-Manager* som er asnvarlig for *Native*-siden. Den største forskjellen er at istedenfor å kommunisere med *JavaScript*-siden med broen vil den utnytte *Native*-funksjonen ved bruk av *JSI* slik at *JavaScript*-siden kan kommunisere direkte gjennom *ref*-funksjoner. Dette resulterer i bedre og mer effektiv ytelse.

3. Turbo-moduler: *Turbo*-moduler er det samme som *native*-moduler i den tidligere arkitekturen, bortsett fra at det blir implementert og oppfører seg annerledes. For det første er *Turbo*-moduler *lazy-loaded*, som betyr at disse modulene kun blir lastet inn når applikasjonen trenger dem istedenfor å laste inn alle sammen på lanseringstid.

4. CodeGen: Blir brukt til å lage statiske typer av *JavaScript*-kode slik at *Fabric* og *Turbo*-modulene blir bevisste på disse typene og unngår å validere data hver gang det kjøres. Det resulterer i mindre tidsforbruk, bedre ytelse og færre feil ved sending av data.

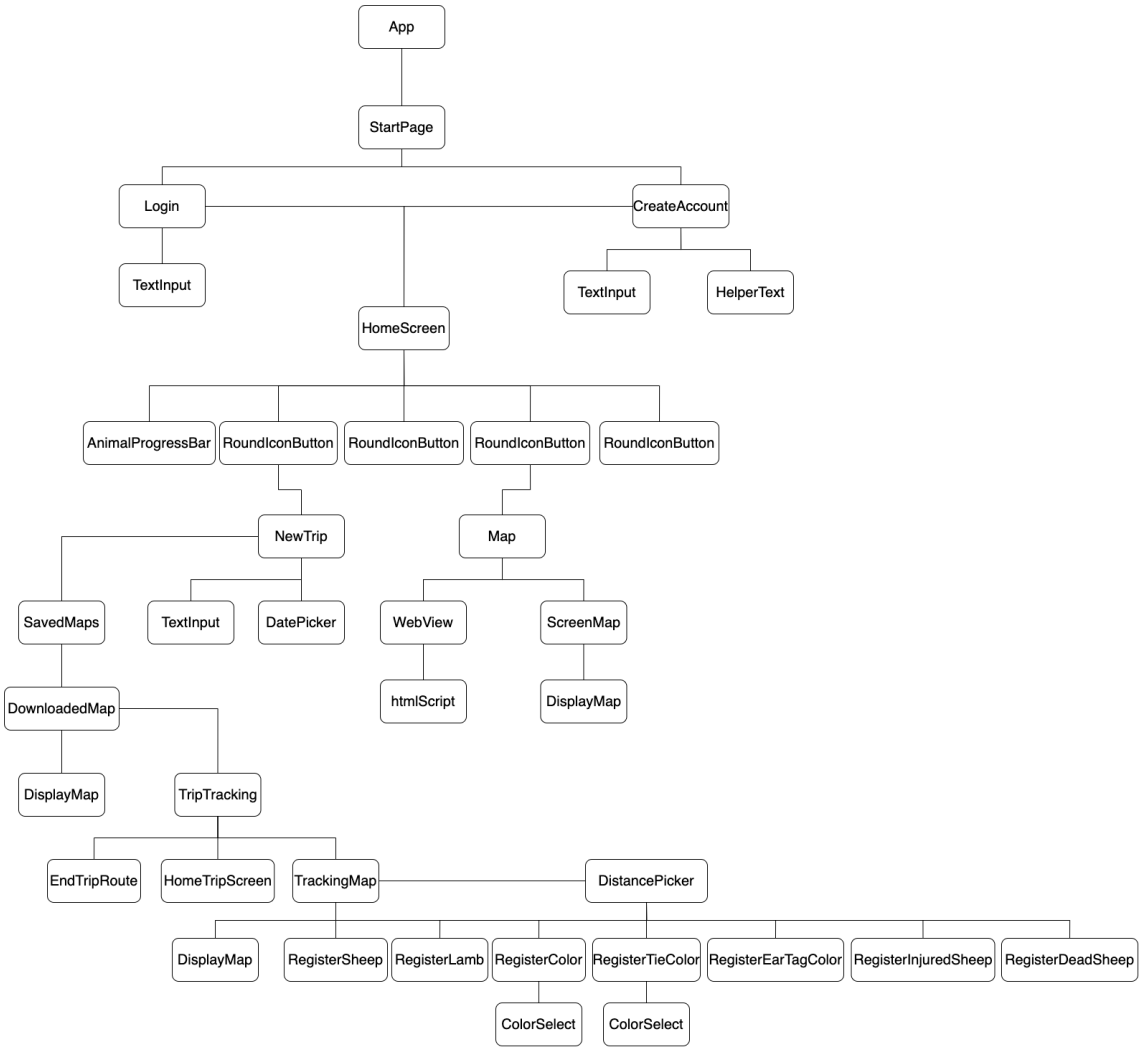
Mer informasjon om den nye arkitekturen til *React Native* finnes her [76] og her [69].

TypeScript

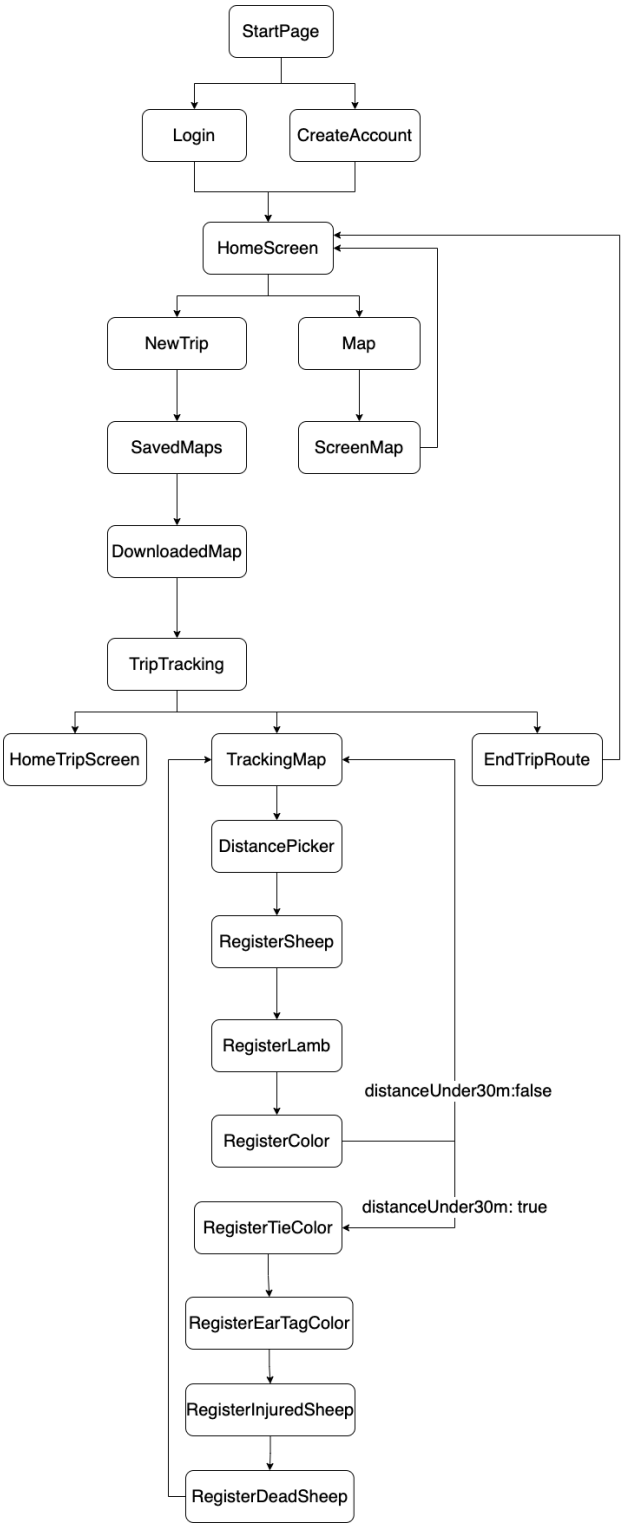
TypeScript er et *open-source* programmeringsspråk som er bygget på *JavaScript*. Det er et såkalt *superset* av *JavaScript* og utvider språket ved å benytte seg av statiske typedefinisjoner [55]. Mer informasjon om *TypeScript* kan leses i Kapittel 5.2.2.

Komponentstruktur

Brukergrensesnittet til mobilapplikasjonen *Bæædar* er sammensatt av flere komponenter. På Figur 4.4 blir det vist en oversikt over hvordan komponentene er satt sammen, mens på Figur 4.5 blir flyten på tvers av komponentene visualisert.



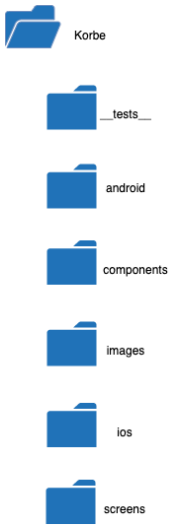
Figur 4.4: Overordnet komponenttre av mobilapplikasjonen *Bæædar*



Figur 4.5: Oversikt av komponentflyt for mobilapplikasjonen *Bæædar*

Mappestruktur

På Figur 4.6 blir mappestrukturen til mobilapplikasjonen visualisert.



Figur 4.6: Oversikt av mappestrukturen til mobilapplikasjonen *Bæædar*

Korbe er den overordnede mappen til hele applikasjonen.

tests er mappen til testing av applikasjonen.

android er mappen med oppsettet for å kjøre applikasjonen på *Android*-enheter.

components inneholder de generelle komponentene som kan bli gjenbrukt og tilpasset etter ønske.

images inneholder bilder som har blitt benyttet i applikasjonen.

ios inneholder oppsettet for å kjøre applikasjonen på *iOS*-enheter.

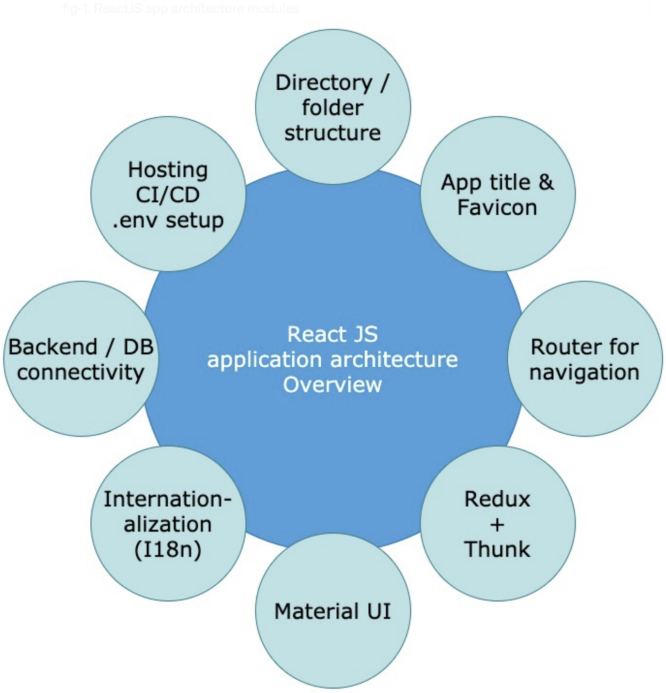
screens inneholder filene til skjermene som blir vist i applikasjonen.

4.3.2 Overordnet arkitektur nettside

Med begrenset tid og en leveringsfrist å overholde, ble arbeidsmengden for stor for å få muligheten til å implementere den tilhørende nettsiden til applikasjonen *Bæædar*. Dette delkapittelet vil derfor omhandle den planlagte arkitekturen for nettsiden, ved bruk av *JavaScript*-biblioteket *React* sammen med *TypeScript*. *React* kan være det logiske valget ettersom mobilapplikasjonen er skrevet i *React Native*. *React* er også hyppig brukt av utviklere verden rundt og såkalt *corporate standard*.

React

React er et *JavaScript*-bibliotek som brukes for å lage brukergrensesnitt for *single-page*-applikasjoner. *Single-page*-applikasjoner er webapplikasjoner som laster inn én *HTML*-side og dynamisk oppdaterer siden etter hvordan brukeren interagerer med applikasjonen [93]. Arkitekturen til *React*-applikasjonen til systemet er tenkt at det kan se slik ut som vist i Figur 4.7.



Figur 4.7: Standard *ReactJS*-applikasjonsarkitektur hentet fra [74]

Mer om *ReactJS* kan leses om her [77].

4.3.3 Datastruktur

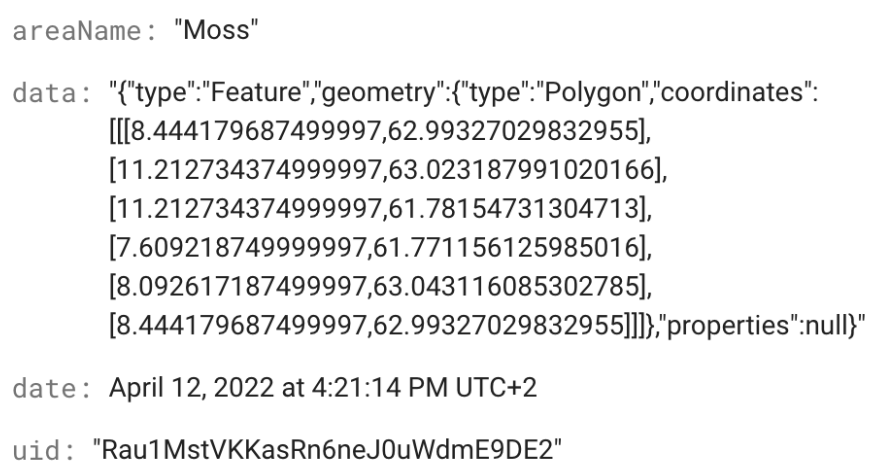
Når brukeren markerer et spesifikt område på kartet i mobilapplikasjonen, blir dette området lagret som et *GeoJSON*-objekt først, før det blir konvertert til en *JSON*-string i databasen. Dette blir lagret sammen med id-en til brukeren som opprettet dette området, datoen og tidspunktet på når dette ble gjort. Datastrukturen blir lagret med brukerid for at det skal være enkelt å hente ut det markerte kartet til en spesifikk bruker senere. Med dato og tidspunkt for når dette kartet ble lagd gjør det mulig å sortere kartene etter seneste dato først. Som beskrevet i Kapittel 5.2.2, blir området lagret som et *GeoJSON*-objekt slik at det enkelt kan konverteres til en *JSON*-string og lagt til i databasen ettersom *Firestore* støtter *JSON*-data. Et utsnitt av datastrukturen blir vist på Figur 4.8.

```
data: "{"type":"Feature","geometry":{"type":"Polygon","coordinates":
       [[[13.273357543945307,65.4374169924667],
       [13.521236572265622,65.42913822686165],
       [13.514370117187495,65.32901550393586],
       [13.270610961914059,65.33245460639117],
       [13.273357543945307,65.4374169924667]]]},"properties":null}"

date: February 17, 2022 at 1:45:24 PM UTC+1

uid: "Rau1MstVKKasRn6neJ0uWdmE9DE2"
```

Figur 4.8: Datastruktur for markert område på kart

Som Figur 4.8 viser, blir koordinatene til det markerte området også lagret. I tillegg er det mulig

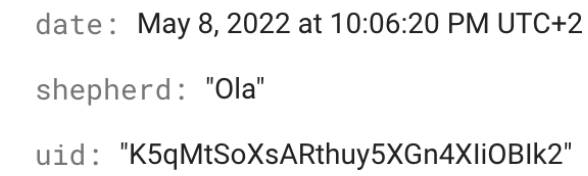
å navngi dette området dersom brukeren ønsker det, sånn at når brukeren skal velge et kart fra listen over nedlastede kart til tilsynstur senere, så står det navn på området i stedet for dato for opprettelsen. For eksempel sånn som på Figur 4.9.



Figur 4.9: Datastruktur for kart med områdenavn

Ved å lagre kartdata på denne måten kan man enkelt hente ut *GeoJSON*-objektet senere for å vise frem kartet på nytt. Mer informasjon om hvordan dette blir gjort finnes i Kapittel 5.2.2.

I databasen finnes det en kolleksjon med tilsynsturer, *trips*. Hver gang det blir opprettet en ny tur, blir det lagd et nytt dokument i kolleksjonen med valgt dato og tidspunkt for turen, navn på gjeter, valgt kart for turen og tilknyttet brukerid. Dette blir vist på Figur 4.10. Foreløpig har ikke kartid blitt tilknyttet tilsynsturen i databasen ennå. Deretter er det tenkt at hvert av disse dokumentene har en kolleksjon med observasjoner, *observations*, fra forskjellige områder på valgt kart. Hvert dokument i denne kolleksjonen skal inneholde registrerte observasjoner på forskjellige steder i løpet av tilsynsturen.



Figur 4.10: Datastruktur for en nyopprettet tilsynstur

4.4 Database

Databasen inneholder følgende tabeller som vist på Figur 4.11:

Figur 4.11: Databasediagram av mobilapplikasjonen *Bæædar*

Ettersom mobilapplikasjonens database er *Cloud Firestore*, gir ikke Figur 4.11 et korrekt bilde av strukturen til *Firestore*. *Cloud Firestore* er en *NoSQL*-database som er dokumentorientert [22], og Figur 4.11 er bare ment for å gi en oversikt av hva databasen inneholder. Det er tenkt at det skal være en tabell med observasjoner fra tilsynsturen, *observation*, som er informasjonen som registreres. Hver tabell med observasjoner representerer informasjon registrert på spesifikke steder i løpet av en tur. Hver tilsynstur, *trip*, kan være tilknyttet flere tabeller med observasjoner og det er *én-til-mange*-relasjon. Videre, er det *én-til-mange*-relasjon mellom *maps* og *trip* for et kart kan bli brukt på flere tilsynsturer, men en tilsyntur kan bare være knyttet til ett kart. Til slutt, har en bruker *én-til-mange*-relasjon til turer og kart. Hver bruker kan ha mange tilsynsturer og markerte kart, men hver tur og hvert kart er unike og kan kun være knyttet til én bruker.

4.4.1 Documents

I *Cloud Firestore* blir data lagret i form av et dokument. Et dokument inneholder felter som har angitte verdier [22]. Hvert dokument blir identifisert med en tilgitt unik brukerid som er generert av *Firestore*, som vist på Figur 4.12.

Figur 4.12: Et dokument som representerer en bruker i *Bæædar*

Hvert dokument i *Firestore* inneholder et sett av nøkkel-verdi-par. Alle dokumenter tilhører en kolleksjon. Dokumenter kan inneholde delkolleksjoner og nøstede objekter, som kan både være primitive datafelt som *strings* eller mer komplekse objekter som lister [22]. Mer komplekse, nøstede objekter i et dokument blir kalt for *maps*. For eksempel, kan det struktures slik som på Figur 4.13:

Figur 4.13: Et dokument som representerer en opprettet tilsynstur med nøstede objekter

Figur 4.13 inneholder følgende:

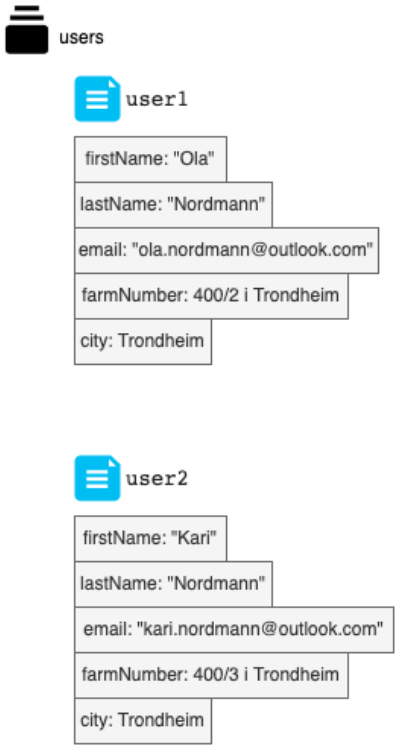
- **trips**: En kolleksjon med opprettede tilsynsturer.
- **trip1**: Et dokument i **trips** som representerer én tilsynstur. Hver tur har:
  - **uid**: Tilknyttet brukerid til den brukeren som opprettet turen.
  - **shepherd**: Navnet til gjeteren som skal gjennomføre denne turen.
  - **createdAt**: Valgt dato og tidspunkt for når tilsynsturen skal gjennomføres.
  - **mapId**: Id til kartet som skal brukes på turen.
- **observations**: En delkolleksjon (*subcollection*) av **trip1** som inneholder alle registrerte observasjoner på en tilsynstur. Hvert dokument i **trips** får en tilknyttet kolleksjon med **observations**.
- **obs1**: Et dokument i **observations** som representerer informasjon som er observert på ett spesifikt sted på tilsynsturen. Informasjonen som skal registreres er basert på kravene i Delkapittel 3.1.3, og er:
  1. **areaCoordinates**: *JSON*-string for det spesifikke området der det blir observert.
  2. **numberOfSheep**: Antall sauer og lam som blir observert.
  3. **numberOfLambs**: Antall lam som blir observert.

4. **numberOfBrownSheep**: Antall brune sauer som blir observert.
5. **numberOfWhiteSheep**: Antall hvite sauer som blir observert.
6. **numberOfBlackSheep**: Antall svarte sauer som blir observert.
7. **numberOfRedTies**: Antall sauer med rødt slips som blir observert.
8. **numberOfBlueTies**: Antall sauer med blått slips som blir observert.
9. **numberOfYellowTies**: Antall sauer med gult slips som blir observert.
10. **numberOfGreenTies**: Antall sauer med grønt slips som blir observert.
11. **earTagColors**: Farger på øremerker til sauene som blir observert. Dette blir lagret som en liste i databasen.
12. **numberOfInjuredSheep**: Antall skadede sauer som blir observert.
13. **numberOfDeadSheep**: Antall døde sauer som blir observert.
14. **distanceUnder30m**: Avstanden fra gjeter til saueflokken. Dersom avstanden er under 30 meter blir den satt til **true**. Hvis avstanden er mer enn 30 meter blir det satt til **false**.

Ettersom hver tilsynstur kan inneholde flere observasjoner fra forskjellige steder under turen er det derfor valgt at hvert dokument i *trips*-kolleksjon har en delkolleksjon med *observations*, hvor hvert dokument i *observations* representerer informasjon fra ett spesifikt sted.

4.4.2 Collections

Hvert dokument i databasen tilhører en større kolleksjon av dokumenter. Kolleksjoner fungerer som *containere* for dokumenter. Det er for eksempel mulig å ha en kolleksjon med brukere som inneholder data til forskjellige brukere hvor hver bruker blir representert som et dokument med en unik id [22]. Et eksempel på dette kan ses på Figure 4.14.



Figur 4.14: En kolleksjon av brukere i *Bæædar*

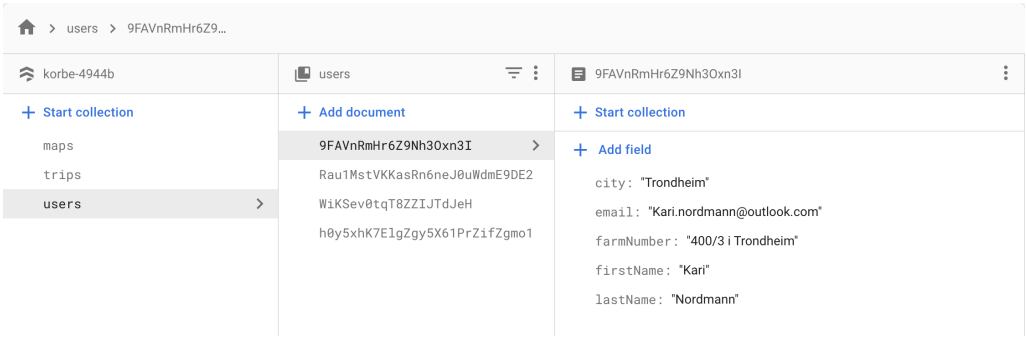
Ettersom *Cloud Firestore* er *schemaless*, har man frihet til å bestemme hva slags felter og datatyper hvert dokument skal inneholde [22]. Dokumenter som tilhører samme kolleksjon kan inneholde felter med forskjellige datatyper. Det er ingen krav om at dokumenter i samme kolleksjon må inneholde lik type data. Likevel, er det vanlig at dokumenter i samme kolleksjon er strukturert på samme måte slik at det blir lettere å hente ut data senere.

Hvert dokument i en kolleksjon får en unik id. Man kan enten lage id-en selv eller la *Firestore* generere tilfeldige id-er til dokumentene. Det er ikke nødvendig å opprette eller slette kolleksjoner. Dersom man sletter alle dokumenter i en kolleksjon, eksisterer ikke den lenger. Med andre ord, en kolleksjon finnes kun hvis den inneholder minst ett dokument [22].

4.4.3 Datastruktur i Cloud Firestore

Cloud Firestore har en hierarkisk datastruktur, det vil si at det er mulig å ha en delkolleksjon som tilhører et spesifikt dokument i en kolleksjon [22]. Dette mønsteret er det ikke mulig å endre på. Data i *Firestore* må alltid følge denne strukturen: kolleksjon - dokument - delkolleksjon - dokument - ... - dokument. Det er for eksempel ikke mulig å ha en kolleksjon i en kolleksjon, eller et dokument i et dokument.

Delkolleksjoner gjør det mulig å strukturere dataen hierarkisk, noe som kan forenkle prosessen med å hente ut spesifikk data fra databasen. Ettersom dokumenter i en delkolleksjon kan inneholde sine egne delkolleksjoner skaper det en nøsting av data. I *Cloud Firestore* kan man nøste data opptil 100 nivåer". Som vist på Figur 4.13, ligner dokumentene mye på *JSON*, og er praktisk talt *JSON*-data med noen små forskjeller. Derfor kan dokumentene i *Firestore* generelt sett bli behandlet som *JSON*-data. I dette tilfellet ser for eksempel kolleksjonen av brukere slik ut som på Figur 4.15:



Figur 4.15: Datastruktur, brukere

Hver bruker får generert en unik id som kan aksesseres med **(...).uid** når data til en spesifikk bruker skal hentes.

Kapittel 5

Implementasjon

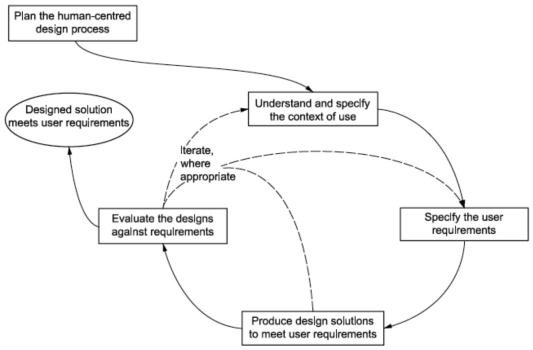
I dette kapittelet blir utformingen av designet og utviklingen av mobilapplikasjonen presentert.

5.1 Design

Store deler av dette kapittelet er hentet fra fordypningsprosjektet høsten 2021. Designet til mobilapplikasjonen ble utformet høsten 2021. I dette kapittelet blir designprosessen og *Figma*-prototypen presentert.

5.1.1 Brukersentrert designprosess

Utformingen av designet på mobilapplikasjonen var en iterativ prosess. På ukentlige veiledningsmøter med Professor *Hvasshovd* ble designet vurdert og testet opp mot de satte kravene for mobilapplikasjonen. Deretter ble de nødvendige endringene gjort til neste iterasjon. Den iterative designprosessen foregikk helt fram til Professor *Hvasshovd* godkjente den endelige løsningen for mobilapplikasjonen. Ettersom Professor *Hvasshovd* i denne situasjonen har fungert som en *kunde*, ble det en form for *brukersentrert designprosess* [9]. Brukersentrert design er en metode for å utforme et produkt i samsvar med brukernes behov og ønsker. Det er en prosess hvor brukerne spiller en sentral rolle i utviklingen av produktet. Ettersom forfatteren av oppgaven hadde tilnærment ingen kunnskap innen domenet til mobilapplikasjonen ble denne metoden valgt for å sørge for at produktet som skulle bli utviklet møtte kravene og behovet til målgruppen. Med tanke på at Professor *Hvasshovd* har tung kunnskap innenfor dette domenet ble han ansett som en bruker av produktet i denne sammenhengen. En visualisering av en brukersentrert designprosess blir vist på Figur 5.1.



Figur 5.1: Brukersentrert designprosess, hentet fra [9]

5.1.2 Håndskisser

I begynnelsen av designprosessen ble det utformet håndskisser av deler av mobilapplikasjonen. Denne skisseringsprosessen er den tiden hvor problemer blir løst og ideer kommer til syne [1]. Denne metoden er spesielt essensiell i idémyldringsfasen for å få visualisert tankene man har. Grunnen til at denne metoden ble valgt i starten var for å få en idé om hvordan applikasjonen kunne se ut og hvilke komponenter som var nødvendige å ha. I tillegg tok det mindre tid å skissere flere alternativer for hånd til hvordan blant annet startsiden, innloggingssiden og hjemskjermen kunne se ut. Håndtegningene er lagt til i Vedlegg .1.

5.1.3 Figma

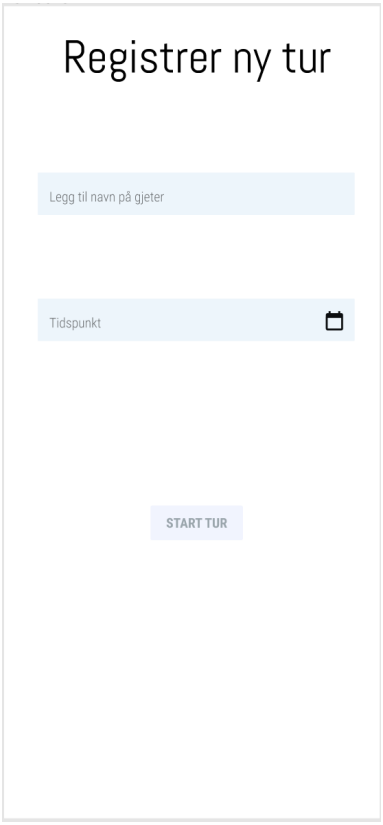
I neste steg av designprosessen ble verktøyet *Figma* brukt for å utforme en prototype for designet av mobilapplikasjonen. *Figma* er et nettbasert redigerings- og prototypeverktøy som bruker vektorgrafikk [15]. Med *Figma* kan man lage interaktive skisser med komponenter og drive med høynivås prototyping av designet. Dette verktøyet ble anbefalt av Professor *Hvasshovd* å bruke for å skissere og lage en prototype av designet som kunne bli testet under den iterative designprosessen.

Skisser

Etter veiledning fra Professor *Hvasshovd* ble det kun utviklet *Figma*-skisser av noen av de viktigste funksjonalitetene til mobilapplikasjonen, som oppretting av en ny tilsynstur, valg av kart og lagrede tilsynsturer. Håndskissene til startsiden, innloggingssiden og hjemskjermen hadde blitt godkjent av Professor *Hvasshovd* og det ble derfor ikke prioritert å lage prototype av disse. Ettersom forfatteren av oppgaven har lite erfaring med *Figma* ble det utfordrende å få utformet en skisse av beskjæring av kart. Derfor ble denne funksjonaliteten heller ikke prioritert, men den har blitt implementert i applikasjonen. Implementasjonen av denne funksjonaliteten blir beskrevet i Kapittel 5.2.2. *Figma*-prototypen kan bli funnet her: https://www.figma.com/proto/Fn3EQrQ5iRAQLzqQLmbXsw/B%C3%A6%C3%A6dar?page-id=0%3A1&node-id=2%3A3&viewport=279%2C48%2C0.62&scaling=scale-down&starting-point-node-id=2%3A3&show-proto-sidebar=1. Videre blir disse skissene presentert.

