Prosjektoppgave i IN2000

Åpent case: Jaktapp med farevarsel

Gruppe 36

Daniel Faour (danieafa), Tobias Myren Foldal (tobiasmf), Emil Damsgård Knutsen (emildkn), Martin Borgaard Waage(maribwa), Waleed Nasir (waleedn) & Trygve Kjensli (trygvkj)

Fagveileder: Johanne Thunes

Institutt for informatikk

Det matematisk-naturvitenskapelige fakultet

UNIVERSITETET I OSLO

21. mai 2021

Sammendrag

I denne rapporten dokumenteres arbeidet med å utvikle en app for Android ved bruk av APIer fra Meteorologisk Institutt og Google i emnet *IN2000 Software Engineering med prosjektarbeid*. Arbeidet er utført av et team på seks studenter med tverrfaglig bakgrunn fra studielinjene programmering og systemarkitektur, design, bruk og interaksjon og digital økonomi og ledelse. Gjennom hele prosjektet er det blitt benyttet en kombinasjon av Scrum og Kanban som smidig utviklingsmetode. Det har vært et 100% digitalt samarbeid ved hjelp av verktøy som Zoom for Scrum-møter, Google Drive for deling av dokumenter og GitHub for versjonskontroll av kode. I løpet av 12 uker har teamet utviklet en app for jegere som ønsker å ha kontroll på terrenget og vindretningen for å få en vellykket jakt. Appen tilbyr dessuten farevarsel, samt et eksklusivt utvalg lokkelyder. Appen er utviklet i Android Studio med Kotlin som programmeringsspråk, og har API-nivå 23. Gjennom hele utviklingsprosessen har det blitt gjort endringer i appens design, funksjonalitet og arkitektur, samt i måten teamet har jobbet og samarbeidet på.

Innhold

1 Prosjektpresentasjon

I dette kurset har gruppen fått en semesteroppgave som går ut på å gjennomføre et app-prosjekt. Hensikten med denne oppgaven er å jobbe i team med å lage en app ut ifra et case gjennom smidige arbeidsprosesser.

Ved prosjektstart fikk gruppen et valg mellom fem konkrete case, samt et et åpent case dersom man ønsket mer rom for egen kreativitet. Fellesnevneren for alle casene var kravet om å bruke minst én datakilde fra Meteorologisk Institutt og at appen skulle utvikles for Android med Kotlin som programmeringsspråk.

Gruppen valgte åpent case. Årsaken var nettopp at gruppen ønsket mer spillerom for kreativitet, og at det fra starten av kom opp ideer om bruksområder for appen som ikke passet med de øvrige casene. Med dette som utgangspunkt endte vi til slutt opp med å lage en jakt-app som har som formål å hjelpe jegere med å lykkes på jakt. Applikasjonen benytter seg av API-et *Nowcast* fra Meteorologisk Institutt til å vise nåværende vindretning og vindstyrke for gitt lokasjon, og Google sitt *Maps*-API til å vise brukerens nåværende posisjon og på kart. Videre har appen funksjonalitet som lar brukeren sjekke om det er registrert farevarsler på sin posisjon via APIet *MetAlerts*, og å spille av et utvalg lokkelyder.

1.1 Rapportens oppbygging

Innledningsvis presenteres prosjektgruppen og dens medlemmer. Deretter følger det en beskrivelse av prosessen knyttet til valg av case, og resultatet av dette. Det formuleres en problemstilling for det valgte caset, og hvordan vi har valgt å gå frem for å løse denne.

I Kapittel 2 er brukerdokumentasjonen. Her spesifiseres først plattformen appen er utviklet for og målgruppen den retter seg mot. Videre følger en detaljert dokumentasjon av appens funksjonalitet og tilhørende skjermbilder av appen i bruk. Kapittelet avsluttes med FAQ / ofte stilte spørsmål.

Deretter følger Kapittel 3 med kravspesifikasjon og modellering. Her presenteres flere brukerhistorier og krav for applikasjonen, og det er laget modeller som fremhever ulike aspekter ved systemet og gir en helhetlig oversikt.

I Kapittel 4 er den tekniske dokumentasjonen. Innledningsvis spesifiseres systemkrav for applikasjonen og hvilken arkitektur som er valgt for kodingen. Deretter følger en teknisk dokumentasjon av hvert element i applikasjonen og hvordan disse er knytte til hverandre. Til slutt forklares hvordan appens design og utforming tar hensyn til prinsippet om universell utforming.

Testdokumentasjonen ligger i Kapittel 5. Først forklares det hva målet med testingen har vært for teamet i dette prosjektet, og hvilke testmetoder som er valgt for å nå dette målet. Underkapitlene 5.1 og 5.2 gir en nærmere beskrivelse av hvordan henholdsvis enhetstesting og integrasjonstesting er anvendt i prosjektet. Testing av ikke-funksjonelle krav er imidlertid gitt i Kapittel 6.4.

Til slutt i rapporten finnes prosessdokumentasjonen. Kapittel 6.1 gjør rede for valg av smidig utviklingsmetode og generell arbeidsform, mens Kapittel 6.2 tar for seg de ulike samarbeids-verktøyene som er brukt for organisering av gruppearbeidet og utvikling av applikasjonen. Kapittel 6.3 gir en grov oversikt over de ulike fasene i prosjektet, med en kronologisk fremstilling av hva som har skjedd når. Prosessdokumentasjonen avsluttes med Kapittel 6.4 og en refleksjon og vurdering av hele prosessen og sågar hele våren 2021. Her diskuteres hvordan samarbeidet har fungert, om kravene i kravspesifikasjonen er oppfylt, tekniske utfordringer og hva vi kunne gjort annerledes i løpet av prosjektet.

Kapittel 7 konkluderer på problemstillingen.

1.2 Om prosjektgruppen

Prosjektgruppen er sammensatt av studenter med både felles og tverrfaglig kompetanse. Tre av studentene er fra programmering og systemarkitektur, to fra design, bruk og interaksjon og én fra digital økonomi og ledelse. Den felles kompetansen kommer fra obligatoriske emner ved IFI, mens den tverrfaglige kompetansen kommer fra studieprogram-spesifikke emner og gruppemedlemmenes personlige interesser. I dette prosjektarbeidet har vi lykkes med å utnytte begge typer kompetanse.

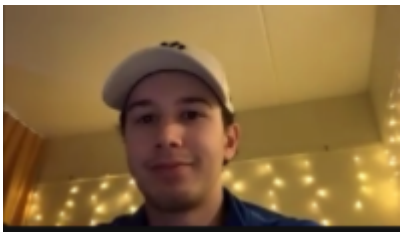
Martin Waage

Studieprogram: Informatikk: Design, bruk og interaksjon.
Interesser: Estetikk og teknologi.
Ønsker å jobbe med design. Startet som en interesse for det estetiske, men har gått gradvis over til teknologi.

Trygve Kjensli

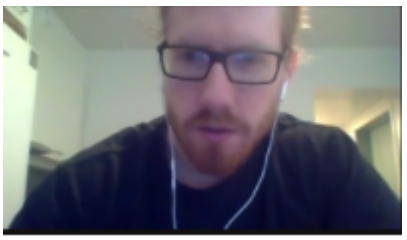
Studieprogram: Informatikk: Programmering og systemarkitektur.
Interesser: Jakt, fiske og naturfotografering.
Master i samfunnsøkonomi fra NTNU, men oppdaget etter hvert en interesse for programmering og begynte på IFI. Vil helst bli den nye Lars Monsen.

Daniel Faour



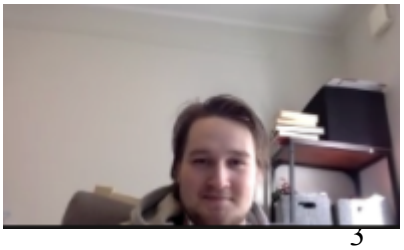
Studieprogram: Informatikk: Design, bruk og interaksjon.
Interesser: Data og teknologi.
Erfaring innen design av nettsider og bruk av digitale verktøy som blant annet Photoshop og Sony Vegas Pro.

Emil D. Knutsen



Studieprogram: Informatikk: Programmering og systemarkitektur.
Interesser: Programmering, musikk
Har en mastergrad som byggingeniør fra NTNU. Fattet gjennom studiet en interesse for programmering, og begynte derfor på IFI.

Tobias M. Foldal



Studieprogram: Informatikk: Programmering og systemarkitektur.

Interesser: Programmering og datateknologi.

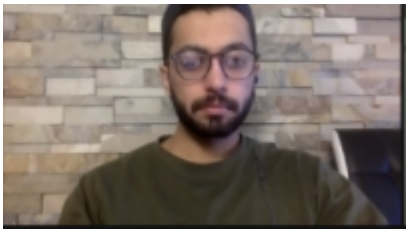
Interessert i programmering og datateknologi, begynte derfor på IFI. Har en anelse av erfaring innenfor webdesign.

Waleed Nasir

Studieprogram: Informatikk: Digital økonomi og ledelse

Interesser: Teknologi og industri.

Ønsker å jobbe med utvikling og design av informasjonsteknologier i fremtiden.



1.3 Valg av case

Prosessen med å velge et case startet med et møte der alle de ulike casene ble gått igjennom i plenum, mens det ble diskutert de umiddelbare ideene medlemmene i teamet fikk. Etter idemyldringen ble alle ideer lagt inn i et felles dokument, hvor det også var mulighet for å komme med flere ideer frem til det neste møtet. Det skulle være lav terskel for å komme med ideer, og det trengte ikke være spesielt utfyllende eller gjennomførbart. Tanken var at selv om ideene kanskje ikke ville fungere, kunne de bidra til at noen andre fikk bedre ideer.

Det ble etter hvert veldig mange ulike ideer, spesielt for det åpne caset. Derfor ble det nødvendig å sortere og eliminere de casene som ikke var interessante eller gjennomførbare. Det var en del ideer som lignet på hverandre, så de ble sortert etter tema. Et eksempel på dette er at det var flere ulike ideer for en tur-app, som inkluderte topptur-samling, hytte til hytte-planlegger, mulighet for å planlegge tur i Oslomarka etc.

Det var flere gode ideer som ville være interessant for teamet å jobbe med, selv etter den første elimineringsrunden. Til slutt sto det mellom syv ulike caser, både av de ferdiglagde casene, og de ulike ideene til åpent case. For å velge mellom disse, ble det gjennomført en digital avstemning der alle i teamet hadde opptil tre stemmer hver som de kunne bruke på de casene de ønsket å jobbe med. Resultatene kom ikke frem til noe entydig svar, da flere av ideene hadde like mange stemmer, og det var liten forskjell på resultatene. På det neste møtet ble det gjennomført en diskusjon der det ble snakket om de ulike ideene, og ideene ble eliminert frem til man satt igjen med jakt-app som det valgte caset.

SVARVALG	SVAR	
1. Case 2 - Farevarsler	50,00%	3
2. Case 5 - Flynerd-app	50,00%	3
3. Nordlys-app	0,00%	0
4. Artobservasjon-app	33,33%	2
5. Jakt-app	66,67%	4
6. Fiske-app	50,00%	3
7. Tur-app	50,00%	3
Totalt antall respondenter: 6		

Figur 1.1: Resultat fra avstemning om case.

1.4 Problemstilling og vår løsning

1. april 2020 sto 516 108 norske menn og kvinner registrert som jegere i Jegerregisteret (SSB, 2020). Ikke alle disse er aktive jegere, men det er grunn til å tro at de av dem som er det, har med seg mobiltelefonen på jakt. Måtene å utøve jakt på er naturligvis mange og varierte, men felles for dem alle er at jegeren må komme seg nærme nok viltet til å kunne avfyre et dødelig skudd. Dette kan enten oppnås ved å smyge seg innpå viltet, eller la viltet komme til deg. I begge tilfeller er det helt avgjørende å ikke bli avslørt av viltets skarpe sanser. De fleste jaktbare arter i Norge har sannsynligvis en nedarvet frykt for menneskets lukter, lyder og skikkelse, og vil flykte umiddelbart hvis en potensiell fare oppdages.

Det jegeren har minst kontroll på i denne sammenheng, er nettopp sin egen lukt. Lukten fraktes med vinden, og kan skremme viltet lenge før jegeren har oppdaget det. En jeger som forsøker å smyge seg innpå viltet, vil sørge for å gå innpå fra en kant som gjør at jegeren har vinden i ansiktet. En jeger som venter på viltet på en post, vil sørge for å plassere seg slik at vinden ikke bærer i retningen jegeren forventer at viltet skal komme fra.

Målet prosjektet var å utvikle en applikasjon som skal hjelpe jegere til å oppnå en vellykket jakt tross jegerens nevnte utfordringer. Dette har vi gjort ved å lage en applikasjon som hjelper brukeren med å holde styr på vindretningen og vindstyrken.

2 Brukerdokumentasjon

2.1 Plattform og tilgjengelighet

Applikasjonen *JaktApp* er utviklet for Android etter semesteroppgavens krav. Prosjektet benytter seg av Android med minimum API-nivå 23 (Marshmallow). Dette betyr at appen kan kjøres på Android 6.0 og nyere, og dermed ca. 62.6% av brukere innenfor Android-plattformen (Stenslund & Almås, 2021). Valget er tatt på bakgrunn av en avveining mellom nytten av moderne funksjonalitet, sikkerhet og hvor bredt vi ønsker å nå.

Applikasjonen kan kjøres på Android-enheter eller på en emulator, men er per tid ikke allment tilgjengelig på Google Play. Det kreves internett-tilgang for å benytte seg av appens kart- og vindvarslingsfunksjonalitet, mens lokkelydfunksjonaliteten er tilgjengelig offline.

2.2 Målgruppe

I dette prosjektet har gruppen valgt å sette fokuset på unge voksne i alderen mellom 20 og 30 år, med interesse for jakt. Formålet med prosjektet er å bidra til en lettere jakt-opplevelse gjennom mobiltelefonen. Mobilforbruket øker betydelig mer utover årene på grunn av den raske utviklingen av smarttelefoner og de nye bruksområdene som dukker opp. Spesielt blant unge voksne under 30 år, som bruker mobiltelefonen nesten ca. 6-8 timer om dagen (Tina Brock, 2020). Det er en gruppe som potensielt er mer interessert i å ta bruk i en app når de er på jakt. Det er også en gruppe av jegere som er relativt unge, og kanskje mindre erfarne enn sine eldre jaktkamerater. Derfor er det også mulig at denne gruppen vil ha en større nytteverdi av å bruke denne appen.

2.2.1 Intervju med målgruppen

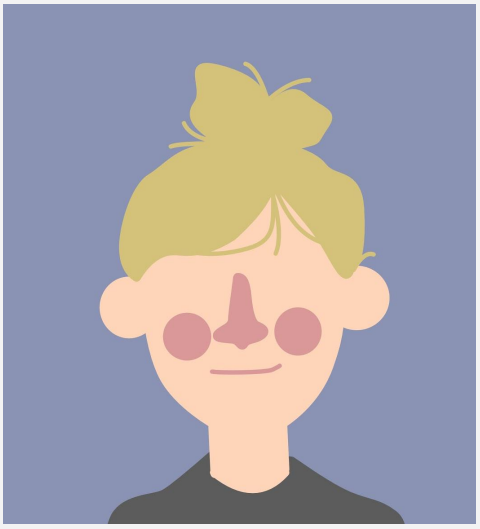
Selv om vi hadde en jeger i teamet som hadde formeninger om hva en jaktapp burde inneholde, ønsket vi å få ekstern input fra en annen i målgruppen. Vi bestemte oss derfor for å gjennomføre et intervju. Vi vurderte at vi ville få problemer med å få mange nok tilbakemeldinger hvis vi i stedet hadde valgt en survey. Intervjuobjektet besvarte generelle spørsmål rundt hvordan en jakt foregår og hvilke hjelpemidler som benyttes underveis, samt hvilke funksjoner som kan være nyttige i en app. Informasjonen fra dette intervjuet bidro til å velge ut hvilke funksjoner appen skulle ha. Det ble gjennomført et semistrukturert intervju

hvor det var planlagt hva Teamet i utgangspunktet ønsket svar på, men som også ga rom for å følge opp temaer som kunne være interessante underveis (Lazar, Feng & Hochheiser, 2017, s.198-199). Intervjuet ble gjort via Zoom, som har vært eneste mulighet i forbindelse med den pågående covid-19 pandemien.

I intervjuet ble det forklart at det på gåsejakt blir benyttet høyttalere med gåselyder for å lokke de til seg. Intervjuobjektet sa også at de brukte noe som heter "blinds" som er en form for kamuflasje for å holde seg skjult for dyra. På spørsmål om hvilken informasjon som er nyttig, kom det fram at dyrenes oppførsel, vindretning, temperatur og terreng er nyttig å ha kunnskap om. For eksempel så er vindretningen viktig fordi gjess alltid lander *mot* vindretningen. Temperatur og vær er viktig fordi dyrene kan befinne seg på ulike steder avhengig av været. Terrenget er nyttig å ha informasjon om hvis man ikke er kjent i området, og fordi man ikke kan se grensene til grunneieren. På spørsmål om hvilke app-funksjoner intervjuobjektet kunne se for seg i en jakt-app, nevnte vedkommende kart og kompass, vær, å kunne kommunisere med andre og f.eks merke av på kart så de andre ser, liste av ulike dyrespor, og dyrelyder.

2.2.2 Brukerscenario

Det ble laget to personaer med brukerscenarioer. Disse er ment å gi et bilde av målgruppen, samt hvilke problemer de ulike funksjonalitetene skal løse (Preece, Rogers & Sharp , 2015, s. 358-359 & 371). Personaene ble basert på intervjuet som ble gjennomført, og det ene er nedenfor mens det andre finnes i Appendiks H.



Kristin

Alder: 30
Bosted: Jessheim
Sivilstatus: Samboer
Yrke: Institusjonskokk

Biografi

Kristin bor på Jessheim sammen med samboeren sin Thomas, og hunden deres Balder. Hun jobber som kokk på et sykehjem, og liker å ta bilder på fritiden. Kristin tok nylig jegerprøven, etter å ha vært med samboeren på jakt for å ta bilder flere ganger tidligere

Motivasjoner	Frustrasjoner
- Samboer og hund - Lære nye ting - God mat	- Rotete kjøkken - Dyrt å komme inn på boligmarkedet

Scenario

Kristin er på gåsejakt med samboer og to av hans venner. De har fått lov av nabobonden å jakte på gås på jordet hans.
Rett før soloppgang setter de ut kamuflasje-senger og gåsebulvaner midt utpå jordet. Kristin sjekker vindretningen for å plassere kamuflasjen og bulvanene i riktig retning, for hun vet at gjessene alltid kommer inn for landing *mot* vindretningen. Deretter venter de på gåseflokkene som pleier å komme trekkende på morningen. Når flokken kommer til syne, spiller Kristin av gåselyder fra en bærbar høyttaler slik at gåseflokken blir lokket til å lande blant de falske gjessene de har satt ut. Akkurat idét gjessene skal lande, spretter de fire opp fra kamuflasjen samtidig og skyter.

2.3 Funksjonalitet

Når applikasjonen er lastet ned og åpnes for første gang, vises en pop-up med valg for tillatelse til bruk av stedstjenester. Etter at tillatelse til bruk av stedstjenester er gitt eller avslått, kommer man til kartsiden (levert av Google). Kartet har funksjonalitet for vind- og farevarsler, men disse er avhengige av tillatelse til bruk av stedstjenester, samt internettilgang. Ved senere oppstart av appen, er alltid kartet det første man møter. I bunnen av enhver side i appen, finnes det en meny / navigasjonsbar. Fra denne kan man raskt bytte mellom appens tre sider; Kart, Innstillinger og Lokkelyder. Første gang appen åpnes, vil det dukke opp en velkomstskjerm som viser brukeren hvor man skal trykke for å benytte seg av funksjonaliteten i kartet.

2.3.1 Kart

Brukeren *har ikke* tillatt bruk av stedstjenester.
Et kart vil vises på skjermen. Brukeren kan benytte siden som et standard Google-kart og kan zoome inn og ut, men ingen annen funksjonalitet utover dette støttes.

Brukeren *har* tillatt bruk av stedstjenester.
Et kart vil vises på skjermen, og brukerens posisjon vil vises som en blå prikk. Kartet vil være zoomet inn på denne prikken. Prikken har en pil som indikerer hvilken retning brukeren holder enheten. Øverst til høyre på skjermen finnes en knapp som fører kartet tilbake til brukerens posisjon (den blå prikken).

