Norwegian University of
Science and Technology

Småviltrapportering

Forfattere
Lars Johan Nybø
Linn-Hege Kristensen
Jaran Godejord

Bachelor i programvareutvikling
20 ECTS
Institute for Datateknikk og Informatikk
Norges teknisk-naturvitenskapelige universitet,

16.05.2018

Veileder Tom Røise

Sammendrag av Bacheloroppgaven

Tittel:	**Småviltrapportering**
Dato:	16.05.2018
Deltakere:	Lars Johan Nybø Linn-Hege Kristensen Jaran Godejord
Veiledere:	Tom Røise
Oppdragsgiver:	Escio AS
Kontaktperson:	Håvard Narvesen, hn@escio.no, 990 35 053
Nøkkelord:	Jakt, mobil, rapporter, kart, offline, statistikk, kryssplattform, API, RESTful, JSON, React Native, Java, JavaScript, SCRUM
Antall sider:	150
Antall vedlegg:	8
Tilgjengelighet:	Åpen

Sammendrag: Målet for denne oppgaven har vært å forenkle jegernes hverdag ved å gi dem en mobilapplikasjon som kan benyttes som et samlet sted for rapportering av småvilt. I tillegg har forvalterne fått et intuitvt grensesnitt for statistiskk. Utfordrende aspekter med oppgaven har hovedsaklig vært å håndtere plotting av småvilt på kart og offlinefunksjonalitet. På server-siden var den største utfordringen å sette opp et API som følger ønsket standard. For å løse oppgaven har gruppen benyttet en rekke spennende teknologier som React Native, Java, JavaScript og RESTful API.

Summary of Graduate Project

Title:	**Småviltrapportering**
Date:	16.05.2018
Authors:	Lars Johan Nybø Linn-Hege Kristensen Jaran Godejord
Supervisor:	Tom Røise
Employer:	Escio AS
Contact Person:	Håvard Narvesen, hn@escio.no, 990 35 053
Keywords:	Hunting, mobile, reports, map, offline, statistics, cross-platform, API, RESTful, JSON, React Native, Java, JavaScript, SCRUM
Pages:	150
Attachments:	8
Availability:	Open

Abstract:

The goal for this assignment was to simplify hunters' everyday lives by providing a mobile application that can be used as a common place for reporting small game. In addition, the region managers have been given an intuitive interface for statistics. Challenging aspects of the assignment have mainly been to manage plotting of small game on a map in addition to offline functionality. On the server side, the biggest challenge was to create an API that meets the desired standard. To solve the task, the group has used a number of interesting technologies like React Native, Java, JavaScript and RESTful API.

Forord

Denne bacheloroppgaven ble skrevet av Jaran Godejord, Lars Johan Nybø og Linn-Hege Kristensen ved NTNU i Gjøvik.

Vi ønsker å takke Escio, spesielt deres representant Håvard Narvesen, for et svært godt samarbeid og god tilgjengelighet under hele prosjektet. Vi ønsker også å takke Tom Røise for regelmessig og verdifull veiledning under prosjektperioden. Videre takker vi alle ansatte, medstudenter og kjente & kjære som stilte opp og hjalp oss med varierte problemstillinger. I tillegg vil vi takke Simon McCallum som stilte med LaTeX-malen vi benyttet til å skrive denne rapporten.

Innhold

Figurer

Tabeller

Listings

Forkortelser

API Application Programming Interface. 4, 5, 13, 19, 20, 23–25, 29, 30, 39, 40, 43, 47, 48, 50, 53, 55–58, 60–63, 65–67, 69, 71, 72, 75, 76, 80, 81, 87–89, 91, 92, 94

APK Android application package. 27

CI Continous Integration. 19

CORS Cross-Origin Resource Sharing. 75

CSS Cascading Style Sheets. 4, 28

EER Enhanced entity–relationship. 69

GUI Graphical User Interface. 13

HATEOAS Hypermedia As The Engine Of Application State. 62

HTML Hypertext Markup Language. 4

HTTP HyperText Transfer Protocol. 25, 62, 76

HTTPS HyperText Transfer Protocol Secure. 25, 62, 75, 76

IDE Integrated development environment. 22, 24, 26, 27

JPA Java Persistence API. 65

JSON JavaScript Object Notation. 4, 23–25, 29, 54, 55, 62–68, 75, 87, 92

JWT JSON Web Tokens. 75, 76, 81

LSD LEAN Software Development. 14, 15

ORM Object Relation Mapping. 24, 63–65, 69, 91

OS Operativsystem. 23, 91, 93

PHP PHP Hypertext Preprocessor. 4, 23, 24

REST REpresentational State Transfer. xiv

RSA Rivest–Shamir–Adleman. 75

RUP Rational Unified Process. 14

SSB Statistisk sentralbyrå. 1–3, 92, 94

SSH Secure SHell. 75

UI User interface. 32

URL Uniform Resource Locator. 67

VPN Virtual Private Network. 75

XP eXtreme Programming. 14, 15, 77

Ordliste

Ajax En funksjon som utfører asynkrone HTTP-forespørsler.. 25, 71, 75

AsyncStorage Et klartekst, asynkront, nøkkelverdi-lagringssystem.. 43, 47, 54, 56, 58, 75

backlog En liste over oppgaver som skal utføres i et utviklingsprosjekt.. 11, 14, 16