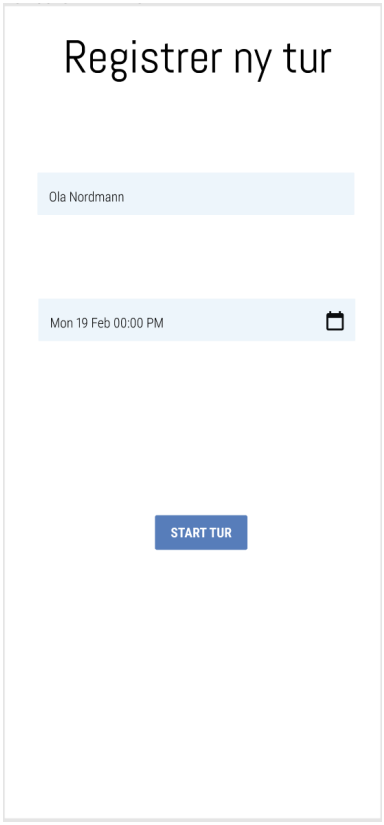
Oppretting av ny tilsynstur

For å opprette en ny tilsynstur må en bruker legge til nødvendig informasjon som navn og tidspunkt for turen, som vist på Figur 5.2.



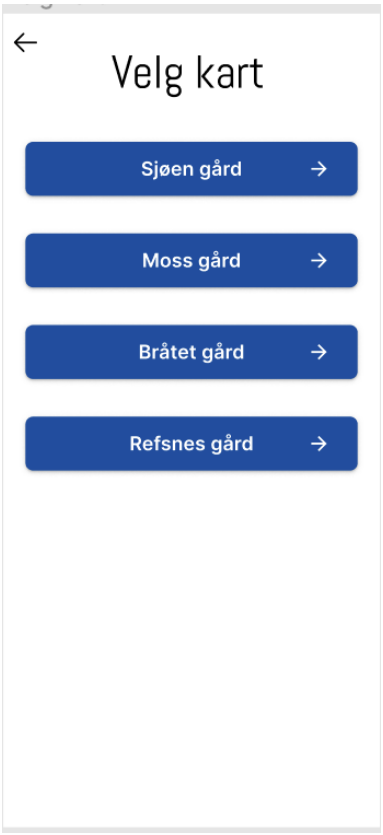
Figur 5.2: Oppretting av ny tilsynstur, hentet fra fordypningsprosjekt høsten 2021

Etter at den nødvendige informasjonen har blitt fylt ut, kan skjemaet for eksempel se slik ut som på Figur 5.3:



Figur 5.3: Eksempel på utfylt skjema, hentet fra fordypningsprosjekt høsten 2021

Deretter skal en bruker velge et kart han/hun ønsker å benytte på tilsynsturen blant ferdignedlastede kart over forskjellige områder (se Figur 5.4).

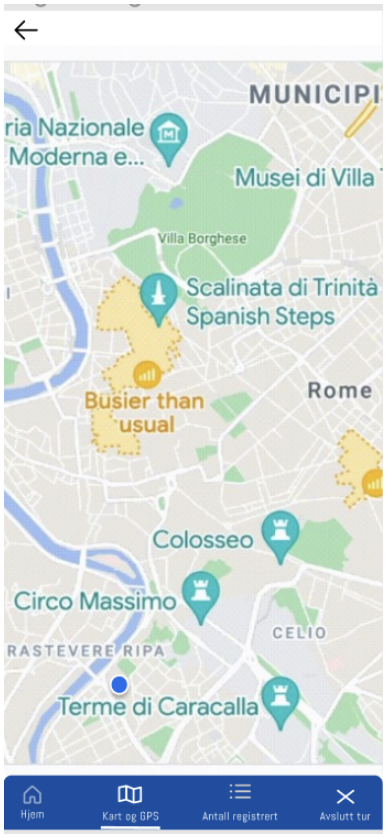


Figur 5.4: Valg av nedlastede kart over områder, hentet fra fordypningsprosjekt høsten 2021

Etter at disse stegene har blitt fullført er brukeren klar til å dra på tilsynsturen.

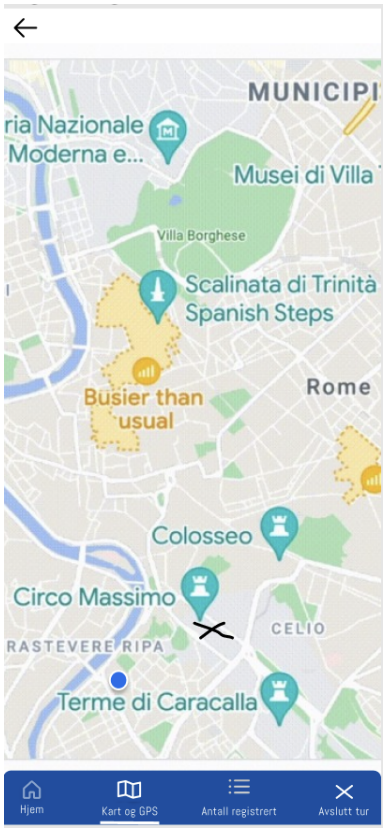
Aktiv tilsynstur

Etter tilsynsturen har blitt startet vil det dukke opp en navigasjonsbar nederst på skjermen, med alternativer som *Hjem*, *Kart og GPS*, *Antall registrert* og *Avslutt tur*. På *Kart og GPS*-skjermen vil brukerens nåværende posisjon på det valgte kartet bli vist. Dette blir vist på Figur 5.5. Et tilfeldig kart over Roma har blitt valgt som et eksempelkart for å demonstrere funksjonaliteten.



Figur 5.5: Oversikt over nåværende posisjon og området, hentet fra fordypningsprosjekt høsten 2021

Ifølge kravspesifikasjonen i Kapittel 3 skal brukeren kunne markere av på kartet hvor han/hun har sett saueflokken. For å skissere dette kravet er det tenkt at brukeren skal kunne sette et kryss der saueflokken befinner seg, som vist på Figur 5.6.

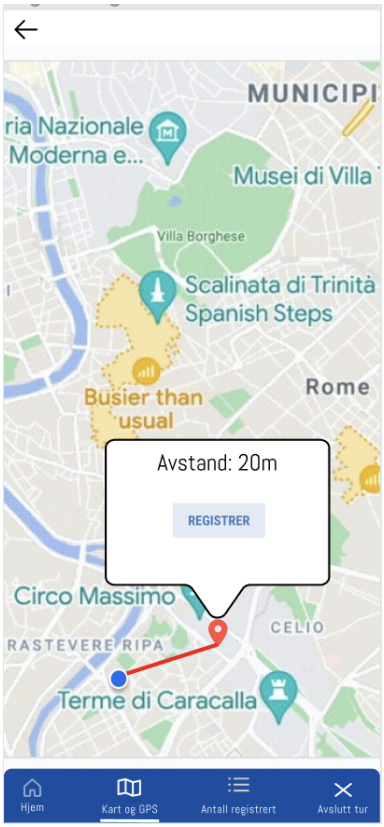


Figur 5.6: Markert område av saueflokken, hentet fra fordypningsprosjekt høsten 2021

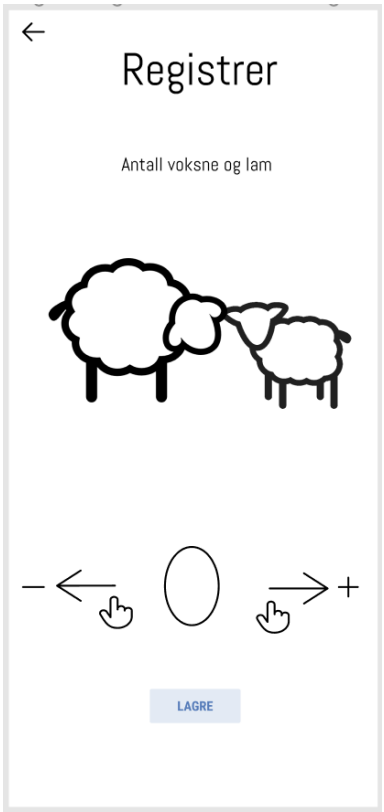
Deretter kan det for eksempel dukke opp en strek fra brukerens posisjon til det satte krysset, hvor avstanden til saueflokken da kan bli estimert som vist på Figur 5.7. Siden avstanden i dette eksempelet er 20m, vil følgende detaljer bli demonstrert på de neste figurene:

- Antall sau og lam (se Figur 5.8)
- Antall lam (se Figur 5.9)
- Antall brune, svarte og gråe sauer (se Figur 5.10, Figur 5.11 og Figur 5.12)
- Farge på slips (se Figur 5.13, Figur 5.14 og Figur 5.15)
- Eierskap (farge på øremerket, se Figur 5.16)

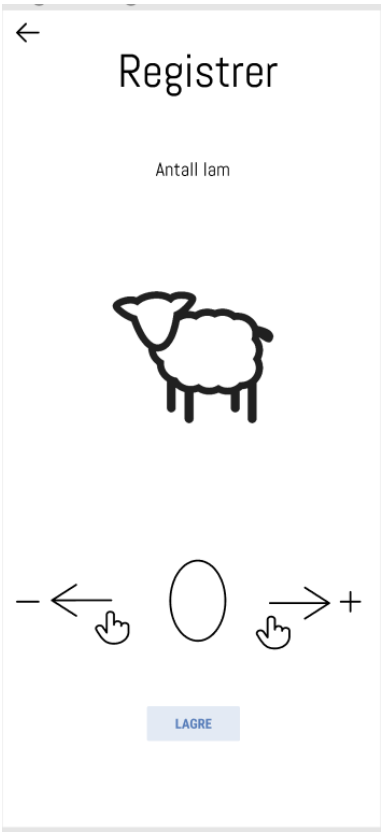
Disse detaljene vil bli registrert ved tastetrykk på +/--knapper, slik at brukeren kan holde et øye med saueflokken mens han/hun teller. Etter at brukeren er ferdig med å registrere trykker brukeren på *Lagre*-knappen for å lagre detaljene og blir deretter sendt til neste informasjon som skal bli registrert. Registrering av antall skadde og døde sauer er tenkt å fungere på samme måte som registrering av antall sau og lam og har dermed ikke blitt demonstrert på *Figma*-prototypen.



Figur 5.7: Avstand fra saueflokk, hentet fra fordypningsprosjekt høsten 2021

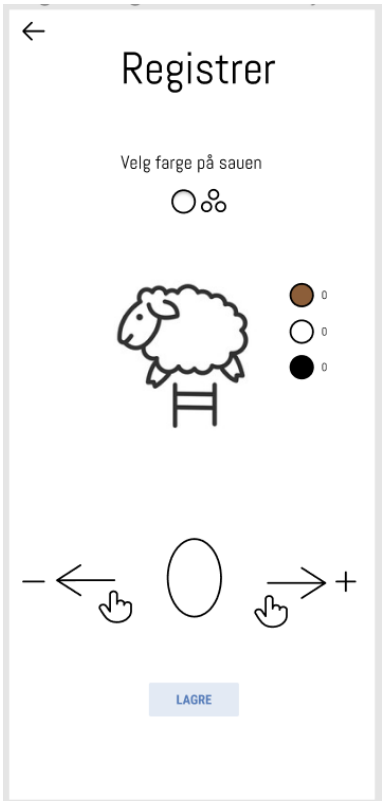


Figur 5.8: Registrering av antall sau og lam, hentet fra fordypningsprosjekt høsten 2021

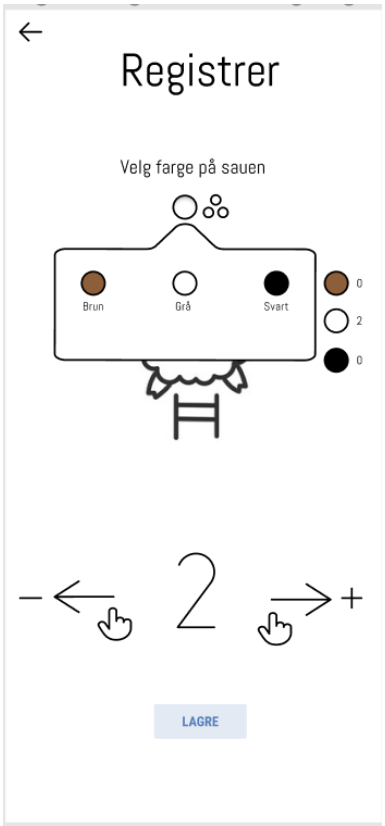


Figur 5.9: Registrering av antall lam, hentet fra fordypningsprosjekt høsten 2021

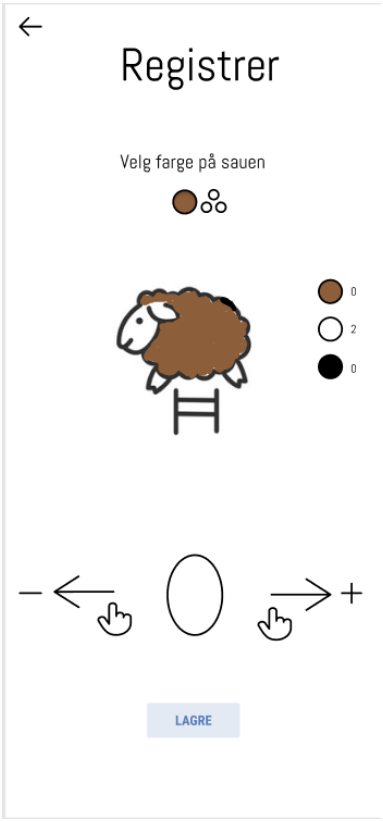
Når brukeren skal registrere antall sau i en farge velger brukeren den fargen det skal registreres på ved å trykke på fargepaletten som vist på Figur 5.11. Underveis på registreringen kan brukeren se antall sauer som har blitt registrert på de tre fargene.



Figur 5.10: Registrering av hvite sauer, hentet fra fordypningsprosjekt høsten 2021

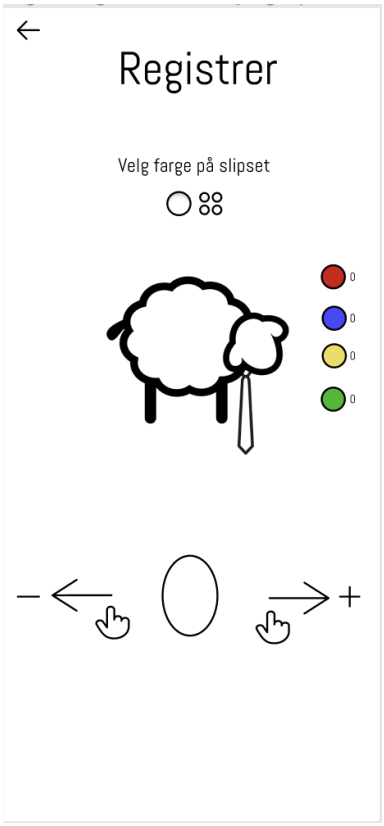


Figur 5.11: Registrering av farge på sau, hentet fra fordypningsprosjekt høsten 2021

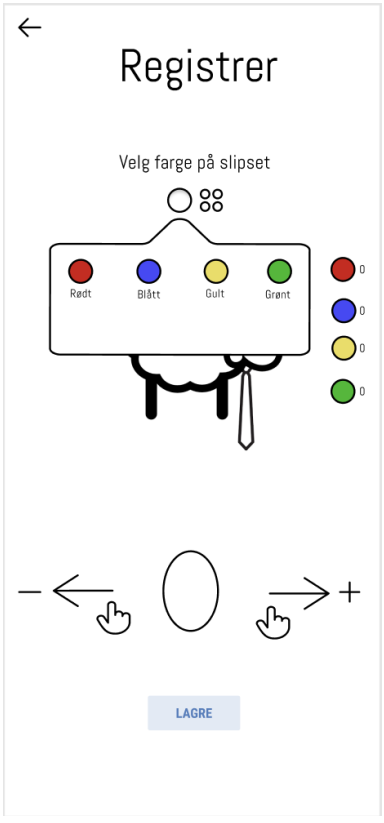


Figur 5.12: Registrering av brune sauer, hentet fra fordypningsprosjekt høsten 2021

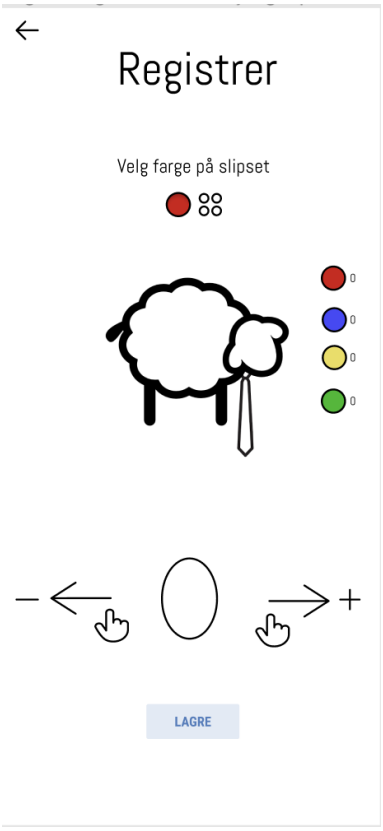
På samme måte som på registrering av farge på sau velger brukeren farge på slipset det skal registreres på som vist på Figur 5.14. Fargen på slipset til sauen indikerer antall lam en sau har.



Figur 5.13: Registrering av farge på slips, hentet fra fordypningsprosjekt høsten 2021

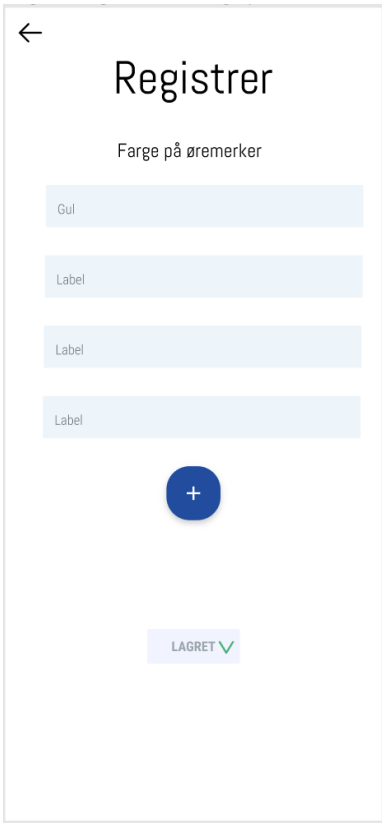


Figur 5.14: Valg av farge på slips, hentet fra fordypningsprosjekt høsten 2021



Figur 5.15: Registrering av rødt slips, hentet fra fordypningsprosjekt høsten 2021

Når brukeren skal registrere fargen på øremerket til sauene skal brukeren oppgi de forskjellige fargene han/hun observerer som vist på Figur 5.16. Fargen på øremerket til sauene gir en indikasjon på hvilken gård sauene tilhører.

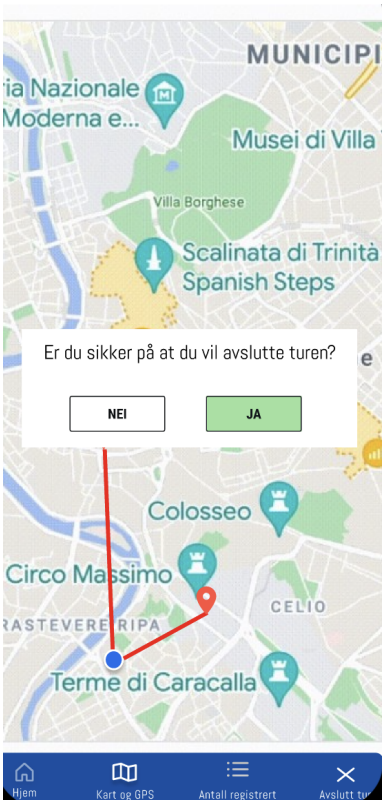


Figur 5.16: Registrering av farge på øremerket, hentet fra fordypningsprosjekt høsten 2021

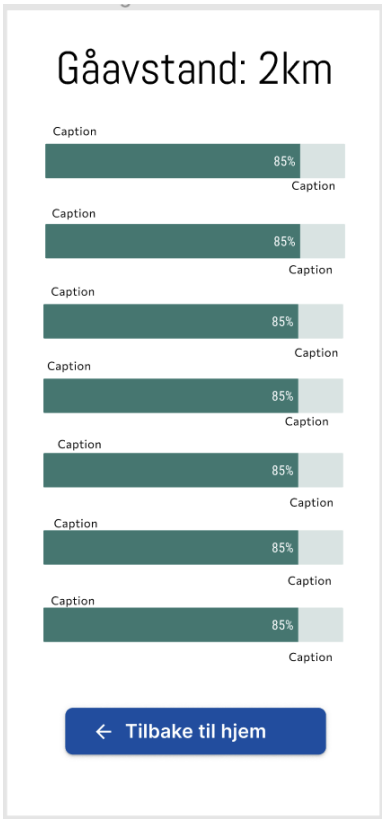
Antall registrert-skjermen på navigeringsbaren gir brukeren en oversikt over informasjonen som har blitt registrert hittil i den pågående turen, som vist på Figur 5.17.

Figur 5.17: Oversikt over antall registrert, hentet fra fordypningsprosjekt høsten 2021

Når brukeren er ferdig på tilsynsturen trykker han/hun på *Avslutt tur* nede til høyre på navigasjonsbaren. Da vil det dukke opp en *popup*, hvor brukeren enten trykker på ja eller nei for å avslutte turen. Dette blir vist på Figur 5.18. Etter at tilsynstyren har blitt avsluttet vil et sammedrag av alt som har blitt registrert på turen bli vist, som på Figur 5.19. Foreløpig står det *caption*, men det er planlagt at det senere skal stå prosentandeler av for eksempel hvor mange av de registrerte sauene som er lam, hvite eller skadde.



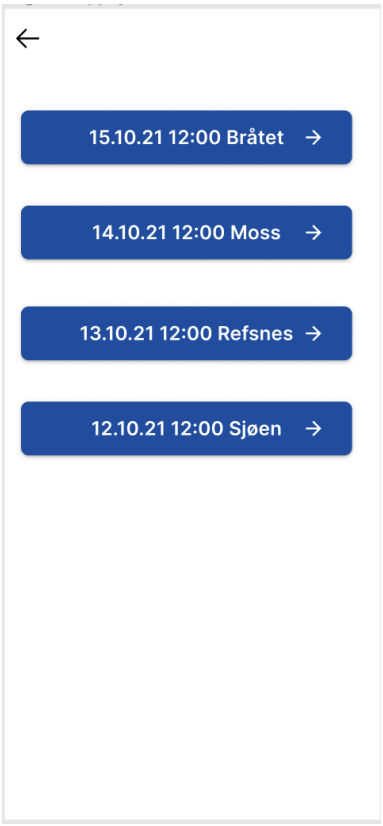
Figur 5.18: Avslutte tilsynstur, hentet fra fordypningsprosjekt høsten 2021



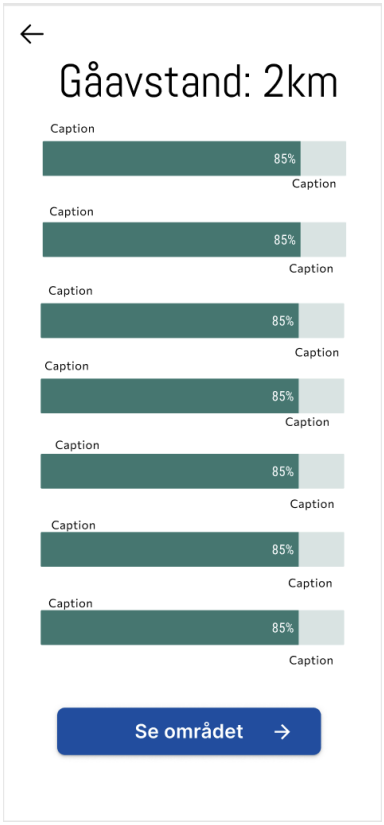
Figur 5.19: Sammendrag av tilsynstur, hentet fra fordypningsprosjekt høsten 2021

Lagrede tilsynsturer

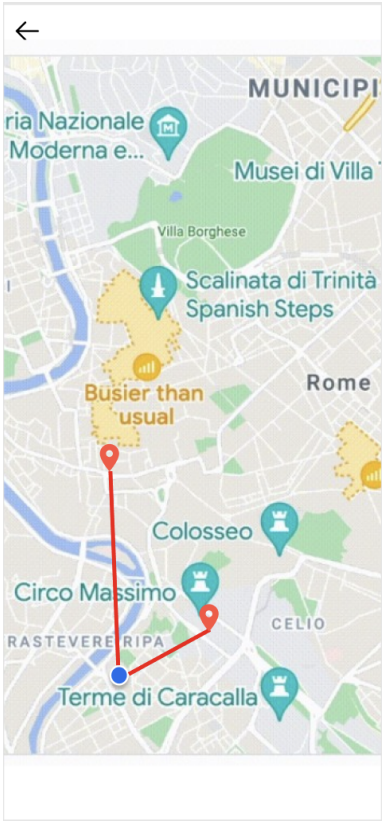
Gjennomførte tilsynsturer blir lagt til i en mappe som brukeren kan gå inn på for å se detaljer fra turene. Dette blir vist på Figur 5.20. For å se informasjonen fra en tilsynstur kan brukeren trykke seg inn på den, som vist på Figur 5.21. Trykker brukeren seg videre på *Se området*-knappen får brukeren en oversikt over hvor det ble gått på tilsynsturen. Dette blir vist på Figur 5.22.



Figur 5.20: Lagrede tilsynsturer, hentet fra fordypningsprosjekt høsten 2021



Figur 5.21: Oversikt av en lagret tilsynstur, hentet fra fordypningsprosjekt høsten 2021



Figur 5.22: Oversikt over gått område, hentet fra fordypningsprosjekt høsten 2021

5.2 Utvikling av applikasjon

Ettersom nettsiden til *Bæædar* ikke har blitt utviklet, vil dette kapittelet omhandle teknologivalg og utfordringer knyttet til utvikling av mobilapplikasjonen.

5.2.1 Smidig utvikling

For å utvikle mobilapplikasjonen *Bæædar* ble det ikke tatt i bruk en konkret utviklingsmetodikk, grunnet at dette prosjektet ble gjennomført individuelt. Likevel, ble det gjennomført en *smidig* utviklingsprosess med litt inspirasjon fra *Scrum*-metodikken [71]. Smidig utviklingsmetodikk er en iterativ tilnærming til utvikling av et produkt. Det er fokus på å levere små, fungerende deler av produktet på hyppig basis i samarbeid med kunden [89]. Med tanke på at Professor *Hvasshovd* har tung kunnskap innenfor dette feltet ble han ansett som en bruker av produktet i denne sammenhengen. På ukentlige veiledningsmøter med Professor *Hvasshovd* ble det vist frem progresjon og gitt tilbakemeldinger på applikasjonen. Til neste møte ble det da gjort endringer på applikasjonen i henhold til tilbakemeldingene og i tillegg gjøremål som Professor *Hvasshovd* kom med. På denne måten ble *Scrum*-metodikken til en viss grad integrert i utviklingsprosessen ved at hvert møte ble en form for *demo* og *sprintplanlegging*, med *sprinter* som varte i en uke.

På grunn av usikkerhet og tilnærmet ingen kunnskap innenfor domenet til produktet ble det valgt en smidig prosjektmetodikk. Uten all nødvendig informasjon om behovene og bakgrunnen til kunden var det mer praktisk å ha en iterativ prosess for å få tilbakemeldinger på funksjonalitet og progresjon for å vite om utviklingen gikk i riktig retning. I tillegg var det flere krav som ble lagt til underveis som ikke var satt i kravspesifikasjonen i utgangspunktet.

5.2.2 Valg av teknologi

Mobilapplikasjonen ble utviklet med *cross-platform*-teknologi. Systemet er utviklet med kun én kodebase som fungerer på flere platformer [83]. Ettersom mobilapplikasjonen *Bæædar* skal fungere på både *Android* og *iOS*, ble rammeverket *React Native* valgt som teknologi for å utvikle applikasjonen siden det har god støtte for *cross-platform*-teknologi. I tillegg har forfatteren av oppgaven et godt kjennskap til *JavaScript*-biblioteket *React* fra før av. Sammen med *React Native* ble programmeringsspråket *TypeScript* brukt. Valget av database landet på *Google Cloud Firestore*, en skydatabase som *Firebase* tilbyr.

Front-end

React Native-rammeverket og programmeringsspråket *TypeScript* ble brukt som frontend-teknologi for å utvikle mobilapplikasjonen. *React Native* er et *open-source-UI*-rammeverk som støtter *cross-plattform*-utvikling av mobilapplikasjoner, og er basert på *React* [11]. Det som skiller *React Native* fra *React* er blant annet at *React Native* bruker *UI*-komponenter for å utvikle *JavaScript*-kode, i tillegg til et sett med komponenter for både *iOS*- og *Android*-plattformer [86]. Derimot er *ReactJS* et *open-source JavaScript*-bibliotek for å lage brukergrensesnitt for webapplikasjoner. Felles for*React Native* og *ReactJS* er at de bruker de samme designprinsippene. Designgrensesnittet er derimot forskjellig.

Komponentbiblioteker som ble brukt sammen med *React Native* var *React Native Paper* og *React Native Vector Icons*. I tillegg ble *React Navigation* - en *plugin* for *React Native* for å navigere mellom ulike skjermer benyttet.

React Native

Med *React Native* har man muligheten til å lage *React*-applikasjoner for mobiler ved å bruke *JSX* - en utvidelse av *JavaScript* og *ES6*-basert syntaks [4]. Fordeler med å bruke *React Native* er blant

annet at:

- Mobilapplikasjoner blir utviklet med webteknologi, noe som gjør det enkelt for hvilken som helst webutvikler å forbedre sine ferdigheter innen *React*.
- *React Native* tilbyr *cross-plattform*-utvikling for mobilapplikasjoner.
- Det er et mobilrammeverk som kompilerer app-komponenter med *Native*-APIer [91]. De samme byggeklossene som blir brukt til *iOS*- og *Android*-utvikling benyttes også av *React Native*. *React Native* kommuniserer direkte med utvalgte komponenter for *iOS* og *Android* og genererer kode til *Native*-APIer. *React Native* bruker en separert prosess fra brukergrensesnittet for å gjøre dette, noe som øker ytelsen til applikasjonen.
- Det er basert på *React* og innehar *React* sitt konsept om gjenbrukbare komponenter.
- *Native*-kontroller og *Native*-moduler til *React Native* forbedrer ytelsen [91]. *React Native* tilbyr et sett med *Native*-moduler som er skrevet i *Objective-C* og *Java*. Disse modulene kan derimot ikke brukes på *Android*. Likevel, kan resten av kodebasen blir gjenbrukt i begge plattformer.
- *React Native* tilbyr enkle verktøy til feilsøking (debugging).

React Native-rammeverket har også noen ulemper. Ulemper med *React Native* er blant annet at:

- Det finnes lite dokumentasjon. Ettersom *React Native* er et relativt nytt og ungt rammeverk, er det fortsatt lite dokumentasjon som er tilgjengelig, spesielt for integrasjon med tilleggsverktøy [91]. I dette tilfellet med utvikling av *Bædar* var det vanskelig å finne dokumentasjon på hvordan man skulle laste inn et kart-API og modifisere det med tilleggsverktøy, uten å bruke *Google Maps*, *Apple Maps* og lignende. For eksempel, finnes det ikke støtte for å bruke *React Native* med *Leaflet*, et verktøy som blir hyppig brukt av utviklere til å lage dynamiske kart. Mer om dette kan leses senere i Kapittel 5.2.4.
- *Native*-moduler kan forhindre utvikling av *cross-platform*-applikasjon ettersom man trenger *Objective-C*, *Java* eller begge for at det skal fungere.
- Det er begrenset antall tredjepartskomponenter. *React Native* har et begrenset tilbud av funksjoner som utviklere kan implementere i applikasjonene sine.
- Synkroniseringen mellom *React Native* og de nye *SDK*-ene tar ofte lang tid. Hver gang *iOS* eller *Android* oppdaterer *SDK*-ene sine må *React Native*-teamet integrere et kodebibliotek til den nye programvaren. Til tross for at dette skjer forholdsvis raskt, er det ikke mulig å oppdatere alle APIene samtidig.
- Det kan oppstå ustabilitet, kompatibilitetsproblemer og feilmeldinger [91]:
  - Reload-feil
  - Inkompatibiliteter mellom *community*-bilbioteker og forskjellige versjoner av *React Native*
  - Emulatorproblemer
  - Problemer med respons av navigering
  - Det at man stadig vekk trenger å installere pakker på nytt
  - Forskjellige feilmeldinger som kan være vanskelige å tolke

React Native Paper

React Native Paper er et komponentbibliotek for *React Native*-applikasjoner som er *cross-platform*, slik at designet vil være likt på både *iOS*- og *Android*-enheter. *React Native Paper* har en kolleksjon av ferdiglagde komponenter for *React Native* som kan tilpasses etter ønske til mobilapplikasjoner

[6]. Det er et komponentbibliotek som følger *Google Material Design Guidelines* [32] og er enkelt å bruke. Flere av komponentene som *React Native Paper* har samsvarte også med komponentene som hadde blitt utformet på *Figma*-skissene i Kapittel 5.1. Av disse grunnene ble dette komponentbiblioteket valgt.