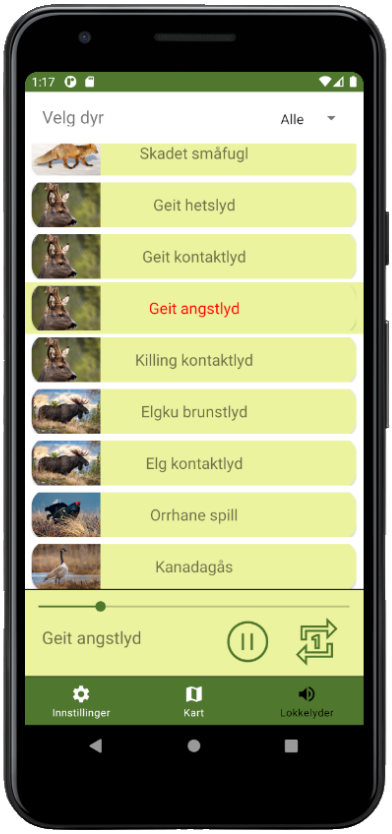
Rett under posisjonsknappen er det en grønn knapp som skrifter karttype mellom ordinært kart og satellittkart. Under denne igjen er det en gul trekantformet knapp som brukeren kan trykke på for å se om det er registrert noen farevarsler i fylket vedkommende befinner seg i. Denne knappen vises såfremt brukeren ikke har skrudd av farevarsler i appens innstillinger.

Varsel for vindretning og vindstyrke på brukerens posisjon er presentert øverst på skjermen. Her er vindretningen visualisert med en pil og gitt med tekst (for eksempel NV (nord-vest)). Vindstyrken er gitt i meter per sekund.

Figur 2.1: Brukergrensesnitt for kart og farevarsel

2.3.2 Lokkelyder

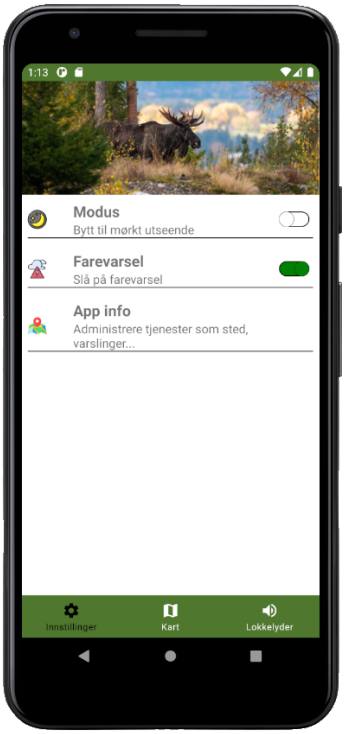
Til høyre på navigasjonsbaren er det en knapp for å komme til siden med lokkelyder. På denne siden møtes brukeren med en liste over alle lokkelydene som er lagt inn i systemet. Hvilket dyr lokkelyden er ment til å lokke, er gitt av bildet på lyden. Øverst til høyre på siden kan brukeren dessuten trykke på en pil for å få opp en meny for valg av dyr. Ved valgt dyr, vil kun lydene som er ment for å lokke dette dyret vises i listen over lokkelyder. Når brukeren klikker på en lyd i listen, vil lyden markeres og tittelen på lyden dukker opp på lydavspilleren nederst på skjermen. Lyden er nå klar til avspilling, og kan settes igang ved å trykke på play-symbolet. Trykker man på den samme knappen enda en gang, pauses avspillingen. Til høyre for play/pause-knappen er det en knapp for automatisk avspilling av den valgte lyden i løkke.



Figur 2.2: Brukergrensesnitt for avspilling av lokkelyd

2.3.3 Innstillinger

Til venstre på navigasjonsbaren er det en knapp for å komme til innstillinger. Her kan brukeren velge mellom lyst og mørkt fargetema for applikasjonen, velge å skru av eller på stedstjenester og velge å skru av eller på farevarsler.



Figur 2.3: Brukergrensesnitt for innstillinger

2.4 FAQ

Må jeg tillate bruk av stedstjenester?

-Nei, men flere av appens hovedfunksjonaliteter avhenger av det. Du kan ikke få informasjon om vind- eller farevarsler uten å gi tillatelse til bruk av stedstjenester. Lokkelydene fungerer derimot utmerket uten informasjon om din posisjon.

Hvor er værvarselet hentet fra?

-Meteorologisk Institutt sin varslingstjeneste "Nowcast".

Hvor nøyaktig er vindvarselet?

-Det gir en indikator på generell vindretning, men husk at formasjoner i landskapet, trær og bygninger rundt deg kan påvirke vinden lokalt.

Lagres data om meg i applikasjonen?

-Nei.

3 Kravspesifikasjon og modellering

Basert på intervjuet som ble gjennomført, personaene og deres scenarioer (se Kapittel 2.2 Målgruppe), formulerte vi et sett med brukerhistorier og rangerte disse etter grad av viktighet. Disse ga grunnlag for en liste med funksjonaliteter som vi kunne kategorisere som enten "nice to have" eller "must have". Både brukerhistoriene, og dermed kravene, har vært dynamiske og formulert relativt åpent - de har blitt endret etter hvert som vi utførte både egne tester og brukertester. Dette har vært i tråd med prinsippet om smidig utvikling.

Det var mange ideer å velge mellom, og på grunn av tidsbegrensningen til prosjektet er det mange funksjonaliteter som er "nice to have" som ikke er blitt realisert. Blant disse er blant annet funksjonalitet for å kunne kommunisere med andre jegere, markere observerte dyr på kartet, registrering av fellinger etc.

I Kapittel 3.1 og 3.2 presenterer vi brukerhistoriene og kravene vi endte opp med å velge. Basert på brukerhistoriene og kravene som faktisk ble implementert, har vi laget ulike modeller som fremhever ulike aspekter ved systemet og interaksjonen mellom systemet og dets aktører. Use case-diagram, tekstlig beskrivelse, sekvensdiagram og klassediagram gir til sammen en helhetlig representasjon av systemet.

3.1 Brukerhistorier

I tabellen nedenfor er alle brukerhistorier med høy prioritet implementert i appen, mens for brukerhistoriene med middels prioritet er det varierende grad av implementasjon. Brukerhistoriene med lav prioritet er ikke implementert, men det kan være lagt til rette for enkel implementasjon ved utvidelse av appen på et senere tidspunkt.

#	**Brukerhistorier**	**Prioritet**
1	Som bruker ønsker jeg å kjapt kunne se vindretningen på min posisjon slik at jeg vet hvilken retning menneskelukten min bærer av sted.	Høy
2	Som bruker ønsker jeg å sjekke om det er farevarsler for området jeg skal jakte i slik at jeg kan unngå unødvendig risiko.	Middels
3	Som bruker ønsker jeg å kunne spille av lokkelyder fra mobiltelefonen slik at jeg kan lokke viltet inn på skuddhold.	Høy

4	Som bruker ønsker jeg å navigere til nøkkelfunksjoner i appen med færrest mulig klikk slik at appen er enkel å håndtere med kalde fingre.	Høy
5	Som bruker ønsker jeg å komponere mine egne spillelister av lokkelyder slik at jeg kan sette sammen lydsekvenser som min erfaring tilsier at fungerer best.	Lav
6	Som bruker ønsker jeg å spille inn mine egne lokkelyder slik at jeg kan tilpasse lydene nøyaktig etter mitt bruk og forholdene der jeg jakter.	Lav
7	Som bruker ønsker jeg å kunne sette i gang lydavspilling og deretter låse skjermen slik at jeg kan ha enheten i lomma.	Middels
8	Som bruker ønsker jeg å kunne styre lydavspilling fra låseskjermen slik at jeg raskt kan starte eller stoppe lyden på riktig tidspunkt.	Middels
9	Som bruker ønsker jeg å kunne endre fargeinnstillinger i appen fordi jeg jakter på forskjellige tider av døgnet med ulike lysforhold.	Middels
10	Som bruker ønsker jeg å kunne endre valg for bruk av stedstjenester slik at jeg har kontroll over min egen data.	Høy
11	Som utvikler og vedlikeholder ønsker jeg at koden er kommentert nøye slik at den er enkel å forstå.	Høy
12	Som bruker ønsker jeg en guide som forteller meg hvordan jeg bruker appen funksjonalitet.	Middels

3.2 Krav

3.2.1 Funksjonelle krav

De funksjonelle kravene som er listet opp i tabellen nedenfor er formulert ut ifra brukerhistoriene i forrige delkapittel. Tabellen gir en oversikt over hvilke krav som er implementert og hvilke som ikke er det. Noen av kravene er listet opp selv om de viste seg å ikke være aktuelle å implementere i de siste iterasjonene av prosjektet. Dette gjøres for å vise hva vi ville prioritert dersom det var mer tid til rådighet.

#	**Funksjonelle krav**	**Status**
1	Appen skal gi mulighet til å vise brukerens posisjon på kart.	Implementert
2	Appen skal gi vindretning og vindstyrke for gitt posisjon i sanntid.	Implementert

3	Appen skal kunne spille av en lokkelyd med maksimalt tre klikk på enheten.	Implementert
4	Appen bør kunne spille inn lokkelyder.	Ikke implementert
5	Appen bør kunne lage spillelister av lokkelyder.	Ikke implementert
6	Appen skal kunne styre lydavspilling på låst skjerm.	Ikke implementert
7	Appen skal kunne informere brukeren om farevarsler på sin posisjon.	Implementert
8	Appen skal kunne endre fargemodus mellom lyst og mørkt tema.	Implementert
9	Appen skal kunne endre valg for stedstjenester.	Implementert
10	Appen skal fungere selv om skjermen roteres	Implementert
11	Lydavspilling skal fortsette selv om brukeren går ut av appen eller låser skjermen.	Implementert
12	Appen skal ha en velkomstskjerm som gir brukeren informasjon om funksjonalitet i appen.	Implementert

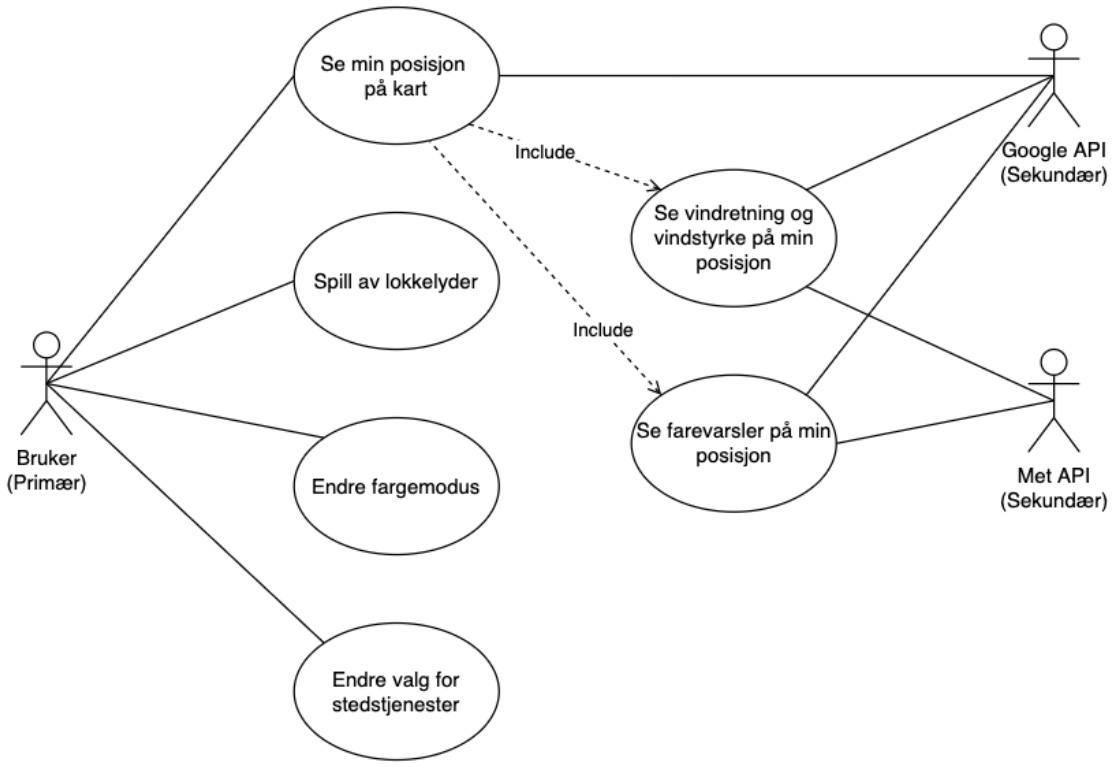
3.2.2 Ikke-funksjonelle krav

I tabellen nedenfor er det listet opp ikke-funksjonelle krav til appen. Noen av disse kravene er gitt i forkant fra oppgavebeskrivelsen til prosjektet, mens andre er utformet ut ifra hva som er ansett som god skikk eller hva som er spesifikt relevant for typen app vi har valgt å lage.

#	**Ikke-funksjonelle krav**	**Type**
1	Appen skal være universelt utformet.	Produktkrav
2	Appen skal ha et enkelt brukergrensesnitt. Brukeren skal ikke trenge ekstern opplæring for å benytte seg av alle appens funksjonaliteter.	Produktkrav
3	Appen skal ikke lagre posisjonen til brukeren.	Produktkrav
4	Appen skal laste inn innhold for kartsiden på under 4 sekunder.	Produktkrav
5	Appens offline-funksjoner skal fungere når brukeren ikke har internett-tilgang.	Produktkrav

6	Appen skal ikke bruke mer enn 50 MB lagringsplass.	Produktkrav
7	Appen skal utvikles ved bruk av smidig arbeidsmetodikk.	Organisatorisk krav
8	Appen skal utvikles i Android Studio med Kotlin som programmeringsspråk.	Organisatorisk krav
9	Koden skal kommenteres nøye slik at en utvikler vil forstå, vedlikeholde og videreutvikle koden uten vanskeligheter.	Organisatorisk krav
10	Koden skal ha en arkitektur (MVVM) som gjør vedlikehold og videreutvikling i fremtiden enklere.	Organisatorisk krav

3.3 Use case-diagram



Figur 3.1: Use case-diagram

3.4 Tekstlig beskrivelse

Nedenfor er tekstlige beskrivelser for alle interaksjonene som er vist i use case-diagrammet. Disse gir en mer presis beskrivelse av flyten i systemet og eventuelt alternativ flyt hvis det skjer avvik fra det som regnes som hovedflyten.

Navn:	**Se min vindvarsel og farevarsel på min posisjon på kart.**
Aktører:	Bruker, System, Google API, Met API.
Prebetingelse:	Har lastet ned appen "NN"
Postbetingelse:	Se brukers nåværende posisjon på kart, vindvarsel på denne posisjonen og eventuelle farevarsler i fylket som brukeren befinner seg i.
Hovedflyt:	1. Bruker: Åpner appen. 2. System: Viser informasjonsside. 3. Bruker: Trykker OK. 4. System: Ber om tillatelse til å bruke stedstjenester. 5. Bruker: Trykker "gi tillatelse". 6. System: Zoomer inn til riktig område på kartet og viser blå prikk på brukerens posisjon. 7. System: Viser pil for vindretning og tall for vindstyrke. 8. System: Viser pop up-symbol for farevarsel. 9. Bruker: Trykker på pop up-symbol for farevarsel. 10. System: Viser liste med farevarsler for fylket brukeren befinner seg i.
Alternativ flyt 1:	Steg 5: Brukeren trykker "avvis" A.5.1 System: Viser kart uten brukerens posisjon. A.5.2 System: Vind- og farevarsel vises ikke.
Alternativ flyt 2:	Steg 6: Ikke kontakt med Google-API A.6.1 Google API returnerer NULL. A.6.2 System: Viser Toast "Kan ikke laste inn kart".
Alternativ flyt 3:	Steg 7: Ikke kontakt med Met-API A.7.1 Met API returnerer NULL. A.7.2 System: Viser data fra forrige kall, eventuelt ingenting hvis første kallet feilet.
Alternativ flyt 4:	Steg 8: Ingen farevarsler registrert i brukerens fylke. A.8.1 Met API returnerer en tom liste med farevarsler.

Navn:	**Spill av lokkelyder**
Aktører:	Bruker, System
Prebetingelse:	Har lastet ned appen og er inne på siden “Lokkelyder”
Postbetingelse:	Lyd spilles av på enheten.
Hovedflyt:	1. Bruker: Velger kategori fra Spinner, eller lar den stå på “Alle”. 2. System: Viser en liste med lokkelyder innenfor valgt kategori i et RecyclerView. 3. Bruker: Trykker på ønsket lokkelyd. 4. System: Forbereder tilhørende lydfil i en MediaPlayer. 5. Bruker: Trykker på play-symbol. 6. System: Spiller av lydfil.
Alternativ flyt:	Steg 6: Lydavspilling foregår allerede i en annen applikasjon. A.6.1 Lokkelyden spilles av over den andre lyden.

Navn:	**Endre fargemodus**
Aktører:	Bruker, System
Prebetingelse:	Har lastet ned appen og er inne på siden “Innstillinger”. Fargemodus i appen står på “lyst modus”
Postbetingelse:	Fargemodus er endret i appen.
Hovedflyt:	1. Bruker: Trykker på slider som aktiverer mørkt modus. 2. System: Endrer fargekoden i hele appen til mørkt tema.

Navn:	**Endre valg for stedstjenester**
Aktører:	Bruker, System
Prebetingelse:	Har lastet ned appen og er inne på siden “Innstillinger”
Postbetingelse:	Tillatelse for stedstjenester er tillatt eller avvist.
Hovedflyt:	1. Bruker: Trykker på knapp “app info”. 2. System: Åpner Info om appen i enhetens innstillinger. 3. Bruker: Trykker på “Tillatelser”. 4. System: Viser liste over tillatte og avviste valg. 5. Bruker: Velger å tillate eller avvise bruk av stedstjenester.

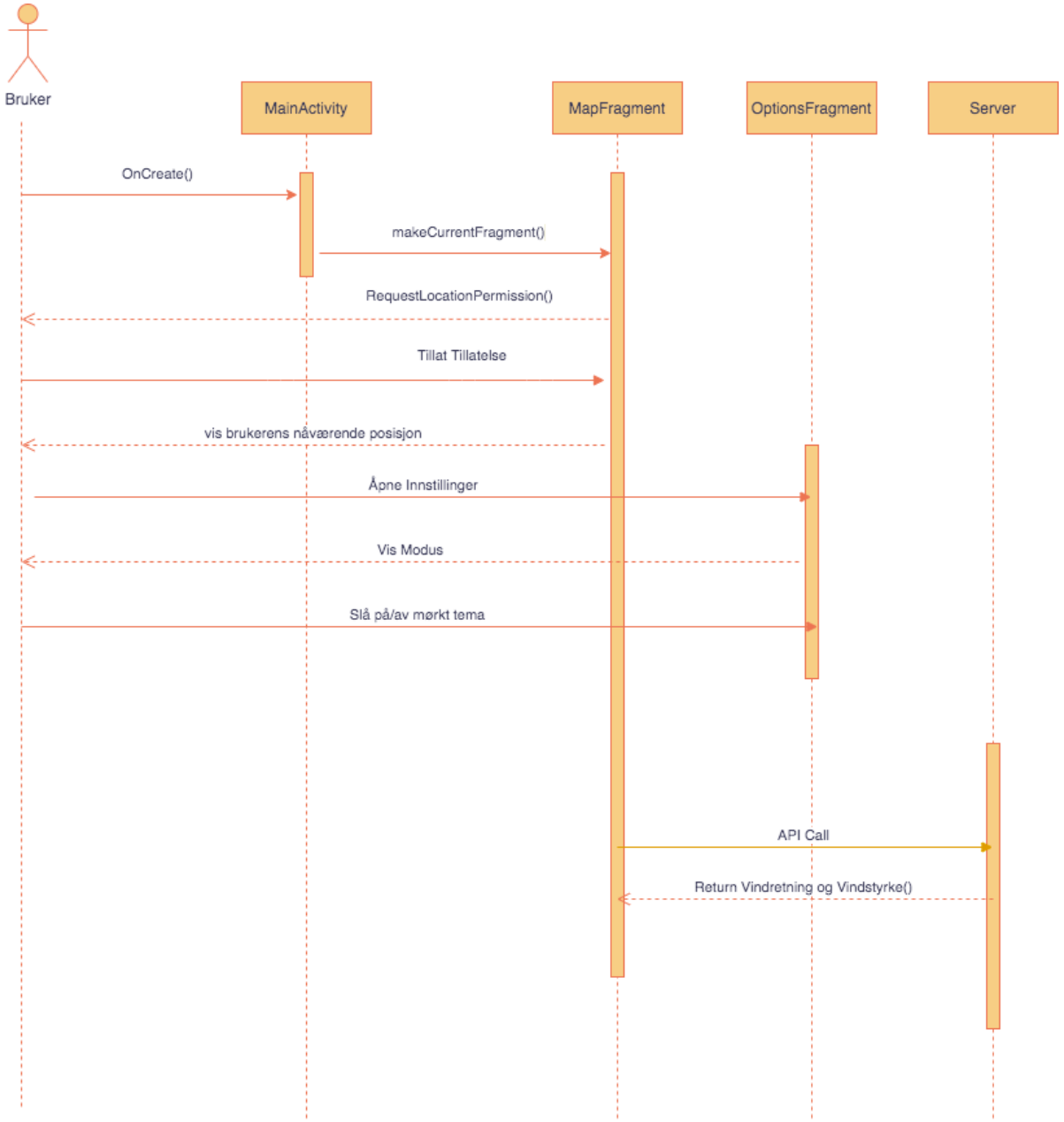
3.5 Sekvensdiagram

Vi lagde sekvensdiagram for noen av de viktigste use casene for å modellere flyten i use casene. De har blitt oppdatert underveis i prosjektet.