Boundary En kommunikasjonskanal mellom to prosesser eller entiteter i et dataflyt-diagram som er utsatt for angrep. 75

Burndown Chart Et diagram som viser en oversikt over gjenstående, estimert arbeidsmengde i et utviklingsprosjekt.. 17, 94

code review En metode for å sikre god kodekvalitet. En annen utvikler leser gjennom koden som er skrevet for å forsikre seg om at koden opprettholder avtalt standard og at logikken stemmer.. 15, 18, 19, 77–79, 81–83

endepunkt En link som peker til en eller flere ressurser i ett . 63, 81, 87

eXtreme Programming En smidig utviklingsmodell som fokuserer på programmering og hurtige leveranser.. xii, 14

Filter-Chain En rekke filtre som utføres sekvensielt på en ressurs. 64

heat map En fremstilling av data som en serie med fargetoner som illustrerer forholdene mellom en gitt mengde over en fysisk avstand.. 9, 20, 40, 41, 72

JSON Schema Et skjema som beskriver alle lovlige verdier og strukturer for JSON-data i ulike sammenhenger.. 87

KanBan En smidig utviklingsmetode som fokuserer på å begrense antall operasjoner som utføres samtidig, for å sikre at en operasjon er ferdig før en annen starter.. 14, 15

LEAN Software Development Et utviklingsrammeverk som fokuserer på å redusere mengden unødvendige prosesser. Rammeverket skal øke effektiviteten til utviklingsteamet.. xi, 14

Linter Et verktøy som går gjennom skrevet kode og leter etter feil.. 27, 77

Misuse Case En invers versjon av Use Case. Dette er en beskrivelse av en uønsket handling eller funksjon.. 7, 10, 61, 75

mock Å lage en kopi eller etterlikning av noe.. 30, 61

OAuth 2.0 En industri-standard for å autorisere eksterne brukeres tilgang til ressurser. 76, 81

overhead Overflødig bruk av tilgjengelige ressurser.. 71

pair programming En smidig metode der to utviklere jobber med samme kode. Én utvikler skriver kode og den andre observerer og vurderer koden som skrives.. 77

pull request En forespørsel om å slå sammen to grener i et versjonskontroll-system. 18, 19, 77–79

Rapid Prototyping En rask, iterativ prosess for å designe grafiske brukergrensesnitt.. 15

Rational Unified Process En omfattende utviklisngsmodell som definerer en stor mengde utviklingsprosesser.. xi, 14

RESTful Et system som følger REpresentational State Transfer (REST)-standarden; klienten er ansvarlig for å opprettholde sin egen tilstand. . 4, 5, 13, 23, 24, 29, 43, 62, 75, 87, 92, 94

scrum En smidig utviklingsmetode for programvareutvikling basert på inkrementell leveranse.. 14–16

single page Et konsept der en webløsning består av én hovedside og innholdet på denne endres dynamisk.. 25, 71

stack En abstrakt datatype som inneholder et sett med elementer. Stack-en har to hovedfunksjoner; push legger et element på toppen av stack-en og pop fjerner det øverste elementer fra stack-en.. 45

Use Case En beskrivelse av en funksjonalitet som en applikasjon skal kunne utføre.. xiii, 7, 8

1 Introduksjon

1.1 Bakgrunn

Oppdragsgiveren bak denne oppgaven er Escio AS. De er et teknologiselskap som er en del av en større bedriftsklynge: NT6. Escio jobber med å utvikle skreddersydde løsninger for interesserte kunder, gi digital rådgivning og leie ut prosjektledere. Videre er Escio lokalisert på Hamar og Gjøvik. Nå har Escio et ønske om å utvikle en mobilapplikasjon som digitaliserer og støtter jegerens obligatoriske innrapporterings-oppgave. De ønsker også en web-løsning som viser statistikk knyttet til data fra jegernes rapporter.

I året 2017 ble det registrert totalt 142 100 aktive jegere. Av disse var 84 000 av dem jegere av småvilt i Norge [1]. For å få tillatelse til å jakte i Norge må jegeren ha betalt jegeravgiften og motta et jegeravgiftskort. Jegerne jakter i bestemte sesonger, og sesongene er spesielt bestemt for hver art det jaktes på. Det er Miljødirektoratet som definerer disse jakttidene. Videre må de forholde seg til et stort sett med regler og aktører [2].

Brønnøysundregistrene krever at hver jeger sender inn en rapport etter endt jaktsesong dersom jegeren har betalt jegeravgiften. Videre benyttes deres sider til å la en bruker se informasjon om han/hun som jeger i tillegg til å betale jegeravgiften og bestille jegeravgiftskort [3]. I dag foregår innrapporteringen elektronisk via Statistisk sentralbyrå (SSB) sine sider [4]. Videre må jegeren kjøpe et jaktkort for å få tillatelse til å jakte innenfor et bestemt område. Disse jaktområdene er enten administrert av en offentlig forvalter (f.eks. Fefo og Statskog) eller en privat grunneier (kan være grunneierlag). De fleste forvaltere eller grunneiere krever også at jegeren rapporterer hva som har blitt felt og eventuelt observert innen deres område. Både kjøp av jaktkort og innsending av rapport for et bestemt jaktområde foregår i dag via tjenesten inatur.no. I følge Escio får ikke alle forvaltere og grunneiere alt av ønsket informasjon fra denne tjenesten. Derfor er det noen som krever ytterligere informasjon (typisk via brev) i tillegg til rapporten som leveres på inatur.no.