React Native Vector Icons

React Native Vector Icons er et bibliotek som blir brukt til å vise fram ikoner fra *Material Design Icons*. Mange av komponentene i *React Native Paper* som har ikonegenskap krever dette biblioteket for å kunne bli lastet inn riktig [6].

React Navigation

React Navigation er et navigasjonsverktøy for å bytte mellom forskjellige skjermer og håndtere navigasjonshistorikken for *React Native*-applikasjoner [14]. *React Navigation* har en *native stack navigator* som fungerer på en lignende måte som loggen til en nettside. Den legger til og henter ut ting fra navigasjonsstabelen samtidig som en bruker interagerer med den. Dette gjør at brukeren kan se og navigere mellom forskjellige skjermer. En viktig forskjell på hvordan navigasjon fungerer på en nettside og hvordan *React Navigation* opererer i *React Native*-applikasjoner er at *native stack navigator* i *React Navigation* animerer navigasjonen mellom ulike skjermer. Denne navigasjonen blir animert på en måte som er forventet å se på *Android*- og *iOS*-enheter.

Ettersom *React Native* ikke har en innebygd logghistorie som en nettside har ble *React Navigation* benyttet. Dette verktøyet er et av de mest anerkjente navigasjonsbibliotekene som blir brukt av utviklere til å implementere funksjonalitet for å navigere mellom forskjellige skjermer i *React Native*-applikasjoner [95].

TypeScript

TypeScript er et *open-source* programmeringsspråk som er bygget på JavaScript. Det er et såkalt *superset* av JavaScript og utvider språket ved å benytte seg av statiske typedefinisjoner [55]. *TypeScript* er et svært populært programmeringsspråk som blir benyttet av de fleste utviklere til applikasjonsutvikling. Fordeler med å bruke *TypeScript* i stedet for *JavaScript* er blant annet:

- **Statiske typer:** Med *TypeScript* kan man gi en variabel en type, for eksempel *string*. Etter at typen til en variabel er deklarert, kan ikke variabelen endre på typen sin og kan kun ta inn spesifikke verdier med samme type. Med slik statisk typedefinisjon kan kompilatoren varsle om type-relaterte feil, både på syntaksbaserte og semantiske feil. Dette resulterer i tidlig feildeteksjon, noe *JavaScript* mangler [70]. Dersom man ikke vet eller har en fastsatt type til en variabel kan den bli definert som typen *any*.
- **Tidlig feildeteksjon:** Med kompilatoren som kan sjekke koden, fange opp varsler og feil under utvikling synker sannsynligheten for feil og uforventet oppførsel av systemet under kjøretid [31].
- **Forbedret lesbarhet:** Med *TypeScript* sin statiske lesning og sitt grensesnitt blir koden mer lesbar. Dette øker kodeoptimaliseringen. Grensesnittet til *TypeScript* kan som sagt bli brukt til å definere typer og i tillegg implementere disse i klasser, noe som resulterer i en mer informativ kodebase. *JavaScript* derimot krever manuell deteksjon av feil.
- **Optimalisert kode:** *TypeScript* har bedre kodeorganisering og objektarrangert programmeringsprosedyrer [70]. De følgende funksjonalitetene kan føre til økt utviklingshastighet:
  - **Typeannotering:** Automatisk sjekk av verdier for hver statiske type
  - **Generisk:** Mulighet for å skrive generaliserte metodeformer

- **API dokumentasjon:** Med *Visual Studio Code* kan utviklere se parametertyper automatisk og og spore variabler
- ***Intellisense*:** Et instrument som er utviklet av *Microsoft* for kodenavigering som tilbyr automatisk kodefullføring.

De ovennevnte egenskapene har ikke *JavaScript*. En ulempe med å bruke *TypeScript* istedenfor *JavaScript* er derimot at *TypeScript* bruker tid på å kompilere koden. Ettersom *TypeScript* først må bli konvertert til *JavaScript*-kode for å kunne brukes i nettlesere er det bedre å bruke *JavaScript* hvis det er snakk om en liten kodebase. For at det skal bli lettere å videreutvikle *Bæædar* senere, ble *TypeScript* valgt for å gjøre koden mer lesbar for andre utviklere. I tillegg kan kodebasen til applikasjonen potensielt vokse seg større, noe som kan være enda en grunn til å benytte *TypeScript* fra begynnelsen av. Mer informasjon om hvordan *TypeScript* fungerer i praksis finnes her [55] og her [70].

Back-end og database

Som backend-teknologi landet valget på *Firebase*. *Firebase* er en *Google*-plattform som tilbyr en rekke funksjonaliteter til utvikling av mobil- og webapplikasjoner. Det er verktøy som gjør det mulig for en utvikler å få dekket store deler av tjenestene man vanligvis må utvikle selv [81]. Disse tjenestene involverer som regel håndtering av autentisering, oppsett av database og lignende. Med *Firebase* er disse tjenestene skybaserte og blir vedlikeholdt av *Google*. Med tanke på tidsbegrensningen og omfanget av prosjektet ble derfor *Firebase* valgt. Ved å benytte funksjonalitetene *Firebase* har kunne fokuset bli satt på utvikling av applikasjonen. For å utvikle *Bæædar* ble *Cloud Firestore* og *Firebase Authentication* benyttet.

Cloud Firestore

Cloud Firestore er en *NoSQL*, dokumentorientert database som er fleksibel og skalerbar [21]. Med en *klient-server*-arkitektur synkroniserer *Firestore* klientapplikasjonene på tvers av hverandre. Databasen er skybasert og bruker *native SDK* for å koble seg til applikasjoner. I tillegg er det *offline*-støtte, slik at man slipper å være avhengig av å være tilkoblet internett for å synkronisere data. Data på applikasjonen blir *cachet*, slik at det fortsatt er mulig å skrive, lese, lytte til og hente ut data selv om klientenheten ikke er koblet til internett. Når enheten igjen blir koblet til internett synkroniseres dataen automatisk til *Firestore* [16].

En av hovedgrunnene til at *Cloud Firestore* ble valgt var at det ikke var nødvendig for applikasjonen å ha en backend. Som nevnt, tilbyr *Firebase*-plattformen en rekke tjenester for håndtering og vedlikehold av for eksempel autentisering av brukere og database. Dermed kunne tiden bli brukt til utvikling av selve applikasjonen og brukergrensesnittet. Videre, var *offline*-støtten en stor motivator for å velge *Firestore*. Som beskrevet på krav **NFK-01** og **NFK-05** i Kapittel 3, skal mobilapplikasjonen fungere uten mobildekning og data skal bli automatisk synkronisert til databasen når sauebonden er tilkoblet internett igjen etter tilsynsturen. Ved å bruke *Cloud Firestore* blir disse kravene oppfylt.

Som nevnt i Kapittel 4, er *Cloud Firestore* fleksibel på den måten at datamodellen i databasen består av hierarkiske datastrukturer. Data blir lagret i dokumenter, som er organisert i kolleksjoner. Fordeler med en slik datamodell er at det er mulig å hente ut spesifikke dokumenter, eller alle dokumenter i en kolleksjon, med enkle spørringer. Disse spørringene kan inkludere filtrering, sortering og til og med en kombinasjon av filtrering og sortering slik at det kun blir hentet ut dokumenter som oppfyller disse kriteriene. I tillegg er det mulig å sette en grense på antall dokumenter man ønsker å hente ut fra en kolleksjon. Med spesifikke parametre som kan bli lagt til i spørringen unngår man å hente ut en hel kolleksjon for å kun få tak i ett dokument. Med *Cloud Firestore* er det også mulig å legge til *realtime*-lyttere slik at databasen holder seg oppdatert til enhver tid. På denne måten er det ikke nødvendig å hente ut alt fra databasen hver gang noe oppdateres. Data i *Firestore* er dessuten indeksbasert [21]. Mer om strukturen til *Firestore* kan leses i Kapittel 4.

Cloud Firestore har også et sett med sikkerhetsregler, *Firestore Security Rules*, som kontrollerer tilgang til dokumenter og kolleksjoner i databasen [28]. Det er mulig å utforme sine egne regler, for eksempel at alle har skrivetilgang til hele databasen, eller at det kun er tillatt å utføre operasjoner på et spesifikt dokument. Sikkerhetsreglene endres manuelt i *Firestore*-konsollen. For *Bæædar* ser sikkerhetsreglene slik ut som på Figur 5.23:

```
1  rules_version = '2';
2  service cloud.firestore {
3  match /databases/{database}/documents {
4   match /{document=**} {
5    allow read, write: if request.auth != null;
6    }
7   }
8  }
```

Figur 5.23: Sikkerhetsregler for *Bæædar*

service cloud.firestore vurderer reglene til *Firestore* og forhindrer konflikter mellom *Firestore Security Rules* og regler for andre tjenester som *Cloud Storage*.

match/databases/database/documents spesifiserer at reglene skal matche hvilken som helst *Firestore*-database i dette prosjektet.

match /document=** gir brukeren tilgang til alle dokumenter i alle kolleksjoner i databasen.

allow read, write: if request.auth != null spesifiserer at kun innloggede brukere har tilgang til å lese og skrive til databasen. Disse brukerene blir autentisert ved hjelp av *Firebase Authentication*.

En ulempe med å bruke *Cloud Firestore* er at det ikke finnes spørringer til aggregering av data [80]. Det er for eksempel ikke mulig å telle, summere, finne maksverdi og gjennomsnitt av alle verdier i en kolonne. *Firestore* er som nevnt en *NoSQL*-database. Fordeler med *SQL* er at det tilbyr fleksible måter å formulere spørringer på. Det er mulig å slå sammen data fra forskjellige tabeller, *join*, aggregere data og det finnes en rekke andre funksjoner som kan bli brukt. Likevel, kan disse *SQL*-spørringene kreve mye jobb og overskride kapasiteten til *hardware* dersom det er snakk om store tabeller. *Firestore* tilbyr derimot ikke like fleksible spørringer, men spørringene vil alltid skalere mye bedre enn *SQL*-spørringene [80]. I stedet kan man lage en *composite index*. En *composite indrex* holder en sortert *mapping* av alle dokumentene i en kolleksjon, basert på en organisert liste av felter til en indeks [26]. Med andre ord, *Firestore* tar og kombinerer flere felter sammen. Det største problemet med dette er når man må kombinere flere områder og filtrere på forskjellige felter på en fleksibel og uforutsigbar måte. Hver unike, komplekse kombinasjon av områder med filtre vil da kreve en ny *composite index*. Ettersom disse indeksene må bli opprettet manuelt i *Firebase*-konsollen, kan det bli litt innviklet.

Andre ulemper er blant annet at *Firestore* har en grense på skrivefrekvensen til et dokument på 1 skrivehastighet per sekund. Det er den maksimalt støttede skrivehastigheten til et dokument. Denne grensen kan skape problemer etter hvert dersom man prøver å overstige skrivehastigheten [80]. Dette problemet er ikke nødvendigvis lett å observere mens man utvikler applikasjonen, og er noe man burde være oppmerksom på. Et vanlig tilfelle hvor denne grensen kan bli problematisk er hvis man ønsker å bruke et dokument til opptelling av hendelser som skjer raskt. Dermed er ikke *Cloud Firestore* den optimale teknologien å gå for hvis man utvikler en applikasjon som har verdier som er i stadig endring.

I tillegg har *Firestore* en grense på hvor store dokumentene kan være. Mengden av data som et dokument kan inneholde er $1MB$ (megabyte) [80]. Som regel er ikke dette problematisk, men dersom man skal lagre datatyper som *map*- og *array*-felter som kan vokse seg større og større over tid uten en gitt øvre grense, kan det skape store skaleringsproblemer for applikasjonen. For eksempel, hvis man skal lagre en liste over alle følgerene til en bruker på *Instagram* kan det virke intuitivt å lage et *array* med strenger av id-en til alle følgerene i dokumentet til en spesifikk bruker. Etter hvert som en bruker blir mer og mer populær vil den øvre grensen på $1MB$ bli nådd, og da vil ikke de nye følgerene bli skrevet til databasen.

En annen svakhet ved *Cloud Firestore* er ytelsen til *offline*-støtten. Når applikasjonen er koblet til internett er det høy hastighet på spørringer og data blir skrevet og hentet ut raskt. Når disse spørringene derimot blir utført *offline*, med *Firebase SDK* som kun bruker lokalt *cachet* data, er det ingen indekser tilgjengelige til bruk [80]. Det går mye raskere å hente ut et dokument hvis man har iden til det dokumentet for da trenger *Firestore* bare å finne dokument-iden med en *hashmap*-struktur. Dersom en spørring ikke bruker iden, kan spørringen matche flere dokumenter. Dette kan ha en stor innvirkning på ytelsen til *Firestore*. Uten lokale indekser må *Firestore SDK* skanne alle dokumentene som er *cachet* for å finne riktig match, noe som kan gjøre at spørringene går veldig sakte. Hvor dårlig ytelsen er kommer an på hvor stor datamengden er [40].

Firebase Authentication

Firebase Authentication er en *Firebase*-funksjonalitet som brukes til autentisering av brukere. Den tilbyr *backend*-tjenester, enkle *SDK*-er og ferdiglagde *UI*-bibliotek til å autentisere brukere i applikasjonen [23]. *Firebase Authentication* støtter autentisering ved bruk av passord, telefonnummeret og populære tredjepartssapplikasjoner som *Google*, *Facebook* og *Twitter*. Det er hovedsakelig to måter å integrere *Firebase Authentication* på; *FirebaseUI Auth* og *Firebase Authentication SDK*.

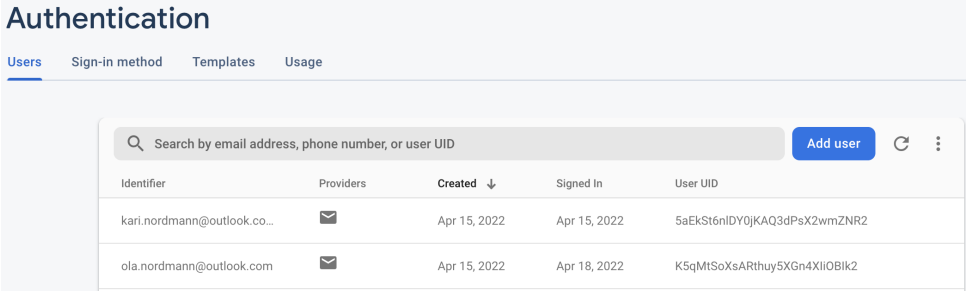
FirebaseUI Auth er en *open source*, *drop-in* autentiseringsløsning som *Firebase Authentication* tilbyr. Dette *UI*-et kan man bruke som innloggingssystem på applikasjonen. Den håndterer *UI*-flyten for innlogging med e-post og passord, telefonnumre og tredjepartssapplikasjoner [23]. *FirebaseUI Auth*-komponenten implementerer de beste praksisene for autentisering på mobilenheter og nettsider, noe som kan maksimere transfigurering for innlogging og oppretting av bruker for ens applikasjon.

Dersom man ønsker å tilpasse designet på innloggingssystemet til sitt eget kan man integrere *Firebase Authentication SDK* manuelt i applikasjonen sin [23]. Det er en rekke autentiseringsmetoder man kan velge mellom:

- **Autentisering med e-post og passord:**
  Tilbyr metoder for å opprette og håndtere brukere som bruker e-post og passord til innlogging. *Firebase Authentication* sender også mailer om nullstilling av passord.
- **Identitetsintegrasjon via ekstern applikasjon:**
  Autentiserer brukere ved å integrere trdjepartsapplikasjoner. *Firebase Authentication SDK* tilbyr metoder som gjør at brukere kan logge seg inn med *Google*, *Facebook*, *Twitter* og *Github*.
- **Autentisering med telefonnummer:**
  Autentiserer brukere ved å sende SMS til deres mobilenheter.
- **Integrasjon av tilpasset autentiseringssystem:**
  Kobler applikasjonens innloggingssystem til *Firebase Authentication SDK* og får tilgang til *Firebase Realtime Database* og andre *Firebase*-tjenester, som i dette tilfellet er *Cloud Firestore*.
- **Anonym autentisering:**
  Bruker funksjoner som krever autentisering uten at brukere må logge seg inn først ved å opprette midlertidige anonyme kontoer. Hvis brukeren senere velger å opprette en bruker kan man oppgradere den anonyme kontoen til en vanlig brukerkonto, slik at man kan fortsette der man slapp.

Firebase Authentication SDK fungerer på den måten at man først får autentiseringsinformasjon fra brukeren [23]. Denne informasjonen kan være brukerens e-post og passord, eller en *OAuth*-token fra en ekstern tredjepartsapplikasjon. Deretter blir autentiseringsinformasjonen sendt videre til *Firebase Authentication SDK*. *Backend*-tjenestene vil da verifisere denne informasjonen og gi en respons tilbake til klienten. Etter at man har fått logget seg inn har man tilgang til brukerens profilinformasjon, og man kan kontrollere brukerens tilgang til lagret data i andre *Firebase*-produkter, som *Cloud Firestore* i denne sammenhengen.

Til denne applikasjonen ble *Firebase Authentication SDK* brukt. Grunnen til det var at det var enkelt å sette seg inn i, istedenfor å måtte sette seg inn i *FirebaseUI Auth*, finne ut av funksjonaliteten fungerer og gjøre justeringer på eksisterende kode for å tilpasse designet til innloggingssystemet til *Bædar*. *Firebase Authentication SDK* ble derfor integrert og tilpasset innloggingsflyten til mobilapplikasjonen *Bædar*, med e-post og passord. Når brukeren taster inn e-post og passord vil denne informasjonen da bli sendt via *Firebase Authentication SDK* for å verifisere brukeren. Hvis autentiseringen er vellykket, vil brukere få tilgang til resten av dataen i applikasjonen. Hvis ikke, vil det komme opp en feilmelding og brukeren får ikke tilgang til applikasjonen. I konsollen til denne applikasjonen i *Firebase* kan man se alle autentiserte brukere i applikasjonen som vist på Figur 5.24.



Figur 5.24: *Firebase Authentication* i *Firebase*-konsollen til *Bædar*

Firebase Authentication kommer også med noen begrensninger. Det er begrensninger på antall autentiseringsoperasjoner som kan bli kjørt i løpet av en viss tid [24]. Operasjoner med begrensinger er:

- **Oppretting av ny brukerkonto:**
  100 kontoer/IP-adresse i timen
- **Sletting av brukerkonto:**
  10 kontoer per sekund
- **Brukerkontoer per prosjekt:**
  - Anonyme brukerkontoer:
    100 millioner
  - Registrerte brukerkontoer:
    Ubegrenset
- **E-post begrensninger:**
  - E-post for adresseverifisering:
    * *Spark plan*-grense:
      1000 e-poster per dag
    * *Blaze plan*-grense:
      100000 e-poster per dag
  - E-post for adresseendring:
    * *Spark plan*-grense:
      1000 e-poster per dag
    * *Blaze plan*-grense:
      10000 e-poster per dag

- E-post for passordendring:
    - *Spark plan*-grense:
      150 e-poster per dag
    - *Blaze-plan*-grense:
      10000 e-poster per dag
- E-poster med link til innlogging:
    - *Spark plan*-grense:
      2000 e-poster per dag
    - *Blaze-plan*-grense:
      25000 e-poster per dag

- **Begrensinger for generering av e-post lenker:**
    - Lenker til adresseverifisering:
        - *Spark plan*-grense:
          10000 e-poster per dag
        - *Blaze-plan*-grense:
          1000000 e-poster per dag
    - Lenker til passordendring:
        - *Spark plan*-grense:
          1500 e-poster per dag
        - *Blaze-plan*-grense:
          100000 e-poster per dag
    - Lenker til innlogging:
        - *Spark plan*-grense:
          20000 e-poster per dag
        - *Blaze-plan*-grense:
          250000 e-poster per dag

- **Begrensninger for innlogging av telefonnummer:**
    - Brukerinnlogging:
      1600/prosjekt/minutt
    - SMS-meldinger med verifiseringskode:
        - 50 meldinger/IP-adresse/minutt
        - 500 meldinger/IP-adresse/time
        - 1500 meldinger/prosjekt/minutt
    - Verifiseringsforespørsler:
      150 forespørsler/IP-adresse/time

- **API-begrensinger:**
    - Operasjoner per tjenestekonto:
      500 forespørsler/sekund
    - Operasjoner per prosjekt:
        - 1000 forespørsler/sekund

* 10 millioner forespørsler/dag

I tillegg er det en øvre grense på antall brukere som kan bli autentisert gratis. Blir grensen oversteget vil autentiseringstjenesten koste penger.

Kart og GPS

For å lage funksjonalitet for kart i henhold til kravspesifikasjonen i Kapittel 3 ble norgeskartet fra *Kartverket* brukt sammen med kartbiblioteket *Openlayers*. For å spore og loggføre GPS-posisjon er det planlagt å bruke *React Native Background Location*-plugin fra *Transistorsoftware*.

OpenLayers

OpenLayers er et *open-source JavaScript*-bibliotek som i utgangspunktet blir brukt for å vise fram dynamiske kart i nettlesere [63]. *OpenLayers* tilbyr et rikt API som gjør det mulig å lage enkle, samt veldig komplekse kartapplikasjoner med mye fleksibilitet. Det er støtte for formater som *GeoJSON*, *GeoRSS*, *KML*, *GML*, *WMS* og *WFS*. I dette prosjektet ble *GeoJSON* brukt som format for kartdata. *GeoJSON* er et åpent standardformat som representerer enkle geografiske funksjoner og deres attributter [94]. Den er basert på *JSON*-formatet og kan bli brukt for å kode geografiske datastrukturer. Ettersom data i *Cloud Firestore* kan bli behandlet som *JSON*-data var det veldig optimalt at *OpenLayers* støttet dette formatet slik at det er lett å lagre kartdata i databasen. Nærmere info om hvordan OpenLayers kan implementeres kan leses her [34].

I dette prosjektet ble *OpenLayers* brukt sammen med *Kartverket* sitt API for norgeskart [37]. På *GitHub*-kontoen til *Kartverket* finnes det eksisterende kode på hvordan API-et kunne bli lastet inn med *OpenLayers* [38]. Her blir norgeskartet lastet inn sammen med *xyz* og *OSM - Open Street Map* i bakgrunnen. *OSM* er et kart over hele verden som er modifiserbar og gratis å bruke, og er basert på innsamlede data fra GPSer og andre ressurser [68]. Et bilde av hvordan dette ser ut i mobilapplikasjonen blir vist på Figur 5.25.

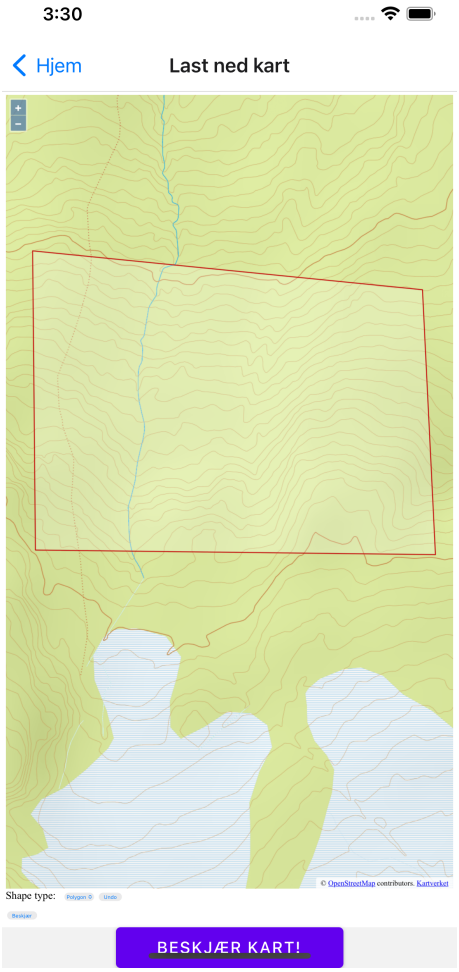
Figur 5.25: Kartfunksjonen i *Bædar*

For å bruke *OpenLayers*-funksjonalitet i en *React Native*-applikasjon måtte koden for funksjonaliteten bli skrevet som et *HTML*-script. Deretter måtte dette scriptet bli gjengitt i *WebView*-komponenten til *React Native* for å bli vist frem [44].

Det var først tenkt at *OpenLayers* sin *DragBox*-funksjonalitet skulle brukes for å markere et område på kartet [66]. Denne funksjonaliteten gjør det mulig for en bruker å lage en vektorboks ved å trykke og dra på kartet, og dermed blir området innenfor boksen markert. *DragBox* blir som regel brukt til å zoome inn på et spesifikt området. Til mobilapplikasjonen var nettopp denne funksjonaliteten ønsket, ettersom en sauebonde skulle kunne markere et område på kartet og deretter kun få opp det markerte området som skal bli lastet ned på mobilen til tilsynstur. I tillegg ga dette et mer ryddig design ved at den boksen ble rektangulær og brukeren kun bestemte størrelsen på den. Derimot var ikke dette mulig å få til, siden det ikke finnes mobilstøtte for *dragbox* i *OpenLayers*. Denne funksjonaliteten hadde bare støtte for webapplikasjoner, hvor man da kunne tegne vektorboksen ved å holde nede enten *Ctrl*- eller *Command*-knappen og dra boksen ut på kartet. Et eksempel på hvordan dette fungerer finnes her [64].

For å implementere funksjonalitet for å markere et spesifikt område på kartet som beskrevet på Tabell 3.1 i Kapittel 3, ble *Draw*-funksjonaliteten til *OpenLayers* benyttet [67]. Denne funksjonaliteten blir brukt til å tegne geometriske figurer på kartet. Man kan velge mellom å tegne *Point, LineString, Polygon* eller *Circle*. En visualisering av hvordan dette fungerer i praksis blir vist her [65]. Til denne mobilapplikasjonen ble valget begrenset til et polygon og en sirkel, ettersom hovedfunksjonen er å markere et spesifikt område, ikke et spesifikt punkt på kartet. Et eksempel på hvordan det ser ut i mobilapplikasjonen blir vist på Figur 5.26. Markering av et spesifikt punkt er planlagt å bli brukt på det nedlastede, valgte kartet underveis på tilsynsturen. For å kunne lagre

et spesifikt, markert område på kartet ble *Draw*-funksjonaliteten sin *writeFeatures* brukt. Denne funksjonen ble brukt til å konvertere det markerte området til et *GeoJSON*-objekt, slik at det kunne bli lagret i *Cloud Firestore*-databasen som en *JSON*-string. Ved å konvertere området til dette formatet, var det også enkelt å hente det fra databasen senere og vise det på kartet på nytt, ettersom *OpenLayers* tilbyr støtte for *GeoJSON*-format.



Figur 5.26: Markering av et område med *Draw* i *Bæædar*

Ulemper med *OpenLayers* er at mye av dokumentasjonen er utdatert. Mange av eksemplene som finnes på hjemmesiden viser gamle metoder å implementere funksjonalitetene på som ikke fungerer optimalt lenger. API-dokumentasjonen er oversiktlig og strukturert, men inneholder så mye informasjon at det kan bli tungvint og forvirrende i begynnelsen å skulle finne fram til spesifikke ting man tenker å implementere. I tillegg er ikke *OpenLayers* sitt fellesskap så utbredt. Siden dette biblioteket som regel blir brukt til å utvikle komplekse *GIS*-applikasjoner, er fellesskapet mye mindre sammenlignet med for eksempel *Leaflet*. Det finnes spesielt svært lite dokumentasjon på hvordan funksjonalitet skal implementeres i mobilapplikasjoner. Det var dermed vanskelig å finne oppdaterte eksempler på nettet om hvordan *OpenLayers* skulle implementeres i *React Native*. En stor del av tiden gikk til å undersøke om det eksisterte framgangsmåter på internett om implementasjon av *OpenLayers*.

React Native Background Geolocation

For å spore lokasjon og loggføre dette i applikasjonen var det planlagt å bruke *React Native Background Geolocation* av *Trasistorsoftware* [88]. Ettersom GPS og lokasjonssporing ikke har blitt

implementert vil dette delkapittelet handle om hvordan *React Native Background Geoloaction* fungerer.

React Native Background Geolocation er et *cross-platform plugin* til lokasjonssporing i bakgrunnen og *geofencing*. Den bruker APIer for bevegelsessensorer for å detektere når mobilenheten er i bevegelse og når enheten står stille [87]. Når mobilen er i bevegelse, vil *plugin*-en automatisk starte å registrere og loggføre en lokasjon i henhold til den konfigurerte *distanceFilter* (metere). Når mobilen blir detektert som stillestående, vil *plugin*-en automatisk slå av lokasjonstjenestene for å spare på strømmen og batteritiden. Grunnen til at *React Native Background Geolocation* blir ansett som et verktøy å bruke er at det finnes en del dokumentasjon og eksempler på hvordan det fungerer på internett. I tillegg tilbyr det *geofencing* som kan avgrense et geografisk område slik at lokasjonen da blir sporet innenfor dette området. *React Native Background Geolocation* er gratis å bruke på iOS-enheter, mens *Android*-modulen krever en lisens. En komplett guide på hvordan det settes opp og implementeres finnes her [87].

Problemer som kan forekomme med *React Native Background Geolocation* er:

- **Tilgang til lokasjonssporing i bakgrunnen på *iOS* 13+ - enheter:** På *iOS* 13 og senere versjoner har det blitt forbudt å be om tillatelse til lokasjonssporing i bakgrunnen direkte fra brukeren ved hjelp av *pop-ups* [96]. Brukere må nå selv gå inn på innstillinger på mobilenheten og gi tillatelse, noe som ikke er lett å oppdage at det er nødvendig å gjøre.
- **Lokasjons/tidsbasert-sporing:** Dersom det er ønskelig med sporing i sanntid kan det være utfordrende å få implementert, spesielt hvis man ønsker å få den nyeste lokasjonsinformasjonen hver gang. Det er ikke nødvendigvis enkelt å implementere eller at det møter ens krav til hva denne informasjonen skal inneholde [96]. Derfor kan man heller bruke tidsbasert sporing, hvor lokasjonsdata blir sendt til backend på ett minutt hvis enheten er stillestående, og på sekunder med kortere intervaller hvis enheten er i bevegelse. Ulempen med denne metoden er derimot at den bruker opp mye strøm på mobilenheten.
- **Strømmodus på mobilenheter:** Ettersom lokasjonen til enheten blir sporet kontinuerlig, vil operativsystemene på mobilenhetene automatisk slå på strømmodus, spesielt på *Android*-enheter. Når dette skjer vil det hindre lokasjonssporing hvis applikasjonen kjører i bakgrunnen.

Mer informasjon om installasjon og konfigurering finnes her [96].

5.2.3 Evaluering av valgt teknologi

Det finnes alltid flere teknologier som kunne blitt tatt i bruk for å utvikle mobilapplikasjonen *Bæædar*. De teknologiene som det er kjennskap til og som har flere av de samme funksjonalitetene som de valgte for prosjektet er blant annet *Apachae Cordova*, *Leaflet* og *Microsoft Azure*.