Se min posisjon på kart

Endre fargemodus

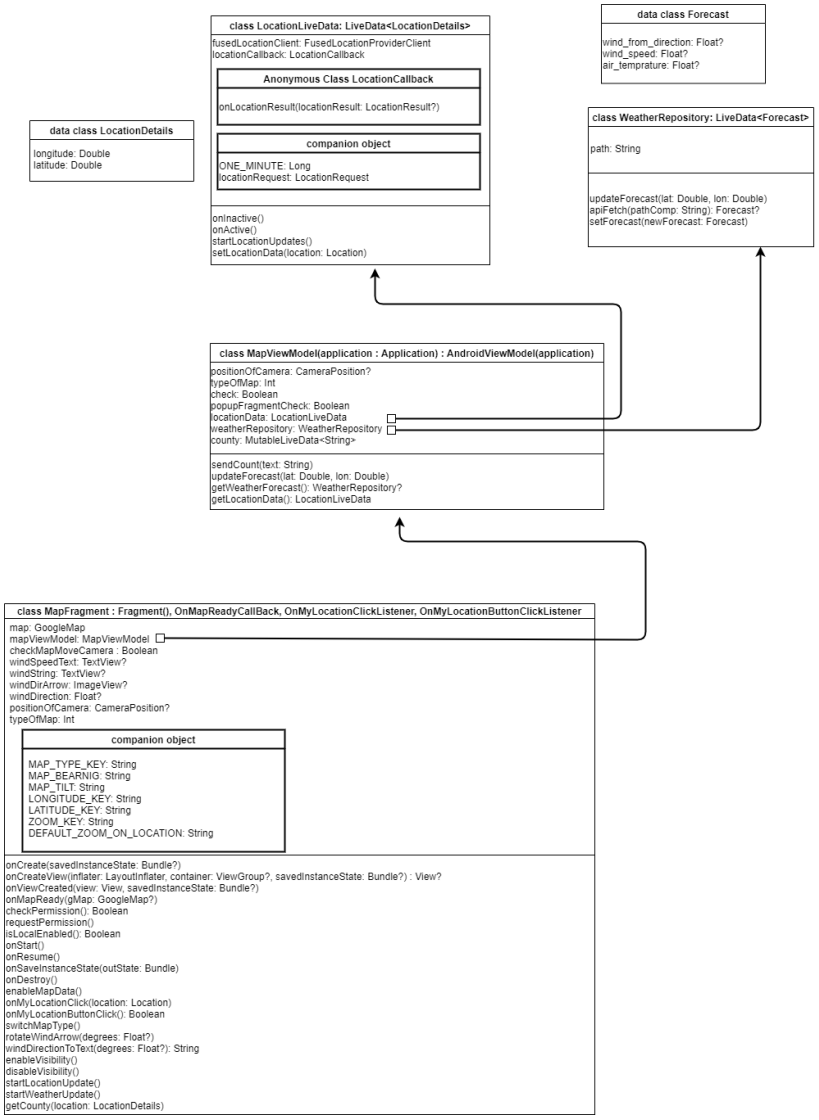
Se vindretning og vindstyrke på min posisjon



Figur 3.2: Sekvensdiagram

3.6 Klassediagram

Figur 3.3 viser et klassediagrammet for MapFragment, mens klassediagram for de øvrige fragmentene finnes i Appendiks B.



Figur 3.3: Klassediagram, MapFragment.

4 Produktdokumentasjon

Den følgende tekniske dokumentasjonen skal være til hjelp for oss selv og andre utviklere for å skjønne hvordan de ulike komponentene i applikasjonen er bygget opp og fungerer. Innledningsvis listes det opp hvilke krav som må og bør være oppfylt for at applikasjonen skal kunne kjøres. Deretter gis et overordnet bilde av hvordan arkitekturen i kildekoden er, før vi dykker ned i hvordan hver av funksjonalitetene i appen er bygget opp og fungerer "under panseret". Til slutt diskuteres det hvordan løsningene som er valgt svarer på ikke-funksjonelle krav om universell utforming.

4.1 Systemkrav

Kravene som stilles til maskinvaren for at applikasjonen skal kjøre er:

- Android som operativsystem.
- Støtte for minimum API-nivå 23.
- 50 MB tilgjengelig lagringsplass.

I tillegg bør følgende krav være oppfylt:

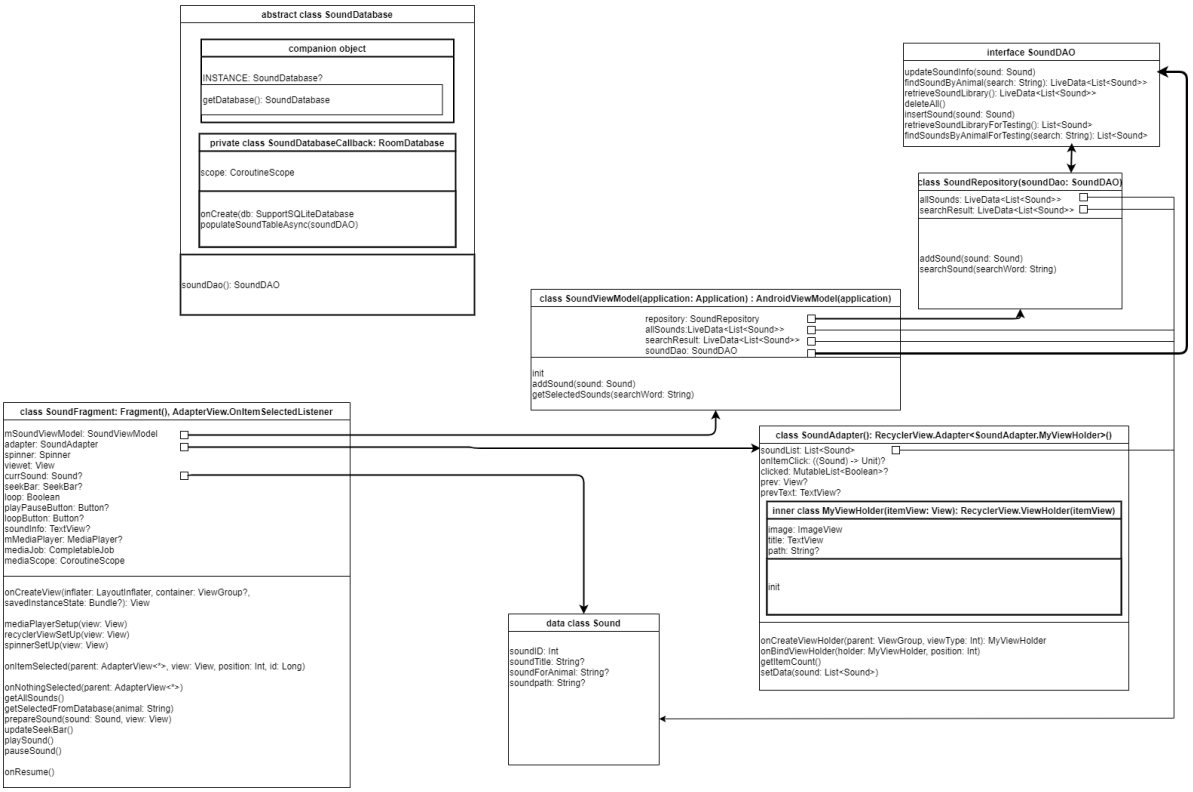
- Stedstjenester på enheten er aktivert.
- Enheten har internett-tilgang.
- Tilgang på serveren som applikasjonen er hostet på (UiOs proxy server).

4.2 Arkitektur og design patterns

Applikasjonsarkitektur spiller en viktig rolle når det kommer til å strukturere kildekoden slik at man oppnår lav kobling og høy kohesjon. Kobling er et mål på hvor sterkt knyttet et objekt er til andre objekter, og det er et mål å begrense antallet av slike avhengigheter. Kohesjon handler om ansvarsområdet til et objekt, og det er ønskelig at hvert objekt har et definert og begrenset ansvarsområde (Sommerville, 2016, s.170). Det finnes flere arkitekturer og design patterns som hjelper å sikre disse målene, som for eksempel Model View Controller (MVC) og Model View ViewModel (MVVM).

For dette systemet er det valgt å bruke en MVVM-arkitektur. Det innebærer at man separerer i tre lag; Model, View og ViewModel. Model styrer dataene, hvordan de lagres og

aksesseres. View viser dataene fra Model, og samler input fra bruker. ViewModel tolker inputene fra bruker og hvilke data som er relevante. Den styrer hvilke data Viewet skal vise (Martini, 2021, s.48). Begrunnelsen for dette valget, er først og fremst at det er anbefalt for android. Det vil skape en konsistent og pålitelig brukeropplevelse, også når brukeren kommer tilbake til appen ved senere anledninger (Android developers, 2021). Det er viktig for jakt-appen at systemet ikke glemmer hvilke innstillinger som brukeren har aktivert, og at man unngår at alt starter på nytt mellom aktivitetene. Det er også mye ulike data som tas inn i appen, som for eksempel værdata, farevarsler og posisjon, og ViewModel er hensiktsmessig for å håndtere dette på en ryddig måte. Nedenfor vises klassediagrammet for SoundFragment som illustrerer bruken av MVVM-arkitekturen i applikasjonen.



Figur 4.1: Klassediagram, SoundFragment.

4.3 Funksjonalitet

4.3.1 Funksjonalitet for Google Maps

Funksjonaliteten for kartet er bygget på Google Maps-plattformen. Vi valgte å benytte denne plattformen fordi det var det mest vanlige kartet å implementere, og dermed plattformen med størst tilgang på ressurser for å løse eventuelle problemer. Det er to måter å implementere et Google Map i en app på; man kan enten benytte seg av et SupportMapFragment, eller et MapView. Appen benytter et SupportMapFragment siden det håndterer kartets lifecycle selv, og dermed er enklere å jobbe med. Kartet blir asynkront lastet inn i onCreateView() metoden. Når kartet er ferdig lastet inn kaller den på onMapReadyCallback()-metoden, og kartet, av typen GoogleMap, blir lagret i variabelen *map*. Dette gjør at kartet kan mainpuleres senere, som for eksempel ved å vise et kart av satellittbilder.

I enableMapData() aktiveres My Location layer - en tilleggsfunksjon på Google Maps - som viser brukerens posisjon som en blå prikk på kartet. My Location layer har også funksjonalitet for knappen som sentrerer kameraet på brukerens posisjon, gjennom setOnMyLocationButtonClickListener(). Disse blir aktivert dersom brukeren har gitt appen tillatelse til bruk av stedstjenester, som sjekkes gjennom checkPermission(), og dersom kartet er initialisert.

MapFragment har en observator på hver av metodene getLocationData() og getWeatherForecast() i fragmentet sin viewModel, som heter MapViewModel. Bruken av et ViewModel sørger for at MapFragment opprettholder lav kobling. Disse observatørene henter oppdaterte data om henholdsvis brukerens posisjon, som lagres i et LocationDetails objekt, og data om vindretning og vindhastighet hentet fra MET api, som lagres i et Forecast objekt. Dataen blir hentet fra klassene LocationLiveData og WeatherRepository, som er repositories i henhold til MVVM-arkitekturen.

Observatørene startes gjennom metodene startLocationUpdate() og startWeatherUpdate() i fragmentet sitt onResume()-metode. Dette er for å håndtere tilfeller hvor en bruker starter appen og ikke gir appen tillatelse til å bruke posisjonen, men deretter gir tillatelse i Android sine innstillinger og går tilbake inn i appen. Da vil appen kalle på onResume(), og

observatørene må startes her dersom brukeren skal få se data om vinden. Av samme grunn kalles enableMapData() i onResume().

```
override fun onResume() {
    super.onResume()
    Log.d( tag: "Map", msg: "resume")

    if(isLocalEnabled()) {
        enableMapData()
        if (checkPermission()) {
            if the user has given permission and location services, and not observers, observers on
            the location api and weather data api
            if(!mapViewModel.getLocationData().hasObservers()) {
                Log.d( tag: "Map", msg: "Location observer started")
                startLocationUpdate()
            }
            if(mapViewModel.getWeatherForecast() != null) {
                if (!mapViewModel.getWeatherForecast()!!.hasObservers()) {
                    Log.d( tag: "Map", msg: "Weather observer started")
                    startWeatherUpdate()
```

Figur 4.2: Kodesnipp som viser onResume()-metoden.

Første gang getLocationData() mottar en oppdatert posisjon, flytter kameraet seg til brukerens posisjon gjennom map.moveCamera. Dette skjer bare én gang, nemlig første gang brukeren åpner kartet. Hver gang posisjonen oppdateres, hentes navnet på fylket brukeren er i gjennom et Geocoder-objekt. Dette gjøres på en separat tråd med Kotlin Coroutines-biblioteket fordi appen hang dersom dette kallet ikke fant navnet på fylket mens den kjørte på hovedtråden. Fylkesnavnet lagres i en variabel county i MapViewModel slik at fragmentet for farevarsler får tilgang til den. Når posisjonen oppdateres kalles også updateForecast()-metoden i MapViewModel med brukerens posisjon, gitt i breddegrad og lengdegrad. Denne metoden starter et kall på en egen tråd til MET API-et for å få værdata.

```
private fun startLocationUpdate() {
    mapViewModel.getLocationData().observe(viewLifecycleOwner, {  it: LocationDetails!
        if(it != null){
            Log.d( tag: "startLocationUpdate",  msg: "lon: ${it.longitude}")

              calls the method to update the weather data from Met api, when the location has been updated
            mapViewModel.updateForecast(it.latitude, it.longitude)
```

Figur 4.3: Kodesnipp som viser observatøren for posisjonsdata.

Når getWeatherForecast() mottar et oppdatert Forecast-objekt, vil vindretningen og vindhastigheten vises øverst på skjermen. Vindretningen vises både som en pil og som tekst. Pilen er et ImageView, kalt windDirArrow. Denne roteres gjennom rotation-egenskapen, som roterer ImageView-et mellom 0 og 360 grader. Vindretningen fra MET API-et går fra -180 til 180 grader. For å konvertere til riktig variant av gradene kalles rotateWindArrow().

Denne tar også høyde for kartets rotasjon gjennom egenskapen *bearing*. For at pilen skal rotere når brukeren roterer kartet, kalles rotateWindArrow() også når kartet roteres med map.setOnCameraMoveListener(). Denne hendelses-lytteren blir satt i enableMapData(), siden den metoden krever at kartet er initialisert og at tillatelse til å bruke posisjonen er gitt.

```
private fun startWeatherUpdate() {
    mapViewModel.getWeatherForecast()?.observe(viewLifecycleOwner, { it: Forecast!
        if(it != null) {

            gets and displays the wind visuals
            windDirection = it.wind_from_direction
            val windText =
                    "${windDirectionToText(windDirection)} ${it.wind_speed} m/s"
            enableVisibility()
            Log.d( tag: "weather in map",  msg: "Angle: ${it.wind_from_direction} $windText")
            if(::map.isInitialized) {
                rotateWindArrow(windDirection)
            }
            windSpeedText?.text = windText
```

Figur 4.4: Kodesnipp som viser observatøren for data om vind.

4.3.2 Henting av brukerens posisjon

Brukerens posisjon blir hentet ved hjelp av FusedLocationProviderClient, som gir appen et simpelt API som kombinerer GPS og Wi-Fi signal for å finne brukerene posisjon. GPS eller Wi-Fi kan frakobles og API-et vil fortsatt kunne gi posisjon på en effektiv måte. Dette blir håndtert i klassen LocationLiveData, som er en subklasse av LiveData<LocationDetails>. Her blir en variabel, kalt fusedlocationProvider, av typen FusedLocationProviderClient hentet gjennom getFusedLocationProviderClient(). Dette er en statisk metode fra klassen LocationServices, som tilbys gjennom Google Play Services API.

Klassen har også et companion object som lager en variabel locationRequest av typen LocationRequest. LocationRequest-et spesifiserer intervallet på kall til API-et, som blir satt til et minutt, og hvilken prioritet kallet skal ha. Vi har valgt å gå for høy nøyaktighet, dette sikrer at posisjonsdataen er så nøyaktig som mulig, men dette bruker mer strøm enn andre valg som ofrer nøyaktighet for mer batteri effektivitet. Begrunnelsen for valget er at en jeger er svært avhengig av å ha riktig informasjon om vind og vindstyrke, og jo mer nøyaktig posisjonen er, desto mer nøyaktig blir dataen.

For å starte hentingen av posisjonen blir startLocationUpdate() metode kalt av MapFragment, som aktiverer observatøren, og kaller klassens onActive() metode.

onActive() kaller på requestLocationUpdates(). Denne metoden tar inn locationRequest, locationCallback, og en looper som argument. Looperen gjør at locationRequests fortsetter så lenge mapFragment er aktivt. Variabelen locationCallback er av typen LocationCallback, en anonym klasse. Denne klassen overskrider metoden onLocationResult() som bestemmer hva som skjer når man får posisjonen. Dersom posisjonen ikke er null, kalles metoden setLocationData(). Her lages et LocationDetails objekt, fra posisjonen og LocationLiveData, som arver fra LiveData, sin value settes lik dette LocationDetails objektet.

```
private fun startLocationUpdates() {

      requests location updates on a loop based on the time interval
    fusedLocationClient.requestLocationUpdates(locationRequest, locationCallback, Looper.getMainLooper())
}

  callback when the location is updated
private val locationCallback = object : LocationCallback() {
    override fun onLocationResult(locationResult: LocationResult?) {
        if(locationResult == null) {
            return
        }
          sets the location to the observable value
        for (location in locationResult.locations) {
            setLocationData(location)
```

Figur 4.5: Kodesnipp som viser startLocationUpdates() og den anonyme klassen LocationCallback.

Til sist overskrives metoden onInactive(). I denne metoden fjernes posisjons oppdateringen gjennom removeLocationUpdates(). Dette er for å sikre at appen ikke bruker unødvendig mye strøm på å finne posisjonen til brukeren når en er i andre fragmenter enn kart, eller når appen er lukket.

4.3.3 Henting av værdata fra MET API

Værdata hentes i et repository når map-fragmentet er aktivt, i klassen WeatherRepository, som er en subklasse av LiveData<Forecast>. Det blir gjort et get-kall hver gang MapFragment mottar en ny location. MapFragment kaller metoden updateForecast(), med breddegrad og lengdegrad som argumenter, gjennom MapViewModel. Denne metoden kaller metoden apiFetch(), som henter data om vindretning, vindhastighet og temperatur fra MET-API-et *Nowcast*, ved bruk av brukerens breddegrad og lengdegrad. Get-operasjonen blir gjort i en egen tråd gjennom coroutine biblioteket for å forbedre ytelsen til appen. Deretter blir JSON-data deserialisert til objekter som blir brukt i appens repository, ved hjelp av Gson. Metoden apiFetch() returnerer et Forecast objekt, som inneholder data om

vindretning, vindhastighet og temperatur eller null hvis responsen fra serveren ikke var vellykket. Dersom apiFetch() ikke returnerte null, kaller updateForecast() på metoden setForecast() med Forecast objektet som argument. setForecast() setter WeatherRepository sin value til å være lik dette Forecast objektet. Henting av værdata er gjort i en egen klasse som bare MapViewModel vet om, for å følge MVVM arkitekturen. Dette gir objektene lav kobling og høy kohesjon. Det samme gjelder for henting av brukerens posisjon.