Grunneier og forvalter bestemmer hvilke arter det skal rapporters på innen eget område. Basert på analyse av tidligere rapporter definerer de hvor mange jaktkort som skal selges i kommende sesong. En rapport består av en dato, et jaktområde, en eller flere arter og antall felt og observert av hver art.

1.2 Avgrensning

Oppgaven er avgrenset til utvikling av en mobilapplikasjon for jegerne og et web-grensesnitt for forvalterne. Videre er også følgende avgrensninger definert:

- Mobilapplikasjonen skal ikke omhandle de sosiale aspektene ved jakt, da det finnes andre applikasjoner som fokuserer på dette.
- Produktet skal ikke kobles direkte opp mot offentlige tjenester i løpet av utviklingsprosessen ettersom behandlinsperioden for dette er for lang.

- Videre er oppgaven avgrenset til den gruppen av jegere som jakter småvilt (ca. 60 % av alle jegere). Derfor vil mulige arter i applikasjonen kun bestå av småvilt. Eventuell statistikk som genereres av mobilapplikasjonen vil også kun illustrere data fra småviltjakt.
- Produktet skal kun støtte de offentlige forvalterne og ikke private aktører (grunneiere og grunneierlag).

1.3 Oppgavebeskrivelse

Vår oppgave var å forenkle jegerens hverdag i felt. Hovedsakelig innebar dette digitalisering av småviltrapportering. Dersom rapporteringen gjøres enklere for jegeren vil dataene som sendes inn ha økt presisjon. Dette vil gjøre statistikken over felte og observerte dyr mer korrekt. Videre sparer digitaliseringen tid for både jeger og forvalter. Som en bonus vil applikasjonen også spare miljøet ved å unngå papirbruk.

Mer konkret skulle vi utvikle en applikasjon til mobiltelefon som jegeren kan bruke til å rapportere dagens (eller tidligere dagers) fangst. Mobilapplikasjonen skulle være så enkel og intutitv som mulig, og derfor hadde vi mye fokus på brukertesting og brukervennlighet. Videre skulle applikasjonen kunne knytte sammen minst to av aktørene som tidligere var isolert for jegeren; SSB og rapportering til forvaltere. Jegere skulle kunne benytte applikasjonen for å samle rapport-dataene på et felles sted.

Utenom den velkjente og obligatoriske rapporteringen skulle jegeren få mulighet til å presisere på et kart hvor han/hun har gjort sine fellinger. Dette vil være svært verdifull informasjon for forvalterne ettersom det vil gjøre deres arbeid og anaylser mye enklere. Videre skulle gruppen også utvikle en web-løsning som illustrerer statistikk som genereres basert på jegernes dataer i systemet. Se mer om plan for prosjektet i prosjektplanen i vedlegg E.

1.4 Brukergrupper

I vår oppgave finnes det flere ulike brukergrupper som gruppen måtte forholde seg til. Dette er jegere, forvaltere og lesere av denne rapporten.

Jegere

Jegere er vår primære brukergruppe. Det er disse brukerne vi prioriterer og utvikler for i denne oppgaven. Jegeren vil først og fremst benytte applikasjonen innenfor jaktsesongene. Det siktes inn på at rapportering skal være så enkelt at jegeren på samme dag oppretter en rapport av dagens fangst. Videre er de fleste jegere norsktalende og godt voksne mennesker. Dette vil gruppen ta i betraktning ved å designe med hensyn til universell utforming. Les mer om hvordan universell utforming ble håndtert i seksjon 5.3.

Forvaltere

Den andre brukergruppen gruppen forholder seg til er forvaltere. Forvalterne har ansvar for bestemte jaktområder og definerer hva jegerene skal rapportere til dem. De har også ansvar for å selge en bestemt mengde med jaktkort per sesong basert på analyse av tidligere jaktrapporter. Dette foregår via inatur.no og brev. For dem vil det være essensielt å motta rapporter på en god og oversiktlig måte. Det vil også være viktig å presentere statistikk inutitivt. Dersom gruppens produkt blir tatt i bruk kan rapporterings-portalen

for småvilt på inatur.no elimineres.

Lesere av rapporten

En siste brukergruppe er lesere av denne rapporten. Dette er spesielt sensor og veileder, men det kan også være produkteier. I fremtiden vil også denne rapporten kunne leses av andre utviklere som skal jobbe videre med gruppens produkt. I tillegg kan også rapporten leses av andre studenter under deres egne bachelor-prosjekter.

1.5 Eksterne systemer

Videre finnes det noen eksterne systemer som det er ønskelig at integreres med produktet gruppen utviklet. Disse er SSB og Brønnøysundregistrene.