Vurdering av Apache Cordova og React Native

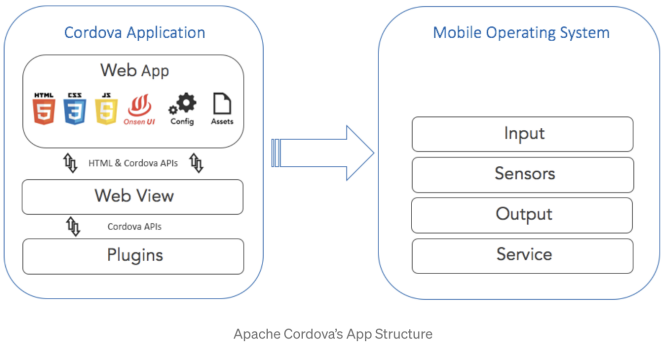
Apache Cordova er et *open-source* rammeverk som blir brukt for å utvikle *cross-platform* mobilapplikasjoner ved bruk av webteknologi som *HTML5*, *CSS3* og *JavaScript* [5]. Applikasjonen blir kjørt i en *webcontainer* på mobilen. *Cordova* har *plugins* som tillater å bruke *JavaScript* for å aksessere mobilens API, for eksempel for å bruke kamera, GPS-posisjon og andre mobilfunksjonaliteter. *Cordova* er et rammeverk som har eksistert i mange år og det finnes mange tilgjengelige ressurser på internett. I tilleg er *Cordova* integrerbar med *UI*-rammeverk som *Sencha Touch*, *Dojo Mobile* og *jQuery Mobile* [3].

I likhet med *Cordova*, er *React Native* som sagt også et *open-source* rammeverk, men med fokus på å utvikle *native cross-platform* mobilapplikasjoner. Det essensielle med å utvikle *React Native*-applikasjoner er å bruke *React*s designmønster og ha muligheten til å lage gjenbrukbare

komponenter. Når applikasjonen kompileres blir *React UI*-komponenter kompilert ned til plattformens *native UI view*-elementer. Til forskjell fra *Cordova* er *React Native* relativt nytt og det finnes derfor ikke like mange tilgjengelige ressurser på internett sammenlignet med *Cordova*.

Sammenligning av ytelse

Ved å sammenligne *Apache Cordova* og *React Native* på ytelse ser man at *Cordova* har bedre tidsytelse for bygging og produserer mindre byggepakker enn *React Native* når det skal kjøres. *Cordova* tilbyr to API-lag som gjør det mulig for mobilapplikasjonen å kommunisere med *Cordovas WebView* for rendering og for *WebView* å kommunisere med *Cordova plugins* [13]. Ettersom applikasjonen kjøres i en *wecontainer* slipper man å måtte bygge applikasjonen til mobilenheten for å teste systemet, noe som er nødvendig å gjøre med *React Native*-applikasjoner. Applikasjonen kan bli testet direkte i nettleseren, noe som kan være tidsbesparende [82]. I tillegg er det kun *plugins* som man spesifikt har tatt i bruk i applikasjonen som kompileres og blir lagt til [13]. En oversikt over applikasjonsstrukturen til *Cordova* kan ses på Figur 5.27.



Figur 5.27: Overordnet arkitektur av *Cordova*-applikasjonen tatt fra [13]

Ettersom *Cordova* er skrevet med standard webteknologier vil tiden det tar å utføre og rendere applikasjoner være mer enn *native*-applikasjoner [82]. Spesielt for store applikasjoner som krever mye data og funksjonalitet er ikke *Cordova* et optimalt valg. Dette skyldes at *JavaScript* er et *interpreted* programmeringsspråk, ikke et kompilert programmeringsspråk [90]. I stedet for at det kjøres gjennom en kompilator som oversetter språket til bytekode som maskinen forstår og kan utføre, er det en *interpreter* i nettleseren som leser over *JavaScript*-kode, tolker hver linje med kode og kjører det. Dette gjør at prosessen tar lang tid. I tillegg bruker *Cordova HMTL* og *CSS* til design og visualisering av applikasjonen, som også er *interpreted* språk.

React Native bruker derimot lengre tid på å bygge opp applikasjonen, samt at pakkestørrelsen er større. Likevel, har *React Native* innebygd støtte for *native*-funksjonalitet, som kamera, mens *Cordova* krever *plugins* for å tilby støtte til *native*-funksjonaliteter. Dette er en av grunnene til at *React Native*-applikasjoner er større når de bygges og kompileres, men det gir bedre ytelse i selve applikasjonen [13]. Som nevnt, er *React Natives UI*-komponenter koblet til *native UI views*, noe som gjør at det går raskere å rendere sammenlignet med *Cordova* [13]. Mer om arkitekturen til *React Native* kan leses i Delkapittel 4.3.1.

En annen ting med *React Native* er at det ikke trenger håndtering av *views*. *JavaScript*-pakken og nødvendige biblioteker blir lastet inn i *JavaScript* VM-en (*Virtual Machine*), analysert og generert til bytekode. Mer om *JavaScript* VM kan leses her [58].

Sammenligning av feilsøkingsmetoder

Som nevnt, har *Apache Cordova* mange tilgjengelige ressurser på internett og derfor god dokumentasjon på hvordan man kan forberede det lokale miljøet for *debugging* (feilsøking). *Cordova* kommer

ikke med noen nedlastede sett med verktøy til debugging, men det er mulig å bruke eksisterende verktøy i nettleserne som:

- *Safari* sin *Web Inspector med Debugging for iOS*
- *Chrome DevTools med Remote Debugging for Android*

Disse vektøyene gjør det enklere å feilsøke og oppdage feil uten å måtte gjenoppbygge hele applikasjonen for hver gang [13]. De gir tilgang til:

- *JavaScript Debugger*
- *Network Inspector*
- *DOM Inspector*

For *React Native* fungerer også *Chrome DevTools* bra til debugging av *Javascript*-kode. Det eneste *Chrome DevTools* mangler er funksjonalitet til å gå igjennom hele *React*-komponentens hierarkiske struktur [13]. For *ReactJS*-applikasjoner finnes det derimot *React Developer Tools Chrome extension* som gjør det mulig å å gjøre nettopp dette, men for *React Native*-applikasjoner har denne utvidelsen blitt gjort om til en applikasjon som man må laste ned og bruke i stedet. Alt av feilsøking i *React Native* gir også utviklere tilgang til en ekstra menyfane til å aktivere andre tillegg som:

- Ytelsesovervåkning
- Elementinspeksjon
- Systemsporing
- *Hot Reloading*
- *Live Reloading*

Oppsummering

Basert på informasjonen over, har både *Apache Cordova* og *React Native* sine fordeler og ulemper. Begge rammeverkene bruker *cross-platform*-teknologi og egner seg utmerket til mobilutvikling på hver sin måte. Til dette prosjektet var det *React Native*-rammeverket som ble valgt, mest på grunn av et godt kjennskap til *ReactJS* fra før av. Derfor virket det naturlig å fortsette i *React*-miljøet. Med tanke på tidsbegrensing og omfanget av prosjektet, ble det prioritert å bruke minst mulig tid på å lære seg et helt nytt og ukjent rammeverk. Likevel, har *React Native* bydd på en del utfordringer. Alt fra implementasjon av *Kartverket* sitt API til nedlasting av tredjepartspakker som er inkompatible med *React Native*-versjonen som blir brukt i applikasjonen. Mer om utfordringer rundt *React Native* kan lese i Kapittel 5.2.4.

Vurdering av Leaflet og OpenLayers

Leaflet er et *open-source JavaScript*-bibliotek for å utvikle mobilvennlige interaktive kart [42]. Biblioteket har de fleste kartfunksjonalitetene utviklere trenger, blant annet zoom-knapper, markører og popups, konvertering til *GeoJSON* og vektorlag som polygoner og sirkler [41]. I tillegg fungerer *Leaflet* på tvers av plattformer, med god mobilstøtte. Ettersom *Leaflet* er et populært kartbibliotek som er hyppig brukt av mange utviklere har den et veldokumentert, oppdatert API, samt svært mange tilgjengelige ressurser på nettet om hvordan det skal implementeres. Det er lite som må settes opp før man kan ta i bruk biblioteket og enkelt å lære seg.

I likhet med *Leaflet*, er også *OpenLayers* et *open-source JavaScript*-bibliotek. *OpenLayers* er kraftigere enn *Leaflet*, og tilbyr langt flere mer avanserte kartfunksjonaliteter. Likevel, er det mer

komplisert å lære seg ettersom det er en del grunnleggende ting man må ha på plass for å ta det i bruk [85]. Til forskjell fra *Leaflet* finnes det litt mindre god dokumentasjon på hvordan *OpenLayers* skal fungere.Det eksisterer en del eksempler på forskjellige funksjoner dette biblioteket har, men mye av denne dokumentasjonen er utdatert. Mer informasjon om *OpenLayers* finnes i Kapittel 5.2.2.

Sammenligning av bruksområde

Leaflet egner seg best til applikasjoner som har enkle krav til kartinteraksjon. Den håndterer enkle kartfunksjonaliteter som zooming og markering på kart godt. Dersom man er ute etter flere avanserte funksjoner, må man bruke *Leaflet*s *plugins* [85].

Ettersom *Leaflet* er et høyere-nivås API enn *OpenLayers*, krever det mindre *JavaScript*-kode for å sette det opp og få det til å kjøre [85]. Som nevnt, har *Leaflet* velfungerende standard kartfunksjonalitet. Dersom det er ønskelig å implementere mer avansert funksjonalitet kan det hende at det må bli skrevet en del unødvendig kode for å tilpasse funksjonaliteten. Hvis ikke, må man finne en *plugin* som tilbyr den funksjonaliteten man ønsker. Med andre ord, *Leaflet* er ikke så lett å tilpasse etter behov.

OpenLayers er derimot lettere å tilpasse etter behov og ønske. Med et lavere-nivås API, krever det mer *JavaScript*-kode for å initialisere og sette opp kartet, men det betyr også at det er enklere å modifisere koden og tilpasse det etter krav. Dersom applikasjonen skal ha en del avanserte kartfunkjsonaliteter, er *OpenLayers* det optimale valget. I tilleg til å ha alle standardfunksjonalitetene som *Leaflet* har, tilbyr *OpenLayers* en god del flere. Likevel, er dokumentasjonen til *OpenLayers* omfattende og vanskelig å forstå seg på. Med et stort og komplekst API fører det til at læringskurven er mye brattere med *OpenLayers* enn *Leaflet*.

Oppsummering

Basert på informasjonen som er beskrevet over, har både *Leaflet* og *OpenLayers* sine styrker og svakheter. Det er tilfeller hvor det er bedre å bruke *Leaflet*, som når applikasjonen kun trenger enkel standard kartfunksjonalitet og man har begrenset med tid på å utvikle applikasjonen. På samme måte er det situasjoner hvor applikasjonen krever mer kompleks funksjonalitet og tilpassing, og da vil *OpenLayers* være det mer optimale valget. Til dette prosjektet var det *OpenLayers* som ble valgt, selv om *Leaflet* hadde vært det mer optimale valget med tanke på at *Bædar* kun trengte standard kartfunskjonalitet. Grunnen til at *OpenLayers* ble valgt var at det ikke finnes støtte for *Leaflet* i *React Native* foreløpig. Etter flere forsøk på implementasjon av *Leaflet* i applikasjonen og flere dager med undersøkelser på internett etter metoder å implementere kartbiblioteket på, måtte valget til slutt lande på *OpenLayers*.

Vurdering av Azure og Firebase

I denne evalueringen blir *Microsoft Azure* og *Google Firebase* sammenlignet. *Microsoft Azure* er en *cloud computing*-plattform som er utviklet av *Microsoft*. Den brukes til å utvikle, publisere og administrere applikasjoner [7]. *Azure* tilbyr forskjellige tjenestedomener man kan bruke:

- **Compute**: Dette domenet blir brukt til å prosessere data i skyen ved bruk av kraftige prosessorer som gir flere instanser av gangen.
- **Storage Services**: Lagringstjenestene blir brukt til å lagre data i skyen med mulighet for skalerering hvis nødvendig.
- **Database**: Databasedomenet blir brukt til å tilby pålitelige relasjons- og ikke-relasjonsdatabaseinstanser som blir håndtert av *Azure*.

- **Networking**: Dette domenet gjør at man kan koble seg til skyen og *on-premise* infrastruktur, og andre tjenester for å forbedre brukeropplevelsen. *On-premise* infrastruktur er et skymiljø som kun er tilgjengelig for en klient.

På samme måte som *Microsoft Azure*, er *Google Firebase* en *cloud computing*-plattform som brukes til å utvikle og kjøre applikasjoner. *Firebase* tilbyr en rekke tjenester, som for eksempel *Cloud Firestore* og *Firebase Authentication* som har blitt brukt på dette prosjektet. Mer informasjon om *Cloud Firestore* og *Firebase Authentication* finnes i Kapittel 5.2.2. Ettersom *Firebase* er noe relativt nytt er det ikke like mange tjenester som tilbys sammenlignet med *Azure*. *Azure* har langt flere tjenester og produkter som kan benyttes.

Sammenligning av databaser

Azure har forskjellige databaser man kan velge mellom:

- **Azure SQL**: En samling av *SQL*-skydatabaser som tilbyr flere fleksible tjenester for applikasjonsmigrasjon, modernisering og utvikling. Disse tjenestene er blant annet **SQL Server on Azure Virtual Machines**, **Azure SQL Managed Instance**, **Azure SQL Database** og **Azure SQL Edge**. Mer om dette kan leses her [53].

- **Azure Cosmos DB**: En *Paas* (*Platform as a service*), skybasert *NoSQL*-database med åpne APIer for enhver skalering. *Azure Cosmos DB* er en serverløs, fullt-administrert *NoSQL*-databasetjeneste for applikasonsutvikling. APIer man kan bruke er blant annet:

  - **Core(SQL) API**
  - **Cassandra API**
  - **MongoDB API**
  - **Gremlin API**
  - **Table API**

  Avhengig av hvilket API man velger å bruke, kan *Cosmos DB* lagre data som nøkkel-verdi, dokumenter og til og med oppføre seg som en grafdatabase. Mer om dette kan leses her [49].

- **Azure Database for PostgreSQL**: En fullt-administrert og intelligent database for *PostgreSQL* som er skalerbar. Mer informasjon finnes her [52].

- **Azure Database for MySQL**: En fullt-administrert og skalerbar database for *MySQL*. Den er enkel å sette opp, håndtere og skalere, med blant annet avansert sikkerhet. Mer informasjon her [51].

- **Azure Database for MariaDB**: En fullt-administrert *community MariaDB*. Denne tilbyr funksjonalitet for å håndtere databasen slik at det ikke er nødvendig for utviklere å gjøre det. Mer informasjon her [50].

Hos *Firebase* er det to forskjellige skybaserte databaser som man kan velge mellom:

- **Firebase Realtime Database**: En skybasert, *NoSQL*-database som synkroniserer data på tvers av klienter i sanntid, og forblir tilgjengelig selv når applikasjonen går *offline*. Data blir lagret i *JSON*-format som *JSON*-objekter, og synkronisert i sanntid til hver klient. Data er strukturert som et *JSON*-tre, hvor data som blir lagt til er en node i den eksisterende *JSON*-strukturen med en assosiert nøkkel. Datastrukturen kan for eksempel se slik ut som vist på Figur 5.28.

```
{
  "users": {
    "alovelace": {
      "name": "Ada Lovelace",
      "contacts": { "ghopper": true },
    },
    "ghopper": { ... },
    "eclarke": { ... }
  }
}
```

Figur 5.28: Datastruktur for en bruker, **alovelace**, tatt fra [25]

Til tross for at *Firebase Realtime Database* tillatter nøsting av data på opp til 32 nivåer, burde man helst unngå det. Hver gang man henter ut data fra databasen henter man også alle barnenodene til den dataen. Det vil i praksis si at hvis man gir noen lese- og skrivetilgang på en node i databasen, gir man dem også tilgang til all dataen som tilhører den noden. Derfor er det *best practice* å prøve og holde datastrukturen så flat som mulig [25].

- **Cloud Firestore**: *Cloud Firestore* er også, som nevnt i Kapittel 5.2.2, en skybasert database med god støtte til mobilutvikling. Til forskjell fra *Firebase Realtime Database* lagres data som kolleksjoner av dokumenter. Mer om datastrukturen kan leses i Kapittel 4.4.

Med utgangspunkt i de spesifiserte kravene for applikasjonen *Bædar* i Kapittel 3, ble *Cloud Firestore* valgt. For det første, var det ikke nødvendig med synkronisering og oppdatering av data i sanntid. Dermed var ikke *Firebase Realtime Database* like aktuell lenger. I tillegg var det bedre med indekserte spørringer med gode muligheter for sortering og filtrering enn dype spørringer med begrenset sortering og filtrerting blant dypt nøstede data. Ettersom *Realtime Database* returnerer hele deltreet hver gang man henter data hadde det vært problematisk å hente for eksempel kartdata, hvor det kun er ønskelig å hente ut feltet med *JSON*-string med data til *GeoJSON*-objektet. Med *Firestore* kan man spesifisere den dataen man vil hente, og til og med begrense antall dokumenter som blir returnert fra en kolleksjon. Videre er *Cloud Firestore* mer skalerbart enn *Realtime Database*, hvor *Firestore* skalerer data automatisk med en øvre grense på 1 million samtidige tilkoblinger og 10000 skrivinger per sekund. *Realtime Database* skalerer kun til rundt 20000 samtidige tilkoblinger og 1000 skrivinger per sekund i én database. Dersom det er ønskelig å skalere mer enn dette kreves det at data blir delt på tvers av flere databaser. Derimot har *Realtime Database* ingen grense på antall ganger man kan skrive til individuelle deler av data, mens *Cloud Firestore* har en begrensning på dette.

Når det kommer til sikkerhet, har *Realtime Database cascading* regelspråk, *Realtime Database Rules*, som separerer autorisasjon og validering. *Cloud Firestore* har *non-cascading* regler som kombinerer autorisasjon og validering med *Cloud Firestore Security Rules*. En ulempe med *Cloud Firestore Security Rules* er at det kan oppstå tilgangsbegrensinger på spørringer, slik at en bruker for eksempel ikke får hentet ønsket data ettersom den inneholder ting som brukeren ikke har tilgang til. Denne begrensningen gjelder ikke for *Realtime Database Rules*. På grunnlag av at datastrukturen til mobilapplikasjonen *Bædar* er slik at de fleste dokumenter er tilknyttet en brukerid og blir hentet basert på brukeriden vil det ikke oppstå tilgangsproblemer. Mer informasjon om forskjeller mellom *Firebase Realtime Database* og *Cloud Firestore* finnes her [19].

En siste sammenligning er mellom *Cloud Firestore* og *Azure Cosmos DB*. Ettersom *Cloud Firestore* og *Azure Cosmos DB* har flere likheter og tilbyr på et vis samme funksjonalitet og tjenester ble det tatt utgangspunkt i disse to. I denne sammenligningen vil det bli tatt utgangspunkt i *Core(SQL)* APIet til *Cosmos DB*, siden dette APIet er anbefalt og holder seg stadig oppdatert med nye funksjonaliteter. *Core(SQL)* støtter både *JSON*-dokumenter og *SQL*-baserte spørringer. Sammenligningen av *Cloud Firestore* og *Azure Cosmos DB* blir vist på Tabell 5.1.

Tabell 5.1: Sammenligning av *Cloud Firestore* og *Azure Cosmos DB*

	Google Cloud Firestore	**Azure Cosmos DB**
Datamodell	Kolleksjoner av *JSON*-lignende dokumenter. *Schemaless* og *NoSQL*. *Firestore* støtter også *ACID*-transaksjoner med serialiserbar isolasjon, men uten noen begrensninger på logisk partisjonering som *Cosmos DB*. *Firestore* har ingen konsepter som logisk partisjonering. Hver spørring i *Firestore* krever en indeks. Som nevnt i Delkapittel 4.4, vil *Firestore* automatisk generere en indeks for hvert felt i dokumentet. For *composite*-spørringer, hvor man skal hente ut mer enn et felt, må man lage en passelig indeks manuelt i *Firestore*-konsollen for å utføre spørringen. Hvis ikke, vil spørringen feile. En annen ting med datamodellen til *Firestore* er at det støtter spesialiserte datatyper, som for eksempel *timestamp* og *geopoint*.	Oppfører seg som en dokumentdatabase med et *SQL*-API. Kolleksjoner i databasen er *containers* [75]. Data i *Cosmos* er representert som *containers* fulle av *items*, hvor hvert *item* er et *JSON*-dokument som er assosiert med *JavaScript*-logikk for å skrive store prosedyrer, utløsere og brukerdefinerte funksjoner. Et *item* kan være et dokument i en kolleksjon, rad i en tabell, node eller en kant i en graf. For hver *container* i databasen kan man spesifisere en *partition key* som *Cosmos DB* bruker for å logisk partisjonere datasettet. Dokumenter med samme *partition keys* er lagret på samme logiske partisjonering. De logsike partisjoneringene definerer omfanget av databasetransaksjoner hvor man kan oppdatere *items* innenfor en logisk partisjonering ved bruk av en transaksjon med en *snapshot*-isolasjon [10].

Lesing av data	Bruker passende *SDK* for å hente ut data fra databasen. Det er hovedsakelig to måter å lese data på: 1. **Lese et dokument med *resource path***: Hvert dokument i *Firestore* er identifisert med en unik *resource path*, som **users/:userId**. Dersom man kjenner til denne kan man hente dokumentet direkte fra databasen. For eksempel som vist på Figur 5.31. 2. **Lage en spørring på en kolleksjon for å hente ut dokument**: Dersom man ikke kjenner til *resource path* til dokumentet, ønsker å hente ut alle dokumenter i en kolleksjon eller et spesifikt antall dokumenter som matcher et sett med vilkår, kan man lage en spørring som gjør nettopp dette. Dette viser eksemplet vist på Figur 5.32, et kodeutsnitt fra *Bædar* hvor alle dokumentene i kolleksjonen *maps* der brukerid er lik iden til den aktive brukeren, sortert med seneste dato først, blir hentet fra databasen. Ingen støtte for aggregering.	Bruker passende *SDK* for å hente ut data fra databasen. Det er hovedsakelig to måter å lese data på: 1. **Dokument-id**: Dersom man har dokument-iden, og eventuelt *partitionKey*, kan man hente ut dataen som vist på Figur 5.29. 2. **Simplifisert SQL-spørring**: Disse spørringene tillatter filtrering, sortering, aggregering og *joins* som vanlig *SQL*. Det er mulig å bruke aggregeringer som **COUNT**, **SUM**, **MAX**, **MIN** og **AVG** på tvers av dokumenter i en *container*. Et eksempel er at man skal finne en bruker med e-posten deres i databasen som vist på Figur 5.30. Denne spørringen vil returnere en liste med dokumenter hvor e-post-feltet matcher den spesifiserte parameteren.
Skriving av data	For å legge til et nytt dokument i en kolleksjon kaller man på **set**-metoden for å sette data til det nye dokumentet. Et kodeutsnitt fra applikasjonen *Bædar* er vist på Figur 5.35, hvor det blir opprettet en ny bruker i kolleksjonen *users*. Ved å kalle på denne metoden blir det opprettet et nytt dokument hvis det ikke finnes i kolleksjonen fra før av. For å oppdatere et eksisterende dokument kan man enten bruke **update** eller **set**. Et eksempel tatt fra *Bædar* med **update**-metoden er vist på Figur 5.36, hvor feltet *areaName* blir oppdatert. For å slette et dokument bruker man **delete**-metoden. Se Figur 5.37.	For å lage et nytt dokument til en *container* kaller man på **create**-metoden for *items* i *containeren* og lager et objekt med en passende id [10]. Et eksempel er vist på Figur 5.33, hvor det blir lagd en ny bruker. For å oppdatere et dokument må man referere til det med en id og bruke **patch**-metoden for å oppdatere et felt, **replace** for å erstatte all data og **delete** for å slette dokumentet. Et eksempel på dette er vist på Figur 5.34.

Kostnader og priser	Hos *Cloud Firestore* er det en *pay-per-use* kostnadsstruktur [29]. Man betaler for: • **Antall dokumenter man leser, skriver og sletter** • **Hvor stor lagringsplass databasen bruker** • **Hvor mye nettverksbåndbredde man bruker** *Firestore* har en gratis kvote i oppstartsfasen [30], som tilbyr: • **1 GiB** Lagret data • **50 000** Dokumentlesninger per dag • **20 000** Dokumentskrivninger per dag • **20 000** Dokumentslettinger per dag • **10 GiB** Nettverksutganger per måned Dersom man overgår denne kvota, koster det: • **Dokumentlesninger**: **$0.06** per 100000 dokumenter • **Dokumentskrivninger**: **$0.18** per 100000 dokumenter • **Dokumentslettinger**: **$0.02** per 100000 dokumenter • **Datalagring**: **$0.18/GiB/måned** per dag. Mer informasjon om kostnader og priser kan leses her [29].	Hos *Azure Cosmos DB* betaler man for operasjonene man utfører på databasen og for lagringsplassen man bruker for data [54]: • **Databaseoperasjoner**: Hvor mye man betaler for databaseoperasjoner avhenger av hva slags *Azure Cosmos*-konto man bruker: – **Provisioned throughput**: Man spesifiserer gjennomstrømningen man trenger i *Request Units* per sekund (RU/s), slik at *Azure Cosmos DB* tildeler ressursene som kreves for å få til dette. Prisen er på **$0.008**/time per 100 RU/s. – **Serverless**: I serverløs modus trenger man ikke å sørge for noen gjennomstrømning når man lager ressurser. I slutten av fakturaperioden blir man fakturert for antall *Request Units* man har brukt i databaseoperasjonene. Prisen er på **$0.282/1M**RU • **Lagring**: Man blir fakturert for den totale mengden av lagringplass (i *GB*) som data og indekser bruker i en tilgitt time. Lagringen blir fakturert etter det faktiske forbruket. *Cosmos DB* tilbyr også gratisalternativer: • **Azure Cosmos DB free tier**: De første 1000 RU/s og 25*GB* lagringsplass er gratis. • **Azure free account**: $200 i *Azure*-kreditt for de første 30 dager og begrenset kvantitet av gratis tjenester i 12 måneder. • **Try Azure Cosmos DB for free**: Man lager en *Azure Cosmos DB*-konto, setter opp en database og kolleksjoner og kjører eksempelapplikasjonen. Den er gratis i en måned, med mulighet for å fornye kontoen så mange ganger man vil. • **Azure Cosmos DB emulator**: Et lokalt miljø som emulerer *Azure Cosmos DB*-tjenester for utviklingsformål. Det er mulig å bruke den for å utvikle og teste applikasjoner lokalt, uten å måtte opprette et *Azure*-abonnement eller betale eventuelle kostnader. Mer informasjon om kostnader og priser finnes her [54].

```
db.container(containerId).item(id, partitionKey).read();
```

Figur 5.29: Lesing av et dokument i *Azure Cosmos DB*, tatt fra [10]

```
db.container("Users")
  .items.query({
    query: `SELECT * FROM Users u WHERE u.email = @email`,
    parameters: [{ name: "@email", value: "ola.nordmann@outlook.com" }],
  })
  .fetchAll();
```

Figur 5.30: Simplifisert *SQL*-spørring i *Azure Cosmos DB*

```
db.doc(`users/${userId}`).get();

// Works with nested collections as well.
db.doc(`users/${userId}/addresses/${addressId}`).get();
```

Figur 5.31: Henting av et dokument med spesifisert *resource path* i *Cloud Firestore*, tatt fra [10]

```
firestore()
.collection('maps')
.where('uid', '==',auth().currentUser?.uid)
.orderBy('date', 'desc')
.get()
```

Figur 5.32: Henting av vilkårlig data med *Cloud Firestore*

```
db.container("Users").items.create({
  type: "User",
  id: "1",
  name: "Robert E. O. Speedwagon",
  email: "robert@speedwagon.foundation",
  age: 50,
  authProvider: "google",
});
```

Figur 5.33: Oppretting av ny bruker i *Azure Cosmos DB*, tatt fra [10]

```
// Overwrite only the age field
db.container("Users").item("1").patch({
  age: 55,
});

// Completely overwrite the existing data.
db.container("Users").item("1").replace({
  type: "User",
  id: "1",
  name: "Joseph Joestar",
  email: "joseph@speedwagon.foundation",
  age: 18,
  authProvider: "facebook",
});

// Delete this document from the container.
db.container("Users").item("1").delete();
```

Figur 5.34: Metoder for å oppdatere og slette et dokument i *Azure Cosmos DB*, tatt fra [10]

```
auth()
.createUserWithEmailAndPassword(email,password)
.then(()=> {
    console.log('User account created & signed in');
    navigation.navigate('Hjem');
    firestore().collection('users').doc(uid).set({
        firstName,
        lastName,
        email,
        city,
        farmNumber,
    });
})
```

Figur 5.35: Oppretting av nytt dokument i *Cloud Firestore*

```
firestore()
.collection('maps')
.doc(docId)
.update({
    areaName: text
});
```

Figur 5.36: Oppdatere et felt i et dokument

```
// Delete the document.
db.collection("users").doc("john").delete();
```

Figur 5.37: Sletting av et dokument i *Cloud Firestore*, tatt fra [10]

Sammenligning av brukeradministrasjon og autentisering

Hos *Azure* kan man bruke *Azure Active Directory B2C* (*Azure AD*) for å autentisere brukere ved innlogging. Denne autentiseringstjenesten tilbyr blant annet engangsinnlogging og multifaktor-autentisering [47]. Som nevnt i Kapittel 5.2.2, tilbyr *Firebase* tjenesten *Firebase Authentication* som har en rekke autentiseringsmetoder man kan implementere i applikasjonen. En sammenligning av autentiseringstjeneste blir vist på Tabell 5.2.

	Firebase Authentication	**Azure Active Directory B2C**
Egenskaper	• Innebygd funksjon til å identifisere om brukeren er logget inn eller ikke. • Mulighet til å bruke et tilpasset token for å registrere nye brukere • Mulighet for å bruke tredjepartsapplikasjoner til autentisering, som for eksempel *Facebook*, *Github* og *Twitter*. • Mulighet for å tilpasse til eget innloggingssystem • Autentisering med passord • Autentisering med telefonnummer • Anonym autentisering • Fleksibel, drop-in brukergrensesnitt - bruker *best practice* for autentisering på mobilenheter En ulempe er blant annet begrensninger på antall autentiseringsoperasjoner som kan bli kjørt på en viss tid. Mer informasjon finnes i Kapittel 5.2.2 og [23].	• Tilgjengelige *plugins* og pakker som reduserer utviklings- og konfigurasjonstiden. • Kompatibilitet for engangsinnlogging • Felles identitet for flere applikasjoner • Bedre sikkerhet med multifaktorautentisering • Forskjellige, tilgjengelige innstillinger for autentisering som selvbetjent tilbakestilling av passord, aktivering av *smart lockout* osv. En ulempe med *Azure AD* er at det ikke er lett å tilpasse autentiseringen etter ønske. Mer informasjon finnes på [47].
Kostnader og priser	To prismodeller å gå utfra: • *Spark plan* med en fast beløpsgrense • *Pay-as-you-go plan* hvor man betaler for hver verifikasjon Mer informasjon finnes i Kapittel 5.2.2 og [23].	Har fire versjoner: • Gratisversjon: Inkludert i abonnementet til blant annet *Azure*, *Dynamics 365*, *Intune* og *Power Platform*. • *Office 365*-apper: Abonnementene til disse appene inkluderer også gratisversjonen. • *Premium P1*: Har flere funksjonaliteter og tillegg. Denne versjonen er inkludert i *Microsoft 365*, og koster ellers $6 per bruker/måneden. • *Premium 2*: Inneholder alle funksjonaliteter og tillegg som finnes i *Azure AD*. Denne verjsonen er inkludert i *Microsoft 365*, og koster ellers $9 per bruker/måneden. Mer informasjon om priser og kostnader finnes her [48].