4.3.4 Henting av farevarsler

Henting av farevarsler foregår i et PopupFragment som legger seg i ytterste lag i MapFragmentet når den er initialisert gjennom en advarsel-knapp. Her benyttes det av data fra MetAlerts API-et for farevarsler, som er et av kravene til dette prosjektet. Fragmentet består av tre hoveddeler: overskrift, innhold og én knapp. Overskriften «Farevarsler i Området» gir brukeren et inntrykk over hva innholdet i fragmentet består av. Naturligvis brukes knappen med teksten «LUKK» til å lukke fragmentet. I innholdet brukes det en RecyclerView hvor det settes inn CardView for hver farevarsel med diverse informasjon fra MetAlerts API-et.

MetAlerts API-et skriver ut hoveddataen i form av et RSS-format med kanaler som inneholder objekter (“item”) for hver farevarsel. Denne måten å gi ut data på er vanlig brukt for nyheter og værmeldinger, men i dette prosjektet ligger kun interessen i dataen i identifiserende CAP-lenker, som befinner seg i hvert farevarsel-objekt. Disse lenkene henter ut CAP-informasjon, som vil si mer detaljerte attributter, som for eksempel kategori, område (i form av polygoner), og faregraden. For å spare tid og ressurser benytter JaktAppen seg av et offentlig og gratis API, “rss2json”. Dette API-et tar inn inndata fra RSS-filen fra MetAlerts API-et og gjør det om til en Json-fil. Slik blir det lettere å hente ut data fra CAP-lenken.

I tillegg er farevarslene fra RSS-filen sortert både med og uten fylker. Det vil si at man har muligheten til å kun hente ut farevarsler fra fylket brukeren befinner seg i ved å bruke fylkeskode. JaktAppen benytter seg av denne GET-funkjsonen på grunn av det ikke er gunstig å vise brukeren farevarsler for andre siden av landet. Dette gjøres før konvertering fra RSS til JSON. Dersom maps-API-et ikke finner et fylke, viser den alle farevarsler i Norge for å forebygge feilmeldinger.

For å hente fylkekode benytter JaktAppen seg av en lokal XML stringlist som inneholder alle fylker. Et fylke-string ser for eksempel slik ut: “Oslo 03”. På den måten sammenligner man stringen hentet fra MapFragmentet (gjennom en metode som benytter seg av MVVM-arkitekturen) med stringlisten, og henter ut tilhørende fylkeskode. Deretter blir fylkeskoden satt inn i GET-funksjonen som nøkkelverdi.

```
for (i in countyList.indices) {
    if (countyName.toLowerCase(Locale.ROOT) in countyList[i]!!.toLowerCase(Locale.ROOT)) {
        response = countyList[i]
    }
}
val string = response ?: return null

// only getting numbers from string. example "Oslo 03" -> "03"
return string.filter { it.isDigit() }
```

Figur 4.6: Kodesnipp som viser metoden for å hente ut fylkeskode.

Etter konverteringen fra RSS til JSON er det nå mulig å hente CAP-lenkene ved å lage objekter for hvert item ved å bruke “gson”- og “fuel”-utvidelsene prosjektet bruker i Kotlin. Objektene blir satt inn i en lokal liste “warningList” for å lagre dataen til neste steg i fragmentet.

CAP-lenkene gir som forklart ovenfor mer utfyllende data om hver farevarsel, men i et XML-format. Her brukes det en standard XMLPullParser for å hente ut ønsket data fra XML-filene og lager objekter ved bruk av dataklasser. Disse objektene blir innført i en lokal liste i XMLPullParseren, og deretter hentet ut i PopupFragmentet og ført inn i nok en lokal liste, “alertList”, for å få tilgang til alle farevarsel-objektene med spesifikk data.

```
coroutineScope.launch {  this: CoroutineScope
    val response = getData()

    activity?.runOnUiThread {
        val inputStream: InputStream = response.byteInputStream()
        val listOfAlerts = InfoXmlParser().parse(inputStream)

        for (i in listOfAlerts.indices) {
            alertList.add(listOfAlerts[i] as Info)
        }
        Log.d( tag: "LIST OF ALERTS (method)", alertList.size.toString())
        (infoJob as CompletableJob).complete()
    }
}
}
```

Figur 4.7: Kodesnipp som viser bruk av XMLPullParseren i PopupFragment.

Til slutt blir alle farevarsel-objektene lagret i alertList skrevet ut i CardViews som plasseres inn i RecyclerViewet til PopupFragmentet ved bruk av en RecyclerView-Adapter.

4.3.5 Avspilling av lokkelyd

Lokkelydene er lagret lokalt på enheten. Ved hjelp av et SQLite-basert håndteringssystem for Android som heter Room, er informasjon om lokkelydene lagret i en database som kan aksesseres uten at brukeren trenger internettforbindelse.

I dataklssen Sound spesifiseres databasetabellen. Hver entitet i databasen har fire attributter: id (primærnøkkel), tittel på lyden, hvilket dyr lyden er ment å lokke og til slutt en fil-sti til der selve lydfilen ligger lagret.

Selve databasen konstrueres og styres i den abstrakte klassen SoundDatabase. Her spesifiseres det først at databasen skal inneholde Sound-objekter fra dataklassen nevnt ovenfor. Konstruksjonen av databasen skjer i metoden getDatabase() som ligger inni et Companion-objekt (det vil bare finnes én instans av denne klassen). Deretter pre-populeres databasen med entiteter når getDatabase() kaller på *.build()*. Denne operasjonen skjer asynkront ved hjelp av Kotlins CoroutineScope.

```
    fun getDatabase(
        context: Context,
        scope: CoroutineScope
    ): SoundDatabase {
        INSTANCE ?: kotlin.run {
            INSTANCE = Room.databaseBuilder(context.applicationContext, SoundDatabase::class.java,  name: "sound_database")
                .fallbackToDestructiveMigration()
                .addCallback(SoundDatabaseCallback(scope))
                .build()
        }
        return INSTANCE!!
    }
}
```

Figur 4.8: getDatabase() bygger databasen.

Interaksjonen med databasen skjer ved bruk av et *data access object* (DAO). Når databasen er opprettet, kan spørringer kjøres mot databasen, og disse spørringene skjer via nettopp vårt SoundDAO interface. I dette interfacet har vi også metoder for å opprette, oppdatere og slette entiteter i databasen. Dette er fordi det skal ligge til rette for å implementere at brukeren selv kan legge inn lokkelyder på et senere tidspunkt. Det er imidlertid kun metodene som henter entiteter som er i bruk i koden per nå. For eksempel vil metoden

findSoundsByAnimal() kjøre en spørring mot databasen der den ber om alle entiteter som matcher et gitt søkeord. Metoden returnerer en liste med *Live Data*-objekter.

```
@Dao
interface SoundDAO {
    // Hent ut alle lyder. Brukes når SoundFragment starter opp
    @Query( value: "SELECT * FROM sound_table ORDER BY soundID ASC")
    fun retrieveSoundLibrary(): LiveData<List<Sound>>
```

Figur 4.9: SQL-spørringene er formulert i interfacet SoundDAO.

Videre finnes det en klasse SoundRepository som fungerer som et grensesnitt som prosesserer SoundDAO-spørringer mens resten av appen holdes utenfor det som har med databasen å gjøre. Dette er gjort for å følge objektorienterte prinsipper som lav kobling og høy kohesjon. I SoundRepository etableres en forbindelse med SoundDAO, og SundDAO sin metode *retrieveSoundLibrary()* kalles for å hente alle lydene fra databasen til en liste *allSounds* med Live Data-objekter.

```
class SoundRepository(private val soundDao: SoundDAO) {

    val allSounds: LiveData<List<Sound>> = soundDao.retrieveSoundLibrary()
```

Figur 4.10: SoundRepository prosesserer SoundDAO-spørringer.

For å gjøre lydfilene tilgjengelig for fragmentet *Lokkelyder* som skal bruke dem, brukes en ViewModel. I klassen SoundViewModel knyttes det kontakt med SoundRepository og data fra SoundRepository sin allSounds lagres i en variabel med samme navn i ViewModelen. Denne listen herfra være tilgjengelig for ulike fragmenter i appen, og vil dessuten automatisk oppdateres når underliggende data endres. Denne arkitekturen er dermed i tråd med MVVM-prinsippene som er presentert i Kapittel 4.2.

```
class SoundViewModel(application: Application) : AndroidViewModel(application) {

    // Knytter ViewModel med SoundRepository
    private val repository: SoundRepository

    val allSounds: LiveData<List<Sound>>
    var searchResult: LiveData<List<Sound>>

    init {
        val soundDao = SoundDatabase.getDatabase(application, viewModelScope).soundDao()
        repository = SoundRepository(soundDao)
        allSounds = repository.allSounds
        searchResult = repository.searchResult
    }
```

Figur 4.11: SoundViewModel gjør data fra databasen tilgjengelig for fragmentet som skal bruke dem.

I SoundFragment initialiseres SoundViewModel og det settes opp en observatør (*observer)* på SoundViewModel sin allSounds-liste. På denne måten vil lydene som er tilgjengelig i fragmentet alltid være dynamisk oppdatert opp mot ViewModelen og dermed også opp mot den underliggende databasen.

```
    private fun getAllSounds() {
        mSoundViewModel.allSounds.observe(viewLifecycleOwner, { sound ->
            adapter.setData(sound)
        })
    }
```

Figur 4.12: En observer sørger for at dataene som vises er dynamisk oppdatert mot databasen.

Når brukeren går inn i SoundFragment vil en Spinner for valg av dyr være forhåndsvalgt på "Alle". Da hentes allSounds gjennom ViewModelen, og brukeren får presentert et RecyclerView med alle lydene som ligger i databasen. Hvis brukeren velger et gitt dyr i Spinneren, vil det kjøres en spørring mot databasen med dette dyret som spørreord, og brukeren vil få presentert bare lydene som er ment for dette dyret i RecyclerViewet.

Når brukeren trykker på en lyd, vil det opprettes et MediaPlayer-objekt som bruker det valgte Sound-objektet sin filsti til å forberede avspillingen av lyden. Lyden spilles imidlertid ikke av før brukeren trykker på play-knappen.

4.3.6 Endre appens innstillinger

Når man klikker på innstillinger, vises et recyclerView med tre elementener:

1. Modus: Bytt til mørkt utseende
2. Farevarsel: Slå på farevarsel
3. App info: Viser applikasjonsinformasjon med bla. innstillinger for stedstjenester

1. Modus:

I modus er det implementert en button/knapp «switch» som brukes for å bytte til mørkt utseende. Den er by default «av». Det betyr at appen har lyst utseende når den startes for første gang. Når man endrer til mørkt utseende, blir strengverdien omgjort fra 0 til 1. Dette er en SharedPreferences-verdi, og gjør listeverdien «checked int» til 1.

```
if(state)
{
    AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_YES)
    MyPreferences(mContext).darkMode = 1
}
else{
    AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_NO)
    MyPreferences(mContext).darkMode = 0
}
```

Figur 4.13: Kodesnipp som viser if-testen for om darkmode er aktiv.

```
var darkMode = preferences.getInt(DARK_STATUS,  defValue: 0)
    set(value) = preferences.edit().putInt(DARK_STATUS, value).apply()
```

Figur 4.14: Kodesnipp som viser lagringen av verdien til darkmode.

Når applikasjonen åpnes etter første gang, er modus angitt etter hva verdien i SharedPreferences er. Hvis den er 0, vil det si at switch knapp er av. Derfor vil det være lyst utseende. Omvendt, hvis verdien er 1, er switch knapp på, og dermed har applikasjonen mørkt utseende.

Bildet som er øverst i innstillinger blir angitt i henhold til applikasjonsmodus. Det vil enten vært lyst eller mørkt avhengig av verdien til SharedPreferences. Hvis verdien til SharedPreferences for mørkt utseende/dark mode er 0, betyr det at det er lyst utseende. Det

vil da velge bilde «elg.png» fra drawable, og sette det inn i imageView. Tilsvarende hvis verdien er 1, velges det bilde «elg_mork.png» fra drawable, og blir lagt inn i imageView. Welcome activity tar i bruk den eksisterende modus ved å sjekke hva den er gjennom SharedPreferences DarkMode-verdien.

2. Farevarsel:

I Farevarsel er switch knapp by default «på. Farevarsler finnes i kartfragmentet. Vi kontrollerer farevarsler gjennom switch-knappen ved å bruke verdien *SharedPreferences*. SharedPreferences er by default 1, dvs. switch knappen er på. Hvis man slår av farevarselet, endres SharedPreferences-verdien til null.

```
var WarningMode = preferences.getInt(Warning_status,  defValue: 1)
    set(value) = preferences.edit().putInt(Warning_status, value).apply()
```

Figur 4.15: Kodesnipp som viser lagringen av verdien til farevarsel knappen.

Når kartfragmentet startes, blir SharedPreferences-verdien sjekket. Hvis den er 1, vises det farevarsel-ikonet på høyre side av kartet. Ved verdien 0, skjules farevarsel-ikonet fra kartet.

3. App info

Det tredje innstillingsalternativet brukes til å åpne enhetens app-innstillinger, hvor brukeren kan administrere databruk, tillatelser (som stedstjenester) og varslinger. Dette innstillingsalternativet åpnes ved hjelp av Intent.

```
val intent = Intent(Settings.ACTION_APPLICATION_DETAILS_SETTINGS)
val uri: Uri = Uri.fromParts( scheme: "package", mContext.getPackageName(),  fragment: null)
intent.data = uri
mContext.startActivity(intent)
```

Figur 4.16: Kodesnipp som viser åpning av enhetens innstillinger.

4.4 Universell Utforming

Et viktig tema i IN2000-kurset, er universell utforming. Universell utforming går ut på at man skal designe en løsning som tar hensyn til flest mulige brukere på en mest likeverdig måte. Formålet ved implementasjon av universell utforming er motvirke diskriminering på

grunn av nedsatt funksjonsevne (Lid, 2020). I dette prosjektet bestemte vi oss derfor for å implementere design med fokus på brukeren og bruksområdet i jaktappen.

Det norske regelverket bekjemper også diskriminering i digitale løsninger ved å bruke WCAG-standarden. Standarden er en forskrift med mange forskjellige minimumskrav som skal følges når man lager en løsning. Både applikasjoner og nettsteder skal følge minst 35 av 61 suksesskriterier i WCAG 2.1 standarden (Uutilsynet, 2012). Med tanke på at dette prosjektet har begrenset med tid på å utvikle app-løsningen er det svært usannsynlig å utvikle en app som følger minstekravet fullstendig. Derfor mener gruppen at det er viktigere å sette fokus på noen spesifikke krav fremfor mange forskjellige krav uten ordentlig implementasjon.

4.4.1 Eksempler på tilpasning til diverse WCAG-minimumskrav

Først og fremst er det implementert en enkel navigasjonsløsning for å ta hensyn til minimumskravene 1.4.3 og 3.2.3 i WCAG-standarden, som til sammen går ut på å lage en navigasjonsløsning med en konsekvent rekkefølge og bruk av kontraster for å gi brukeren feedback (Uutilsynet, 2021). Navigasjonsløsningen i JaktApp-en baserer seg av tre knapper med identifiserbare ikoner og tekst under. Når brukeren interagerer med en av knappene, blir ønsket side vist. Det som skjer i bakgrunnen i applikasjonen er at fragmentet tilhørende knappen brukeren trykker på bytter ut tidligere fragment i fragment-view i MainActivity. Et av de mest typiske eksemplene på universell utforming i en applikasjon med visuelt grensesnitt er å gi brukeren feedback og vise tydelig akkurat hvor brukeren befinner seg i applikasjonen. Derfor er en nedre navigasjonsbar en god løsning. Navigasjonsbaren viser brukereren hvor man befinner seg i applikasjonen til all tid ved å fylle fargen til ikonet ut i fra hvor brukeren er. I tillegg er navigasjonsbaren lett tilgjengelig for tommelen eller andre fingre ved bruk av smarttelefonen med en hånd.

Figur 4.17: Navigasjonsbar plassert på bunnen av appen

En applikasjons struktur er definert gjennom valg av farger og kontraster. I WCAG-standarden legges det vekt på bruk av farger og kontraster i punktene 1.4.1 og 1.4.3,

hovedsaklig mellom tekst og bakgrunn. I JaktApp-en benyttes en fargepalett som gir applikasjonen bedre dybde og bidrar til å styrke brukerens forståelse av applikasjonen og dens innhold. Ved bruk av gode farge- og kontrastkombinasjoner er det enklere for brukeren å skille forskjellige elementer og funksjoner fra hverandre slik at brukeren skal kunne problemfritt navigere seg rundt i applikasjonen og oppfatte informasjon i innholdet. Det er for eksempel ikke tilrådelig å sette lys tekst over lys bakgrunn. Enkelte brukere vil få problemer med å lese teksten, og tilsvarer at designløsningen eksluderer svaksynte brukere.

Figur 4.18: Sound-fragment med fargekontrast på tekst ved seleksjon

Et annet eksempel på fargesignaler i applikasjonen er hvordan faregraden til farevarslene i popup-menyen er implementert. Rammefargen til farevarsel-elementene baserer seg på faregraden. For eksempel om faregraden er "Alvorlig" vises bakgrunnsfargen som oransje, og gul dersom faregraden er "Moderat". Årsaken til at denne signaliseringen er implementert er å gi brukeren en rask feedback på hvilke farevarsler som er mer alvorligere enn andre. Det er også en tekstlig beskrivelse av faregraden om brukeren skulle for eksempel være fargeblind.

Figur 4.19: Popup-fragment - farevarsel-elementer med fargekode

I den nyeste utgaven av standarden, WCAG 2.1, blir det introdusert et nytt minimumskrav 1.3.4. Dette minimumskravet går ut på at brukeren selv skal kunne velge hvilken

visningsretning på applikasjonen vedkommende ønsker å bruke (Uutilsynet, 2021). Altså brukeren skal kunne velge mellom liggende eller stående retning til enhver tid. I JaktApp-en er denne funksjonaliteten implementert fra begynnelsen av prosjektet slik at dette ikke skulle bli en utfordring. Det meste er bygget opp av fragmenter i en hovedaktivitet, hvor alt er organisert gjennom constraints.

Figur 4.20: Popup-fragment i liggende modus

Nok et eksempel fra Popup-fragmentet er håndteringen av statusbeskjeder. Formålet med statusbeskjeder er klargjøre og sette oppmerksomheten til brukeren på potensielle endringer og oppdateringer i applikasjonen. Dette går innenfor punkt 4.1.3 i den nye WCAG 2.1-utgaven. I JaktApp-en er det kun tre hoveddeler som benytter seg av dynamiske API-kall: kart, vindvarsel og farevarsler. I Popup-fragmentet for farevarsler hentes det data fra Met API-et. Derfor er det viktig å fortelle brukeren hvilket stadium i GET-prosessen man befinner seg i. For eksempel ved start av innhenting av farevarsler skrives det ut i popup-fragmentet at farevarsler laster inn. Dersom det ikke skulle være noen farevarsler tilgjengelig i det brukeren ønsker å få opp farevarsler, skrives det ut at det ikke finnes farevarsler for øyeblikket.

5 Testdokumentasjon

Et prinsipp i testing at det er umulig å teste for alt. I stedet bør testingen legges opp etter risikovurderinger og prioriteringer i hver enkelt prosjekt (Vihovde, 2021). Målet med testingen i vårt prosjekt var å oppdage så mange feil som mulig, at brukerens forventninger møtes og at teamet selv skulle få litt erfaring med ulike testverktøy. Vår vurdering var at sikkerhet i systemet var lite relevant fordi appen ikke lagrer noe informasjon om brukeren, og testing som sikkerhetsmekanisme har derfor ikke vært i fokus.