Statistisk sentralbyrå

I dag benyttes som nevnt SSB sider for å la en jeger sende inn hans/hennes årlige rapport. Dersom gruppens produkt blir tatt i bruk kan rapporterings-portalen deres også elimineres, og de kan benytte data fra gruppens database til å illustrere statistikk o.l. Videre har gruppen sendt en e-post til SSB angående tilgang til deres data. Dataene de har om hver enkelt jeger er naturligvis konfidensielle. Les hele svaret som et av gruppemedlemmene mottok fra SSB i vedlegg G. Integrasjon diskuteres i mer detalj i seksjon 10.3.

Brønnøysundregistrene

Som nevnt benytter en jeger Brønnøysundregistrene hovedsaklig for å administrere jegeravgiften. I tillegg viser også tjenesten brukeren informasjon om han/hun som jeger. Derfor er det relevant å integrere gruppens produkt med Brønnøysundregistrene sine systemer. Les mer om dette i seksjon 10.3.

1.6 Hvorfor vi valgte oppgaven

Gruppen valgte denne oppgaven av flere grunner. For noen av gruppemedlemmene er jakt et svært interessant tema. Flere av gruppemedlemmene har også kjennskap til jakt gjennom andre i vennekrets og familie som jakter. Dette gjør denne oppgaven mer spennende, ettersom gruppen opplever at vi kan utvikle noe som kan være til hjelp for både kjente og kjære. I tillegg er jakt et kjent tema, og det føles fascinerende å være med på å utvikle et verktøy som kan benyttes av svært mange mennesker i Norge. Oppdragsgivers motiv var å forenkle og digitalisere en jegers gjøremål, og dette var noe gruppen ønsket å være en del av.

Videre ble gruppen enda mer sikker på oppgaven etter en prat med oppdragsgiver på første møte. Det ble arrangert et foredrag av de ulike oppgavene i november 2017, og det var på dette tidspunktet at gruppen fikk muligheten til å diskutere oppgaven i mer detalj. Gruppen fikk et svært godt inntrykk av oppdragsgiver og ble tryggere på at dette var en oppgave for oss.

1.7 Kompetanse

Tidligere emner

Gjennom denne utdanningen har gruppen opparbeidet seg et svært godt grunnlag for å løse denne oppgaven. Gruppen har hatt matte for informatikk og lært mye grunnleggende om informasjonssikkerhet. Gruppen har også lært om sikkerhet gjennom nettverksemner. Videre gir det oss en stor fordel i denne oppgaven at gruppen har hatt emner som systemutvikling og objektorientert systemutvikling. Gjennom studiet har gruppen også hatt et dedikert database-fag, men databaser og databasemodellering ble også benyttet i andre emner.

Utenom dette har gruppen hatt spesielt stor nytte av de ulike programmerings-emnene som har vært obligatoriske i studiet. Gruppen har lært spesielt mye om objektorientert programmering og systemutvikling. Innenfor dette området har gruppen lært seg språk og teknologier som C++, Hypertext Markup Language (HTML), Cascading Style Sheets (CSS), PHP Hypertext Preprocessor (PHP), JavaScript (jQuery), Java og Java for Android.

Annen erfaring

Linn-Hege jobbet for IT-tjenesten som utvikler sommeren 2017. I denne perioden utviklet hun to web-løsninger skrevet i HTML, PHP og JavaScript (jQuery). Hun modellerte også en database for den ene web-løsningen med MySQL. Den største tjenesten (Orego) er i drift i dag.

Lars Johan har på eget initiativ utviklet egne hobby-prosjekter ved siden av studiene. Prosjektene inkluderer flere websider basert på ulike teknologier, spillutvidelser og selvstendige applikasjoner. Han har også engasjert seg som fagassistent i både *Grunnleggende Programmering* og *Objektorientert Programmering*.

Jaran har utviklet diverse nettsider og utvidelser til dataspill han har vært interessert i på fritiden. Han liker å se etter sårbarheter i programvarer og å finne ut hvordan han kan få programmer til å oppføre seg unormalt.

Nye teknologier

I dette prosjektet var det flere nye teknologier vi ikke hadde benyttet tidligere som vi måtte forholde oss til for å løse oppgaven på en god måte:

- **React Native** - React Native skulle benyttes for å utvikle en mobilapplikasjon. Dette er et rammeverk som benytter React og JavaScript for å bygge en native app. Rammeverket er spesielt preget av React som er et helt ukjent bibliotek for gruppen. Les mer om hvorfor React Native ble valgt i seksjon 4.1.1 og 10.1.1.
- **Node.js** - For å kjøre koden skrevet i React Native måtte Node.js benyttes. Les mer om Node.js i seksjon 4.2.1.
- **Xcode** - Xcode ble benyttet for å kompilere og teste mobilapplikasjonen i et iOS-miljø. Gruppen måtte forsikre seg om at applikasjonen også kunne kjøres på en iOS-enhet ved å benytte en iOS-emulator. Les mer om Xcode i seksjon 4.2.1
- **RESTful API** - Det var også et krav fra oppdragsgiver at gruppen utviklet et RESTful Application Programming Interface (API) for kommunikasjon mellom backend og frontend-tjenestene (mobilapplikasjon og web-løsning). Tjenesten skulle også benyttet JavaScript Object Notation (JSON) som standard. Les mer om hvordan

API-et ble implementert i seksjon 6.2.