Tabell 5.2: Sammenligning av *Firebase Authentication* og *Azure Active Directory*

Oppsummering

Med utgangspunkt i informasjonen over, har både *Azure* og *Firebase* sine fordeler og ulemper. *Azure* fungerer godt hvis man har et stort prosjekt og trenger flere tjenester og tillegg [36]. I tillegg har *Azure* bedre programvaresikkerhet med høyere sikkerhetsstandarder enn *Firebase*. *Azure Cosmos DB* er en veldig fleksibel *NoSQL*-database som tilbyr en rekke APIer og kompatibilitetsmoduser. *Core*-APIet som det det har blitt tatt utgangspunkt i tilbyr et simplifisert *SQL*-språk som støtter filtrering, sortering og aggregering som vanlig *SQL*. Dette har ikke *Cloud Firestore*. *Firestore* har kun et API, som støtter mange av de samme spørringsmønstrene, bortsett fra aggregering på databasenivå. Dersom man sammenligner kostnadene til *Cosmos DB* og *Cloud Firestore* på samme arbeidsmengde er *Cosmos DB* billigere å bruke enn *Firestore*. Likevel, kan det være litt for komplekst å sette seg inn i *Azure* med tanke på alle produktene og tjenestene *Azure* har og den omfattende dokumentasjonen. En annen ting er at *Azure Active Directory B2C* krever at man har et *Azure*-abonnement for å bli brukt, og det er i tillegg vanskelig å tilpasse autentiseringen etter ønske. *Firebase* er derfor en god plattform å starte med, spesielt for utvikling av mobilapplikasjoner for *Android* og *iOS*. Det er mindre dokumentasjon å sette seg inn i og dermed kan man komme raskere i gang med utviklingen av produktet, noe som passet godt til dette prosjektet. *Firebase Authentication* er også enkel å tilpasse til applikasjonen og integrere i innloggingssystemet. Til tross for at *Firebase* ikke har like god sikkerhet og personvern som *Azure*, har ikke *programvaresikkerhet* blitt satt som et krav for dette prosjektet. Ettersom *Bæædar* er tilnærmet en *MVP*, ble ikke sikkerheten sett på som noe essensielt. Ved videre utvikling av applikasjonen bør derimot programvaresikkerhet bli tatt i betraktning.

5.2.4 Utfordringer

Underveis i utviklingsprosessen av mobilapplikasjonen *Bæædar* ble det møtt på noen tidkrevende og utfordrende problemer som bremset framgangen i utviklingen. Disse utfordringene involverte blant annet kompatibilitetsproblemer med *React Native*, implementasjonsvansker av *Kartverket* sitt API og begrensning på utviklingen basert på omfanget av prosjektet.

Kompatibiletetsproblemer med React Native

En svakhet ved *React Native* som ble nevnt i Delkapittel 5.2.2 er at det er en del kompatibilitetsproblemer med biblioteker, *plugins* og forskjellige versjoner av *React Native*. Det var flere tilfeller hvor pakker måtte installeres på nytt med andre versjoner enn de nylig oppdaterte versjonene siden *React Native*-versjonen som ble brukt ikke støttet disse pakkene. Det ble satt av en del tid til feilsøking og prøve å finne ut av hvilke versjoner av pakkene som var kompatible med *React Native*, ettersom det ikke nødvendigvis sto i feilmeldingene hvilke versjoner som var kompatible.

Innimellom var det også nødvendig å installere pakker flere ganger på nytt før de kunne brukes i applikasjonen. Til tross for at det kom opp melding om at pakkene hadde blitt installert, var det hendelser hvor *React Native* ikke var i stand til å finne pakkene da de skulle bli brukt.

Implementasjon av interaktivt kart

React Native har en del dokumentasjon på hvordan man kan implementere *Google Maps* og *Apple Maps* i en *React Native*-mobilapplikasjon. Når det kommer til implementasjon av kart fra *Kartverket* derimot eller et eksternt API, er det lite dokumentasjon å finne. Et *Javascript*-bibliotek som hadde blitt anbefalt å bruke for å laste inn kartet fra *Kartverket* var *Leaflet* [42]. *Leaflet* er kjent for å ha god støtte til mobilutvikling og utforming av dynamiske kart. Det var dermed en stor ulempe at dette ikke kunne tas i bruk ettersom det ikke har blitt utviklet støtte av dette kartbiblioteket til *React Native* som beskrevet tidligere.

Som nevnt, ble *OpenLayers* brukt i stedet [63]. På samme måte som *Leaflet* gjør dette biblioteket det mulig å implementere dynamiske kart. Det er derimot ikke samme grad av mobilstøtte som med

Leaflet, spesielt ikke for *React Native*-applikasjoner, slik at alle funksjonaliterer *OpenLayers* har må implementeres som et *HTML*-script for å kunne bruke det med *React Native*. Deretter gjengis dette i *WebView*-komponenten til *React Native* [44]. Denne måten å implementere *OpenLayers*-funksjonalitet på kom med noen utfordringer. Dersom det var en feil i implementasjonen var det vanskelig å oppdage det ettersom det ikke kom opp feilmeldinger på det. Mer om hvordan dette ble gjort finnes i Kapittel 5.2.2.

En av de største utfordringene var å finne en god måte å lagre et valgfritt markert område på kartet på i databasen, spesielt i hvilket format som egnet seg best å lagre kartdataen i. Det skulle være enkelt å hente det og fremstille data i applikasjonen senere. Etter flere runder med undersøkelser og testing på hvilken metode som fungerte best til å lagre kartdata landet valget på *GeoJSON-formatet* [94], som beskrevet i Kapittel 5.2.2. Hvordan det kunne konverteres til et *GeoJSON*-objekt hadde *OpenLayers* eksempler på. Likevel, måtte koden til denne metoden, som nevnt, implementeres i *HTML*-scriptet. Det gjorde det utfordrende å finne ut av hvordan kartdata inni *HTML*-scriptet kunne bli hentet ut og sendt til databasen. Dataen måtte på et vis kunne bli hentet ut og sendt til *WebView*-komponenten, og deretter til databasen. Det var lite å finne på forskjellige nettsteder på hvordan dette kunne gjøres. Etter flere dager med testing og forsøk med forskjellige implementasjoner ble det oppdaget at *React Native* sin *WebView*-komponent hadde en "onMessagefunksjon som tillot kommunikasjon mellom *HTML*-scriptet og komponenten. På denne måten kunne kartdataen sendes fram og tilbake.

Det var i tillegg andre *OpenLayers*-funksjonaliteter som ble testet ut, blant annet *Dragbox* som ble nevnt i Kapittel 5.2.2. Det ble undersøkt om det fantes en mobilversjon av *Dragbox*-funksjonen, men til slutt måtte det bli forkastet. Basert på det som har blitt analysert av *OpenLayers*, ser det ut til at det er lite mobilstøtte i *OpenLayers*. De fleste funksjoner fungerer best på webapplikasjoner, ikke mobilapplikasjoner. Med et stort og komplekst dokumentasjon som *OpenLayers* har, hvor flere av eksemplene var utdaterte, var det tidkrevende å få til implementasjon av interaktivt kart.

Liste med nedlastede kart

For å oppfylle krav **FK-14** som beskrevet i Kapittel 3 skulle hele kolleksjonen med nedlastede kart, *maps*, bli hentet fra databasen og listet opp i applikasjonen slik at brukeren kan velge blant områder som har blitt lastet ned fra før av. *Card* -komponenten til *React Native Paper* ble brukt for å lage en trykkbar liste som inneholdt forskjellige kart [6]. Deretter skal hvert kartdokument fra *maps*-kolleksjonen bli vist når brukeren trykker seg videre på et av "kortene" som blir listet opp. Det var litt utfordrende å finne en god metode for å knytte hvert kort til hvert sitt unike kartdokument. Det som var tenkt innebar å ha en liste med *DisplayMap*-komponenten, som bruker kardata fra databasen til å vise det nedlastede, markerte området på kartet, og lage en *Card*-komponent til hvert visualisert kart på listen. Denne framgangsmåten viste seg å være resultatløs og fungerte ikke optimalt. Foreløpig blir det kun hentet ett kartdokument fra databasen for å vise fram denne funksjonaliteten.

Tidsbegrensning og omfang av prosjekt

Med tanke på at det var begrenset med tid og en leveringsfrist å overholde, var omfanget av prosjektet for stort til å få utviklet et fullverdig produkt. I tillegg ble det møtt på noen tidkrevende utfordringer som nevnt tidligere. Det resulterte i at kun deler av kravspesifikasjonen ble oppfylt og at mobilapplikasjonen ble et *Minimal Viable Product*.

Kapittel 6

Brukertesting

For å teste ut hvor godt funksjonalitetene til mobilapplikasjonen *Bæædar* fungerte, ble det gjennomført brukertesting på medstudenter og romkamerater. Funksjonalitetene som ble testet var *oppretting av bruker*, *nedlasting av kart* og *registrering av informasjon på tilsynstur*. I dette kapittelet blir den valgte testmetoden, gjennomføringen av brukertestene, hendelser og reaksjoner fra brukerne presentert.

6.1 Testmetode

For å brukerteste mobilapplikasjonen ble *Guerilla*-metoden for brukertesting benyttet [2]. *Guerilla*-metoden er en enkel form for brukertesting, hvor man går til et offentlig sted, som for eksempel en kafé, og spør tilfeldige personer om de har lyst til å teste ut prototypen. Det er som oftest en kort brukertest, hvor deltagerne får gratis kaffe i bytte mot å gjennomføre testen. Denne testmetoden er lett å gjennomføre og gir faktiske tilbakemeldinger fra brukere. *Guerilla*-metoden fungerer best i det tidlige stadiet av utviklingsprosessen for å se om applikasjonen beveger seg i riktig retning.

Grunnen til at *Guerilla*-metoden ble benyttet var for å få direkte tilbakemeldinger og respons på mobilapplikasjonen. Med tanke på at *Bæædar* er i et relativt tidlig stadie av utviklingsprosessen var formålet med brukertestene å validere om konseptet i dette prosjektet hadde potensiale og identifisere styrker og svakheter ved funksjonalitetene som hadde blitt implementert til fremtidig videreutvikling.

Mobilapplikasjonen ble testet på 7 deltagere i alderen 22-25 år, hvor 4 var kvinner og 3 var menn. 4 av 7 deltagere hadde en form for tilknytning til landbruk, og varierende teknisk bakgrunn. Disse deltagerne ble valgt ut tilfeldig, og alle som ønsket å teste applikasjonen og hadde tid kunne gjøre det. Varigheten på hver brukertest var rundt 15-20 minutter.

6.2 Gjennomføring av brukertest

Før hver brukertest av en funksjonalitet ble deltageren satt i et scenario og gitt oppgaver han/hun skulle fullføre som sauebonde. Brukerne skulle navigere seg rundt på applikasjonen på egenhånd og ble bedt om å tenke høyt. Først ble *oppretting av ny bruker* testet, deretter *nedlasting av kart* og til sist *registrering av informasjon på tilsynstur*.

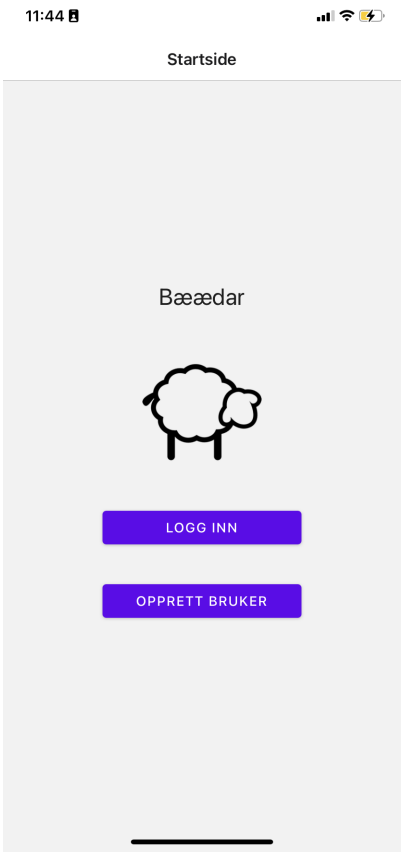
6.2.1 Oppretting av ny bruker

Brukeren ble vist startsiden, hvor han/hun hadde valget mellom å logge inn eller opprette en ny bruker som vist på Figur 6.1. Brukeren ble bedt om å opprette en ny bruker på applikasjonen.

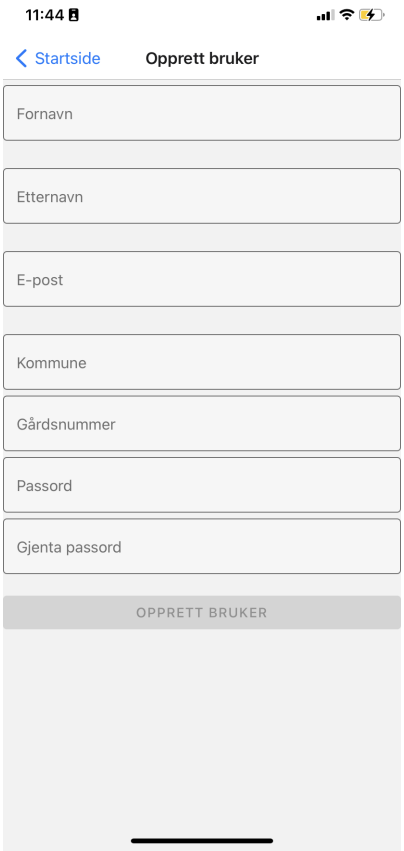
Informasjonen som måtte fylles ut var:

- Fornavn
- Etternavn
- E-postadresse
- Kommune
- Gårdsnummer
- Passord

Oppretingsskjemaet blir vist på Figur 6.2.



Figur 6.1: Startsiden til mobilapplikasjonen *Bæædar*



Figur 6.2: Oppretting av ny bruker på mobilapplikasjonen *Bæædar*

Ettersom flertallet av deltagerne ikke hadde en stor tilknytning til gårdsbruk, var de usikre på hva de skulle fylle ut som gårdsnummer. Dette resulterte i noen tilfeldige numre som ble satt som gårdsnumre. Det var ingen som klarte å lage passord som oppfylte passordkravet, som var på minimum 6 bokstaver/tegn, på første forsøk. Flere av deltagerne måtte spørre om hva som var passordkravet, siden den autogenererte feilmeldingen fra *Firebase Authentication* som dukket opp på skjermen kun varslet om at passordet var for svakt. Det var også noen av deltagerne som reagerte på at passordfeltene var synlige og at den første bokstaven som ble tastet inn var en stor bokstav. For øvrig, gikk prosessen som forventet og var vellykket. Brukerne antok at opprettelsen av bruker var vellykket ettersom de ble sendt til hjemskjermen hvor det stod "Velkommen!", som vist på Figur 6.3.

6.2.2 Nedlasting av kart

Etter at brukeren hadde klart å opprette en ny bruker og fikk tilgang til applikasjonen, ble han/hun tildelt en ny oppgave som skulle løses. Brukeren ble gitt et scenario hvor han/hun var en sauebonde som skulle på tilsynstur. Før tilsynsturen måtte han/hun laste ned et kart over området det skulle turen skulle gå i.

Figur 6.3: Hjemskjerm til mobilapplikasjonen *Bæædar*

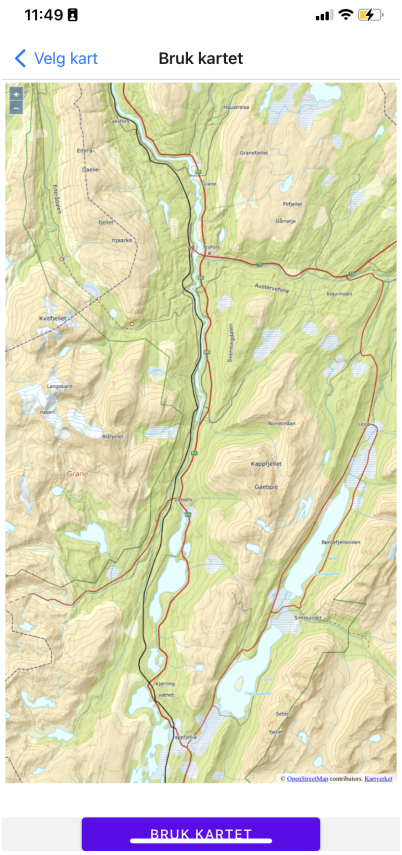
Alle brukerne forstod at man skulle trykke på *Last ned kart* som vist på Figur 6.3 for å laste ned et kart over området for tilsynsturen. Da brukerne kom inn på skjermen ble et norgeskart vist som på Figur 6.4. Flere av brukerne reagerte positivt på at det var mulig og enkelt å zoome inn på kartet med fingrene. Deretter hadde de fleste et forvirret ansiktsuttrykk ettersom de ikke helt forstod hva som skulle skje videre. Under halvparten forstod at han/hun skulle markere av et område på kartet. Av disse hadde flertallet problemer med å treffe den siste kanten av polygonet for at området skulle bli markert. Dersom markeringen av området er vellykket går ruten fra å være blå til rød, noe de fleste ble overrasket over siden det ikke hadde blitt gitt noe indikasjon på at dette kom til å skje. Den andre parten av brukerne begynte å zoome inn på kartet, på det området de ønsket å bruke på tilsynsturen. Deretter prøvde de å trykke på *Beskjær kart*-knappen og forventet at området da skulle bli markert. Da de fikk opp en autogenerert feilemelding fra applikasjonen ble de fleste enda mer forvirret. Etter et par minutter begynte brukerne å trykke på kartet og da forstod de hvordan kartet skulle bli markert. Neste steg var da å gi området et navn og deretter ble brukerne sendt tilbake til hjemskjermen, som vist på Figur 6.5.

Figur 6.4: Nedlasting av kart på mobilapplikasjonen *Bæædar*

Figur 6.5: Nedlastet kart på mobilapplikasjonen *Bæædar*

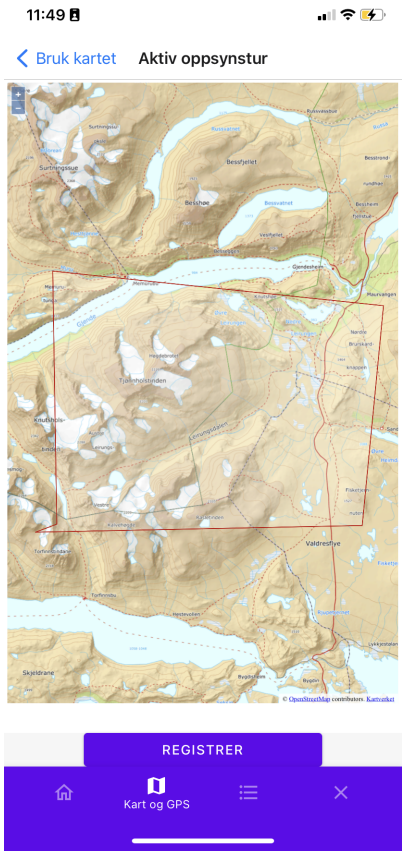
6.2.3 Registrering av informasjon på tilsynstur

Etter at kartet hadde blitt lastet ned, skulle brukerne på tilsynstur. Det neste på agendaen var da å opprette en ny tilsynstur og registrere informasjon på turen. Brukerne fikk litt informasjon om hvordan en tilsynstur foregår og hva slags informasjon som skal registreres basert på avstanden mellom sauebonden og saueflokken. Da brukerne skulle velge kartet som skulle bli brukt på tilsynsturen var kartet zoomet inn på et annet tilfeldig sted enn på det markerte området, som vist på Figur 6.6.



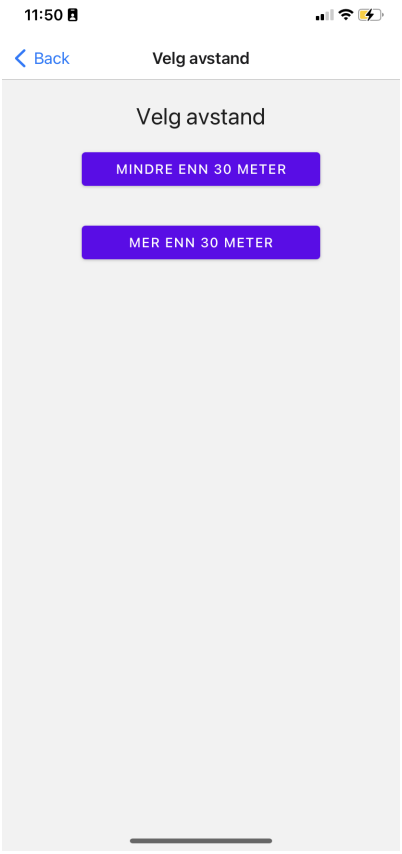
Figur 6.6: Valgt kart til tilsynstur på mobilapplikasjonen *Bæædar*

Det oppstod forvirring blant flere av brukerne og enkelte antok at de måtte markere kartet på nytt. Da brukerne begynte å zoome ut så de omsider at det markerte området befant seg der og trykket seg videre. Da dukket det opp en navigasjonsbar med alternativene *Hjem*, *Kart og GPS*, *Antall registrerte* og *Avslutt tur*. Deretter trykket brukerne seg inn på *Kart og GPS*, fikk opp kartet og en knapp nederst på skjermen hvor det stod *Registrer* som vist på Figur 6.7.



Figur 6.7: *Kart og GPS* på navigasjonsbaren på mobilapplikasjonen *Bæædar*

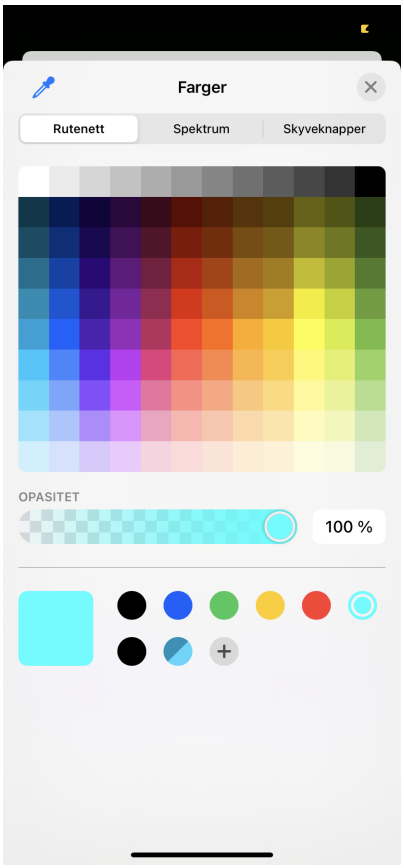
Brukerne ble bedt om å forestille seg at de var på tilsynstur og observerte en saueflokk like ved som de ønsket å registrere informasjonen til. Etter at brukerne trykket på *Registrer* dukket det opp to alternativer: *mindre enn 30 m* og *mer enn 30 m*, som vist på Figur 6.8.



Figur 6.8: Valg av avstand på tilsynstur på mobilapplikasjonen *Bæædar*

Registreringen av antall voksne sau og lam og antall lam var forståelig for alle brukerne. Registrering av antall sau i forskjellige farger skapte forvirring hos mange. Da det grensesnittet dukket opp gikk brukerne inn på fargepaletten for å velge en farge, men deretter var det usikkerhet rundt å trykke på *Lagre*- eller *Ferdig*-knappen som vist på Figur 6.9 og Figur 6.10. Brukerne som trykket på *Ferdig*-knappen ble overrasket da de ble sendt tilbake til *Kart og GPS*-skjermen. De som hadde valgt å registrere informasjon som var *mindre enn 30m* unna forventet å komme seg videre i prosessen og registrere de neste detaljene. De andre brukerne som trykket kun på *Lagre*-knappen ble sendt videre til registrering av antall sauer med forskjellige farger på slips. Dette var misvisende for de som hadde valgt *mer enn 30m* ettersom registreringsprosessen skulle ha blitt avsluttet etter dette.

Figur 6.9: Registrering av farge på sau på tilsynstur på mobilapplikasjonen *Bæædar*



Figur 6.10: Fargepalett til valg av farge på sau på mobilapplikasjonen *Bæædar*

6.3 Konklusjon

Med utgangspunkt i responsen og reaksjoner fra de syv brukertestene er det en del forbedringer og endringer som må iverksettes. Funksjonaliteten for å opprette en ny bruker var veldig intuitiv og lett å forstå seg på, og alle deltagerne klarte å lage seg en bruker i applikasjonen. Funksjonaliteten for nedlasting av kart var derimot lite intuitiv og de fleste av deltagerne hadde problemer med å markere det valgte området. Det burde ha vært mye mer veiledning og informasjon i applikasjonen som forklarer brukeren hvordan markeringen fungerer. Registrering av informasjon på tilsynstur trenger også ytterligere forbedringer, spesielt registreringsprosessen med antall sauer i forskjellige farger og med forskjellige farger på slipset. Fargepaletten burde begrenses til kun de fargene det er mulig at sauene og slipsene kan ha, og måten å registrere antall til de fargene burde bli forandret. I tillegg kan det være ideelt å lage to separate brukerflyt til registrering av informasjon for avstand *mer enn 30m* og *mindre enn 30m*.

Kapittel 7

Resultater

I dette kapittelet blir resultatene fra brukertestene beskrevet i Kapittel 6 lagt frem. Resultatene er basert på situasjoner fra brukertestene og hvordan brukerne opplevde mobilapplikasjonen *Bæædar*.

7.1 Brukertest av oppretting av ny bruker

Kjønn	Alder	Tilknytning til landbruk	Tilbakemeldinger og respons
Mann	22	Har en tante som har en gård	Likte at det kom opp validering av e-post-felt. Skulle ha vært det samme med passord-felt.
Kvinne	21	Ingen tilknytning	Standard skjema for oppretting av ny bruker. Dermed lett å gjenkjenne og vite hvordan prosessen fungerer. Syntes det var deilig at man blir direkte logget inn etter at man har opprettet brukeren.
Kvinne	23	Har besteforeldre som er fra en gård	Veldig lettvint å lage en ny bruker. Siden man blir direkte logget inn etter opprettelsen av bruker og kommer til forsiden, antok hun at opprettelsen gikk bra. Likte godt denne flyten, ettersom man slipper å måtte logge seg på etter at man har lagd en ny bruker samtidig som brukeren får en bekreftelse på at opprettelsen gikk igjennom.
Mann	24	Ingen tilknytning	Likte at det var et standard skjema for opprettelse av ny bruker som de fleste kan gjenkjenne. Passord-feltene burde ikke stå i klartekst. Disse feltene burde være spesifikt av type *passord-felt*.
Mann	24	Ingen tilknytning	God flyt. Et standard skjema som er lett å følge.
Kvinne	24	Kommer fra en gård	Naturlig og logisk. Skjønner at opprettelsen gikk bra siden brukeren blir sendt til forsiden hvor det står "velkommen".
Kvinne	25	Har et søskenbarn som eier et landeområde hvor sauebønder har sauene sine	Veldig intuitivt. Standard skjema, lett å forstå seg på. Passord-feltene burde være av type *passord-felt* og ikke stå i klartekst. I tillegg burde det komme en form for *feedback* mens man skriver inn passordet. En validering på at passordet som blir lagd ikke er "sterkt" nok. Hadde litt problemer med at tastaturet la seg over det ene feltet som skulle fylles ut. Var bra at det ikke var mulig å lage bruker med samme e-post flere ganger.

Tabell 7.1: Brukertest av oppretting av ny bruker

7.2 Brukertest av nedlasting av kart

Tabell 7.2: Brukertest av nedlasting av kart

Kjønn	**Alder**	**Tilknytning til landbruk**	**Tilbakemeldinger og respons**
Mann	22	Har en tante som har en gård	Markering av kart var litt innviklet. Det var vanskelig å treffe punktet på polygonet presist for at området skulle bli markert. Ble også veldig forvirret av feilmeldingen generert av *React Native* som dukket opp. Var bare masse uforståelig kode. Det burde heller vært en feilmelding som var forståelig for brukeren, som for eksempel forklarte brukeren hvordan området skulle markeres. Det gikk ellers greit å laste ned kartet og gi et navn til området.
Kvinne	21	Ingen tilknytning	Burde opplyse brukeren om at området er markert når ruten blir rød. I tillegg burde det være mer informasjon om hvordan markeringen skal gjøres, at man må treffe punktene. Ble forvirret av at inn-zoomingen på markert område viste et tilfeldig sted og lurte på om hun måtte markere på nytt. Likte at det kom opp at man kunne gi et navn til området og at man får en bekreftelse på at området har blitt lastet ned og hvor man blir sendt videre etter dette.
Kvinne	23	Har besteforeldre som er fra en gård	Forstod ikke i starten at man faktisk skulle markere et område på kartet. Trodde at man bare skulle zoome inn på et området og tilpasse det til rammen på skjermen først, trykke på *beskjær*-knappen og deretter ble området "beskjært".
Mann	24	Ingen tilknytning	Markering av området gikk bra. Likte at ruten til det markerte området ble rød som en respons på at det hadde blitt markert.
Mann	24	Ingen tilknytning	Det var vanskelig å treffe det siste punktet på markeringen og han hadde en del problemer med dette. Det burde ha kommet feilmelding til bruker når brukeren gjør feil på kartbeskjæring. Inn-zoomingen var også forvirrende siden det ikke viste det nylig, markerte området, men et tilfeldig område på kartet. Da ble han usikker på om det var nødvendig å markere området på nytt.

Kvinne	24	Kommer fra en gård	Fikk inntrykk av at man skulle zoome inn på kartet til det ble tilpasset rammen til skjermen og deretter trykke på *beskjær*. Forstod etterhvert at man skulle markere et område på kartet, men det burde være noe som indikerer hvordan man skal markere kartet. Hun kom med et forslag om å bytte navnet på knappen til *marker området* istedenfor *beskjær kart*. Det burde heller ikke være tekniske feilmeldinger som kommer opp, men i stedet feilmeldinger som alle brukere kan forstå. Savner funksjonalitet for å kunne søke opp området sitt og deretter zoomer kartet automatisk inn på dette området, istedenfor at man må finne området på egenhånd på kartet. En funksjonalitet lignende *Google Maps*.
Kvinne	25	Har et søskenbarn som eier et landeområde hvor sauebønder har sauene sine	Trodde at man skulle trykke på *beskjær kart* for å markere et område etter at man har zoomet inn på ønsket område. Da dette ikke gikk trodde hun at det var fordi hun hadde prøvd å beskjære et område som ikke var en gård. Det burde komme en informativ feilmelding om hvordan man skal markere på kartet og eventuelt hva man gjør feil. Det burde også gjøres noe med selve markeringsmetoden ettersom det var vanskelig å treffe den siste kanten. Denne prosessen kunne vært mer brukervennlig. Likte derimot at polygonet ble rødt da man hadde klart å markere området, og at man fikk en bekreftelse på at man hadde klart å laste ned kartet etter å ha gitt området et navn.

7.3 Brukertest av registrering av informasjon på tilsynstur

Tabell 7.3: Brukertest av registrering av informasjon på tilsynstur

Kjønn	**Alder**	**Tilknytning til landbruk**	**Tilbakemeldinger og respons**
Mann	22	Har en tante som har en gård	Var lettvint å starte en ny tilsynstur. Registrering av voksne og lam gikk bra. *Ferdig*-knappen var misvisende. Forstod ikke helt om han skulle trykke på *Ferdig* eller *Lagre* mens han registrerte antall sauer i forskjellige farger. Måten for å avslutte tilsynturen på var heller ikke helt intuitiv. Syntes *X*-ikonet på navigeringsbaren virket mer som en utloggingsknapp enn å avslutte tilsynstur. Han begynte å trykke på *Hjem*-knappen på navigeringsbaren i stedet.