I Kapittel 5.1 og 5.2 gjør vi rede for hvordan vi har brukt testmetodene enhetstesting og integrasjonstesting til å nå målet vårt med testingen. Selv om testene kan oppdage noen feil, kan de ikke skjekke at det ikke eksisterer andre feil. For å oppdage noen av feilene som ikke oppstår under kjøring, benyttet vi oss av statisk testing og teknikken "walkthrough". Denne teknikken går ut på at teamet går gjennom koden i plenum, og den som har skrevet koden leder gjennomgangen (Vihovde, 2021). Fordelen med denne øvelsen er at den kan gjøres uten at man er avhengig av å ha kjørende kode. Vi brukte teknikken for å finne ut av kryptiske bugs og gjøre oss kjent med hverandres kode, men også som en måte å teste om de ikke-funksjonelle kravene, som for eksempel arkitektur, var ivaretatt.

5.1 Enhetstesting

Enhetstesting er en testmetode for programvare hvor enhetene, de individuelle komponentene i programvaren, testes. Dette brukes for å sørge for at koden fungerer som den skal, for å finne bugs og for å forebygge at bugs oppstår i fremtiden (Mikhalchuk, 2020). Enhetstester skrives gjerne som funksjoner i koden, og så sjekker man verdiene eller oppførselen som disse funksjonene gir i ulike scenarioer. Ved god og konsekvent bruk at denne type tester, kan man forhindre å skape problemer for eksisterende kode når man legger til noe nytt. I store prosjekter er det vanlig å automatisere slik testing for å eliminere menneskelige feil (som for eksempel å glemme å kjøre testen).

I vårt prosjekt konkluderte vi med at arbeidet ved å implementere omfattende og "korrekt" enhetstesting vil overstige nytten av det. Det er fordi omfanget av prosjektet er relativt lite og en slik implementasjon ville derfor tatt for mye av disponibel tid. Videre er det slik at appens tre hovedbestanddeler er nokså uavhengige av hverandre i koden, noe som reduserte

sjansen for at det kunne oppstå ødeleggende konflikter på tvers av disse bestanddelene. Vi valgte derfor å bruke Android Studios Logcat-funksjonalitet for å nettopp sjekke tilstander og returverdier for de ulike funksjonene i koden. Dette fungerte som en lettvektig og rask enhetstesting, men løsningen var naturligvis ikke like robust som skikkelig enhetstesting skal være.

Vi har likevel implementert enhetstester til noe av funksjonaliteten i appen, og gjort oss kjent med prinsippet om testdrevet utvikling. Testdrevet utvikling handler om å skrive testen før man implementerer funksjonen som skal bestå testen. Til dette brukte vi JUnit 4-rammeverket sammen med Google sitt Truth-bibliotek. JUnit er det mest populære og brukte enhetstest-rammeverket for Java.

Enkle enhetstester

Det er implementert enkle enhetstester for følgende:

- Sound-objekter. Testen sjekker at objekter som opprettes av denne typen har en ID som settes lik 0, og at alle attributter blir gitt verdier.
- Location-informasjon. Testens formål er å sjekke at en gitt lokasjon som returneres fra API-kallet og skal brukes videre har breddegrad, lengdegrad og høyde som ligger innenfor teoretisk mulig verdiområde.
- Vindvarsel. Testens formål er å sjekke at en gitt vindretning og vindstyrke som returneres fra API-kallet har lovlige verdier.

Instrumentert enhetstest - testing av Room-database

For å bygge databasen, er man nødt til å ha en "Context" (som en Android-spesifikk klasse), og dermed kan ikke dette testes med en lokal enhetstest, men må i stedet testes på en fysisk enhet (eller emulator). Denne type tester kalles "instrumented unit tests", og kjører normalt mye saktere enn lokale enhetstester, men er viktig for vår applikasjon nettopp fordi databasen bruker lagring lokalt på enheten den kjører på. Måten vi har valgt å teste databasen på kjører imidlertid relativt raskt fordi den ikke krever at en aktivitet eller et fragment er laget før testen kan kjøres.

Testen ligger i mappen "AndroidTest" → "soundDatabase". Her opprettes en database som kun brukes til testing. Det er to hovedfunksjonaliteter i koden som er knyttet til databasen; å legge inn entiteter i databasen, og å hente disse ut i etterkant. Testen "writeAndReadSound"

legger inn et Sound-objekt i databasen, og sjekker deretter om objektet finnes der når funksjonen som henter entitetene i databasen returnerer. Sjekken gjøres ved bruk av Thruth-biblioteket. Testen "searchForSound" sjekker på tilsvarende måte at spørringer med gitt søkeord fungerer i databasen.

5.2 Integrasjonstesting

Integrasjonstesting er testmetode som hvor enhetene/modulene blir testet sammen, altså etter hver eneste modul/funksjon har blitt testet individuelt. Det er hovedsakelig to forskjellige type integrasjonstesting: Enhet/komponent integrasjonstesting og system integrasjonstesting (softwaretestingfundementals, 2020).

For vårt prosjekt var det enhet/modul integrasjonstesting som var mest relevant, dette går ut på å teste grensesnittet mellom enhetene/modulene i appen (softwaretestingfundementals, 2020) og avdekke feil mellom disse. Mye av integrasjonstesting blir ofte gjort automatisert, til likhet med enhetstesting konkluderte vi med at arbeidet ved å implementere en automatisert integrasjonstesting ville overstige nytten av det. Dette er igjen fordi prosjektet er relativt lite og det meste av testingen kunne bli gjort "for hånd".

Integrasjonstestingen som ble gjort var relativt lite arbeid i seg selv, siden det var svært lite informasjon som gikk fra funksjon til funksjon, altså ingen informasjon går fra for eksempel kart funksjonen til lyd funksjonen og vice versa. Det blir derimot byttet informasjon mellom modulene i MVVM arkitekturen i samme funksjon, her ble det testet på lik måte som enhetstestingen, etterhvert som data fra repository ble mottatt fra modell og remote data source og sendt videre til viewModel ble de vist på brukergrensesnittet der, da kunne vi sammenligne verdiene der med verdier som var skrevet ut med Logcat-funksjonen i de andre komponentene i arkitekturen.

Det ble ikke relativt få bugs eller feil fra denne testingen, dette stammer fra god implementasjon av MVVM-arkitekturen og god enhetstesting som gjorde at alle komponentene til prosjektet kommuniserte som de skulle.

6 Prosessdokumentasjon

Når prosjektet startet, satte teamet seg ned og begynte planlegging av prosessen. Det ble bestemt hvilken smidig metode som skulle benyttes, hvilke regler som skulle gjelde innad i teamet, hvilke verktøy som skulle brukes, og det ble laget en plan over ukene vi hadde til rådighet. Vi brukte god tid på dette for å gjøre prosessen så ryddig og oversiktlig som mulig senere.

I dette kapittelet følger en beskrivelse av vår tilnærming til smidig arbeidsmetodikk og hvordan vi har valgt å legge opp våre sprinter og møter. I kapittel 6.2 gis en oversikt over hvilke arbeidsverktøy som er benyttet gjennom prosjektet. Deretter følger Kapittel 6.3 med en gjennomgang av hvordan arbeidet med både appen og rapporten forløpt, delt inn i faser og sprinter. Til slutt vil Kapittel 6.4 gi diskusjon og refleksjon rundt organiseringen av prosjektet, samarbeidet, utfordringene og lærdommene vi tar med oss videre.

6.1 Smidig utviklingsmetodikk

Smidige prosessmodeller velges som regel når det foreligger en lettere og mer dynamisk kravspesifikasjon enn det typisk gjør når man velger plandrevne prosessmodeller. En smidig prosessmodell kjennetegnes ved at både planlegging og levering skjer inkrementelt, og at det dermed er lettere å gjøre endringer i krav underveis i prosjektet. Videre stilles det ikke så strenge krav til formelle prosjektdokumenter - det er kontinuerlig samarbeid og dialog med kunden som vektlegges. I vårt prosjekt er det naturligvis ikke eksistert en reell kunde, så vi har i stedet forholdt oss til en mer “tenkt kunde”.

6.1.1 Hybrid av Scrum og Kanban

Prosessmodellen vi valgte å bruke for dette prosjektet var en blanding av Scrum og Kanban, også kjent som Scrumban. Denne hybrid-modellen brukte vi både for koding av appen og rapportskriving. Vi valgte å gjøre det slik fordi vi tenkte at å dele opp prosjektet i mindre biter ville gjøre arbeidet mer inspirerende, men også mer oversiktlig. Siden det i tillegg var begrenset med tid til planlegging, intervjuer og undersøkelser i forkant av utviklingsprosessen, anså vi det som svært sannsynlig at kravspesifikasjonene vi hadde

utformet måtte endres underveis. Både Scrum og Kanban gjør det veldig enkelt å tilpasse prosjektet til å inkludere nye krav etter hvert som behov og ideer dukker opp.

Scum har rollebasert prosjektstyring. En Scrum Master leder prosjektprosessen. Prosjektet gjennomføres med tidstyrte, korte arbeidsøkter (to-fire uker) som kalles sprinter. Sprintene består av deloppgaver beskrevet som brukerhistorier. Det er rollen Product Master som har ansvar for disse. Vi hadde imidlertid ikke disse tydelig definerte rollene i vår prosjektgruppe. Vi valgte heller å gå for Kanbans tilnærming som går ut på at rollene er mer flytende og at ingen har noe særskilt ansvar for progresjon i prosjektet. Oppgaver ble løst og ferdigstilt i en kontinuerlig flyt. Selv om vi opererte med begrepet "sprint" i tråd med Scrum, ble altså oppgaver løst på en måte som ligger nærmere Kanban-modellen. Våre sprinter handlet mer om å skape motivasjon for å ferdigstille deler av prosjektet og å nå små delmål.

6.1.2 Sprint

Sprintene våre varte i en uke, fra fredag til fredag, med rullerende rolle som møtefasilitator/referent. Såpass korte sprinter fungerte for oss siden vi aldri hadde oppgaver som *måtte* fullføres i løpet av en sprint, men at det samtidig ga insentiv til å ha synlig progresjon i prosjektet for hver eneste uke, noe som vi anså som nødvendig i et så kortvarig prosjekt. Grunnen til at vi utformet sprintene på denne måten var at vi ikke er vant til å jobbe med store prosjekter innenfor koding, vurderte vi det som sannsynlig at vi noen ganger ville dele opp en oppgave i for store biter, og dermed ta på oss for mye arbeid i en sprint. Derfor hentet vi altså inspirasjon fra Kanban, hvor man jobber med en oppgave til den er ferdig. Vi tillot oss å la oppgaver og delmål bli med over fra en sprint til den neste dersom de var for store til ferdigstilles i løpet av en sprint. På fredag etter endt sprint hadde vi til å begynne med retrospektiv-møter der vi evaluerte hvordan sprinten hadde gått og hvordan samarbeidet hadde fungert. Det viste seg etter hvert å være unødvendig å ha denne evalueringen så ofte (les mer i Kapittel 6.5 Evaluering av prosessen).

6.1.3 Stand up-møter

Stand up-møtene fra Scrum, eller "daily scrum", er ment som en arena der teamet får vite om eventuelle utfordringer som har oppstått siden sist og hvor teamet kan koordinere sin innsats. Vi hadde ikke slike møter hver dag som navnet antyder, men i stedet tre ganger i uken; mandag, onsdag og fredag. Vi valgte denne løsningen fordi vi alle hadde andre fag

som vi også måtte arbeide med, og det ville derfor blitt for mange møter hvor lite eller ingenting hadde skjedd siden sist hvis vi skulle hatt daglige møter. Ved å begrense oss til tre møter i uken, fikk altså alle muligheten til å jobbe med andre fag en dag, uten å stresse med at de også måtte ha gjort noe på prosjektet til morgendagens stand up møtet, eller å føle på noe skyld for at de ikke hadde gjort noe. Til gjengjeld varte våre scrum-møter lenger enn hva et tradisjonelt daily scrum ville gjort.

På daily stand up startet vi med at hvert gruppemedlem fortalte hva vedkommende hadde gjort siden sist og eventuelle problemer som hadde oppstått. Møtefasilitatoren skrev ned alle relevante opplysninger i møtereferatet for dagen. Deretter gikk vi gjennom spørsmålene eller problemene som hadde blitt nevnt, og forsøkte å oppklare de mest trivielle av dem i plenumet, noe som er et prinsipp hentet fra Kanbans fokus på flyt. Hvis problemet var av en større type, avtalte vi hvem som skulle jobbe med problemet til neste gang. Til slutt i møtene tok vi en runde på hvilke oppgaver hver av oss skulle gjøre til neste møte, og dette ble også skrevet ned i referatet. Referatet fungerte dermed som en primitiv Kanban-tavle med hva som var gjort og hva som skulle gjøres.

6.2 Arbeidsverktøy

Det finnes mange digitale hjelpemidler for å kunne arbeide så smidig som mulig, og vi har benyttet oss av noen av dem. I en situasjon hvor teamet ikke har kunnet møtes fysisk, har særlig digitale samarbeidsplattformer vært nødvendig. Nedenfor lister vi opp de digitale verktøyene som er brukt i prosjektet.

6.2.1 Organisering

Det ble opprettet en Google Drive-mappe for alle dokumenter i prosjektet (bortsett fra kode). Alle medlemmene hadde tilgang til denne mappa, og alt av dokumenter ble sortert i mapper for å ha det så ryddig som mulig, slik at det skulle være lett å finne det man leter etter. Kommentar-funksjonaliteten i Google Docs gjorde det enkelt å få og gi tilbakemeldinger på arbeidet kontinuerlig. I begynnelsen ble Trello ble brukt til kanban boards, så alle i teamet skulle kunne ha oversikt over hva hvilke oppgaver som må gjøres, og hvor man er i prosessen. Til videosamtaler ble Zoom brukt mest da dette var det medlemmene var mest kjent med. I kommunikasjon med veileder, ble Microsoft Teams benyttet. For rask kommunikasjon brukte vi Facebook Messenger. Her delte vi for eksempel

Zoom-link før møtene, stilte spørsmål og kom med påminnelser. All kommunikasjon foregikk over nett, da det ikke var mulighet for å jobbe sammen fysisk.

6.2.2 Utvikling

Vi benyttet vi GitHub for versjonskontroll av koden. Android studio støtter integrasjon av Git, slik at det var enkelt å gjøre Git-operasjoner kontinuerlig mens vi arbeidet i utviklingsmiljøet. Dette har vært svært nyttig underveis for å få oversikt over hvilke endringer som er gjort, og hvem som har gjort dem. Det har også gitt mulighet for å gå tilbake i koden hvis det er gjort endringer som har skapt problemer som ikke var der tidligere. Vi har utviklet applikasjonen i flere forgreninger, eller brancher, med *master branch* som hovedgren. Dette var nødvendig i situasjoner der vi måtte jobbe parallelt med ulike deler av programmet, og det var uvisst hvordan delene til slutt burde kobles sammen. Denne arbeidsmetoden tillot oss dessuten å eksperimentere mer med koden uten at vi trengte å bekymre oss for at det vi gjorde ville ødelegge for andre i teamet. Når vi hadde kode som var testet og oppførte seg som forventet, ble koden pushet til master. For forsøkte å ikke pushe for mye kode til master hver gang.

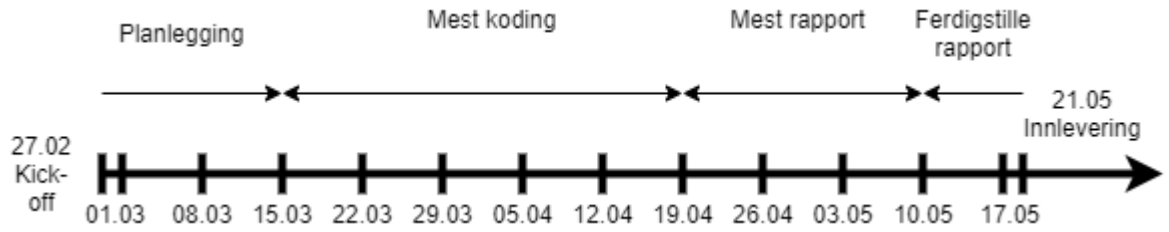
6.2.3 Design

Diagrams.net ble brukt for å lage diagrammer, og enkle skisser av appen og hvordan den skulle fungere. Proto.io, som er et prototype-verktøy, ble også brukt til å lage skisser. Til fargepalett ble det tatt utgangspunkt i en noen autogenererte paletter fra coolors.co, men det ble gjort endringer for å tilpasse det til appen. For å lage logoen til prosjektet og noen enkle ikoner, ble vectr.com brukt. Dette var nyttig fordi det ga mulighet for å lett endre størrelsen på figuren, uten at den ble uklar, pixlete etc.

6.3 Prosjektfaser

Prosjektet kan deles inn i tre hovedfaser; planlegging av prosjekt, periode med hovedfokus på koding av app, og periode med hovedfokus på rapport. Dette kapittelet er følgelig strukturert på samme måte. Planleggingsfasen satte grunnlaget for resten av prosjektet. De to andre fasene innebar jobb med alt som var nødvendig for å lage appen og skrive rapport, men det var ulikt hvor mye ressurser som ble brukt på ulike oppgaver. I det som heretter omtales som “kodefasen”, ble det meste av teamets ressurser brukt til å kode da vi anså det

som nødvendig å få på plass noe håndfast vi kunne bygge videre på. I "rapportfasen" var det meste av det tekniske på plass, og det satt av mer ressurser enn tidligere til å jobbe med rapporten, dokumentere koden og tegne ulike diagrammer. Til slutt i dette delkapittelet ligger en brukertest vi gjennomførte mot slutten av kodefasen. Denne gir et eksempel på hvordan vi arbeidet smidig med design og funksjonalitet i appen.



Figur 6.3: Tidslinje for prosjektfaser

6.3.1 Planleggingsfasen

De to første ukene ble hovedsakelig brukt til planlegging av prosjektet og å bli kjent med gruppen. Planleggingen gikk i første omgang ut på å bestemme hvordan arbeidet skulle organiseres og gjennomføres. Vi brukte nesten to hele to møter til dette. Deretter fulgte en grundig prosess for valg av case. Det ble også gjennomført et intervju med målgruppe for appen i denne perioden.

Det ble tidlig bestemt at vi skulle møtes annenhver dag i uka, og at disse skulle holdes kl. 12 på mandager, onsdager og fredager. Dette var så teamet kunne møtes ofte nok til å gi oppdateringer underveis om hvordan den enkelte lå an med arbeidet, og hva som ble gjort. Møtene skulle være typiske scrum-møter der vi først skulle gå igjennom hva alle har gjort siden sist, så ta opp eventuelle problemer eller hendelser, og til slutt sette opp oppgaver til neste gang.

I forlengelsen av møteplanleggingen, ble det laget en plan for prosjektet. I planen ble alle uker i prosjekttiden listet opp, og vi førte inn datoer og tidspunkt for møter og andre viktige hendelser som innleveringsfrister, påskeferie etc. Planen inneholdt også oversikt over hvilket medlem i gruppa som var møteansvarlig/møtefasilitator for hver uke. Å ha møteansvar innebar at man skulle skrive referat, være ordstyrer og opprette zoom-møter. Denne planen skulle gjøre det oversiktlig å se hvor mye tid det var igjen av prosjektet slik at vi kunne tilpasse planlagt arbeidsmengde både for hver sprint og mer langsiktig.

Etter vi hadde valgt case, ble det laget noen skisser av designet til appen for å illustrere hvordan de ulike funksjonalitetene kunne se ut. Disse skissene hjalp oss med å forstå hverandres ideer og peke ut en felles retning, og ble utgangspunkt for mer detaljerte tegninger senere i prosjektet. Deretter var det tid for å gjennomføre et intervju med en person i målgruppa. Målet med intervjuet var å få en uavhengig vurdering av hvilke funksjonaliteter som kunne være nyttige i en jaktapp.

6.3.2 Kodefasen

Apputviklingen, eller kodefasen, var den største fasen i prosjektet. Vi tok utgangspunkt i skissene og intervjuet fra planleggingsfasen og definerte arbeidsområder som vi fordelte til gruppemedlemmene. Til å begynne med var det tre hovedområder i appen det var naturlig å bemanne - nemlig innstillinger, kart og lokkelyder. Vi tok en runde i gruppa på hvilket område hver av oss kunne tenke å jobbe på, og fant en naturlig fordeling ut ifra dette. Som utgangspunkt en begynne med lokkelyd-funksjonaliteten, en skulle sette seg inn i Met-APIet og en skulle sette seg inn i Google sitt kart-API. En fjerde person tok på seg oppgaven med å sette opp tre fragmenter som var knyttet til en såkalt "navigation bar". På denne måten kunne vi arbeide parallelt med de tre områdene uten at arbeidet til én person var for avhengig av arbeidet til en annen. Samtidig som kodingen var i gang, jobbet resten av teamet med innhold til rapporten.