1.8 Rammer

- Mobilapplikasjonen skal utvikles med React Native etter sterkt ønske fra produkteier.
- Produkteier ønsker at applikasjonen støtter de mest utbredte versjonene i løpet av de to siste årene og generelt nyere mobiler (Samsung Galaxy S7 og nyere). React Native støtter Android versjon 4.1 (API 16) og oppover. I tillegg støttes også iOS versjon 8.0 og oppover [5].
- Det skal benyttes et RESTful API for kommunikasjon mellom klientene (mobilapplikasjonen og web-løsningen) og backend. Det kreves at standaren beskrevet på jsonapi.org [6] benyttes under utforming av API-et.
- Applikasjonen skal brukertestes (helst i flere iterasjoner) i løpet av utviklingsperioden.
- Applikasjonen skal utvikles ved hjelp av smidige arbeidsmetoder.

1.9 Roller

I prosjektet ble det naturlig å dele inn produktet i ulike deler og ansvarsområder. Dermed fikk hvert medlem i utviklingsgruppen hvert sitt ansvarsområde. Til tross for dette var det ingen tvang i forhold til arbeidsoppgaver. Noen gruppemedlemmer jobbet gjerne på flere deler av produktet og gruppen hadde mye fokus på å oppdatere hverandre på hver sin kant. Utenom dette forholdt gruppen seg til produkteier og veileder. Slik var ansvaret fordelt:

- **Håvard Narvesen** var gruppens kunde og oppdragsgiver. Han fungerte som produkteier og hadde regelmessig møter med gruppen for å definere utviklingens retning, ny funksjonalitet og fokusområder. Møtene ble holdt i slutten av hver sprint (annen hver uke). Det er også verdt å nevne at han personlig har erfaring fra jakt og delte denne kompetansen med gruppen underveis i utviklingsprosessen.
- **Jaran Godejord** var hovedsakelig utvikler. I de tilfelle Lars ikke hadde mulighet fikk han ansvar som referent. Ellers hadde han delt ansvar for mobilapplikasjonen.
- **Lars Johan Nybø** var utvikler og referent. Som refererent var det hans oppgave å føre notater underveis i alle møter. Som utvikler hadde han hovedansvar for backend og web-løsningen.
- **Linn-Hege Kristensen** var prosjektleder for oppgaven og fungerte som gruppens Scrum Master. Innen utvikling hadde hun delt ansvar for mobilapplikasjonen. Som prosjektleder og Scrum Master hadde hun ansvar for at utvikling og oppgaver ble utført som planlagt.
- **Tom Røise** var gruppens veileder gjennom hele utviklingsprosessen. Han deltok i regelmessige møter med gruppen angående gruppens fremgang og prosess. Møtene ble som regel holdt annen hver uke, tirsdag 14.30. Tom veiledet gruppen ved å svare på faglige spørsmål, lese gjennom rapport, stille krav til dokumentasjon o.l.

1.10 Om rapporten

Kildekoden som det refereres til er lagt ved denne rapporten som en egen ZIP-fil lastet opp i Blackboard. I rapporten benyttes det ulike engelske begreper, fremmedord og forkortelser. Disse er klikkbare og er registrert under egen ordliste. Innholdsfortegnelsen er klikkbar og leder deg til kapittelet du velger. Referanser til kapitler, seksjoner, figurer, tabeller og vedlegg i teksen er også klikkbare. I de tilfellene der figurer eller tabeller virker uleselige er det mulig å se dem bedre ved å øke størrelsen på PDF-en. Videre er det verdt å nevne at ved referanse til «mobilapplikasjonen» menes både iOS- óg Android-implementasjoner ettersom applikasjonen er utviklet med en kryssplattform-teknologi. Rapportens strukturt er beskrevet i tabell 1.

Kapittel	**Beskrivelse**
1 Introduksjon	Beskriver oppgaven og gruppens utgangspunkt.
2 Kravspesifikasjon	Beskriver kravene som ble stilt til produktet.
3 Utviklingsprosessen	Beskriver hvordan gruppen jobbet sammen gjennom prosjektet og hvorfor.
4 Teknologier	Kartlegger de teknologiene gruppen benyttet under utviklingsprosessen.
5 Programvaredesign	Beskriver hvordan gruppen utarbeidet produktets design. Arkitektur og brukergrensesnitt diskuteres.
6 Implementering	Beskriver hvordan de ulike delene av produktet ble implementert og hvorfor.
7 Kodekvalitet	Beskriver hvordan gruppen forsikret seg at koden hadde god kvalitet.
8 Testing	Beskriver de verktøyene og metodene gruppen tok i bruk for å teste produktet og hvorfor.
9 Veien til produksjon	Beskriver det som gjenstår før produktet kan produksjonssettes.
10 Utfall	Beskriver resultatene av arbeidet. Gruppen reflekterer over disse og andre alternativer.