Kvinne	21	Ingen tilknytning	Trodde at *Card*-komponenten var trykkbar, men det er bare pilen på komponenten som kan trykkes på. Bortsett fra at inn-zoomingen viste feil sted, likte hun at man kunne se det markerte området på kartet som skulle bli brukt. Trodde først at man måtte markere området på nytt siden kartet var zoomet inn på et tilfeldig sted, men forstod at det ikke var nødvendig da hun zoomet ut og så sitt markerte område. Ble også forvirret av *Lagre*- og *Ferdig*-knappen, trodde at begge knappene hadde samme funksjon. I tillegg var registrering av antall sauer i forskjellige farger litt vanskelig å forstå. Opplevde *X*-ikonet på navigeringsbaren som veldig intuitiv og forstod at man skulle trykke på den for å avslutte turen. Derimot var *Hjem* på navigeringsbaren misvisende. *Hjem* skal i utgangspunktet vise statistikken for siste tilsynstur på det samme området, men hun trodde at knappen skulle sende brukeren tilbake til forsiden.
Kvinne	23	Har besteforeldre som er fra en gård	Trodde at *Card*-komponenten var trykkbar, men det er bare pilen på komponenten som kan trykkes på. Var litt usikker på om man måtte markere området på nytt siden kartet var zoomet inn på et tilfeldig sted. Kom med tilbakemelding om at inn-zoomingen må på plass for å unngå forvirring. Syntes *Lagre* og *Ferdig* var misvisende på registrering av antall sauer i forskjellige farger. Forstod ikke hva som var forskjellen mellom de to funksjonene. Forstod heller ikke hvordan registreringen fungerte, at man skulle trykke på *Lagre* etter hver registrering av en farge og deretter velge en ny farge. Foreslo at de tre fargene (svart, brun og hvit) kunne ha vært der fra før av og blitt vist, og fargepaletten skulle ha blitt begrenset til valg mellom disse tre fargene. Dette kunne også blitt gjort for registrering av antall sauer med forskjellige farger av slips. Likte veldig godt brukergrensesnittet med å trykke på +/–knappene for å telle, og størrelsen på knappene gjorde at det var enkelt å trykke på dem.

Mann	24	Ingen tilknytning	Tastatutet forsvant ikke da han trykket på *DatePicker*-komponentene og skulle velge dato og tidspunkt for tilsynsturen. Inn-zoomingen på kart burde det rettes opp i. *Registrer*-knappen på kartet var heller ikke veldig intuitiv. Forstod hvordan registreringsprosessen av antall sauer i forskjellige farger foregikk, men syntes *Ferdig*-knappen var misvisende og forvirrende. Forstod ikke denne knappens funksjon og hensikt. Den knappen burde ha gitt en slags *feedback*, en *popup* som spør brukeren om han/hun er helt sikker på at registreringen er ferdig og informasjon om hvor brukeren blir sendt videre etter det. Måten å avslutte tilsynsturen på var veldig intuitiv, forstod med en gang at man skulle trykke på *X*-ikonet på navigeringsbaren. Han tenkte "Kryss ut, få det vekk" da han så på det ikonet. Likte veldig godt brukergrensesnittet, spesielt +/--knappene for å telle. Var bra at de var store og lette å trykke på, i tilfelle sauebonden får våte fingre på tur.
Mann	24	Ingen tilknytning	Trodde at *Card*-komponenten var trykkbar, men det er bare pilen på komponenten som kan trykkes på. Syntes *Lagre* og *Ferdig* var misvisende på registrering av antall sauer i forskjellige farger. Forstod ikke hva som var forskjellen mellom de to funksjonene. Registreringen av antall sauer i forskjellige farger gikk bra, og syntes det meste virket intuitivt. Syntes at knappene var enkle å trykke på. Det burde være et sammendrag av hva som har blitt registrert når man er ferdig med registreringsprosessen på det spesifikke området og en form for *feedback* når man ønsker å avslutte registreringen på det området.
Kvinne	24	Kommer fra en gård	Trodde at man kunne trykke på selve grafikken til statistikken på *Hjem*-navigeringsskjermen. Syntes *Lagre* og *Ferdig* var misvisende på registrering av antall sauer i forskjellige farger. Registreringsprosessen av antall sauer i forskjellige farger var også litt innviklet. Burde ha stått "registrer sauer og velg farge" på den skjermen for å indikere at sauer i spesifikke farger skulle registreres. Fargepaletten burde også begrenses til kun de fargene det er mulig å registrere sauer og slips på sauene i.

Kvinne	25	Har et søskenbarn som eier et landeområde hvor sauebønder har sauene sine	Trodde at *Card*-komponenten var trykkbar, men det er bare pilen på komponenten som kan trykkes på. Syntes *Lagre*-knappen på noen av registreringsskjermene var plassert litt for høyt oppe. *Lagre* og *Ferdig* var misvisende på registrering av antall sauer i forskjellige farger. Det burde være et sammendrag av hva som har blitt registrert når man er ferdig med å registrere på det spesifikke området og en form for *feedback* når man skal avslutte registreringen på det området. Det burde komme en *popup* som for eksempel sier "Er du sikker?" når man trykker på *Ferdig* sånn at man ikke avbryter flyten ved feiltrykk. Generelt sett, burde det være mer veiledning og hjelp å få under registreringsprosessen. Mer respons og informasjon om hva som skal skje videre.

Kapittel 8

Diskusjon

I dette kapittelet blir resultatet av prosjektet og brukertestene diskutert og evaluert med utgangspunkt i problemstilling, mål og kravspesifikasjon. Ettersom det kun er mobilapplikasjonen *Bædar* som har blitt utviklet og nettsiden fortsatt ikke er påbegynt, vil det kun bli tatt utgangspunkt i mobilapplikasjonen i diskusjonen og evalueringen.

8.1 Evaluering av brukertest

Etter gitt råd og veiledning av Professor *Hvasshovd* ble det utført brukertester på medstudenter og romkamerater. Mobilapplikasjonen ble brukertestet på 4 kvinner og 3 menn i alderen 22-25 år, hvor 4 av 7 deltagere hadde en slags tilknytning til landbruk og gård. Testmetoden som ble brukt var *Guerilla*-metoden for brukertesting [2]. Deltagerne ble valgt tilfeldig, alle som hadde tid og lyst til å gjennomføre en brukertest kunne gjøre det. Varigheten på hver brukertest var på 15 til 20 minutter. Etter hver brukertest fikk deltagerne en belønning i form av godteri.

Guerilla-metoden fungerer best i det tidlige stadie i utviklingsfasen, noe *Bædar* befinner seg i og av den grunn ble denne metoden valgt. Den blir brukt til å samle opp personlige meninger og inntrykk av ideer og konsepter, og for å vite om applikasjonen beveger seg i riktig retning. I tillegg får man se hvordan brukerne interagerer med applikasjonen og deres reaksjoner. Alle brukertestene ble observert og dokumentert. Disse testene ga et godt innblikk i styrker og svakheter ved den nåværende applikasjonen og hva som kan forbedres og implementeres med tanke på funksjonalitet og brukervennlighet. Deltagerne tenkte også høyt og ga gode tilbakemeldinger på applikasjonen som kan brukes til videre utvikling.

Ulemper ved *Guerilla*-metoden er at varigheten på brukertestene er som regel kort. Man er begrenset til en liten del av brukerflyten og får ikke testet applikasjonen like mye i dybden som en formell brukertest [43]. Dermed burde denne metoden antakelig ikke bli brukt i fremtidig utvikling av applikasjonen når applikasjonen er forbi dette tidlige stadiet. En formell brukertest vil gi bedre innsikt og dypere analyse av applikasjonen, og kan ikke erstattes av denne metoden. Videre, ble ikke mobilapplikasjonen testet på de reelle brukerne som faktisk skal bruke systemet, det vil si sauebønder. Av den grunn kan det hende at den responsen og de tilbakemeldingene som har blitt gitt ikke samsvarer med det en faktisk sauebonde ville ha gitt.

Ettersom det kun var deler av funksjonaliteten til applikasjonen som hadde blitt implementert, ble det testet:

- Oppretting av ny bruker
- Nedlasting av kart
- Oppretting og registrering av tilsynstur

De ovennevnte funksjonalitetene ble prioritert å implementere og teste ettersom de ble sett på som de viktigste i henhold til oppgavebeskrivelsen i Kapittel 1.2. Det er essensielt å kunne opprette en bruker for å få tilgang til applikasjonen. Videre, var en sterkt ønsket funksjonalitet det å kunne laste ned et markert området av et kart som skal brukes på tilsynstur. Til sist, er oppretting av tilsynstur og registreringsmetoden av sauer det som har størst betydning for å dekke behovet til sauebøndene. Det er kjerneflyten, essensen med hele applikasjonen. Deltagerne av brukertestene ble gitt oppgaver de skulle fullføre og ble forklart hvordan en tilsynstur blir gjennomført av sauebønder i praksis på forhånd. En oppgave kunne være: "Du er en sauebonde som ønsker å laste ned et kart over området du skal på tilsynstur i". Da skulle deltageren på egenhånd navigere seg rundt på applikasjonen og finne ut av hvordan oppgaven skulle fullføres.

Til tross for at det ble gjennomført syv brukertester på personer med varierende teknisk bakgrunn, og noen med tilknytning til gårdsbruk, hadde det mest optimale vært å utføre brukertester på sauebønder. Ettersom formålet med applikasjonen er å dekke sauebøndenes behov, og at det er de som er målgruppen, hadde brukertesting på sauebønder gitt mer innsiktsfulle og verdifulle tilbakemeldinger på applikasjonen. Selv om deltagerne av bruketestene ble bedt om å forestille seg at de var sauebønder og ble gitt bakgrunnsinformasjon om gjennomføring av tilsynstur, er ikke det sammenlignbart med sauebønder som har ansvar for å følge opp sauene hver beitesesong. Uten innspill fra målbrukergruppen av applikasjonen, er det usikkert om applikasjonen blir utviklet i riktig retning.

8.2 Evaluering av endelig løsning

I dette kapitlet skal resultatet bli analysert og vurdert opp mot kravspesifikasjonen. For å evaluere resultatet av applikasjonen blir det tatt utgangspunkt i interaksjoner og tilbakemeldinger fra brukertestene.

8.2.1 Evaluering av Bæædar

Mobilapplikasjonen *Bæædar* er først og fremst et *Minimal Viable Product*. Det ble prioritert å implementere den nødvendige funksjonaliteten ved applikasjonen for å teste ut potensialet til konseptet av en slik applikasjon. Ettersom en sammenligningstest av *Bæædar* og eksisterende innsamlingsmetoder, som penn og papir, ikke var mulig å gjennomføre ble i stedet hver bruker fortalt hvordan informasjonsinnsamlingen foregår i dag og stilt spørsmål om deres tanker rundt *Bæædar* som en potensielt bedre innsamlingsmetode for sauebønder. Tilbakemeldinger som var felles hos de fleste brukerne var at konseptet om å lage en slik applikasjon som kan erstatte den eksisterende innsamlingsmetoden hadde høyt potensialet. Applikasjonen var oversiktlig og enkel å bruke, og det er en bedre måte å loggføre informasjonen som blir registrert på en applikasjon enn med for eksempel penn og papir. Det er lettere å miste oversikt på hvilken informasjon som har blitt registrert på papir. Likevel, vil ikke den nåværende tilstanden av applikasjonen kunne dekke sauebøndenes behov. Det mangler fortsatt en del funksjonalitet som må implementeres og feilrettinger av funksjoner som ikke fungerte som forventet. Med ytterligere forbedring av nevnte punkter vil applikasjonen kunne erstatte nåværende innsamlingsmetoder.

8.2.2 Avvik fra Figma-skisser

Ikke alle brukergrensesnittene er implementert i henhold til *Figma-skissene* som ble presentert i Kapittel 5.1. For det første, ble ikke avstanden mellom sauebonde og saueflokk automatisk beregnet som det var planlagt for i begynnelsen. Med tanke på tidsbegrensning og arbeidsmengde var denne funksjonen for omfattende å oppnå. I stedet ble det implementert et grensesnitt for at brukeren kunne velge avstanden selv som vist på Figur 6.8. Funksjonaliteten for at avstanden kan autoberegnes er derimot noe som muligens kan bli utviklet senere hvis det er ønskelig.

Videre, ble ikke fargepaletten til registrering av farge på sau og slips utviklet i henhold til *Figma-*

prototypen. Som det blir beskrevet nærmere i Kapittel 8.2.3 under, opplevde flere av deltagerne av brukertestene at et komplett fargepalett med alle farger ble litt omfattende. Brukerne ønsket heller at fargevalget skulle bli begrenset til de fargene det var mulig å registrere, slik som på skissene i Kapittel 5.1. Med en tidsfrist å overholde ble det heller besluttet å implementere en ferdiglagd fargekomponent fra [39] for å kunne demonstrere registrering og fargevalg. Fargekomponenten blir vist på Figur 6.10. Til fremtidig arbeid er derimot en begrenset fargepalett mer brukervennlig å ha.

8.2.3 Validering av funksjonelle krav

I dette delkapittelet vil de funksjonelle kravene som er relevante til de ferdigimplementerte funksjonene bli evaluert. Dette blir vist på Tabell 8.1.

Tabell 8.1: Validering av funksjonelle krav for mobilapplikasjon

FK-ID	**Beskrivelse**	**Kommentarer**
FK-01	Sauebonden skal kunne se et kart over hele Norge	Deltagerne syntes at brukergrensesnittet av kartet og måten det ble framstilt på var fint.
FK-02	Sauebonden skal kunne klippe ut en bit av norgeskartet og laste det ned på applikasjonen	Markering av kart var litt innviklet. Ikke alle brukerne forstod at man måtte trykke på kartet og lage et polygon for å markere et område. Det var enkelte brukere som trodde at man bare trengte å zoome inn på ønsket område på kartet og deretter trykke på *beskjær* for å markere området. I tillegg måtte brukeren treffe kanthjørnet av polygonet presist for at området skulle bli markert, noe som var litt utfordrende. Alle brukerne likte at man kunne gi navn til det valgte området, og at det kom en bekreftelse på at kartet hadde blitt lastet ned.
FK-06	Sauebonden skal kunne registrere antall voksne og lam han/hun observerer	Dette gikk som forventet. Grensesnittet var intuitivt og det var lett å forstå knappene brukeren måtte trykke på for å telle sauene som ble observert.
FK-07	Sauebonden skal kunne registrere antall lam han/hun observerer	Dette gikk som forventet. Grensesnittet var intuitivt og det var lett å forstå knappene brukeren måtte trykke på for å telle lam som ble observert.
FK-08	Sauebonden skal kunne registrere antall sau i ulike farger han/hun observerer	*Lagre*-knappen og *Ferdig*-knappen var forvirrende. Å ha et helt fargepalett med alle farger ble litt for stort utvalg å ha. Ikke alle brukerne forstod hvordan denne funksjonen fungerte og at man måtte trykke på *Lagre* etter registrering av antall sauer til en spesifikk farge før man skulle registrere sauer til neste farge.
FK-09	Sauebonden skal kunne registrere antall sau etter fargen på slipset sauene han/hun observerer	Å ha et helt fargepalett med alle farger ble et litt for stort utvalg å ha. Ikke alle brukerne forstod hvordan denne funksjonen fungerte og at man måtte trykke på *Lagre* etter registrering av antall sauer til en spesifikk farge før man skulle registrere sauer til neste farge.
FK-10	Sauebonden skal kunne registrere hvilke farger sauene har på øremerkene sine	Dette gikk som forventet. Grensesnittet var intuitivt og brukerne skrev inn fargene på øremerkene. Det var også mulig å legge til flere skrivefelter.
FK-11	Sauebonden skal kunne zoome inn på den delen av kartet han/hun vil klippe ut	Det var lett å zoome inn på kartet. Det var mulig for brukeren å zoome inn med fingrene, noe alle deltagerne likte veldig godt.

FK-12	Sauebonden skal kunne se hvilke områder han/hun har lastet ned fra før av	Det var mulig å se det markerte området som hadde blitt lastet ned. Derimot var kartet zoomet inn på feil sted, slik at brukerne måtte zoome ut for å finne det markerte området og deretter zoome inn igjen på det.
FK-13	Sauebonden skal kunne velge blant områder som har blitt lastet ned fra før av	Da listen med nedlastede kart dukket opp trodde de fleste brukerne at man kunne trykke på *kortet* for å gå inn på det valgte kartet, men de skjønte ganske raskt at det var *pil*-ikonet til høyre på kortet man måtte trykke på for å gå videre. Siden inn-zoomingen på det markerte området ikke fungerer som det skal og viser et tilfeldig sted på kartet i stedet, var det enkelte som først trodde at man måtte markere området på nytt.
FK-14	Sauebonden skal kunne velge dato og tidspunkt for tilsynsturen	Dette var ganske intuitivt og brukerne forstod at man skulle trykke på *kalender*-ikonet for å velge dato og tidspunkt. De likte godt at dato og tidspunkt også ble satt automatisk til daværende dato og tidspunkt av brukertesten. Hos noen av brukerne dukket tastaturet på mobilen opp samtidig som datovelgeren, slik at datovelgeren ble delvis dekket til.
FK-16	Sauebonden skal kunne logge inn på applikasjonen	Grensesnittet og flyten for oppretting av bruker gikk som forventet og var veldig intuitivt. Det var et standard skjema for utfylling av brukerinfo som var enkel å følge. Passordfeltene skulle ha vært skjult og det skulle ha vært informasjon om hva slags krav det var til passord. E-postadressen til brukere ble også validert, slik at det ikke var mulig å opprette en bruker som eksisterte fra før av.
FK-17	Sauebonden skal kunne registrere forskjellige detaljer som han/hun observerer basert på avstanden til saueflokken	Det var to forskjellige valg brukerne fikk da de skulle registrere: *mindre enn 30m* og *mer enn 30m*. Disse knappene var lette å forstå. *Lagre*-knappen i **FK-09** skulle sende brukeren videre i registreringsprosessen dersom avstanden var mindre enn 30m. *Ferdig*-knappen i **FK-09** skulle avslutte registreringen etter at antall sauer i forskjellige farger hadde blitt registrert dersom avstanden var mer enn 30m. Derimot oppstod det forvirring rundt disse knappene og de fleste brukerne endte opp med å trykke på *Ferdig*-knappen hver gang.

FK-18	Sauebonden skal kunne registrere antall skadde sauer han/hun observerer	Dette gikk som forventet. Grensesnittet var intuitivt og det var lett å forstå knappene brukeren måtte trykke på for å telle antall skadde sauer som ble observert.
FK-19	Sauebonden skal kunne registrere antall døde sauer han/hun observerer	Dette gikk som forventet. Grensesnittet var intuitivt og det var lett å forstå knappene brukeren måtte trykke på for å telle antall døde sauer som ble observert.
FK-22	Sauebonden skal kunne avslutte tilsynsturen når han/hun er ferdig	De fleste brukerne forstod at man skulle trykke på *krysset* på navigeringsbaren for å avslutte tur. Brukerne med mindre teknisk bakgrunn syntes ikke det var like intuitivt å skulle trykke på *krysset* for å avslutte turen. Enkelte trodde at man skulle trykke på *Hjem*-ikonet på navigeringsbaren for å avslutte turen.
FK-23	Sauebonden skal kunne opprette en bruker på applikasjonen	Grensesnittet og flyten for oppretting av bruker gikk som forventet og var veldig intuitivt. Det var et standard skjema for utfylling av brukerinfo som var enkel å følge. Passordfeltene skulle ha vært skjult og det skulle ha vært informasjon om hva slags krav det var til passord.

Tilbakemeldinger og forslag til videreutvikling

FK-02: Sauebonden skal kunne klippe ut en bit av norgeskartet og laste det ned på applikasjonen
En av grunnene til at dette kravet ikke ble fullstendig tilfredsstilt kan være at det var ingen indikasjon på brukergrensesnittet om hvordan kartet skulle markeres. Det var ikke en "boks" som var på skjermen fra før av som ga en form for antydning til markeringsmåten på kartet. Videre, kom det ikke opp et punkt på kartet før man trykket på skjermen. Dermed var det vanskelig for brukerne å vite hvordan et område skulle markeres. Andre antagelser til at denne funksjonen ikke fungerte som forventet kan være at det var misvisende at det sto *beskjær kart* på knappen enn *marker kart*. Det var derfor ikke overraskende at enkelte brukere antok at de skulle zoome inn på ønsket område og da *beskjære* kartet ved å trykke på knappen. Det bør tas i betraktning hvor vill-ledende en knapp kan være med feil tekst.

Tilbakemeldinger fra brukertestene var:

- **Beskrivelse av hvordan man skal markere på kartet:** Det burde stå informasjon om at man må trykke på kartet og lage punkter for å tegne polygonet, spesielt at man må treffe det siste kanthjørnet presist for at området skal bli markert. I tillegg burde det stå at området blir rødt dersom markeringen var vellykket. Generelt burde det være tilstrekkelig mer veiledning.
- **Informativ feilemelding ved feilmarkering:** Foreløpig kommer det opp en feilmelding generert av systemet dersom man har markert feil og prøver å laste ned kartet. Denne feilmeldingen inneholder bare masse teknisk kode som er uforståelig for brukeren. Det burde derfor komme opp en informativ feilemelding til brukeren hvor det står hva som har blitt gjort feil. I tillegg burde være mer respons på brukerinteraksjon med mobilapplikasjonen slik at brukeren får vite hva som skjer videre.
- **Rette opp i inn-zoomingen på markert område:** Etter at brukeren har markert ønsket område, skal man kunne trykke seg videre for å se dette området før man laster det ned.

Denne funksjonen bør forbedres slik at det blir zoomet inn på riktig sted. Imidlertid, skaper denne funskjonen forvirring hos brukerne.

- **Bytte tekst på beskjæringsknappen:** Å ha teksten *Beskjær kart* på knappen var forvirrende for flere av brukerne. Det fikk brukerne til å tro at de kunne markere på kartet ved å kun zoome inn på området. Et forslag fra en av brukerne var å bytte teksten på knappen til *Marker området*.

FK-08: Sauebonden skal kunne registrere antall sau i ulike farger han/hun observerer / FK-09: Sauebonden skal kunne registrere antall sau etter fargen på slipset sauene han/hun observerer

Metoden for å registrere farger på sau viste seg å være lite intuitiv. Det kan komme av at det i denne situasjonen følgelig var tilnærmet ingen veiledning på hvordan fargene skulle registreres. Uten forklaring på hvordan denne prosessen skulle foregå, og hva hensikten med *Lagre*- og *Ferdig*-knappen var, var det vanskelig å forstå seg på funksjonaliteten. Det var ingen form for indikasjon på hva slags funksjon hver av de knappene hadde og hva som var forskjellen mellom dem. Teksten på de knappene har i tillegg omtrent samme ordbetydning. Dermed kan det lett oppstå forvirring rundt hvilken knapp man skal trykke på etter å ha valgt farge.

Tilbakemeldinger til funksjonen med fargevalg var:

- **Begrense fargevalget til kun 3 farger ved registrering av farge på sau:** Med tanke på at det bare finnes brune, svarte og hvite/gråe sauer, burde valget bli begrenset til disse tre fargene som vist i *Figma*-skissene i Kapittel 5.1. Grunnen til at det ble valgt et helt fargepalett var at det var for begrenset med tid til å utvikle fargekomponenten som vist på skissene. Derfor ble det prioritert å implementere en ferdiglagd fargekomponent i stedet for å kunne demonstrere prosessen med fargevalg.
- **Begrense fargevalget til kun 4 farger ved registrering av farge på slips på sau:** Med tanke på at fargene slipsene på sauene kan ha er rødt, blått, gult og grønt burde valget bli begrenset til disse fargene som vist på *Figma*-skissene i Kapittel 5.1. Til denne funksjonaliteten ble det følgelig brukt en ferdiglagd fargekomponent som beskrevet i punktet over.
- **Informativ tekst som beskriver hvordan man velger farge og registrerer:** Et forslag fra en av brukerne var at overskriften på denne funksjonen burde være *Velg farge og registrer antall sauer* eller lignende. I tillegg burde det være en beskrivende tekst som forklarer at man må velge farge først, registrere antall sauer også trykke på *Lagre*-knappen før man går videre til neste farge.

FK-16: Sauebonden skal kunne logge inn på applikasjonen
En ting enkelte brukere med teknisk bakgrunn påpekte var at passordfeltene ikke var av typen *passordfelt*, men at de var *string* som ikke er sikkert i det hele tatt. Dermed bør passordfeltene endre type på frontend-siden slik at feltene er skjult når brukeren skriver inn passord. I tillegg burde det være informasjon om hva slags krav passordet har. I denne applikasjonen måtte for eksempel passordet være på minst 6 tegn, men det sto ingenting om dette kravet. Det kom bare opp en feilmelding som var generert av *Firebase Authentication* om at passordet var svakt.

FK-17: Sauebonden skal kunne registrere forskjellige detaljer som han/hun observerer basert på avstanden til saueflokken
Det mest optimale hadde vært å ha to forskjellige flyt. Når man trykker på *mindre enn 30m*, burde alle detaljene som skal registreres komme opp i rekkefølge og fortsette helt frem til detaljene har blitt gjennomgått, ikke at det skal være en *Lagre*- og *Ferdig*-knapp som dukker opp for at brukeren skal ta et valg om man skal fortsette på registreringen eller ikke. Dette var, som nevnt tidligere, svært forvirrende og lite brukervennlig. Når man trykker på *mer enn 30m* burde brukeren bli sendt til en annen flyt. Da skal registreringen stoppe opp etter at registrering av farger på sauene er ferdig. Det burde da kun være en *Ferdig*-knapp. En annen ting som alle brukerene ønsket var at etter at man var ferdig med å registrere observasjoner rundt et spesifikt område burde det dukke

opp et sammendrag av hva som har blitt registrert i det området slik at man får oversikt. Deretter burde det komme opp et valg til, i form av for eksempel en *popup* som spør brukeren om han/hun er helt sikker på at han/hun er ferdig med registreringen av det området.

Legge til profilside
Det var et ønske om å ha en profilside på mobilapplikasjonen slik at man kunne se den informasjonen som var lagt til på brukeren. I tilfelle brukeren registrerer feil adresse eller flytter er det praktisk om det er mulig å endre på denne informasjonen direkte i applikasjonen.

8.2.4 Validering av ikke-funksjonelle krav

Det er kun de relevante kravene som ble testet under brukertestingen som blir evaluert. Det blir vist på Tabell 8.2.

NFK-ID	Beskrivelse	Kommentarer
NFK-03	Mobilapplikasjonen skal være intuitiv nok til at det tar mindre enn 10 minutter på å lære seg hvordan man bruker det	Hver brukertest varte i gjennomsnitt 15 minutter. Enkelte deltagere brukte mer enn 15 minutter ettersom nedlasting av kart ikke var like intuitiv som forventet.
NFK-04	Registrering av observasjoner skal ikke ta mer enn 3 minutter	De fleste deltagerne ga tilbakemeldinger på at flyten var veldig god, og at det var rask responstid på brukergrensesnittet. I tillegg var grensesnittet for det meste intuitivt og lett å forstå. Som nevnt på **FK-08**, **FK-09** og **FK-17** fungerte ikke registreringsmetoden av fargevalg helt optimalt. Det var mindre intuitivt og *Lagre*- og *Ferdig*-knappene skapte bare forvirring.

Tabell 8.2: Validering av ikke-funksjonelle krav for mobilapplikasjon

Ved å gjøre de nødvendige endringene og forbedringene av de funksjonelle kravene til brukergrensesnittet kan **NFK-03** og **NFK-04** bli oppfylt.

8.3 Evaluering av problemstilling og mål

Det overordnede målet for prosjektet er, som beskrevet i Kapittel 1.2.1, å utvikle et konsept som kan digitalisere og effektivisere oppfølging av sau på utmarksbeite under beitesesongen. For å evaluere mobilapplikasjonen opp mot målet vil applikasjonen bli vurdert etter de tre forskningsspørsmålene i Kapittel 1.2.2, med utgangspunkt i brukertestene og kravspesifikasjonen.

8.3.1 Evaluering av problemstilling og forskningsspørsmål

Ettersom mobilapplikasjonen ikke har blitt testet på en sauebonde vil besvarelsen på forskningsspørsmålene være basert på tilbakemeldinger fra brukertestene og hvorvidt kravene som er satt har blitt oppnådd.

F1: Hvor godt dekker applikasjonen *Bæædar* sauebøndenes behov ved oppfølging av sau på utmarksbeite?
Med tanke på at mobilapplikasjonen er en *MVP* og fortsatt er i et tidlig stadie i utviklingsfasen er det en del funksjonalitet som gjenstår. Som beskrevet i problemstillingen for dette prosjektet i Kapittel 1.2.2 er det et ønske om å flytte registreringen av observasjoner på papir over til et

databasert verktøy. Innholdet i rapporten og kartet over hvor bonden har gått på tilsynsturen blir deretter overført til nettsiden, hvor en standardisert rapport produseres ved sesongens slutt og sendes til myndighetene. Hvilke funksjoner dette systemet skulle ha er beskrevet i Kapittel 3. Status på kravene til applikasjonen er vist på Tabell 8.3.

FK-ID	**Beskrivelse**	**Prioritet**	**Status**
FK-01	Sauebonden skal kunne se et kart over hele Norge	Høy	Ferdig
FK-02	Sauebonden skal kunne klippe ut en bit av norgeskartet og laste det ned på applikasjonen	Høy	Ferdig
FK-03	Sauebonden skal kunne se sin egen posisjon på kartet	Høy	Ikke startet
FK-04	Sauebonden skal kunne merke av på kartet hvor han/hun har sett saueflokken	Middels	Ikke startet
FK-05	Sauebonden skal kunne se streker på kartet som viser hvor han/hun har gått	Høy	Ikke startet
FK-06	Sauebonden skal kunne registrere antall voksne og lam han/hun observerer	Høy	Ferdig
FK-07	Sauebonden skal kunne registrere antall lam han/hun observerer	Høy	Ferdig
FK-08	Sauebonden skal kunne registrere antall sau i ulike farger han/hun observerer	Høy	Ferdig
FK-09	Sauebonden skal kunne registrere antall sau etter fargen på slipset sauene han/hun observerer	Høy	Ferdig
FK-10	Sauebonden skal kunne registrere hvilke farger sauene har på øremerkene sine	Høy	Ferdig
FK-11	Sauebonden skal kunne zoome inn på den delen av kartet han/hun vil klippe ut	Middels	Ferdig
FK-12	Sauebonden skal kunne se hvilke områder han/hun har lastet ned fra før av	Høy	Pågår
FK-13	Sauebonden skal kunne velge blant områder som har blitt lastet ned fra før av	Høy	Pågår
FK-14	Sauebonden skal kunne velge dato og tidspunkt for tilsynsturen	Middels	Ferdig
FK-15	Sauebonden skal kunne se sine tidligere tilsynsturer	Middels	Ikke startet
FK-16	Sauebonden skal kunne logge inn på applikasjonen	Høy	Ferdig
FK-17	Sauebonden skal kunne registrere forskjellige detaljer som han/hun observerer basert på avstanden til saueflokken	Høy	Ferdig
FK-18	Sauebonden skal kunne registrere antall skadde sauer han/hun observerer	Høy	Ferdig
FK-19	Sauebonden skal kunne registrere antall døde sauer han/hun observerer	Høy	Ferdig
FK-20	Sauebonden skal kunne ta bilde av skadde sauer og døde sauer han/hun observerer	Lav	Ikke startet
FK-21	Sauebonden skal kunne se en oversikt over hva som har blitt registrert underveis på tilsynsturen	Middels	Ikke startet
FK-22	Sauebonden skal kunne avslutte tilsynsturen når han/hun er ferdig	Middels	Ferdig
FK-23	Sauebonden skal kunne opprette en bruker på applikasjonen	Høy	Ferdig

Tabell 8.3: Status på funksjonelle krav for mobilapplikasjon

I henhold til status på kravene som er satt for at mobilapplikasjonen *Bæædar* skal bli et velfungerende og nyttig produkt for sauebønder, dekker ikke applikasjonen deres behov foreløpig. For det første, har ikke **FK-03** og **FK-05** med **høy** prioritet blitt implementert. Som følge av dette er ikke mobilapplikasjonen i stand til å spore og loggføre lokasjonen til en sauebonde på tilsynstur. Videre, eksisterer ikke den tilhørende nettsiden til applikasjonen per nå og derfor er det ikke mulig

å produsere en standardisert rapport ved sesongens slutt.

For det andre, med utgangspunkt i brukertestene som ble gjennomført, var det en felles enighet blant deltagerne om at applikasjonen sånn som den er nå ikke vil tilfredstille sauebøndenes behov. De implementerte funksjonalitetene fungerer ikke optimalt og det er en del forbedringer og endringer som må iverksettes som beskrevet over i Kapittel 8.2 før mobilapplikasjonen er et produkt som dekker bøndenes behov.