I det følgende er en grov oversikt over hva hver av sprintene inneholdt. Det var likevel slik at noen arbeidsoppgaver i én sprint ble videreført til den neste på grunn av ulike utfordringer. Sprintene listet opp nedenfor er derfor mest ment å gi et inntrykk av rekkefølgen ting ble gjort, mens selve arbeidsmetodene i praksis lignet mer Kanban-tilnærmingen og flere arbeidsområder ble jobbet på parallelt.

Sprint 1 - komme i gang

I den første sprinten var det overordnede målet at vi ville opprette en app som viste noe (hva som helst) på skjermen. Vi ble enige om at det mest hensiktsmessige var nettopp å implementere en navigation bar hvor det var mulig å navigere mellom tre helt blanke sider. Mens en på gruppa jobbet med dette, begynte de øvrige i "kodegjengen" å eksperimentere

og leke seg i sine egne brancher med sine arbeidsområder. Dette var for å unngå dødtid mens den grunnleggende strukturen ble opprettet.

Sprint 2 - vise kart

Da navigasjonsbaren fungerte på en tilfredsstillende måte, satt vi som nytt mål å vise kartet på skjermen. Dette målet viste seg å være relativt trivielt isolert sett, men siden vi ønsket at det skulle fungere i tråd med MVVM-arkitektur, ble målet noe mer omfattende. Samtidige mål for sprinten var å hente inn værdata fra Met-API, og begynne arbeidet med lokkelyd-databasen og å se nærmere på hvilke farger som kunne passe i appen. Da det var tid for "sprint review", kunne vi konkludere med en fungerende meny, et kart som ble vist på skjermen uten brukerens posisjon og at værdata ble skrevet ut med Logcat.

Sprint 3 - database for lokkelyder

Målet for denne sprinten var å pushe database-arbeidet til "master branch" og at denne koden var i henhold til god MVVM-arkitektur. Det ble implementert en instrumentert enhetstest for å sikre at opprettelsen av databasen fungerte. Deretter ble det lagt inn dummy-lydfiler som senere skulle erstattes med ordentlige lokkelyder. I denne sprinten ble det også jobbet med å få til brukerens posisjon på kartet - noe vi til slutt fikk til. Fragmentet for innstillinger fikk dessuten sitt første design og fargepaletten for appen ble valgt.

Sprint 4 - dark mode og lydavspilling

På grunn av påskehøytiden ble denne sprinten lenger enn vanlig. Gruppen ønsket å redusere arbeidet i påsken, og det ble avtalt å avlyse møtene denne uka. Sprinten fortsatte uken etter påske. Et av målene var å fikse at kartet skulle vise plasseringen og automatisk zoome inn på brukerens plassering. I tillegg skulle det være et grovt fungerende innstillinger-fragment med en knapp for dark mode som faktisk fungerte. Videre ble det spilt inn og lagt til ordentlige lydfiler og bilder til lokkelyd-funksjonen, og det ble implementert en play-knapp som spilte av lydene. Det ble også avgjort at det ville være nok tid til å lage en funksjon for farevarsler i appen, så arbeidet med dette begynte.

Sprint 5 - bugfix og farevarsler

Den andre uka etter påske var det igjen tid for å oppdatere målene våre. Stadig hadde vi noen kryptiske bugs med kartet, samt mindre bugs i de to andre fragmentene. For eksempel viste det seg vanskelig å få til en fornuftig endring av farge for enkelte elementer i appen

ved skifte til dark mode. Målene for sprinten dreide seg derfor i stor grad om å fikse bugs, men også å få pilen for vindretning til å fungere, hente inn data for farevarsler fra API og vise disse i et pop up-fragment oppå kartet. Det ble nødvendig med en del samjobbing utenfor møtene for å fikse bugs i løpet av denne sprinten.

6.3.3 Rapportfasen

Sprint 6 - brukertest

Etter femte sprint, skiftet fokuset til teamet i større grad over til rapportskriving. Rapporten har hele veien blitt jobbet med, men i resten av sprintene ble mer av teamets ressurser satt av til rapportskriving. I sprint 5 var målet å få gjennomført brukertest med en person i målgruppen. Vi var heldige å få muligheten til å gjennomføre en fysisk brukertest i en realistisk setting. Brukertesten ga oss bekreftelser på ting vi visste om fra før, men også mye ny og nyttig informasjon. Med utgangspunkt i resultatene fra testen, ble det gjort enkelte endringer i appen. Dette dreide seg blant annet om fargevalg og plassering av knapper (se brukertest nedenfor i Kapittel 6.3.4).

Sprint 7 - kodedokumentasjon

I denne sprinten var målet å skrive det tekniske kapittelet om funksjonalitet i appen. Arbeidsfordelingen ga seg selv - den som hadde hatt hovedansvaret for en funksjonalitet, skulle skrive om denne. Deretter leste vi gjennom hverandres tekst for å se om det var mulig å forstå forklaringene for noen som ikke hadde jobbet med det området i appen. Vi valgte også å legge ved en del skjermbilder av koden for at teksten skulle bli enklere å følge. Mot slutten av sprinten meldte seg et behov for å få ekstern tilbakemelding på både struktur og innhold, og vi kontaktet derfor veileder for å få dette.

Sprint 8 - prosessdokumentasjon og enhetstester

I sprint åtte tok vi utgangspunkt i tilbakemeldingene som veileder hadde gitt på rapporten. Det ble nødvendig å diskutere generell struktur i dokumentet, og det ble stilt spørsmål om vi oppfyller kravet om enhetstester i koden. På dette tidspunktet var det kun implementert to enhetstester, og prinsippet om testdrevet utvikling kunne ikke sies å være fulgt veldig nøye opp. Vi konkluderte med at vi med fordel kan skrive et par enhetstester til, men at det viktigste var at prinsippet er forstått og at vi har lært oss hvordan det utføres teknisk. Det

store målet med denne sprinten var derimot å jobbe med prosessdokumentasjonen i rapporten. Til dette arbeidet var møtereferatene våre til stor nytte.

Sprint 9 - sluttføring og presentasjon

Den siste perioden frem mot innlevering av rapporten var å regne som en treukers sprint. Rapporten fremsto som en uslipt diamant, og det var mange detaljer som skulle på plass. Samtidig med arbeidet på rapporten, var det fortsatt detaljer i appen vi gjerne ville fikse. Alle i gruppen hadde dessuten andre innleveringer og eksamener å forholde seg til, så det ble en travel periode. Det ble lagt vekt på å jobbe med hver enkelt del til del ble *helt* ferdig - det var ikke lenger rom til å lage utkast på avsnitt i rapporten.

6.3.4 Brukertest

Brukertestingen har vært begrenset på grunn av restriksjoner knyttet til Covid-19. Vi ønsket i utgangspunktet å gjennomføre brukertester på uavhengige personer i målgruppen som ville komme med objektive tilbakemeldinger på sin brukeropplevelse. Siden målgruppen til vår applikasjon er jegere, og vi ikke har kunnet oppsøke folk i denne gruppen, har mye av den jevnlige brukertestingen foregått innad i teamet på en mer uformell måte.

Mot slutten av utviklingsprosessen fikk vi likevel mulighet til å utføre en mer tradisjonell brukertest på en person i målgruppen, men som riktignok hadde en relasjon til et av gruppemedlemmene. Vi tok høyde for at det kan ha oppstått *bias* som følge av denne relasjonen (Lazar et al., 2017, s.63). Problemet vårt var begrensede muligheter for hvordan å gjennomføre testen, og på denne måten ble testen gjennomført ansikt til ansikt med vedkommende. Håpet var at dette skulle gi bedre observasjoner og tilbakemeldinger enn ved en test utført over nettet. Vi forsøkte å hente ut så mye informasjon som mulig fra denne testen siden vi ikke hadde mulighet til å gjennomføre flere tester. Det kan derfor virke som resultatene i denne ene testen har blitt vektet uforholdsmessig tungt, men den endelige utformingen av appen er altså et resultat av innspill som er kommet både innad og utenfor teamet gjennom hele prosessen, samt at det er forsøkt å oppfylle visse krav og standarder med tanke på universell utforming. Brukertesten bestod av praktiske oppgaver som deltakeren skulle løse, mens medlemmet som utførte testen, målte resultatet. Dette kunne for eksempel være hvor lang tid, samt antall trykk, deltakeren brukte på å finne en lokkelyd til

elg, og spille den av. Etter at deltakeren hadde utført testene, kom vedkommende med tilbakemeldinger på hva han synes om de ulike delene av appen.

Etter testen med bruker var gjennomført ble resultatet oppsummert i ni ulike tiltak for videre arbeid med appen, med varierende grad av prioritering. Fordi deltakeren hadde problemer med å finne vindretningen så kjapt som vi ønsket, ble tiltaket å implementere vindretningen også i tekstlig form. Det andre tiltaket var å implementere en informasjonsside som forteller om appens funksjonaliteter, da deltakeren trengte beskjed om at den gule trekanten var farevarsler. Tredje tiltak var allerede en endring som var planlagt fra før, å endre plassering av knappene. Dette fikk vi også tilbakemelding fra deltakeren om. Deltakeren stusset over at selve kartet ikke hadde en darkmode, så fjerde tiltak ble å fikse dette. Fordi deltakeren synes farevarslene brukte lang tid på å laste inn, ble femte tiltak å forsøke å få dette til å gå fortere. Det sjette tiltaket ble å implementere funksjonen til å bytte til satellittkart i selve kartsiden. Syvende tiltak ble gjøre spinner-menyen i lokkelyder mer tydelig, og åttende ble å tydeligere markere hvor på elementene brukeren skulle trykke i innstillinger, da deltakeren ønsket mere responsive knapper. Deltakeren synes det var litt vanskelig å se hvilken side i appen han befant seg på navigasjonsbaren, så siste tiltaket ble å forsøke å gjøre dette tydeligere ved å for eksempel jobbe med fargekontrast eller symbol.

6.4 Refleksjon og vurdering av prosessen

6.4.1 Organisering

Vi har klart å ha en ryddig og oversiktlig struktur på alle de digitale ressursene. En god planleggingsfase bidro til at vi klarte å spre arbeidsmengden relativt jevnt utover semesteret. Prosjektplanen gjorde det lett å se hvilke uker den enkelte hadde møteansvar, finne tidspunkter for møter, samt hvor mye tid teamet hadde til rådighet. At teamet møttes annenhver dag, selv om det noen ganger ble ganske korte møter, var nyttig for å få løpende tilbakemelding. Det førte dessuten til god progresjon fordi alle følte de måtte ha gjort noe til neste møte. I tillegg ble det skrevet gode møtereferater etter hvert møte, med oversikt over hva den enkelte hadde gjort og skulle gjøre til neste gang. De detaljerte møtereferatene var en del av forberedelsene til “rapportfasen” som vi visste ville komme mot siste tredjedel av prosjektet.

Teamet kunne i større grad ha benyttet seg av Trello-tavlene slik det var planlagt. Et slags Kanban-board var en del av den smidige arbeidsmetoden vi hadde bestemt oss for, men realiteten ble at ingen i gruppa var flinke nok til å oppdatere Trello. Dermed fikk vi ikke utbytte av tavlene, og det ble bestemt at vi bare skulle skrote Trello omtrent halvveis i prosjektet. En mulig grunn til det var fordi møtereferatene var veldig utfyllende, så medlemmene heller benyttet disse for å få oversikt over oppgavene. Dette var litt synd, for vi opplevde egentlig de smidige tavlene i Trello som et godt verktøy for å organisere arbeidet.

6.4.2 Metodikk og samarbeid

Samarbeidet i gruppa har fungert bra, og det har vært lite konflikter. De gangene det har vært uenigheter har disse blitt løst i tråd med Team-avtalen. Dette innebar for eksempel å stemme over store valg, slik som ble gjort ved valg av case. Teammedlemmene har vært flinke til å møte opp til avtaler, og sagt ifra på forhånd de gangene det ikke har vært mulig. Det har også blitt gitt beskjed i de periodene der enkelte medlemmer har hatt mindre arbeidskapasitet i forbindelse med for eksempel eksamener. Dette har gjort det lettere å planlegge og fordele oppgaver, samt at man på forhånd er klar over situasjonen.

Vi gjennomførte noen retrospektiv-møter i tråd med Scrum. Her var fokus på hvordan samarbeidet hadde fungert i forrige sprint - hva som hadde fungert bra og hva som kunne gjøres annerledes i neste sprint. Her kom det for eksempel frem at møtene og samarbeidet ofte hadde en litt for formell eller profesjonell tone. Vi var mer fokusert på arbeidsoppgavene og det som ledet til fremgang i prosjektet fremfor småprat og "teambuilding". Vi forsøkte å ta med oss dette videre, og kunne bekrefte ved et senere retrospektivmøte at vi hadde fått til en mer uformell tone i gruppa.

Utover i prosjektet sluttet retrospektivmøtene å gi oss så mye nytt å jobbe med, og dermed bare var unødvendig tidsbruk. Vi bestemte oss for å slutte med disse møtene, og heller gi tilbakemeldinger fortløpende, både på hva som fungerte og hva som kunne forbedres. Dette fungerte greit for oss, det kom opp både noen ting som var bra og dårlige, men det er mulig disse ville bli oppdaget tidligere dersom vi hadde hatt retrospektiv etter hver sprint. Ved å sette av tid til å gi tilbakemeldinger blir det enklere å gi tilbakemeldinger, spesielt dårlige. Disse er ofte mer konfronterende, og man vil ofte vegre seg mot å gi denne typen

tilbakemeldinger før det virkelig irriterer, som kan være et problem. Vi følte allikevel vår metode fungerte bra for dette teamet, gruppemedlemmene var flinke til å gi tilbakemeldinger fortløpende.

En stor utfordring underveis har vært at vi ikke har hatt mulighet til å møtes fysisk og jobbe sammen på grunn av den pågående pandemien med covid-19. Dette har vi løst ved å møtes digitalt, og å være nøye på å ha det vi gjør digitalt så oversiktlig som mulig. Det er vanskelig å skulle løse problemer når feks en eller to i teamet har en bug som ikke de andre har. Mye hadde potensielt vært lettere å finne ut av hvis vi så hverandre ansikt til ansikt, og kunne jobbe med det sammen. En annen fordel teamet ville hatt av å møtes fysisk er at det ville vært lettere å bli kjent. Det var ingen som kjente hverandre fra før, og teamet sleit litt med at folk var stille og ikke tok ordet, spesielt i starten av prosjektet. Dette bedret seg som nevnt noe over tid.

6.4.3 Vurdering av kravspesifikasjonen

Vi har hatt dynamiske brukerhistorier og utformet kravspesifikasjonen i henhold til disse. Vi hadde utrolig mange ulike ideer til hva appen burde og kunne inneholde, og i tillegg har vi fått innspill gjennom intervjuer. Vi lagde en prioritering av brukerhistoriene, og laget en kravspesifikasjon der brukerhistoriene med lav prioritet typisk ble et "bør"-krav, mens brukerhistoriene med høy prioritet typisk ble et "må"-krav. Hvilke funksjonelle krav som ble implementert og ikke, er gitt i Kapittel 3.2.1.

Brukertesten ga oss tilbakemelding på hva som burde endres. Vi fulgte opp disse tilbakemeldingene og endret der det lot seg gjøre. Det ble endret så selve kartet fikk darkmode. Flere problemer knyttet til ikonene på kartet ble fikset, som for eksempel plasseringen. Menyen i lokkelyder ble gjort tydeligere. Innstillingene fikk slidere, istedenfor at det kom opp valg når man trykket på de. Og det ble implementert en informasjonsside som dukker opp første gang brukeren åpner appen, som gir informasjon om de ulike funksjonalitetene.

I innstillinger blir man sendt til app info, som ligger i innstillinger på selve mobilen, istedenfor å ha en egen innstilling for å endre posisjonsdata. I appens innstillinger på mobilen kan man fint endre dette selv. Dette er en mer kronglete måte å gjøre det, men det

sørget ihvertfall for at brukeren kunne endre tillatelsen sin dersom det skulle være ønskelig. Det ideelle hadde vært at brukeren fikk opp en egen slider for posisjon som ligner sliderene for darkmode og farevarsler.

Hva gjelder de ikke-funksjonelle kravene, er det satt av få ressurser til å teste om alle disse er oppfylt på en formell måte. I tabellen nedenfor gis likevel en vurdering om kravene er oppfylt basert på uformelle tester innad i teamet (gitt i Status-kolonnen).

#	**Ikke-funksjonelle krav**	**Test**	**Status**
1	Appen skal være universelt utformet.	Sjekk app opp mot krav i WCAG.	Godkjent*
2	Appen skal ha et enkelt brukergrensesnitt. Brukeren skal ikke trenge ekstern opplæring for å benytte seg av alle appens funksjonaliteter.	Observer helt ferske brukere av appen.	Ikke utført test
3	Appen skal ikke lagre posisjonen til brukeren.	Sjekk at brukerens posisjon ikke blir lagret lokalt i koden etter at brukeren har lukket applikasjonen.	Godkjent
4	Appen skal laste inn innhold for kartsiden på under 4 sekunder.	Mål tiden det tar å laste inn kartsiden.	Godkjent
5	Appens offline-funksjoner skal fungere når brukeren ikke har internett-tilgang.	Skru av internett på enheten og test funksjonalitet.	Godkjent
6	Appen skal ikke bruke mer enn 50 MB lagringsplass.	Sjekk app-info om databruk på enheten.	Godkjent
7	Appen skal utvikles ved bruk av smidig arbeidsmetodikk.	Dokumenter arbeidsmetoder.	Godkjent
8	Appen skal utvikles i Android Studio med Kotlin som programmeringsspråk.	-	Godkjent
9	Koden skal kommenteres nøye slik at en utvikler vil forstå, vedlikeholde og videreutvikle koden uten vanskeligheter.	Walkthrough av kildekoden.	Godkjent

10	Koden skal ha en arkitektur (MVVM) som gjør vedlikehold og videreutvikling i fremtiden enklere.	Dokumenter hvordan koden og klassene henger sammen og interagerer.	Godkjent**

* Gjelder utvalgte deler av WCAG 2.1

** Etter vår forståelse av MVVM-arkitektur og vårt kunnskapsgrunnlag.

6.4.4 Tekniske utfordringer

I de fleste prosjekter med apputvikling dukker det opp flere tekniske utfordringer underveis i utviklingen. JaktApp-prosjektet er ikke et unntak. Utfordringene, spesielt de større, ble tatt opp i plenum på gruppemøter, slik at teammedlemmene kunne reflektere og feilsøke sammen. Dette fungerte generelt sett veldig bra.

En utfordring som dukket opp var skalering av Popup-fragmentet. I utgangspunktet justerte ikke fragmentet seg etter liggende-modus siden fragmentet ikke plasserte seg i et fragment-view som de andre sidene, men heller som et underliggende element på toppen av lagene i Map-fragmentet. Problemet ble tatt opp i plenum i et gruppemøte slik at gruppen sammen kunne komme frem til løsning på problemet. Løsningen ble å implementere “constrained with” og “constrained height” med prosenter som verdier slik at fragmentet justerer seg automatisk etter høyden og bredden til Map-fragmentet ved rotasjon av applikasjonen.

Et annet problem gruppen støtet på var fargeendringer i Sound-fragmentet. Ved seleksjon av en lyd så endrer bakgrunnen seg bak elementet for å gi brukeren feedback på interaksjonen, og når brukeren trykker på en ny lyd så skal den pålagte bakgrunnen endre seg tilbake. Problemet var at den tidligere programmerte løsningen gjorde det slik at bakgrunnen til forrige valgt element blir gjort om til hvitt. Det førte til at denne løsningen ikke var kompatibel med darkmode, siden elementbakgrunnen ble gjort til hvit for hver lyd man trykker vekk. En god løsning på dette var å lage en underliggende stil i tema-filene i resources for både dark- og lightmode. Slik ble fargeendringene i fragmentet dynamiske uansett hvilke modus brukeren var på. Dette gjaldt også seekbaren, play- og gjenavspillingsknappen, og var løst på samme måte.