Tabell 1: Rapportens struktur og beskrivelse av hvert kapittel

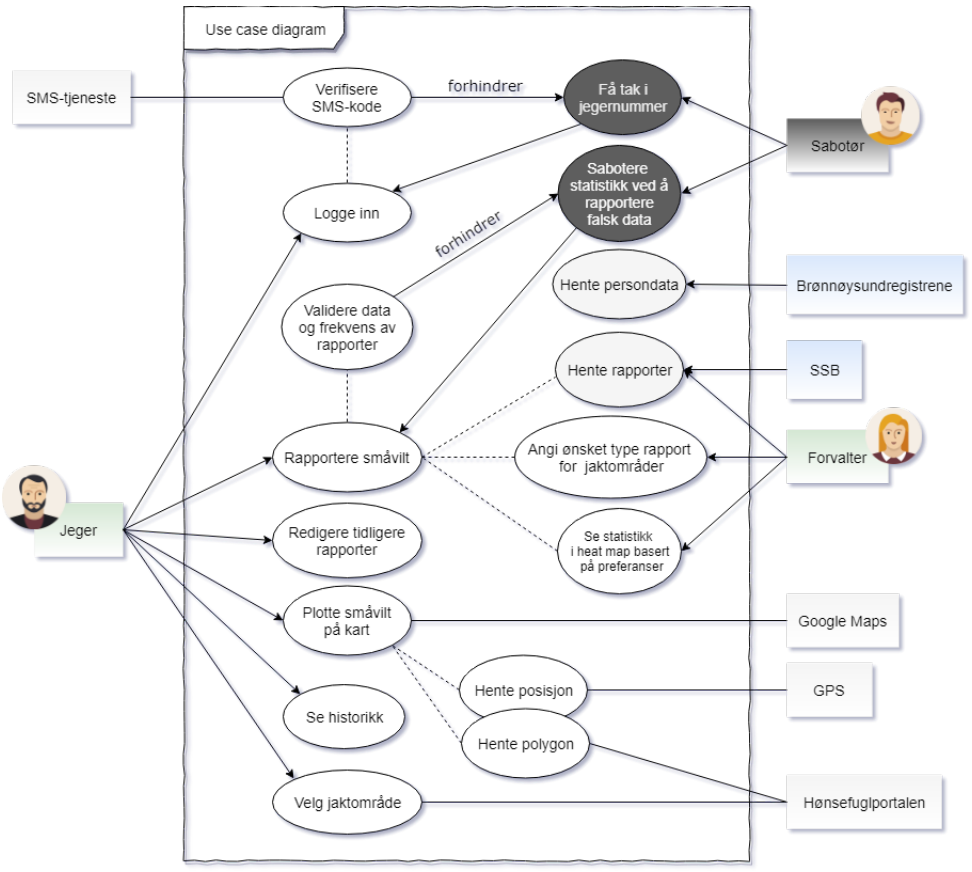
2 Kravspesifikasjon

Dette kapittelet beskrives de ulike kravene som gruppen har definert for prosjektet. Kravene er spesifisert i samarbeid med produkteier.

2.1 Use Cases og Misuse Cases

I prosjektet benyttet gruppen Use Cases ettersom det er svært relevant for arbeidsmetodikken som ble valgt ut. I tillegg definerte gruppen noen Misuse Cases for å illustrere et par sikkerhetsaspekter som gruppen tok hensyn til. Videre ble det utformet et Use Case- og Misuse Case-diagram for å se helheten i den planlagte prototypen.

De Use Case-ene som ikke ble implementert i denne oppgaven har grå bakgrunnsfarge i Use Case-diagrammet vist i figur 1. Disse er likevel illustrert ettersom funksjonaliteten er ønsket i fremtiden. Videre er sluttbrukere markert med et ikon av en person. Offentlige systemer er aktørene farget i blått og sekundære aktører som støttet opp under utviklingen er farget med grått. Utenom dette er Misuse Case farget i svart.



Figur 1: Use Case- og Misuse Case-diagram for produktet

High-level Use Cases

Under finnes en liste over de Use Case-ene som ble implementert i prosjektet. Use Case-ene gruppen jobbet med er beskrevet high-level.

Use Case: Logge inn.
Aktører: Jeger.
Mål: Å identifisere jegeren med hjelp av jegernr. og telefon nr.
Betingelser: Brukeren er registrert med jegernummer i applikasjonen.
Beskrivelse: Jegeren fyller inn jegernr. og trykker på logg inn-knappen. Deretter sendes en sms-kode til brukerens mobiltelefon. Denne koden må brukeren oppgi for å identifisere seg selv.
Variasjoner: Dersom jegeren allerede har logget inn en gang tidligere vil han/hun bli logget inn automatisk.

Use Case: Velge jaktområde.
Aktører: Jeger.
Mål: Definere et aktivt jaktområde for rapportering.
Betingelser: Brukeren har logget inn.
Beskrivelse: Ved førstegangs-innlogging blir bruker bedt om å velge et jaktområde for innrapportering. Dette blir husket til neste gang jeger logger seg inn. Jeger kan også bytte jaktområde.
Variasjoner: Ved førstegangs-innlogging vil brukeren motta en velkomstbeskjed i tillegg til å måtte velge et aktivt jaktområde.