F2: På hvilke måter er det bedre å bruke *Bæædar* til manuell oppfølging av sau på utmarksbeite sammenlignet med eksisterende innsamlingsmetode?
Dagens innsamlingsmetode av informasjon fra tilsynsturer er i følge Professor *Hvasshovd* med penn og papir. Da deltagerne av brukertestene ble stilt spørsmålet om en mobilapplikasjon som *Bæædar* kunne være en bedre måte å registrere observasjoner på svarte alle ja. De ga uttrykk for at gitt at applikasjonen i det minste har alle funksjonene som dagens metode har, og at de nødvendige forbedringene og endringene som har blitt nevnt i Kapittel 8.2 blir gjennomført, vil den kunne erstatte dagens innsamlingsmetode. Enkelte deltagere påstod at det er lett å miste oversikt over hva som har blitt registrert med papir, og at det da var en bedre løsning at alt er loggført på en applikasjon. I følge Professor *Hvasshovd* virker det som en mer tungvint prosess å skulle notere ned detaljer for hånd samtidig som man observerer. I tilegg var det fordelaktig at registreringsprosessen hadde en struktur og fulgte en bestemt rekkefølge, slik at observasjoner ikke ble registrert på en tilfeldig og uoversiktlig måte. Brukergrensesnittet var veldig intuitivt og hadde god flyt, og det var lettvint å telle og registrere sauer, ifølge flere av brukerne.

Markering av området på kart, konseptet om å kunne se sin egen posisjon og kunne spore og loggføre hvor man går på tilsynsturen vil også forbedre rapporteringsprosessen betraktelig. Noen av brukerne nevnte at en nettside hvor man kan se informasjonen fra tilsynsturene senere kunne være veldig nyttig, selv før det ble nevnt at det er lagt en plan for det i videre arbeid. Hvis alle funksjonene blir satt på plass, vil dette systemet potensielt ha stor nytteverdi for sauebønder. Følgelig bør det utføres en sammenligningstest av mobilapplikasjonen og eksisterende innsamlingsmetode i framtidig arbeid for å kunne vurdere dette forskningsspørsmålet ytterligere og få et faktisk estimat på det. Med et estimat på, for eksempel tiden det tar å registrere informasjon med mobilapplikasjonen og med med penn og papir, er det mulig å se i hvor stor grad applikasjonen er mer effektiv enn nåværende registreringsmetode.

F3: Hvor detaljert, strukturert og oversiktlig er informasjonen fra tilsynsturer samlet inn med *Bæædar* sammenlignet med eksisterende innsamlingsmetode?
Uten å vite hvordan informasjonen som blir samlet inn med eksisterende innsamlingsmetoder ser ut i dag, er det vanskelig å sammenligne det med informasjonen som blir samlet inn med *Bæædar*. Dersom det hypotetisk sett blir notert ned i tilfeldig rekkefølge på papir, vil informasjonen samlet inn med applikasjonen, basert på hvordan deltagerne av brukertestene oppfattet det, være mer strukturert og oversiktlig. Detaljnivået vil nok omtrent være det samme, men informasjonen samlet inn med *Bæædar* kan tenkes å være mer presis. En gjennomgående ting som ble nevnt av alle deltagerne var at mobilapplikasjonen opplevdes som svært oversiktlig.

8.3.2 Evaluering av mål for prosjektet

Som beskrevet i Kapittel 1.2.1, var målet for prosjektet:

Utvikle et konsept som potensielt kan ha nytteverdi for både sauebønder og myndighetene. Konseptet skal kunne gjøre det enkelt og effektivt å registrere detaljert og strukturert informasjon om sau på utmarksbeite, samt mulighet for å ta i bruk interaktivt kart på tilsynstur.

Besvarelsene på forskningsspørsmålene indikerer at mobilapplikasjonen *Bæædar* fortsatt har en vei å gå før den er et fullverdig produkt som har nytteverdi for både sauebønder og myndighetene. Likevel, har konseptet utfra brukertestene bevist seg å være godt motatt av brukerne med positiv respons. Store deler av applikasjonen ble oppfattet som oversiktlig og strukturert. Ettersom den tilhørene nettsiden, som blant annet skal automatisk generere ferdigrapporter som skal bli sendt til myndighetene, ikke har blitt utviklet ennå vil ikke *Bæædar* ha like stor nytteverdi for myndighetene. Som sagt, gjenstår det en god del arbeid igjen før *Bæædar* kan erstatte nåværende

innsamlingsmetode. Videre arbeid av systemet kan leses i neste kapittel, Kapittel 9. Det kan dermed konkluderes med at målet for dette prosjektet er til en viss grad oppfylt.

Kapittel 9

Konklusjon og videre arbeid

Over en lengre periode har Professor *Svein-Olaf Hvasshovd* hatt et samarbeid med norske sauebønder for å utvikle teknologiske metoder og verktøy som kan forenkle jobben til sauebønder. Han er stadig i kontakt med sauebøndene og har tung kunnskap innenfor dette feltet. Dermed har mobilapplikasjonen, *Bæædar*, samt planen for den tilhørende nettsiden, blitt utviklet med utgangspunkt i veiledningsmøter med og kravspesifikasjon fra Professor *Hvasshovd*. Foreløpig er produktet som har blitt presentert i dette prosjektet først og fremst et *Minimal Viable Product*. Dersom det er et ønske om å distribuere dette systemet og sette det ut i produksjon, er det nødvendig med ytterligere forbedringer og videreutvikling av systemet. Mobilapplikasjonen er i et relativt tidlig stadie i utviklingsfasen og kun en del av de mest essensielle funksjonalitetene har blitt implementert. Det gjenstår fortsatt en del funksjonalitet i henhold til kravspesifikasjonen i Kapittel 3 som må implementeres for å få en velfungerende mobilapplikasjon som har høy nytteverdi for sauebønder. Utfra brukertestene er det også nødvendig med ytterligere forbedringer av eksisterende funksjonaliteter. I tillegg gjenstår den tilhørende nettsiden til *Bæædar* som ikke har blitt påbegynt. Det vil si at myndighetene ikke har fått dekket sine behov. Likevel, har konseptet om et slikt system utfra brukertestene vist seg å ha et stort potensiale for å forenkle og effektivisere oppfølging av sau på utmarksbeite.

9.1 Videreutvikling av mobilapplikasjon

Mobilapplikasjoen mangler en del viktig funksjonalitet før det kan bli tatt i bruk i praksis av sauebønder. Kravene som gjenstår og status på disse kravene blir vist på Tabell 9.1 og på Tabell 9.2. En nærmere beskrivelse av hvordan det er tenkt at de gjenværende funksjonalitetene kan implementeres vil bli presentert i dette delkapittelet.

FK-ID	Beskrivelse	Prioritet	Status
FK-01	Sauebonden skal kunne se et kart over hele Norge	Høy	Ferdig
FK-02	Sauebonden skal kunne klippe ut en bit av norgeskartet og laste det ned på applikasjonen	Høy	Ferdig
FK-03	Sauebonden skal kunne se sin egen posisjon på kartet	Høy	Ikke startet
FK-04	Sauebonden skal kunne merke av på kartet hvor han/hun har sett saueflokken	Middels	Ikke startet
FK-05	Sauebonden skal kunne se streker på kartet som viser hvor han/hun har gått	Høy	Ikke startet
FK-06	Sauebonden skal kunne registrere antall voksne og lam han/hun observerer	Høy	Ferdig
FK-07	Sauebonden skal kunne registrere antall lam han/hun observerer	Høy	Ferdig
FK-08	Sauebonden skal kunne registrere antall sau i ulike farger han/hun observerer	Høy	Ferdig
FK-09	Sauebonden skal kunne registrere antall sau etter fargen på slipset sauene han/hun observerer	Høy	Ferdig
FK-10	Sauebonden skal kunne registrere hvilke farger sauene har på øremerkene sine	Høy	Ferdig
FK-11	Sauebonden skal kunne zoome inn på den delen av kartet han/hun vil klippe ut	Middels	Ferdig
FK-12	Sauebonden skal kunne se hvilke områder han/hun har lastet ned fra før av	Høy	Pågår
FK-13	Sauebonden skal kunne velge blant områder som har blitt lastet ned fra før av	Høy	Pågår
FK-14	Sauebonden skal kunne velge dato og tidspunkt for tilsynsturen	Middels	Ferdig
FK-15	Sauebonden skal kunne se sine tidligere tilsynsturer	Middels	Ikke startet
FK-16	Sauebonden skal kunne logge inn på applikasjonen	Høy	Ferdig
FK-17	Sauebonden skal kunne registrere forskjellige detaljer som han/hun observerer basert på avstanden til saueflokken	Høy	Ferdig
FK-18	Sauebonden skal kunne registrere antall skadde sauer han/hun observerer	Høy	Ferdig
FK-19	Sauebonden skal kunne registrere antall døde sauer han/hun observerer	Høy	Ferdig
FK-20	Sauebonden skal kunne ta bilde av skadde sauer og døde sauer han/hun observerer	Lav	Ikke startet
FK-21	Sauebonden skal kunne se en oversikt over hva som har blitt registrert underveis på tilsynsturen	Middels	Ikke startet
FK-22	Sauebonden skal kunne avslutte tilsynsturen når han/hun er ferdig	Middels	Ferdig
FK-23	Sauebonden skal kunne opprette en bruker på applikasjonen	Høy	Ferdig

Tabell 9.1: Status på funksjonelle krav for mobilapplikasjon

NFK-ID	Beskrivelse	Kategori	Prioritet	Status
NFK-01	Mobilapplikasjonen skal kunne fungere uten mobildekning	Tilgjengelighet	Høy	Pågår
NFK-02	Mobilapplikasjonen skal kunne fungere på både *iOS* og *Android*	Brukervennlighet	Middels	Pågår
NFK-03	Mobilapplikasjonen skal være intuitiv nok til at det tar mindre enn 10 minutter på å lære seg hvordan man bruker det	Brukervennlighet	Høy	Pågår
NFK-04	Registrering av observasjoner skal ikke ta mer enn 3 minutter	Brukervennlighet	Middels	Ferdig
NFK-05	Registrerte data skal synkroniseres automatisk til databasen etter hver tilsynstur med en gang applikasjonen er koblet til internett igjen	Pålitelighet	Høy	Pågår

Tabell 9.2: Status på ikke-funksjonelle krav for mobilapplikasjon

FK-03: Saubonden skal kunne se sin egen posisjon på kartet
Sauebonden skal kunne se sin egen nåværende GPS-posisjon på det utvalgte området. Som nevnt i Kapittel 5.2.2, er det planlagt å bruke *React Native Background Geolocation* til denne funksjonen. Nærmere informasjon om dette finnes i Kapittel 5.2.2 og her [88].

FK-04: Sauebonden skal kunne merke av på kartet hvor han/hun har sett saueflokken
Under en tilsynstur skal sauebonden kunne sette et merke på det stedet på kartet der saueflokken blir observert. Det er tenkt at *OpenLayers* sin *Draw*-funksjon kan bli brukt til å merke av på kartet. *Draw*-funksjonen har en type som heter *Point*, som tegner et punkt på kartet. Dermed kan dette kravet bli implementert på en lignende måte som **FK-02**, hvor det markerte punktet blir konvertert til et *GeoJSON*-objekt og lagret som en *JSON*-string i databasen. Mer om framgangsmåten kan leses i Delkapittel 5.2.2.

FK-05: Sauebonden skal kunne se streker på kartet som viser hvor han/hun har gått
For å spore og loggføre hvor en sauebonde har gått på tilsynsturen har *React Native Background Geolocation* et *geofencing*-system. Dette systemet visualiserer både hvor man har gått og hvor langt man har gått. Mer om dette systemet finnes her [88].

FK-06 / FK-07 / FK-08 / FK-09 / FK-10
Til tross for at brukergrensesnittet til disse kravene er implementert, har ikke det blitt lagd kolleksjoner i databasen for informasjonen som registreres. Per nå er disse funksjonalitetene kun *proof of concept* for formålet av brukertesting. Det er derfor nødvendig at det blir lagt til implementasjon for å sende dokumentasjonsdata til databasen og knytte det til den tilsynsturen det gjelder. Hvordan strukturen på denne dataen skal organiseres i *Cloud Firestore* blir vist på Figur 4.13 i Kapittel 4.4.

FK-12: Sauebonden skal kunne se hvilke områder han/hun har lastet ned fra før av / FK-14: Sauebonden skal kunne velge blant områder som har blitt lastet ned fra før av
Når sauebonden begynner på en tilsynstur skal han/hun kunne velge det kartet som skal brukes utfra en liste med forskjellige nedlastede kart. Foreløpig har det ikke blitt implementert en god måte å hente disse kartene fra databasen på og liste dem opp i applikasjonen, som beskrevet i Kapittel 5.2.4. Det er for nå bare ett kartdokument som blir hentet fra databasen for å vise fram denne funksjonaliteten.

FK-15: Sauebonden skal kunne se sine tidligere tilsynsturer
Sauebonden skal kunne gå inn på applikasjonen og se en oversikt over tidligere tilsynsturer som har blitt gjennomført, med detaljer om observasjoner og hvor det har blitt gått. For å implementere dette kravet er det tenkt at **FK-05** må være på plass først slik at det finnes en logg over hvor bonden har gått. Videre er det tenkt at observasjonsdata fra turen hentes fra databasen og blir vist på et diagram, som vist på Figur 5.21 i Kapittel 5.1.3.

FK-20: Sauebonden skal kunne ta bilde av skadde sauer og døde sauer han/hun observerer
For å vite hva slags skade som har blitt gjort på sauene og dokumentere det burde det være mulig å ta bilde med applikasjonen. Underveis når sauebonden registrerer skadde og døde sauer kan det være en funksjon hvor sauebonden kan ta bilde, slik at applikasjonen blir koblet til mobilenhetens kamera. Deretter blir bildene lagret som en del av informasjonen for tilsynsturen. For å lagre bildet i databasen som en del av *observations*-delkolleksjonen kan URL-en til bildet bli lagret som en string, slik at man kan hente ut denne stringen fra databasen senere og rendere det i *Image*-komponenten til *React Native*. For å koble en *React Native*-applikasjon til mobilkamera kan man bruke *plugin*-en *React Native Camera* [92].

FK-21: Sauebonden skal kunne se en oversikt over hva som har blitt registrert underveis på tilsynsturen
Underveis på tilsynsturen skal sauebonden kunne trykke på *Antall registrert* på navigeringsbaren og se en oversikt over hva som har blitt registrert underveis på tilsynsturen som vist i Kapittel 5.1 på Figur 5.17. Den lokalt-cachet dataen hentes og listes opp på applikasjonen.

NFK-01: Mobilapplikasjonen skal kunne fungere uten mobildekning
Cloud Firestore har *offline*-støtte som beskrevet i Kapittel 5.2.2, men denne funksjonaliteten har foreløpig ikke blitt testet ut. Det er derfor noe som bør tas i betraktning og bli testet ute i felten.

NFK-02: Mobilapplikasjonen skal kunne fungere på både *iOS* og *Android*
Foreløpig har mobilapplikasjonen kun blitt testet ut på *iOS*-enheter og blitt utviklet med utgangspunkt i en *iOS*-simulator. For å kjøre applikasjonen på en *Android*-enhet må koden for det settes opp i *Android*-mappen til applikasjonen, og det må i tillegg bli kjøpt en lisens for å bruke *React Native Background Geolocation*.

NFK-05: Registrerte data skal synkroniseres automatisk til databasen etter hver tilsynstur med en gang applikasjonen er koblet til internett igjen
Dette kravet forutsetter at **NFK-01** er på plass. Dersom **NFK-01** fungerer som forventet vil *CLoud Firestore* håndtere automatisk synkronisering av data til databasen.

Finpussing av brukergrensesnittet
Til videre arbeid kan utseende til brukergrensesnittet finjusteres. Blant annet kan feltene til innlogging og oppretting av bruker justeres til å bli mindre og midtstilles. Noen av knappene bør det også finpusses på.

Tilbakemeldinger fra brukertestene
Det kan være en idé å ta utgangspunkt i tilbakemeldingene fra brukertestene beskrevet i Kapittel 8.2.3 ettersom disse tilbakemeldingene gir et bilde av hvordan applikasjonen oppleves av brukerne. Det er en del forbedringer og endringer som burde iverksettes for bedre funksjonalitet og brukervennlighet på mobilapplikasjonen.

9.2 Utvikling av nettside

Webapplikasjonen er ikke påbegynt og dermed er status på alle kravene "ikke startet" som vist på Tabell 9.3 og på Tabell 9.4. Planlagt teknologi for nettsiden er *React* med *Typescript*. Et forslag til designet av brukergrensesnittet kan være å benytte seg av komponentbiblioteket *Material UI* [57], siden komponentene er lett å tilpasse etter ønske og at komponentdesignet samsvarer litt med komponentdesignet til *React Native Paper*. En nærmere beskrivelse av hvordan det er tenkt at disse funksjonalitetene kan implementeres vil bli presentert i dette delkapittelet.

FK-ID	Beskrivelse	Prioritet	Status
FK-01	Sauebonden skal kunne logge seg inn på nettsiden for å få tilgang	Høy	Ikke startet
FK-02	Sauebonden skal kunne opprette en ny tilsynsgruppe	Høy	Ikke startet
FK-03	Sauebonden skal kunne invitere andre brukere til sin tilsynsgruppe	Høy	Ikke startet
FK-04	Sauebonden skal kunne bli med i tilsynsgrupper opprettet av andre brukere	Høy	Ikke startet
FK-05	Sauebonden skal kunne se rapporter fra sine egne tilsynsturer	Høy	Ikke startet
FK-06	Sauebonden skal kunne se rapporter fra brukere i samme tilsynsgruppe	Høy	Ikke startet
FK-07	Sauebonden skal kunne se informasjon fra tilsynsturene visualisert på et kart	Høy	Ikke startet
FK-08	Sauebonden skal kunne filtrere og sortere på informasjonen fra tilsynsrapporter som vises basert på observasjonsdetaljer	Lav	Ikke startet
FK-09	Sauebonden skal kunne se ferdiggenererte rapporter med data fra sine egne tilsynsturer	Høy	Ikke startet
FK-10	Sauebonden skal kunne velge mellom forskjellige visualiseringer på kartet	Lav	Ikke startet
FK-11	Sauebonden skal kunne se en oppsummering av hver tilsynstur gjennomført i sesongen i den ferdiggenererte rapporten	Høy	Ikke startet
FK-12	Sauebonden skal kunne se registrert, detaljert informasjon om hver observasjon fra tilsynsturer i den ferdiggenererte rapporten	Høy	Ikke startet
FK-13	Sauebonden skal kunne laste ned den ferdiggenererte rapporten som en PDF-fil	Høy	Ikke startet

Tabell 9.3: Status på funksjonelle krav for nettside

NFK-ID	Beskrivelse	Kategori	Prioritet	Status
NFK-01	Nettsiden skal ha en gjennomsnittlig responstid på 3 sekunder	Ytelse	Høy	Ikke startet
NFK-02	Nettsiden skal en responstid på maksimalt 5 sekunder	Ytelse	Middels	Ikke startet
NFK-03	Nettsiden skal være intuitiv nok til at det tar mindre enn 5 minutter på å lære seg hvordan det fungerer	Brukervennlighet	Høy	Ikke startet
NFK-04	Brukere av nettsiden skal autentiseres for å få tilgang	Sikkerhet	Høy	Ikke startet

Tabell 9.4: Status på ikke-funksjonelle krav for nettside

FK-01: Sauebonden skal kunne logge seg inn på nettsiden for å få tilgang
Innloggingen til nettsiden kan foregå på samme måte som med mobilapplikasjonen; ved bruk av *Firebase Authentication* tilpasset innloggingssystemet til nettsiden. Mer informasjon om hvordan det ble implementert finnes i Kapittel 5.2.2.

FK-02: Sauebonden skal kunne opprette en ny tilsynsgruppe
Det er tenkt at sauebonden skal ha en tilsynsgruppe med flere gjetere. I databasen kan det for eksempel være en kolleksjon med alle opprettede tilsynsgrupper. Hvert dokument i kolleksjonen representerer én tilsynsgruppe, som inneholder felt for:

- Dato og tidspunkt for når gruppen ble opprettet

- En liste med brukeriden til alle brukere som er tilknyttet den gruppen

FK-03: Sauebonden skal kunne invitere andre brukere til sin tilsynsgruppe
For å invitere andre brukere til sin tilsynsgruppe kan det være et søkefelt på nettsiden hvor man skriver inn e-posten til brukeren man ønsker å invitere. Alle brukere kan bli hentet fra databasen og e-postene deres kan bli listet opp i en slags *dropdown*-meny som kun viser e-posten som matcher søkeordet. Deretter kan brukeren trykke på e-posten som dukker opp og legge det til i feltet, og fortsette å skrive inn e-posten til den neste brukeren ønsker å invitere. Dette feltet kan fungere på samme måte som "Til"-feltet på *Microsoft Outlook* når man skal skrive inn e-posten til den man ønsker å sende mail til. Etter at brukeren har lagt til e-postene til alle brukerne han/hun ønsker å invitere, kan brukeren trykke på en knapp som sender en invitasjon på mail til de andre brukerne. I tillegg burde det være mulig å legge til andre brukere senere dersom han/hun ønsker det.

FK-04: Sauebonden skal kunne bli med i tilsynsgrupper opprettet av andre brukere
Dette kravet forutsetter at **FK-03** er implementert. Hvis en bruker kan bli lagt til i en tilsynsgruppe, skal brukeren også kunne legge til andre brukere i sin tilsynsgruppe.

FK-05: Sauebonden skal kunne se rapporter fra sine egne tilsynsturer
Rapporter fra tilsynsturene til sauebonden kan bli vist på samme måte som **FK-15** til mobilapplikasjonen.

FK-06: Sauebonden skal kunne se rapporter fra brukere i samme tilsynsgruppe
Brukeren skal kunne gå inn på en tilsynsgruppe han/hun tilhører, få en oversikt over andre brukere i samme gruppe og kunne trykke seg inn på en av dem. Deretter vil rapporter fra deres tilsynsturer bli vist på samme måte som **FK-05**.

FK-07: Sauebonden skal kunne se informasjon fra tilsynsturene visualisert på et kart
Visualiseringen av tilsynsturene på kartet er tenkt at den skal ha samme funksjonalitet som på **FK-05** for mobilapplikasjonen.

FK-08: Sauebonden skal kunne filtrere og sortere på informasjonen fra tilsynsrapporter som vises basert på observasjonsdetaljer
Det skal være mulig å filtrere på de forskjellige kategoriene det har blitt registrert på, for eksempel antall lam som ble observert. Da skal brukeren få opp en liste med antall lam med dato og tidspunkt som har blitt registrert på de forskjellige tilsynsturene i løpet av sesongen. I tillegg skal det være mulig sortere på informasjonen slik at brukeren for eksempel får opp antall brune sauer først.

FK-09: Sauebonden skal kunne se ferdiggenererte rapporter med data fra sine egne tilsynsturer
Strukturen på de ferdiggenererte rapportene og hva slags data disse rapportene skal inneholde burde utformes i samsvar med myndighetenes kriterier. På denne måten kan man være sikker på at man dekker myndighetenes behov for rapportering av sau på utmarksbeite. Utfra myndighetenes kriterier kan det bli utformet en standard mal for rapporten med ferdigutfylte felter med data hentet fra databasen.

FK-10: Sauebonden skal kunne velge mellom forskjellige visualiseringer på kartet
Sauebonden skal kunne velge mellom forskjellige tilsynsturer han/hun ønsker at det vises en visualisering av på kartet. Dette kravet forutsetter at **FK-05** til mobilapplikasjonen er implementert for at tilsynsturene skal bli loggført.

FK-11: Sauebonden skal kunne se en oppsummering av hver tilsynstur gjennomført i sesongen i den ferdiggenererte rapporten
I likhet med **FK-09** bør oppsummeringen være i samsvar med det myndighetene ønsker at rapporten skal inneholde.

FK-12: Sauebonden skal kunne se registrert, detaljert informasjon om hver observasjon fra tilsynsturer i den ferdiggenererte rapporten
I likhet med **FK-09** bør informasjonen om hver observasjon fra tilsynsturene være i samsvar med det myndighetene ønsker at rapporten skal inneholde.

FK-13: Sauebonden skal kunne laste ned den ferdiggenererte rapporten som en PDF-

fil
Til denne funksjonaliteten finnes det flere *plugins* til *React* som man kan bruke for å eksportere *React*-komponenter til PDF. Blant annet *jspdf*. En guide på hvordan eksporteringen kan implementeres finnes her [35].

9.3 Programvaresikkerhet

Ettersom mobilapplikasjonen i dette prosjektet er en *MVP*, har sikkerheten verken blitt satt som et krav eller blitt prioritert. Derimot har programvaresikkerhet og personvern i senere tid blitt grunnleggende prinsipper i alle applikasjoner. Til videreutviklingen av mobilapplikasjonen og utvikling av nettsiden kunne det ha vært mulig å integrere verifisering av e-post når det blir opprettet en ny bruker, passordendring ved glemt passord og to-faktor-autentisering. Disse tre metodene har blitt vanlig praksis for god sikkerhet hos de fleste applikasjoner. Mer informasjon om prinsippene for programvaresikkerhet finnes her [12].

9.3.1 Verifisering av e-post

Det er mulig å integrere verifisering av e-post i *Firebase Authentication* ved oppretting av ny bruker. I de fleste applikasjoner har det blitt en vanlig praksis at nye brukere må verifisere e-postadressen sin. Da blir det sendt ut en mail til e-postadressen som har blitt brukt til å lage en ny bruker, hvor brukeren da må trykke på linken på mailen for å verifisere e-postadressen sin. Dette stemmer overens med OWASP WSTG-IDNT-02 - prinsippet [12], om at ikke alle kan registrere seg for tilgang. Hvordan dette kan bli implementert blir forklart her [56] og her [27].

9.3.2 Endring av passord

Dersom en bruker har glemt passord kan *password reset email* bli integrert i *Firebase Authentication* [27]. Da blir det sendt en link til e-posten til brukeren som brukeren kan trykke på for å bli sendt til siden hvor han/hun kan lage et nytt passord. For mer informasjon om hvilke retningslinker man burde følge for å lage en sikker tjeneste for passordendring kan man se her [72].

9.3.3 To-faktor-autentisering

For å integrere to-faktor-autentisering i applikasjonen kan man bruke *Google Identity Platform*. *Firebase* støtter denne plattformen. Da blir det sendt en kode på SMS til brukerens mobiltelefon som skal brukes for å logge inn på applikasjonen. En guide på hvordan to-faktor-autentisering kan settes opp finnes her [8]. Dersom det er ønskelig å bruke en annen tjeneste kan man følge retningslinjene for god multifaktor-autentisering her [73].

9.4 Brukertest på sauebønder

Til videreutvikling av mobilapplikasjonen *Bæædar* og utviklingen av nettsiden bør det brukertestes jevnlig. Sauebønder er som sagt målgruppen og dra nytte av dette systemet. Sauebønder kan gi innsiktsfulle tilbakemeldinger om hvordan de opplever systemet. I en smidig utviklingsprosess er det viktig med demo og brukertesting for å kunne utvikle et godt produkt som dekker brukernes behov [71].

Mobilapplikasjonen burde i tillegg bli testet i praksis på en ekte tilsynstur av en sauebonde. På denne måten kan man se hvordan applikasjonen faktisk fungerer ute i felt, og kan gi innholdsrik informasjon om hva som fungerer godt, hva som burde forbedres og hva som eventuelt må forandres på ved applikasjonen.

9.5 Kommunikasjon med fylkesmann

Som nevnt tidligere, bør man ha kommunikasjon med en fylkesmann for å vite hva slags informasjon myndighetene ønsker å hente fra tilsynsturene til sauebøndene. Informasjonen om observasjonene som registreres på mobilapplikasjonen skal også dekke behovet til myndighetene om rapportering av sau på utmarksbeite. Planen videre burde være å sette opp et intervju med en fylkesmann for å vite hva slags informasjon mobilapplikasjonen skal registrere og hva rapporten i slutten av beitesesongen skal inneholde. Dette intervjuet burde i grunnen bli gjennomført før videre utvikling av applikasjonen og nettsiden for å oppdage eventuelle mangler ved applikasjonen og vite hvordan rapporten skal utformes.

Kildeliste

[1] Avocode. *Hand drawing is the most important skill for a designer, says Janis Andzans.* URL: https://blog.avocode.com/hand-drawing-is-the-most-important-skill-for-a-designer-says-janis-andzans-b1b84937043a (sjekket 31.05.2022).

[2] Nick Babich. *Top 7 Usability Testing Methods.* URL: https://xd.adobe.com/ideas/process/user-testing/top-7-usability-testing-methods/ (sjekket 15.05.2022).

[3] Chris Bateson. *Cordova vs. React Native vs. Xamarin: Which Cross-Platform Development Framework Is Best?* URL: https://chrisbateson80.medium.com/cordova-vs-react-native-vs-xamarin-which-cross-platform-development-framework-is-best-14ee98d80d41 (sjekket 20.04.2022).

[4] Vikrant Bhalodia. *What is React Native Development, Advantages and Disadvantages?* URL: https://medium.com/weblineindia/what-is-react-native-development-advantages-and-disadvantages-e5bf052473a7 (sjekket 15.03.2022).

[5] N. Breau mfl. *Overview.* URL: https://cordova.apache.org/docs/en/11.x/guide/overview/index.html (sjekket 18.04.2022).

[6] Callstack. *Cross-platform Material Design for React Native.* URL: https://callstack.github.io/react-native-paper/ (sjekket 15.03.2022).

[7] Archana Choudary. *Azure Tutorial — Getting Started With Microsoft Azure.* URL: https://medium.com/edureka/azure-tutorial-5a97e30ee9a7 (sjekket 22.04.2022).

[8] Google Cloud. *Adding multi-factor authentication to your web app.* URL: https://cloud.google.com/identity-platform/docs/web/mfa (sjekket 13.05.2022).

[9] Yngve Dahl. *Introduksjon til Brukersentrert design.* URL: https://folk.ntnu.no/baldurk/skolearbeid/MMI/Forelesninger%20MMI/11-Brukersentrert%20design%20-%20Intro.pdf (sjekket 19.11.2021).

[10] John Dengis. *Cloud Firestore vs. Cosmos DB: Which NoSQL database is right for you?* URL: https://ezfire.io/blog/cloud-firestore-vs-cosmos-db/ (sjekket 26.04.2022).

[11] Prakash Donga. *14 Reasons to Choose React Native for App Development.* URL: https://www.solutelabs.com/blog/14-reasons-to-choose-react-native-for-app-development (sjekket 21.11.2021).

[12] R. Mitchell E. Saad. *Web Security Testing Guide.* 4.2. The OWASP Foundation, 2020.

[13] Erisu. *Cordova and React Native Comparison.* URL: https://medium.com/the-web-tub/cordova-and-react-native-comparison-3f8bf16cf036 (sjekket 18.04.2022).

[14] Expo, Software Mansion og Callstack. *Hello React Navigation.* URL: https://reactnavigation.org/docs/hello-react-navigation (sjekket 16.03.2022).

[15] Figma. *Figma: the collaborative interface design tool.* URL: https://www.figma.com/ (sjekket 19.11.2021).

[16] Jakub Fijalkowski. *Why Firestore, Part I: Reasons to Love It.* URL: https://leancode.co/blog/why-firestore-firebase-pros-cons-reasons-to-love-it-part-i (sjekket 01.06.2022).

[17] Findmy. *E-bjelle model 2.* URL: https://www.findmy.no/nb/ebjelle-model2 (sjekket 08.11.2021).

[18] Findmy. *Om oss.* URL: https://www.findmy.no/nb/om-oss (sjekket 08.11.2021).

[19] Google Firebase. *Choose a Database: Cloud Firestore or Realtime Database*. URL: https://firebase.google.com/docs/database/rtdb-vs-firestore (sjekket 25.04.2022).

[20] Google Firebase. *Cloud Firestore*. URL: https://firebase.google.com/products/firestore?gclid=CjwKCAiAyPyQBhB6EiwAFUuakmHvEmoVhlfeGeEdhuFPfQKi6ug8-ooW5psBZNXzxCVVWkeSk0qEkxoCpYkQA BwE&gclsrc=aw.ds (sjekket 02.03.2022).