Spinneren i SoundFragment har ikke en dynamisk farge på valgt dyreart/kategori. Dette problemet fremsto ulikt hos noen av gruppemedlemmene. Mens noen kunne se teksten i darkmode, kunne andre bare se teksten i lightmode. Gruppen fant ikke ut en løsning av problemet, og måtte konkludere med at problemet skulle ignoreres siden denne utfordringen ble oppdaget relativt sent i prosjektet.

Tidlig i prosjektet oppdaget gruppen også et unormalt problem; JaktAppen ikke fikk tak i brukerens posisjon. Gruppen fant ut at dette er mest sannsynlig en bug i emulatoren. Dersom man åpner appen i en helt ny emulator er det ikke sikkert at posisjonen kommer opp, selv om emulatorens posisjon er satt. Dette kan man løse enten ved å "wipe" emulatoren og åpne appen på nytt helt til posisjonen dukker opp, eller så kan man gå inn i Google Maps applikasjonen i emulatoren, og finne emulatorens posisjon der. Da vil emulatoren ha "oppdaget" sin posisjon, og posisjonstjenesten vil fungere i appen også. Vi kom frem til at denne buggen skulle ignoreres på grunn av at sjansen for at brukeren aldri har brukt GPS på enheten sin tidligere er svært liten.

Noen tekniske forbedringer som gruppen ønsket å rette opp ble ikke utført grunnet mangel på tid. I utgangspunktet ønsket vi at brukeren skulle få mulighet til å gi applikasjonen tillatelse til å bruke blant annet lokasjonsdata i selve welcome-skjermen. Dersom brukeren ikke ønsker at appen skal få tilgang på telefonens posisjon, ville brukeren kun ha måttet avslå denne forespørselen én gang, siden denne skjermen bare dukker opp første gang man åpner appen. I den nåværende versjonen av appen må brukeren avslå denne forespørselen hver gang appen blir åpnet, som kan være irriterende for brukeren. Dette ble nedprioritert fordi gruppen kom fram til at det var lite sannsynlig at noen ville bruke appen uten å gi tillatelse til å bruke telefonens posisjon, fordi appen krever dette for å finne vindretningen og vindstyrken.

Gruppen ønsket også å flytte metodene checkPermission(), requestPermission(), isLocalEnabled(), og onrequestPermissionResult() fra MainActivity til en egen klasse. Dette hadde gitt mer oversiktlig kode og høyere kohesjon for MainActivity. Det samme gjelder noe av koden i MapFragment, spesiellt metoden getCounty() og observatøren på getLocationData(). Disse kunne for eksempel vært flyttet til PopupFragment eller MapViewModel, som ville forbedret kohesjonen. Siden appen fungerte uten disse

forbedringene på koden, og dette ikke var et enormt prosjekt hvor disse metodene skulle bli brukt i store deler av koden, valgte gruppen å nedprioritere dette.

Applikasjonen har også problemer med enkelte Android API-nivåer over 23. Det hender at JaktAppen krasjer ved oppstart av appen med en ukjent feil, noe som gruppen mistenker er emulator bug. Gruppen har ikke funnet en måte å løse dette på, og valgt gjennom en beslutning å ignorere dette problemet med tanke på at prosjektets krav er at applikasjonen skal klare å kjøre uten feil på utvalgt API-nivå, som i dette tilfellet er API-nivå 23 (Marshmallow).

Til syvende sist har det dukket opp flere tekniske utfordringer av forskjellige kategorier underveis i prosjektet. Gruppen har tatt for seg utfordringene i plenum og reflektert disse sammen for en ordentlig gjennomgang. Slik er det mulig å komme frem til en beslutning og potensielt løse problemene.

7 Konklusjon på problemstilling

I dette prosjektet ønsket vi å utvikle en applikasjon som hjelper jegere med å oppnå en vellykket jakt. Ut ifra egne erfaringer, brukerundersøkelser og diskusjon i teamet har arbeidet resultert i en app med funksjonalitet for å se brukerens nåværende posisjon på et kart samtidig som vindretning og vindstyrke vises på kartet. I tillegg har applikasjonen en lokkelyd-funksjon som gir mulighet til å spille av et utvalg dyrelyder som lokker viltet inn på skuddhold.

Vi mener denne appen vil hjelpe jegeren med å lykkes på jakt.

Referanser

Android developers. (2021, 26.1). *Guide to app architecture*. Hentet 16.3.2021 fra: https://developer.android.com/jetpack/guide

Uutilsynet. *WCAG Standarden.* Hentet 09.04.2021 fra: https://www.uutilsynet.no/wcag-standarden/wcag-20-standarden/86

Tina Brock. (2020, 21.5). Hentet 21.03.2021 fra: https://www.nrk.no/osloogviken/okt-mobilbruk_-mange-kjenner-det-i-mobilarmen-1.15001267

Lazar, J., Feng, J. H. & Hochheiser, H. (2017). *Research methods in human-computer interactions* (Second edition). Cambridge, United states: Morgan Kaufmann publications.

Lid, Inger Marie: *Universell utforming*. *Store norske leksikon*. Hentet 23.04.2021 fra https://snl.no/universell_utforming

Lindsjørn, Y. (2021, 23.2). *IN2000-cases-v21*. Hentet 16.3.2021 fra: https://www.uio.no/studier/emner/matnat/ifi/IN2000/v21/prosjekt-og-casebeskrivelser/

Martini, A. (2021, 16.2). *in2000.2021.02.16-architecturetechnicaldebt*. Hentet 16.3.2021 fra: https://www.uio.no/studier/emner/matnat/ifi/IN2000/v21/forelesninger/

Mikhalchuk, O. (2020, 10.12). *The importance of unit testing, or how bugs found in time will save you money.* Hentet 25.4.2021 fra: https://fortegrp.com/the-importance-of-unit-testing/

Preece, J. Rogers, Y. & Sharp, H. (2015) *Interaction design. Beyond human-computer interaction.* (4th edition) Chichester, West Sussex, United Kingdom. John Wiley & Sons Ltd.

Sommerville, I. (2016). Software Engineering, 10th Edition. University of Lancaster, United Kingdom, University of St Andrews, Scotland: Pearson education

Sommerville, I. (2020). *Engineering software products. An introduction to modern software engineering*. Hoboken, NJ, USA: Pearson education

Statistisk Sentralbyrå (2020). Hentet 18.3.2021 fra: www.ssb.no/statbank/table/03508/

Stenslund, E. og Almås, S. (2021, 14.1)
in2000.2021.01.14.grunnleggende.android.studio.pdf . Hentet 25.3.2021 fra:
https://www.uio.no/studier/emner/matnat/ifi/IN2000/v21/forelesninger

Software testing fundementals (2020). Hentet 25.4.2021 fra:
https://softwaretestingfundamentals.com/integration-testing/#Method

Vihovde, E.H. (2021). *An Introduction to Software Testing.* Hentet fra:
https://www.uio.no/studier/emner/matnat/ifi/IN2000/v21/forelesninger/in2000.2021.02.04.testing.pdf

Appendiks

Appendiks A Intervjuguide innledende, målgruppe

Introduksjon

Fortell om hvem vi er og hvorfor vi intervjuer vedkommende, og takke for deltakelse.

Vi er en gruppe studenter som tar faget IN2000 Software engineering med prosjektarbeid, på Universitetet i Oslo, og skal i den sammenheng lage en app. I vårt prosjekt har vi valgt å fokusere på temaet jakt, og jobber med å lage en app som kan være til nytte på nettopp jakt. Vi har derfor spurt om du vil delta i et intervju, for å gi oss en bedre innsikt i hva en slik app bør inneholde. Hvis det er greit for deg, vil det bli gjort opptak underveis av intervjuet. Dette opptaket vil kun deles med de andre i teamet. Gjengivelser av funn vil bli anonymisert og formulert i en rapport, som vil bli delt med våre faglærere.

Oppvarming

- Kan du fortelle litt om deg selv?
    - Navn, alder etc
- Hvor lenge har du drevet med jakt?

Hoveddel

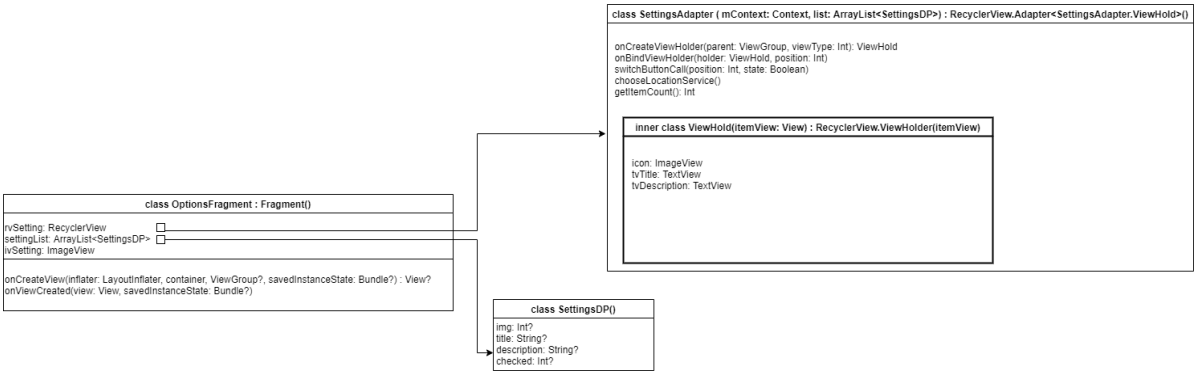
- Kan du ta oss gjennom en typisk jakt?
    - Hvilke oppgaver er det som må løses?
    - Hvor mange er med?
    - Hvilke hjelpemidler benyttes for å utføre ulike oppgaver?
    - Hvilke dyr er det du jakter/har jaktet på?
- Hvilken type informasjon er nyttig for deg å ha på jakt?
- Hvilke funskjoner kan du se for deg at er nyttige i en app når du er på jakt?
- Hvilken påvirkning har vær og vind når du er ute på jakt?
- Hvis du skulle fått farevarsler, hvilke farer er relevant å få varsler om?

Oppsummering

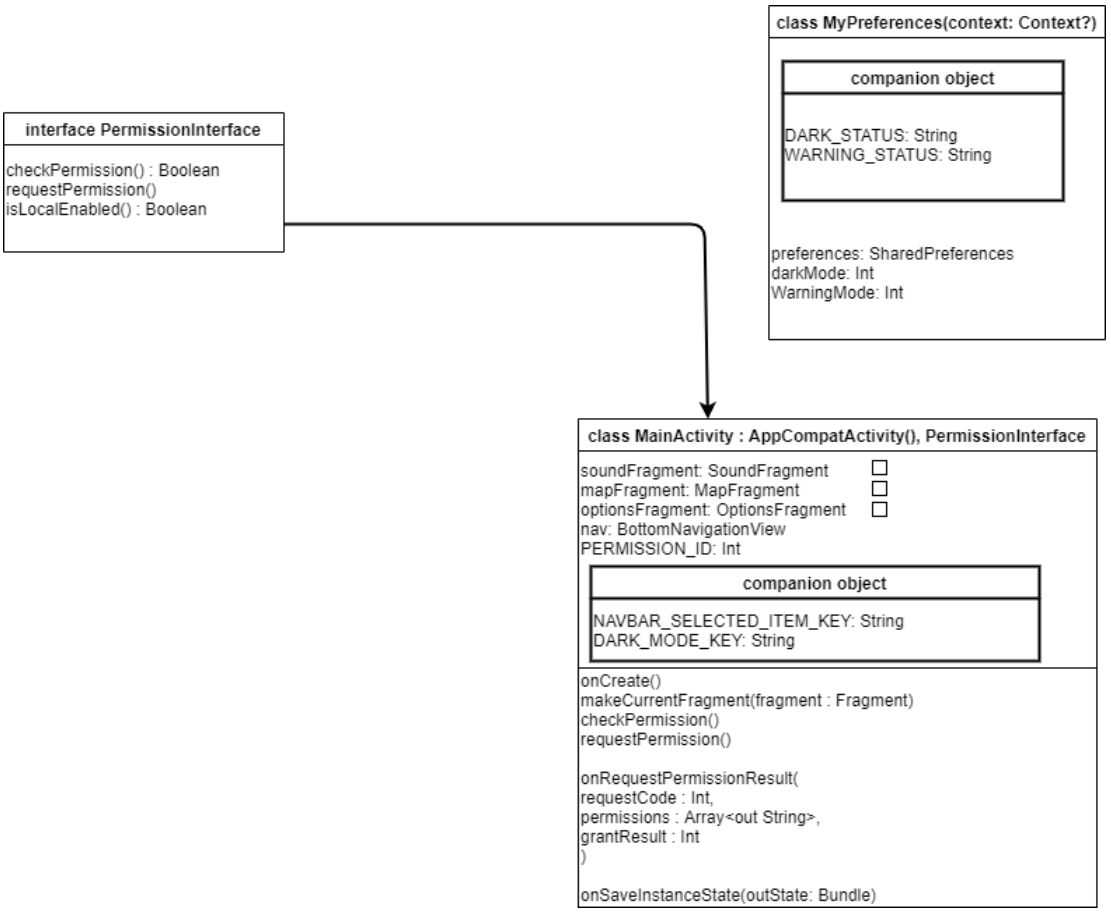
- Er det noe mer du vil legge til?
- Har du noen spørsmål?
- Takke for at deltakeren har villet delta på dette intervjuet, og ønske dem en fin dag videre.

Appendiks B Klassediagram

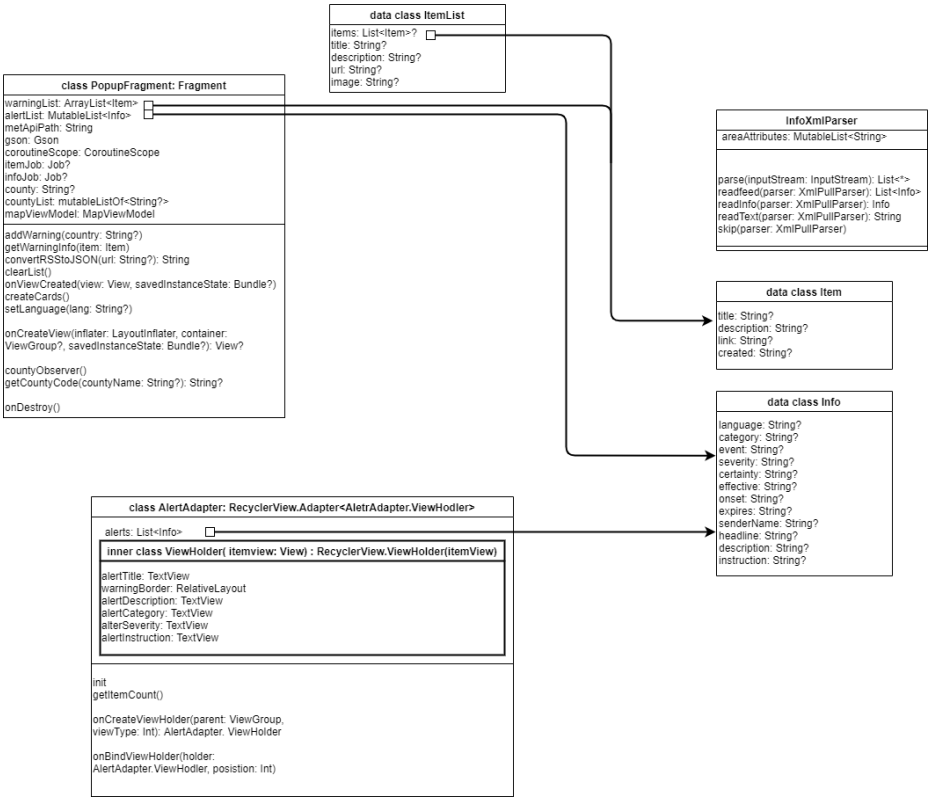
Klassediagram OptionsFragment



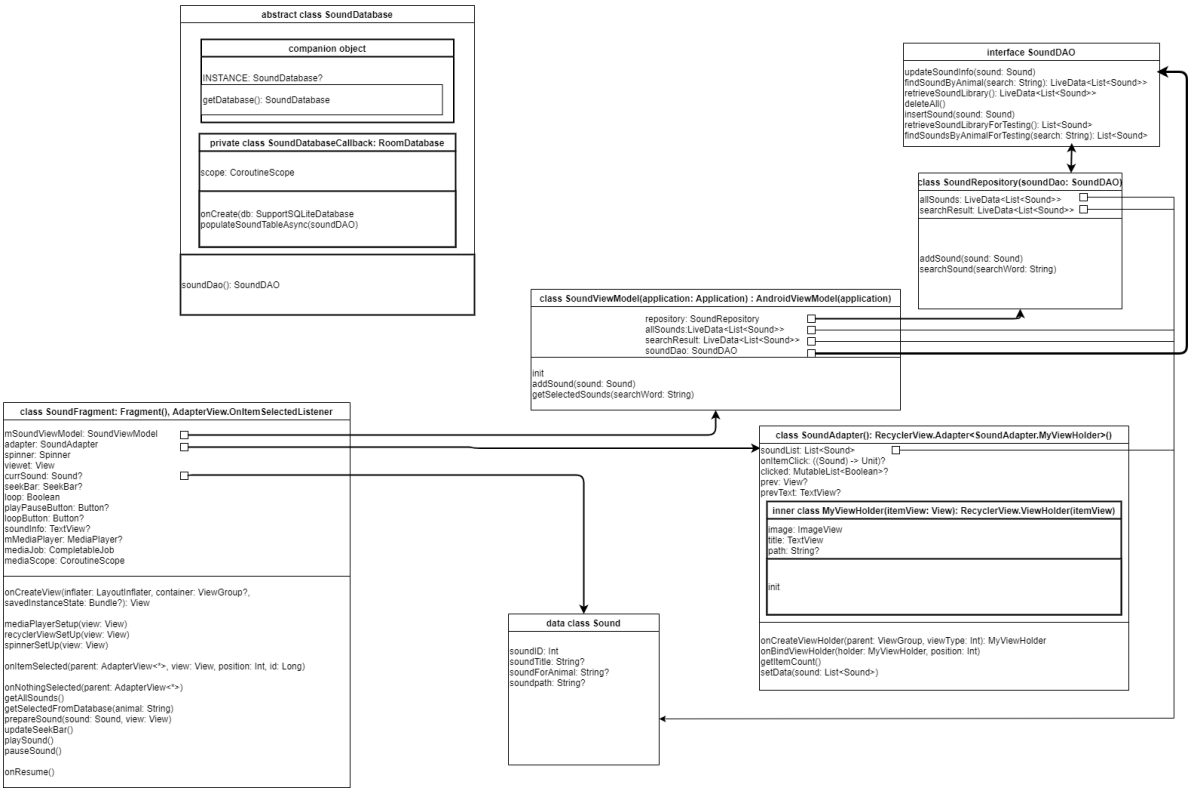
Klassediagram MainActivity



Klassediagram PopupFragment



Klassediagram SoundFragment



Appendiks C Teamavtale for Team 36

Teamavtale

Teammedlemmer:

Emil Knutsen, Tobias Myren Foldal, Martin Borgaard Waage, Trygve Kjensli, Waleed Nasir, Daniel Faour.

Mål:

Vi skal arbeide effektivt som et team i 12 uker og bli godt kjent med hverandre gjennom prosjektet. Teamet skal utvikle en flott app til det caset vi velger i første iterasjon, og skrive en fyldig og konkret rapport sammen.

Oppmøte:

Det forventes at man møter opp klart til møtet i tide. Gjerne gå gjennom agendaen for dagens møte i forkant for å bli ekstra klar slik at teamet ikke bruker unødvendig tid for å komme i gang.

Fravær:

Ved fravær fra møter skal man selv gi beskjed til gruppa om fraværet, minimum 12 timer før møtet, med begrunnelse for fraværet. Unntak for dette gis ved nødsituasjon eller krisetilfeller.

Møter:

Scrum-master rullerer fra uke til uke (se prosjektplan) og vedkommende har ansvar for å lage Zoom-rom. Dagsorden er satt i forkant av møtet og gruppemedlemmene har lest denne. Møtene må startes til avtalt tid. Det er viktig at man ikke kommer for sent. Det er OK å komme sent av og til, og vi gjør ikke et stort nummer ut av det. Vi starter møtene med oppfølging av avtaler fra forrige møte. Det er viktig å gå gjennom oppgavene fra sist gang, viktig å ikke henge ut den som ikke har gjort oppgavene sine. Det er viktig med fasilitering av skriving. Vi må styre prosessen for å få struktur og form på oppgaveskrivingen. Få gruppemedlemmene til å gi feedback. Til slutt gå gjennom om alle punktene er nådd på dagsorden, 20 minutter før slutt.