Use Case: Rapportere småvilt.
Aktører: Jeger.
Mål: Rapportere inn antall småvilt (og art) som ble observert og/eller felt.
Betingelser: Brukeren må være identifisert i applikasjonen og må ha satt et aktivt jaktområde.
Beskrivelse: Når jeger er i terrenget (og etter) skal vedkommende ha mulighet til å registrere antall småvilt (og art) som blir/ble observert og/eller felt.
Variasjoner: Dersom jegeren ikke har internettforbindelse skal det fremdeles være mulig å lagre nye rapporter. Disse blir dermed lagret lokalt frem til jegeren får internettforbindelse igjen.

Use Case: Se historikk.
Aktører: Jeger.
Mål: Få oversikt over alle rapporter som er lagret for valgt område.
Betingelser: Brukeren må være identifisert i applikasjonen og må ha satt et aktivt jaktområde.
Beskrivelse: Jeger har mulighet til å få en oversikt over sine tidligere rapporter. Her kan jegeren bekrefte de opplysningene som finnes.
Variasjoner: Dersom jegeren ikke har internettforbindelse skal det fremdeles være mulig å se en oversikt over tidligere rapporter.

Use Case: Redigere tidligere rapporter.
Aktører: Jeger.
Mål: Mulighet til å endre en lagret rapport for feilrettinger og liknende.
Betingelser: Brukeren må være identifisert i applikasjonen, ha satt et aktivt jaktområde og har registrert minst én rapport.
Beskrivelse: Dersom vedkommende oppdager en skrivefeil eller ønsker å oppdatere en rapport utført tidligere er det mulig ved å åpne rapporten i historikk. Deretter kan innholdet eventuelt endres. Rapporter kan også slettes.
Variasjoner: Dersom jegeren ikke har internettforbindelse skal det fremdeles være mulig å oppdatere eller slette en rapport. Rapporten blir så lagret lokalt frem til jegeren får internettforbindelse igjen.

Use Case: Plotte småvilt på kart.
Aktører: «Rapportere småvilt», «Redigere tidligere rapporter» og jeger.
Mål: Mulighet til å definere på et kart hvor fellinger ble utført.
Betingelser: Brukeren må være identifisert i applikasjonen og må ha satt et aktivt jaktområde. I tillegg må brukeren være i prosessen ved å opprette en ny rapport.
Beskrivelse: Både ved rapportering og ved redigering av en rapport er det mulig for jegeren å åpne et kart for å plassere eventuelle fellinger med hjelp av «pins». Kartet vil vise et farget polygon som illustrerer jaktområdet. Dersom brukeren tillater det vil brukerens plassering hentes. Dersom plasseringen er på innsiden av polygonet vil kartet benytte brukerens lokasjon som kartets senter.
Variasjoner: Denne funksjonen er ikke tilgjengelig dersom jegeren ikke har internettforbindelse.

Use Case: Se statistikk i heat map basert på preferanser.
Aktører: Forvalter og jeger.
Mål: Gi en oversikt over alle innsamlede rapportdata.
Betingelser: Minst én bruker må plottet småvilt på kartet for at noe statistikk skal vises.
Beskrivelse: Web-løsningen skal tilby alle interesserte brukere å se en oversikt over all rapportdata som ligger i databasen. Disse dataene skal blant annet representeres med et heat map og polygoner. Dette kartet skal vise informasjon basert på preferanser som brukeren har satt i et filter.
Variasjoner: Heat map og polygoner oppdateres ved bruk av filter.

Use Case: Angi ønsket type rapport for jaktområder.
Aktører: Forvalter.
Mål: Definere arter som det skal rapporteres på for hvert jaktområde.
Betingelser: Forvalteren har logget seg inn i web-løsningen.
Beskrivelse: Web-løsningen skal la en forvalter definere ønsket rapport for hvert jaktområde som er en del av forvalterens region. Forvalteren skal med andre ord få muligheten til å ramse opp hvilke arter som befinner seg i de ulike jaktområdene.
Variasjoner: Ingen.

High-level Misuse Cases

Videre har gruppen definert et par Misuse Cases som kan forekomme. Dette er uønsket adferd og funksjonalitet som må håndteres. Disse er markert med svart i digrammet i figur 1. I tillegg er de beskrevet high-level under:

Misuse Case: Få tak i jegernummer.
Aktører: Sabotør.
Mål: Å logge inn i applikasjonen selv om vedkommende ikke har et registrert jegernummer selv.
Betingelser: Jegeren har skaffet et jegernummer.
Beskrivelse: Sabotøren prøver å logge seg inn i applikasjonen ved å benytte en annen jegers jegernummer. Sabotøren blir forhindret fra å gjøre dette ettersom innlogging krever identifisering via telefon og SMS-kode.
Variasjoner: Dersom sabotøren også har tilgang til telefonen som tilhører utvalgt jegernummer vil sabotøren klare å logge seg inn.

Misuse Case: Sabotere statistikk ved å rapportere falsk data.
Aktører: Sabotør.
Mål: Å sabotere den nasjonale statistikken over felt og observert småvilt ved å selv rapportere falsk data.
Betingelser: Sabotøren har klart å logge inn.
Beskrivelse: Sabotøren benytter den vanlige rapporteringsmetoden for å rapportere feilaktig eller falsk og uekte data. Noe som forhindrer dette til en viss grad er at applikasjonen validerer data opp mot logiske verdier. I tillegg valideres også frekvensen av nye rapporter for å forsikre at dataene er korrekte.
Variasjoner: Dersom sabotøren rapporterer logiske data og like ofte som en vanlig jeger er det sannsynlig at disse dataene ikke blir registrert som korrupte.