[21] Google Firebase. *Cloud Firestore*. URL: https://firebase.google.com/docs/firestore?hl=en&authuser=0 (sjekket 17.03.2022).

[22] Google Firebase. *Cloud Firestore Data model*. URL: https://firebase.google.com/docs/firestore/data-model (sjekket 09.03.2022).

[23] Google Firebase. *Firebase Authentication*. URL: https://firebase.google.com/docs/auth (sjekket 24.03.2022).

[24] Google Firebase. *Firebase Authentication Limits*. URL: https://firebase.google.com/docs/auth/limits (sjekket 28.03.2022).

[25] Google Firebase. *Firebase Realtime Database*. URL: https://firebase.google.com/docs/database (sjekket 25.04.2022).

[26] Google Firebase. *Index types in Cloud Firestore*. URL: https://firebase.google.com/docs/firestore/query-data/index-overview (sjekket 23.03.2022).

[27] Google Firebase. *Manage Users in Firebase*. URL: https://firebase.google.com/docs/auth/web/manage-users (sjekket 13.05.2022).

[28] Google Firebase. *Structuring security rules*. URL: https://cloud.google.com/firestore/docs/security/rules-structure (sjekket 04.04.2022).

[29] Google Firebase. *Understand Cloud Firestore billing*. URL: https://firebase.google.com/docs/firestore/pricing (sjekket 28.04.2022).

[30] Google Firebase. *Usage and limits*. URL: https://firebase.google.com/docs/firestore/quotas (sjekket 28.04.2022).

[31] GeeksforGeeks. *Difference between TypeScript and JavaScript*. URL: https://www.geeksforgeeks.org/difference-between-typescript-and-javascript/ (sjekket 11.05.2022).

[32] Google. *Introduction*. URL: https://material.io/design/introduction (sjekket 15.03.2022).

[33] R. Greenlaw og E.M. Hepp. *Client-Server Model*. URL: https://www.sciencedirect.com/topics/computer-science/client-server-model (sjekket 03.03.2022).

[34] Mohit Gupta. *Working with OpenLayers 4 — Part 1-Creating the first application*. URL: https://medium.com/attentive-ai/working-with-openlayers-4-part-1-creating-the-first-application-9ab27bbd7a62 (sjekket 05.04.2022).

[35] Uditha Janadara. *Export React Component As a PDF*. URL: https://udithajanadara.medium.com/export-react-component-as-a-pdf-5afba8ba02ee (sjekket 13.05.2022).

[36] Ali Kamalizade. *AWS vs Azure vs Firebase vs Heroku vs Netlify—How To Choose the Best Platform for Web Projects*. URL: https://betterprogramming.pub/aws-vs-azure-vs-firebase-vs-heroku-vs-netlify-how-to-choose-the-best-platform-for-web-projects-482d017de254 (sjekket 03.05.2022).

[37] Kartverket. *API og data*. URL: https://www.kartverket.no/api-og-data (sjekket 12.04.2022).

[38] Kartverket. *example-clients*. URL: https://github.com/kartverket/example-clients (sjekket 12.04.2022).

[39] Jesse Katsumata. *react-native-color-picker-ios*. URL: https://www.npmjs.com/package/react-native-color-picker-ios (sjekket 24.05.2022).

[40] Todd Kerpelman. *Why is my Cloud Firestore query slow?* URL: https://medium.com/firebase-developers/why-is-my-cloud-firestore-query-slow-e081fb8e55dd (sjekket 23.03.2022).

[41] Leaflet. *Features*. URL: https://leafletjs.com/SlavaUkraini/index.html#features (sjekket 21.04.2022).

[42] Leaflet. *Leaflet - an open-source JavaScript library for mobile-friendly interactive maps*. URL: https://leafletjs.com/ (sjekket 21.11.2021).

[43] Guy Ligertwood. *Guerrilla Testing: Hallway Usability Tests for UX*. URL: https://xd.adobe.com/ideas/process/user-testing/hallway-usability-test-guerrilla-testing/ (sjekket 02.06.2022).

[44] T. Malbranche mfl. *React Native WebView - a Modern, Cross-Platform WebView for React Native*. URL: https://github.com/react-native-webview/react-native-webview (sjekket 01.02.2022).

[45] Robin Jan Maly mfl. «Comparison of centralized (client-server) and decentralized (peer-to-peer) networking». I: *Semester thesis, ETH Zurich, Zurich, Switzerland* (2003), s. 1–12.

[46] Landbruks og matdepartementet. *Forskrift om velferd for småfe*. URL: https://lovdata.no/dokument/SF/forskrift/2005-02-18-160/ (sjekket 02.11.2021).

[47] Microsoft. *Azure Active Directory (Azure AD)*. URL: https://azure.microsoft.com/en-us/services/active-directory/#features (sjekket 02.05.2022).

[48] Microsoft. *Azure Active Directory (Azure AD) pricing*. URL: https://azure.microsoft.com/en-us/pricing/details/active-directory/ (sjekket 02.05.2022).

[49] Microsoft. *Azure Cosmos DB*. URL: https://azure.microsoft.com/en-us/services/cosmos-db/ (sjekket 25.04.2022).

[50] Microsoft. *Azure Database for MariaDB*. URL: https://azure.microsoft.com/en-us/services/mariadb/#overview (sjekket 25.04.2022).

[51] Microsoft. *Azure Database for MySQL*. URL: https://azure.microsoft.com/en-us/services/mysql/#overview (sjekket 25.04.2022).

[52] Microsoft. *Azure Database for PostgreSQL*. URL: https://azure.microsoft.com/en-us/services/postgresql/#overview (sjekket 25.04.2022).

[53] Microsoft. *Azure SQL*. URL: https://azure.microsoft.com/en-us/products/azure-sql/#product-overview (sjekket 25.04.2022).

[54] Microsoft. *Pricing model in Azure Cosmos DB*. URL: https://docs.microsoft.com/en-us/azure/cosmos-db/how-pricing-works (sjekket 02.05.2022).

[55] Microsoft. *TypeScript is JavaScript with syntax for types.* URL: https://www.typescriptlang.org/ (sjekket 10.05.2022).

[56] Laurence Moroney. *Email Verification in Firebase Auth*. URL: https://firebase.blog/posts/2017/02/email-verification-in-firebase-auth (sjekket 13.05.2022).

[57] MUI. *Move faster with intuitive React UI tools*. URL: https://mui.com/ (sjekket 04.06.2022).

[58] Node.js. *Node.js v18.0.0 documentation*. URL: https://nodejs.org/api/vm.html (sjekket 20.04.2022).

[59] Nofence. *Hva er Nofence?* URL: https://www.nofence.no/hva-er-nofence (sjekket 08.11.2021).

[60] Nofence. *Om oss*. URL: https://www.nofence.no/om-oss (sjekket 08.11.2021).

[61] Nofence. *Priskalkulator*. URL: https://www.nofence.no/priser (sjekket 22.05.2022).

[62] Dyrebeskyttelsen Norge. *Fanesak tap av sau på beite*. URL: https://www.dyrebeskyttelsen.no/tap-sau-pa-beite/ (sjekket 02.11.2021).

[63] OpenLayers. *A high-performance, feature-packed library for all your mapping needs*. URL: https://openlayers.org/ (sjekket 01.02.2022).

[64] OpenLayers. *Box Selection*. URL: https://openlayers.org/en/latest/examples/box-selection.html (sjekket 12.04.2022).

[65] OpenLayers. *Draw and Modify Features*. URL: https://openlayers.org/en/latest/examples/draw-and-modify-features.html (sjekket 12.04.2022).

[66] OpenLayers. *ol/interaction/DragBox DragBox*. URL: https://openlayers.org/en/latest/apidoc/module-ol_interaction_DragBox-DragBox.html (sjekket 12.04.2022).

[67] OpenLayers. *ol/interaction/Draw Draw*. URL: https://openlayers.org/en/latest/apidoc/module-ol_interaction_Draw-Draw.html (sjekket 12.04.2022).

[68] OpenStreetMap. *Velkommen til OpenStreetMap*. URL: https://www.openstreetmap.org/#map=5/65.401/17.864 (sjekket 12.04.2022).

[69] Rishabh Pancholi. *Fabric Architecture - React Native*. URL: https://medium.com/mindful-engineering/fabric-architecture-react-native-a4f5fd96b6d2 (sjekket 11.05.2022).

[70] Avelon Pang. *TypeScript vs. JavaScript*. URL: https://medium.com/geekculture/typescript-vs-javascript-e5af7ab5a331 (sjekket 11.05.2022).

[71] ScrumGuides.org. *The 2020 Scrum Guide TM*. URL: https://scrumguides.org/scrum-guide.html (sjekket 21.11.2021).

[72] OWASP Cheat Sheet Series. *Forgot Password Cheat Sheet*. 2021. URL: https://cheatsheetseries.owasp.org/cheatsheets/Forgot_Password_Cheat_Sheet.html (sjekket 13.05.2022).

[73] OWASP Cheat Sheet Series. *Multifactor Authentication Cheat Sheet*. 2021. URL: https://cheatsheetseries.owasp.org/cheatsheets/Multifactor_Authentication_Cheat_Sheet.html (sjekket 13.05.2021).

[74] Saurabh Shah. *React JS— Architecture + Features + Folder structure + Design Pattern*. URL: https://medium.com/geekculture/react-js-architecture-features-folder-structure-design-pattern-70b7b9103f22 (sjekket 11.05.2022).

[75] Dinesh Singu. *Introduction to Cosmos DB*. URL: https://medium.com/globant/introduction-to-cosmos-db-8d106bb7207 (sjekket 27.04.2022).

[76] Facebook Open Source. *Fabric*. URL: https://reactnative.dev/architecture/fabric-renderer (sjekket 11.05.2022).

[77] Facebook Open Source. *React - A JavaScript library for building user interfaces*. URL: https://reactjs.org/ (sjekket 11.05.2022).

[78] Facebook Open Source. *Running On Device*. URL: https://reactnative.dev/docs/running-on-device (sjekket 04.06.2022).

[79] Facebook Open Source. *Setting up the development environment*. URL: https://reactnative.dev/docs/environment-setup (sjekket 04.06.2022).

[80] Doug Stevenson. *The top 10 things to know about Firestore when choosing a database for your app*. URL: https://medium.com/firebase-developers/the-top-10-things-to-know-about-firestore-when-choosing-a-database-for-your-app-a3b71b80d979 (sjekket 22.03.2022).

[81] Doug Stevenson. *What is Firebase? The complete story, abridged*. URL: https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0 (sjekket 01.06.2022).

[82] Terasol Technologies. *Introduction to Apache Cordova: How it Works, Architecture, 7 Benefits and Best Examples*. URL: https://terasol.medium.com/introduction-to-apache-cordova-how-it-works-architecture-7-benefits-and-best-examples-9183fd0e98e (sjekket 20.04.2022).

[83] techopedia. *Cross-Platform Development*. URL: https://www.techopedia.com/definition/30026/cross-platform-development (sjekket 21.11.2021).

[84] Telespor. *Elektronisk overvåkning av husdyr*. URL: https://telespor.no/ (sjekket 05.11.2021).

[85] ThinkGeo. *Leaflet vs. OpenLayers 3: Which is the better client-side JavaScript mapping library?* URL: https://thinkgeo.com/blog/leaflet-vs-openlayers-3-which-is-the-better-client-side-javascript-mapping-library (sjekket 21.04.2022).

[86] Thinkwik. *React Native: What is it? and, Why is it used?* URL: https://medium.com/@thinkwik/react-native-what-is-it-and-why-is-it-used-b132c3581df (sjekket 21.11.2021).

[87] Transistorsoftware. *Background Geolocation for React Native*. URL: https://github.com/transistorsoft/react-native-background-geolocation (sjekket 12.05.2022).

[88] Transistorsoftware. *React Native Background Geolocation*. URL: https://www.transistorsoft.com/shop/products/react-native-background-geolocation (sjekket 12.05.2022).

[89] Simon Ullsfoss. *Smidig vs. tradisjonell prosjektmetodikk - hva er best?* URL: https://sprint.no/artikler/smidig-vs-tradisjonell-prosjektmetodikk-hva-er-best (sjekket 01.06.2022).

[90] Stanford University. *What is JavaScript?* URL: https://web.stanford.edu/class/cs98si/slides/overview.html (sjekket 20.04.2022).

[91] Bhumin Vadalia. *ReactJS and React Native - Key Difference, Advantages, and Disadvantages*. URL: https://medium.com/geekculture/reactjs-and-react-native-key-difference-advantages-and-disadvantages-ceb197f4d31d (sjekket 30.03.2022).

[92] Steven Vanderschaeve. *RNCamera*. URL: https://react-native-camera.github.io/react-native-camera/docs/rncamera (sjekket 14.05.2022).

[93] Mike Wasson. *ASP.NET - Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET*. URL: https://docs.microsoft.com/en-us/archive/msdn-magazine/2013/november/asp-net-single-page-applications-build-modern-responsive-web-apps-with-asp-net (sjekket 11.05.2022).

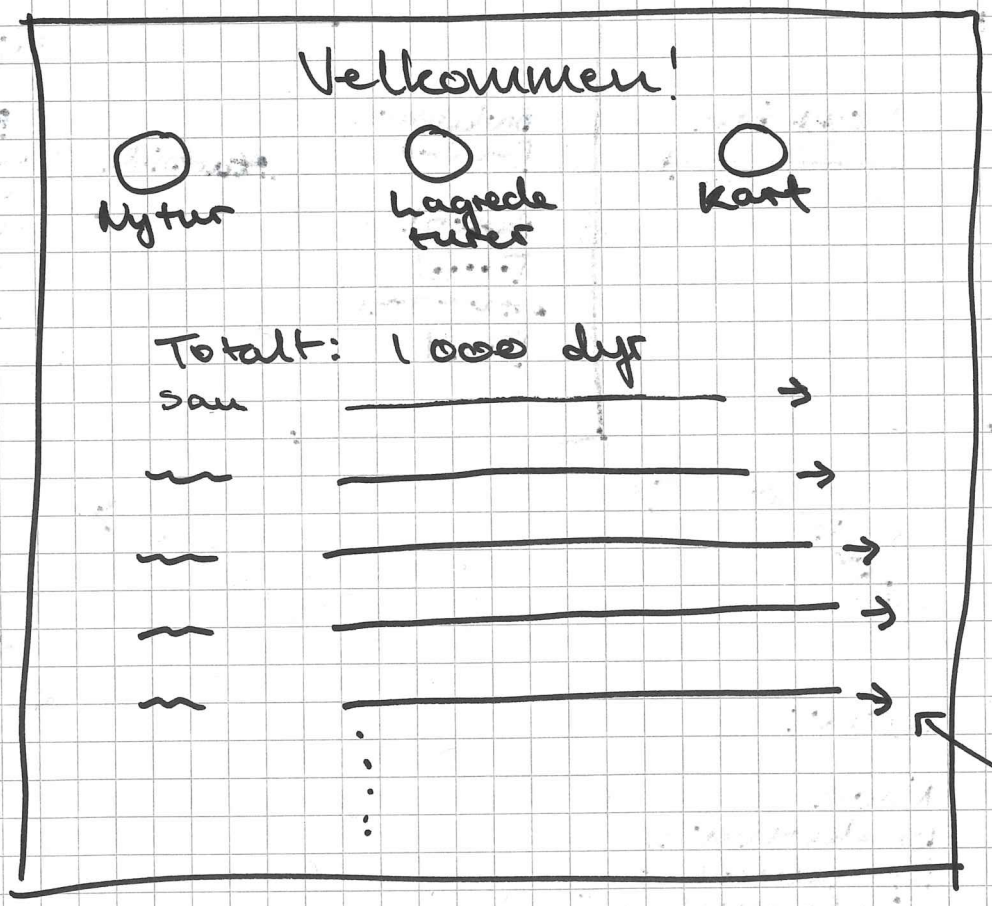
[94] Wikipedia. *GeoJSON*. URL: https://en.wikipedia.org/wiki/GeoJSON (sjekket 16.02.2022).

[95] Emmanuel Yusufu. *React Native navigation: React Navigation examples and tutorial*. URL: https://blog.logrocket.com/navigating-react-native-apps-using-react-navigation/ (sjekket 01.06.2022).

[96] Yusuf Zeren. *Location Tracking with React Native*. URL: https://medium.com/trendyol-tech/react-native-ile-konum-takibi-bfa51c4b2c98 (sjekket 12.05.2022).

Vedlegg

.1 Håndskisser



Flere detaljer →

→ Flere detaljer

Innlogging

Startside

Hvor er bøøø?

Logg in

Lag bruker

Logg inn

Brukernavn

Passord

o Husk meg

OK

Startside

Velkommen!

Siste tur

Navbar:

Ny tur

Norgeskart

Se tidligere turer

Hjelp (?)

Ny bruker

Logg inn

Ny bruker

Krav:

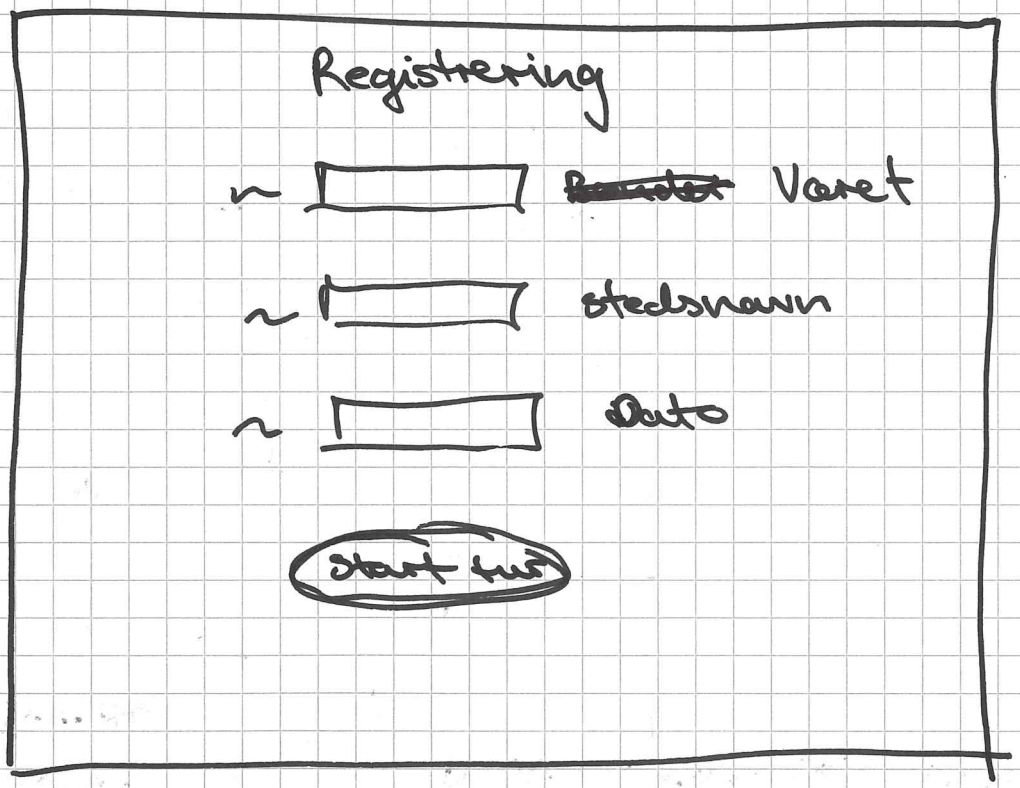
Navn
Brukernavn
Kommune - dropdown?
Gårdsnummer
Mye e-post
Passord
Gjenta passord

Ny tur

Kart

Lagrede turer

Ny oppsynstur



Features:

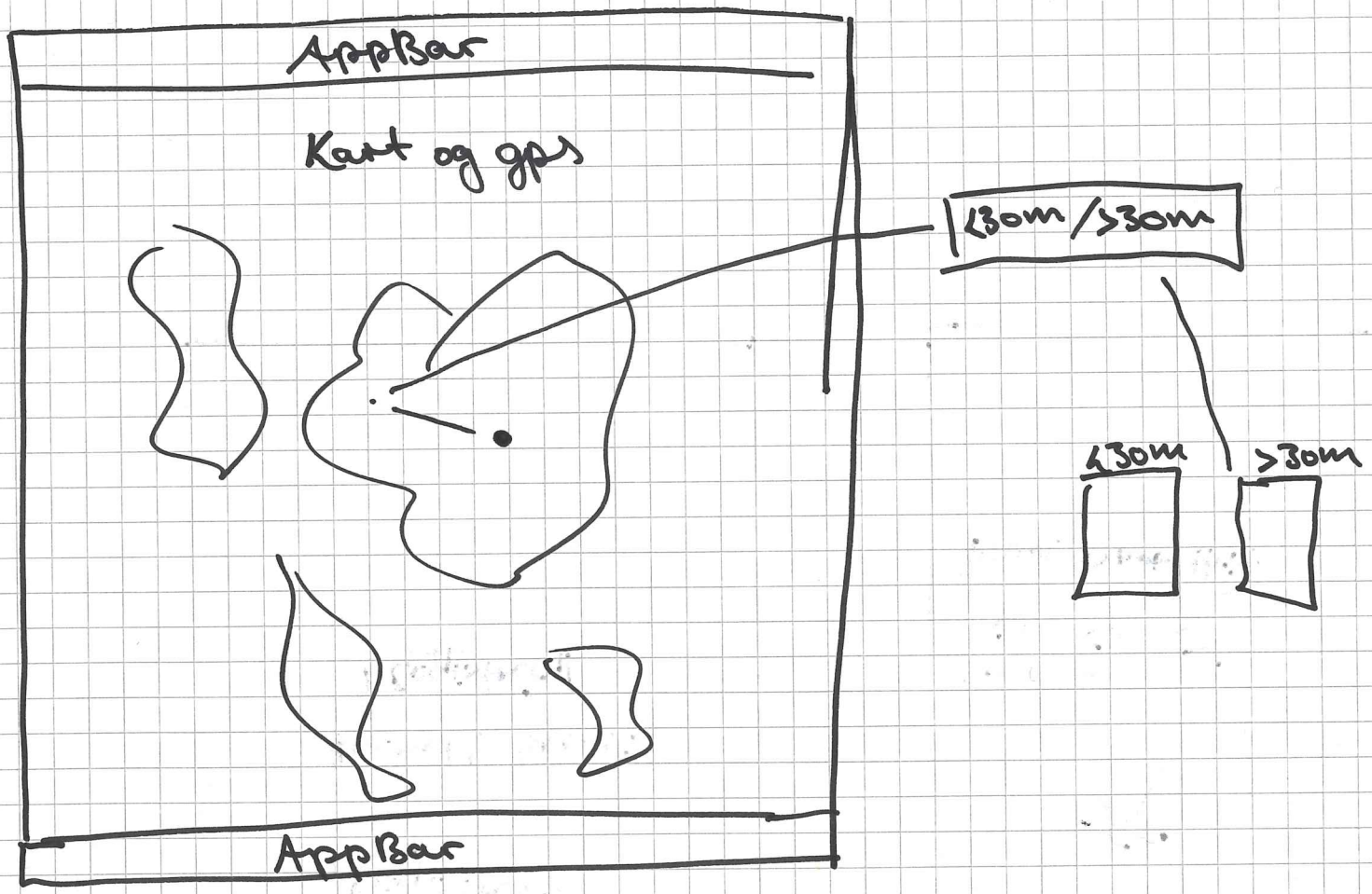
Ta bilde av sau

Mer enn 30m

Mindre enn 30m

GPS

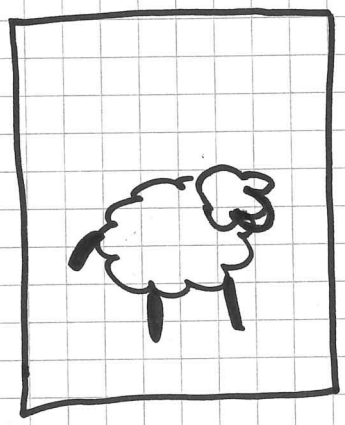
Oppsynstur



Enten AppBar eller BottomNavigation:

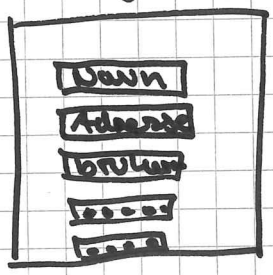
- Avslutt tur
- Data registrert
- Hjem-skjerm

Login



Login/Sign up

Sign up



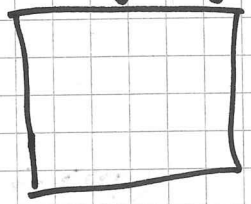
Nawkaar:

GPS

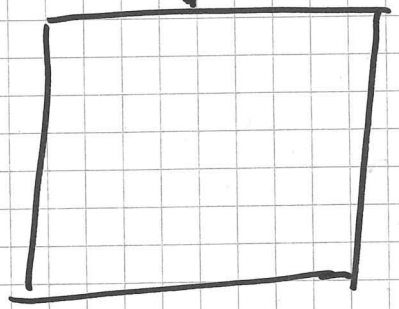
Kart

Registrer sauer

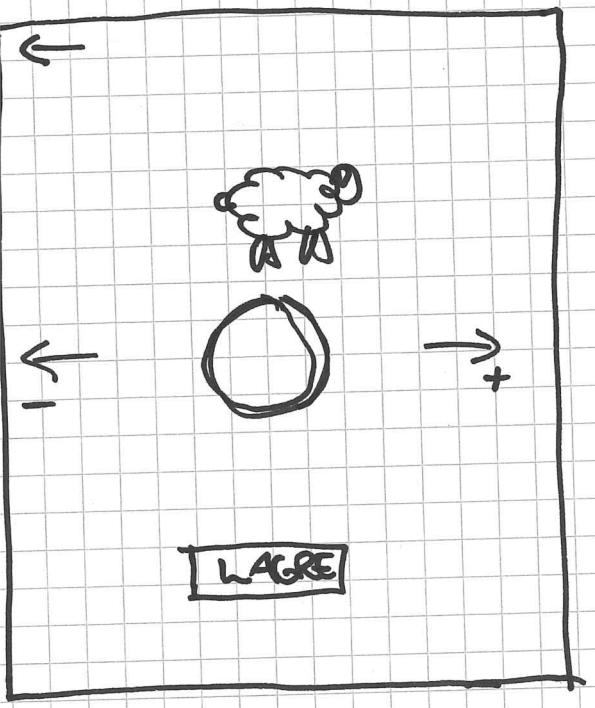
Plugin google (kanskje droppe ?)

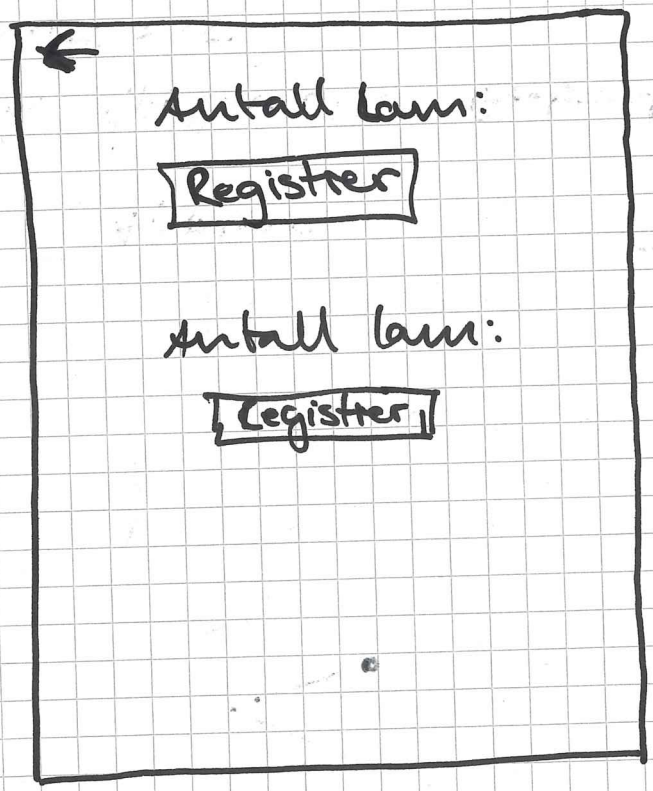


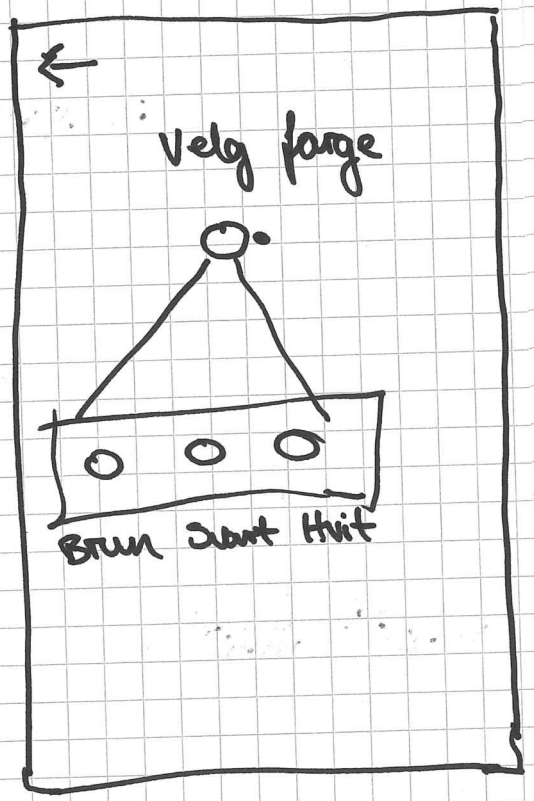
Login/Sign in



Registrering - > 30m







.2 Akronymer og ordliste

MVP = Minimal Viable Product

DOM = Document Object Model

Rendering = transformasjon

API = Application Programming Interface

View = skjermen man ser

JSON = JavaScript Object Notation

Open-source: åpen og tilgjengelig kildekode

Manager: verktøy for automatisk håndtering

Native: innfødt programvare eller dataformat for en bestemt prosessor

Ref functions: funksjoner for å aksessere *DOM* på

Corporate standard = bedriftsstandard

UI = User Interface

Display = vise frem

Container: pakker inn applikasjonskoden på en standard måte til et objekt

SQL = Structured Query Language

NoSQL = Not only Structured Query Language

Cache = mellomlager

Cachet = lagret i mellomlageret

Map = å koble en ting til en annen

Array = en organisert liste

Hashmap-struktur: datastruktur som kan koble spesifikke nøkler til spesifikke verdier

Hardware = maskinvare

Composite index = samensatt indeks

Community = fellesskap

Research = undersøke

Plugin = programvareutvidelse

Geofencing = geogjerde (geografisk avgrenset område)

Web container: ansvarlig for administrering av livssyklusen til servlets, koble en *URL* til en bestemt servlet og sikre at *URL*-forespørselen har de riktige adgangsrettighetene

Best practice: kommersielle eller profesjonelle prosedyrer som blir sett på som riktige eller mest effektive

GiB = Gibibyte

Scope = omfang

Snapshot-isolasjon = øyeblikksbildeisolering

GB = Gigabyte

Dropdown-menu = nedtrekksmeny

GPS = Global Positioning System

GLONASS = Globalnaja Navigatsionnaja Sputnikovaja Sistema (globalt navigasjonssatellittsystem som er russisk)

GNSS = Global Navigation Satellite Systems (navigasjonssatellitt)

LTE-M: laveffekts *wide area network*-radioteknologistandard

IOT = Internet of Things

Narrowband IOT: nettverksteknologi for lav strømnettverk

Proof of conceopt = konseptbevis

SDK = Software Development Kit

GeoRSS: et fornat for å kode en lokasjon som en del av en webfeed

KML = Keyhole Markup Language

GML = Geography Markup Language

WMS = Web Map Service

WFS = Web Feature Service

Schemaless = skjemaløst

Maria Vy Nguyen Vu

Digitalisering av manuell oppfølging av sau på utmarksbeite

NTNU
Kunnskap for en bedre verden