Evaluering av møtene hver fredag; hva fungerer bra, hva kan forbedres? Nevn gjerne 3 ting.

Tidsbruk:

Det vil bli satt av 20 timer i uka til arbeid med prosjektet i utgangspunktet. Antall timer kan variere, med mer å gjøre i perioder før frister.

Forventninger:

Det forventes at alle bidrar i tråd med felles bestemmelser. Man har et individuelt ansvar for å sørge for at egne arbeidsoppgaver blir gjort innenfor de tidsfrister som er satt. Dersom man ikke klarer å utføre sine oppgaver til den fastsatte fristen, forventes det at man informerer resten av gruppen og spør om hjelp. Hvis man blir spurt om hjelp, forventes det at man hjelper, innenfor rimelighetens grenser, hvis ikke man selv sliter med å overholde frister. Om en ikke skulle ha noen oppgaver forventes det at man sier ifra slik at nye oppgaver kan bli fordelt. Det forventes videre at man holder seg oppdatert på prosjektets status og eventuelle endringer på *Google Disk, GitHub og Trello*. Dersom man ikke kommer på møter forventes det at man oppdaterer gruppen om sin framgang på arbeidsoppgaver.

Konflikter:

Overordnet mål med konflikter er at konflikten resulterer i et bedre team, bedre prosesser og tettere samarbeid. Vi forsøker å ta tak i uenigheter før de utvikler seg til konflikter. Hvis konflikter oppstår mellom enkelte gruppemedlemmer, forsøker de involverte å oppklare situasjonen mellom seg før det eventuelt tas opp i plenum. Vi tåler kritikk, og lytter til andres synspunkter. Ved enkle uenigheter som angår hele teamet, benyttes avstemming. Dersom uenigheten ikke kan løses ved avstemming, går et av gruppemedlemmene inn som megler mellom partene. Veileder kontaktes hvis situasjonen eskalerer.

Beslutninger:

Beslutninger som tas, skal det i utgangspunktet være enighet over, der dette ikke er mulig vil vi ta utgangspunkt i avstemning. Hvis det fortsatt ikke er enighet, vil det bli slått kron eller mynt, eller evt at man involverer gruppelærer/veileder som en nøytral tredjepart som kan hjelpe med å mekle mellom partene.

Konflikthåndtering:

I utgangspunktet følger vi følger vi prosedyren for ‘konflikter’. Ved mer alvorlige konflikter oppsøker vi veileder eller faglærer dersom teamet selv ikke klarer å løse konflikten.

Emil Knutsen	Tobias Myren Foldal
_____	_____
Martin Borgaard Waage	Trygve Kjensli
_____	_____
Waleed Nasir	Daniel Faour
_____	_____

Appendiks D Samtykkeskjema intervju

Vi er studenter i kurset IN2000 – Software Engineering med prosjektarbeid, ved Institutt for informatikk, Universitetet i Oslo. Kursveileder er Johanne Thunes, epost: johanthu@ifi.uio.no. Intervjuansvarlig er Daniel Faour, epost: daniea1602@gmail.com, tlf: 47629779.

Prosjektet vårt handler om å velge et case og bygge en app som fyller eventuelle krav til caset. Vi valgte en fri case hvor vi skal utvikle en jakt-app med utvidelser som formidling av farevarsler. Hensikten med intervjuet er å samle inn nok data til å etablere eventuelle krav til systemet og andre ideer som bør reflekteres og vurderes. Prosjektet og innsamlet data blir kun brukt i dette kurset.

Frivillig deltakelse

Alt er frivillig, og du kan trekke deg når som helst ved ønske. Intervjuet skal loggføres gjennom lydopptak/videoopptak, og sikkert notater underveis i tillegg for å huske ting underveis gjennom intervjuet.

Du kan avslutte intervjuet når som helst og/eller trekke tilbake informasjon når som helst som er gitt under eller observasjon. Det vil også si at du kan velge at noen deler av intervjuet ikke skal brukes som blant annet sensitiv informasjon (Siden begrepet «sensitiv informasjon» er veldig subjektivt).

Anonymitet

Du vil forbli anonym. Alt av innsamlet data gjennom observasjon og intervjuet vil bli anonymisert. Det vil si at ingen andre en meg vil vite hvem som er blitt intervjuet, og informasjonen vil ikke kunne tilbakeføres til deg, med mindre du ønsker det.

For å delta i dette intervjuet må du samtykke i deltagelsen ved å undertegne på at du har lest og forstått informasjonen på dette arket.

Samtykke

Jeg har lest og forstått informasjonen over og gir mitt samtykke til å delta i intervjuet

Sted og dato	SIGNATUR FRA DELTAKER

Appendiks E Samtykkeskjema brukerundersøkelse

Formål

Som en del av en studentoppgave ved UiO, skal vi undersøke hvordan brukere i målgruppen til JaktApp bruker og interagerer med vår applikasjon. Denne studien omfatter kun deg, og vil ikke bli publisert.

Hvem er ansvarlig for brukerundersøkelsen?

Trygve Kjensli, student ved Universitetet i Oslo, matematisk-naturvitenskaplige fakultet.
E-post: kjensli.trygve@gmail.com

Hvorfor får du spørsmål om å delta?

I denne undersøkelsen ønsker vi å analysere brukere som er midt i appens målgruppe, og som samtidig har en annen bakgrunn enn oss i utviklingsteamet.

Hva innebærer det for deg å delta?

Dette er en observasjonsstudie. Du vil få tildelt forhåndsbestemte oppgaver som skal utføres én etter én på en tildelt mobiltelefon, mens jeg noterer hvordan oppgavene løses. Du vil bli bedt om å tenke høyt mens du utfører oppgavene.

Det er frivillig å delta

Det er frivillig å delta i denne brukerundersøkelsen. Hvis du velger å delta, kan du når som helst trekke samtykket tilbake uten krav om begrunnelse. Alle opplysninger om deg vil bli anonymisert.

Ditt personvern – hvordan dine opplysninger oppbevares

Opplysningene om deg vil kun bli brukt til formålet som er oppgitt ovenfor. Opplysningene behandles konfidensielt og i tråd med personopplysningsloven. Resultatene vil kun bli brukt i en rapport som en del av studentoppgaven.

Hva skjer med opplysningene dine når prosjektet er avsluttet?

Studentoppgaven lagres i sin helhet på UiO sine krypterte servere. Ingen av dataene som lagres kan knyttes til deg som person.

Dine rettigheter

Så lenge du kan identifiseres i datamaterialet, har du rett til:

- Innsyn i hvilke personopplysninger som er registrert om deg.
- Å få rettet personopplysninger om deg.
- Å få slettet personopplysninger om deg.
- Å klage til personvernombudet eller Datatilsynet om behandlingen av dine personopplysninger.

Hva gir meg rett til å behandle personopplysninger om deg?

Vi behandler opplysninger om deg basert på ditt samtykke.

Hvor kan jeg finne ut mer?

Hvis du har spørsmål til undersøkelsen, eller ønsker å benytte deg av dine rettigheter, ta kontakt med hovedansvarlig (se ovenfor).

Med vennlig hilsen

Trygve Kjensli
(prosjektansvarlig)

Jeg samtykker til:

Å delta i brukerundersøkelsen

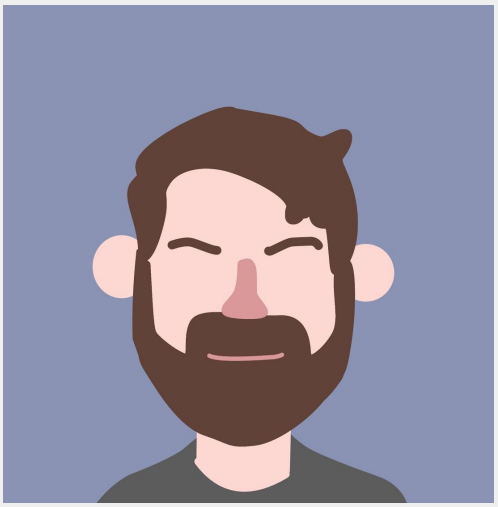
(dato, signert av deltaker)

Appendiks F Prosjektplan

Uke	Møte	Info
22.02 - 28.02		27.02 kick-off
01.03 - 07.03	mandag 14:00 onsdag 12:00 fredag 12:00	Planlegging Team-kontrakt Velge case, verktøy, metodikk Møtereferat: Daniel fredag 05.03 frist for å ha funnet case
08.03 - 14.03	mandag 12:00 onsdag 12:00 fredag 13:00	møtereferat: Martin fredag, møte flyttet til 13 fredag 12.03 oblig 3 innlevering
15.03 - 21.03	mandag 12:00 onsdag 12:00 fredag 12:00	møtereferat: Emil Denne uka starter vi med koding
22.03 - 28.03	mandag 12:00 onsdag 12:00 fredag 12:00	møtereferat: Tobias
29.03 - 04.04	mandag 12:00	møtereferat: Trygve Påske
05.04 - 11.04	mandag 12:00 onsdag 12:00 fredag 12:00	møtereferat: Waleed
12.04 - 18.04	mandag 12:00 onsdag 12:15 fredag 12:00	møtereferat: Daniel
19.04 - 25.04	mandag 12:00 onsdag 12:15 fredag 12:00	møtereferat: Martin brukertest gjennomført mandag
26.04 - 02.05	mandag 12:00 onsdag 12:15 fredag 12:00	møtereferat: Emil 30. april: levere førsteutkast av rapport til grp lærer

03.05 - 09.05	mandag 12:00 onsdag 12:15 fredag 12:00	møtereferat: Tobias møte fredag avlyst
10.05 - 16.05	mandag 12:00 onsdag 13:15 fredag 12:00	møtereferat: Trygve obs utsatt møte onsdag
17.05 - 23.05	tirsdag 12:00 onsdag 14:15 torsdag 14:15 fredag 12:00	møtereferat: Waleed presentasjon av prosjekt denne uka torsdag 20. 13:30 - 15:00 fredag 21.05 innlevering prosjekt

Appendiks G Persona



Andreas
Alder: 25
Bosted: Trondheim
Sivilstatus: Singel
Yrke: Student

Biografi

Andreas bor til vanlig i kollektiv i Trondheim og studerer for å bli kroppsøvingslærer på ungdomsskolen eller videregående. Ved siden av dette jobber han deltid i en sportsbutikk. På fritiden driver han med jakt og fiske. Han er ofte på jakt sammen med broren sin, som har samme interesse. Andreas foretrekker å jakte på storvilt.

Motivasjoner	Frustrasjoner
- Fysisk aktivitet - Naturopplevelser	- For lite tid til å gjøre ting - Dyr offentlig transport

Scenario

Andreas er på elgjakt med elghunden sin. Elghunden finner en stor elgokse i skogkanten og ender i stålos ca 500 meter inn i skogen. Andres må nå forsøke å smyge seg inn på skuddhold. Han sjekker vindretningen på JaktApp for å planlegge hvordan han bør gå for å at elgen ikke skal sanse ham.

Appendiks H Brukertest

Testens formål

Vi forventer å observere om brukeren opplever systemet som brukervennlig, enkelt og forståelig, samt om vedkommende anser funksjonaliteten som nyttig og tilstrekkelig. Spørsmål vi stiller oss i forkant er blant annet:

- Navigerer brukeren seg rett inn på det vedkommende er ute etter, eller er det mye klikking og feiling?
- Forstår brukeren vindvarselet?
- Forstår brukeren symbolene som brukes på kartsiden?
- Er brukeren klar over alle tjenestene som appen tilbyr?
- Tenker brukeren likt som vi har gjort?
- Savner brukeren noe funksjonalitet?

Om deltakeren

Deltakeren i undersøkelsen er en del av målgruppen. Det er en mann på 30 år som har drevet med jakt siden han var ungdom. Deltakeren er venn av et av gruppemedlemmene.

Utforming av undersøkelsen

I forkant av brukertestingen ble det laget en plan for testen som inneholdt oppgaver som testdeltakeren skulle gjennomføre (Lazar et al., 2017, s. 274). Oppgavene ble laget gjennom en diskusjon i teamet, og fokus var at hvert spørsmål skulle rettes mot en spesifikk funksjonalitet i appen. For at testen skulle utføres i riktigst mulig kontekst (Lazar et al., 2017, s. 286), ble testen utført i skogen og testdeltakeren ble bedt om å forestille seg at han var på jakt. Resultatet av oppgavene ble målt gjennom tidtakning, telling av feil og måling av hvor vanskelig deltakeren synes oppgaven var (Lazar et al., 2017, s. 288).

Utførelse av undersøkelsen

Deltakeren fikk låne mobiltelefon hvor applikasjonen er ferdig installert. Deltakeren var fra før vant til å bruke Android-enheter. Deretter ble deltakeren bedt om å tenke høyt mens han utførte oppgavene - spesielt hvis det var noe han brukte mye tid på eller synes er vanskelig. Hvis det var oppgaver deltakeren ikke klarte å gjennomføre, ble han bedt om å forklare hva

han ikke forsto. Etter at oppgavene var utført, ble deltakeren bedt om å leke seg rundt i appen i noen minutter og bite seg merke i ting han likte og ikke likte på et generelt grunnlag.

Oppgaver for deltakeren

1. Finn din lokasjon på kartet. Fortell meg hva vindretningen er og hva vindstyrken er. Pek i hvilken retning det blåser.
2. Sjekk om det er noen farevarsler i området du befinner deg i.
3. Endre kartvisning til satellitt (funksjonen er ikke implementert der vi ønsker ennå).
4. Finn en lokkelyd som lokker til seg elg og spill den av.
5. Endre modus til dark-mode (ikke fullstendig implementert).

Observasjoner

Oppgave 1: Lokasjon og vind

Måling: Tid som brukes på å utføre oppgaven, og hva som eventuelt ble gjort feil.

Resultat:

Deltakeren fant sin lokasjon på kartet umiddelbart. Etter noen sekunder fant han også vindhastigheten: "Vindhastigheten er 2.1, som jeg antar er meter per sekund". Deltakeren hadde problemer med å finne vindretningen. Han forsøkte å snu og vende på telefonen, samt å rotere kartet for å se om dette påvirket pilen på skjermen. Han fikk etter hvert veiledning om at pilen viser hvilken vei det blåser så lenge kartet er rotert med nord opp på telefonen. Han rettet da telefonen slik at nord på kartet samsvarte med nord i virkeligheten (som han visste intuitivt hvor var) og kunne nå peke ut vindretningen.

Oppgave 2: Farevarsel

Måling: Brukerens opplevelse av hvor enkelt det var å finne farevarsel.

Resultat:

Deltakeren kikket først ned på navigasjons-baren for å se hvor det var naturlig å finne et farevarsel, og konkluderte ganske fort at "kart" var det mest naturlige stedet å lete ut fra de valgene som fantes. Han trodde umiddelbart at den gule trekanten øverst i venstre hjørne var en markering på kartet, men da han beveget på kartet så han at det var noe som lå fast på skjermen. Han trykket da på trekanten, fikk opp en tom boks og trykket ganske kjapt på "OK" fordi han ikke fant noe der. Deltakeren fikk da tips om at det kan være nødvendig å vente litt før informasjonen dukker opp i boksen. Han trykket da på trekanten igjen, og etter 4-5 sekunder fikk han opp farevarsler: "Aha, nice! Men det burde vært noe som fortalte meg

at jeg var på riktig sted eller at data lastes inn. Helst burde det gått litt fortere også, men det er kanskje ikke så enkelt å gjøre noe med?".

Oppgave 3: Satellitt

Måling: Hvor forventet brukeren å finne denne funksjonen?

Resultat:

"Jeg forventer å finne den i margen på karet slik som man gjør hos Google. Det er tross alt et Google-kart som benyttes, og da forventer man at en del funksjoner er plassert likt som i Google Maps.".

Oppgave 4: Lokkelyd

Måling: Tid som brukes på å utføre oppgaven, samt antall trykk på enheten.

Resultat:

Deltakeren brukte fire klikk og omlag seks sekunder for å finne frem til lokkelyden og spille den av. Da han klikket på den riktige lyden i listen, stoppet han opp et sekund eller to fordi han forventet at lyden skulle spilles av automatisk. Så oppdaget han play-knappen og trykket på den. Han uttrykte at han likte denne løsningen: "Dette er egentlig veldig lurt. Det er fort gjort å komme bort feil lyd når man scroller i listen med kalde fingre eller hansker på, og hvis man da spiller av feil lyd kan man ødelegge jakta totalt.". Deltakeren la ikke merke til Spinneren øverst på siden som lar deg velge hvilket dyr du vil se lokkelyder for. Han påpekte at skriften var for liten og anonym.

Oppgave 5: Dark mode

Måling: Tid som brukes på å utføre oppgaven, samt antall trykk på enheten.

Resultat:

Deltakeren brukte tre sekunder på oppgaven. Han opplevde det som intuitivt hvor funksjonen var plassert.

Deltakerens frie kommentarer

- "Informasjonen om vind burde vært en mye mer elegant pil (kanskje inni en sirkel) og ved siden av denne kan de stå typ (NØ) 2.1 m/s."
- "Knappene på kartsiden er ikke på linje, og dessuten burde de være samlet på ett sted og ha samme størrelse."
- "Vindretning er veldig greit å ha på jakt da, så dette har jo potensiale."

- “Hvorfor blir ikke selve kartet også mørkt i dark-mode?”
- “Å kunne slå av og på farevarsler burde kunne gjøres med en knapp rett på der det nå står Farevarsel.”
- “Jeg hadde i utgangspunktet ikke tenkt på at denne appen ville inneholde farevarsler, men det har jo en viss nytteverdi.”
- “Knappene i innstillinger funker ikke alltid. Mulig jeg trykker på feil sted. Enten bør det være mulig å trykke overalt, eller så burde dere markere tydeligere hvor det kan trykkes.”
- “Navigasjonsbaren er ganske sexy, men det eneste jeg sliter med å lese er den siden jeg faktisk er på. Kanskje dere burde markere det tydeligere.”
- “Valg-menyen øverst i lokkelyder er som sagt for anonym. Jeg ville hatt samme skriftstørrelse der som på teksten for valgt lokkelyd.”

Evaluering av observasjoner

1. Deltakeren hadde problemer med å finne vindretningen så kjapt og enkelt som vi ønsker.

Tiltak: Implementer vindretning også i tekstform (N, NØ, Ø, SØ, S,... osv). Vindpilen plasseres ved siden av denne teksten. Høy prioritet.

2. Deltakeren ville ikke tenkt at den gule trekanten var et farevarsel med mindre han hadde fått beskjed om å lete etter et farevarsel.

Tiltak: Implementer informasjonsside som forteller om appens funksjonaliteter første gang brukeren åpner appen. Høy prioritet.

3. Deltakeren mente knappene / funksjonene på kartsiden var dårlig plassert og ikke på linje.

Tiltak: Ingen nye - det finnes allerede en plan for hvor knappene til slutt skal plasseres.

4. Deltakeren stusset over at selve kartet ikke ble mørkt i dark-mode.

Tiltak: Undersøk omfang av en slik implementasjon. Middels prioritet.

5. Deltakeren syntes farevarsler brukte litt lang tid på å laste inn.

Tiltak: Mulig med en liten optimalisering i koden. Middels prioritet.

6. Deltakeren ønsket at funksjon for å bytte til satellittkart skulle være på selve kartsiden.

Tiltak: Implementer knappen på kartsiden sammen med de andre kart-funksjonene. Høy prioritet.

7. Deltakeren synes Spinner øverst på lokkelyd-siden var for anonym.

Tiltak: Implementer større tekst på Spinneren. Høy prioritet.

8. Deltakeren ønsket enklere og mer responsive knapper i Innstillinger.

Tiltak: Marker tydeligere hvor på elementene brukeren skal trykke. Implementer av/på-knappen for farevarsler og for dark-mode som en enkel "slider". Høy prioritet.

9. Deltakeren syntes det var litt vanskelig å se akkurat hvilken side han var på i appen i navigasjonsbaren.

Tiltak: Undersøk om det er mulig å bruke farger med høyere kontrast, eller om det er mulig å gjøre tekst/symbol på siden man er på større enn de andre. Middels prioritet.