Expanded Use Case

Hovedoppgaven til mobilapplikasjonen er å la jegeren rapportere inn antall felte og observerte dyr i tillegg til typen dyr. Derfor har gruppen valgt å legge ved en utvidet beskrivelse av denne funksjonaliteten.

Use Case: Rapportere småvilt.
Aktører: Jeger.
Mål: Rapportere inn antall småvilt (og art) som ble observert og/eller felt.
Beskrivelse: Når jeger er i terrenget (eller etter) skal vedkommende ha mulighet til å registrere antall småvilt (og type art) som ble observert og/eller felt på valgt dag. Dette skal også være mulig å gjøre uten internettforbindelse.
Type: Essensiell.
Betingelser: Brukeren må være identifisert i applikasjonen og må ha satt et aktivt jaktområde.

Detaljert hendelsesforløp:

1. Brukeren logger seg inn i applikasjonen.
2. Brukeren kan endre jaktområde dersom ønskelig, ellers benyttes tidligere valg.
3. Brukeren bekrefter deretter at det gjelder dagens dato eller endrer for å gå videre.
4. Brukeren oppgir hva som har blitt felt/observert i løpet av valgt dag og antallet.
5. Dersom brukeren har én eller flere felte dyr kan han/hun plassere disse som markeringer på et kart.
6. Brukeren lagrer rapporten og får se en bekreftende oppsummering av den nye rapporten.

Alternative scenarier:

Jegeren har ikke internettforbindelse
Alternativt er jegeren offline i terrenget ved innrapportering. I dette tilfellet er hendelsesforløpet nokså identisk. Et unntak er at informasjonen lagres lokalt og ikke sendes før brukeren blir online igjen. Det andre unntaket er at brukeren ikke kan plotte fangsten på kartet.

Jegeren tvangslukker appen
Dersom jegeren tvinger avslutning av applikasjonen under rapporteringen vil en ny rapport fremdeles være lagret så lenge brukeren trykket på «Lagre jaktdag» før krasjet oppstod. Dette gjelder uansett om brukeren har internett eller ikke. Ellers må brukeren gjenta prosessen.

2.2 Product backlog

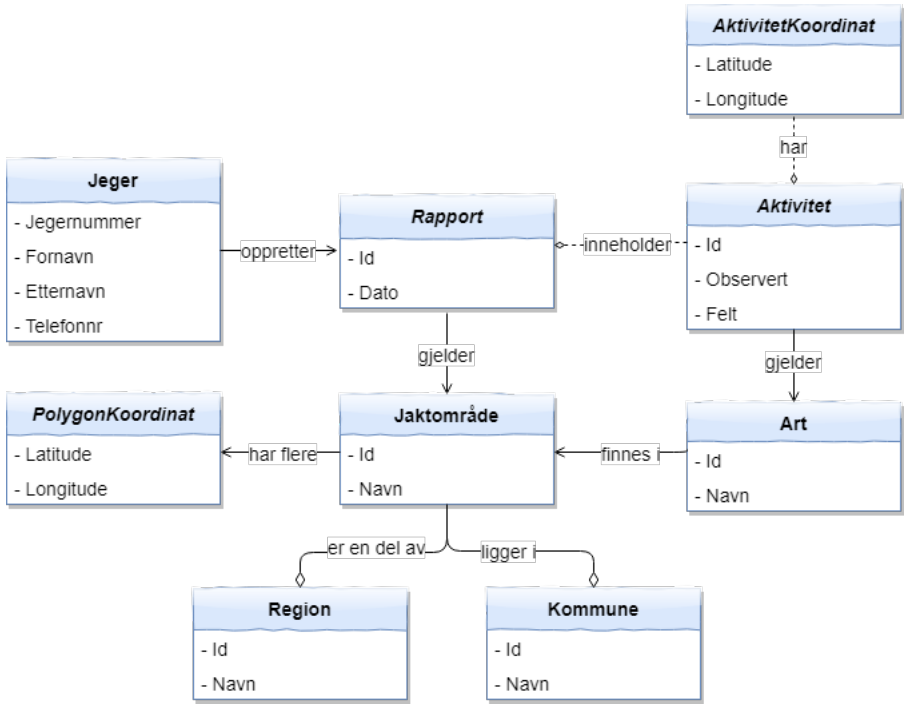
Gruppens product backlog ble definert i Google Spreadsheet og deretter lagt inn i Jira. Gruppen valgte å dele inn større oppgaver, også kalt *issues*, i subtasks for å gi en god oversikt over hva som skulle gjøres ved hver enkelt oppgave. Tabell 2 viser en oversikt over de ulike typene oppgaver som lå i backlog-en. Totalt hadde gruppen opprettet 111 *issues* i Jira ved prosjektavslutning.

Issue	Beskrivelse
User Story	Oppgaver definert av produkteier. Disse var gjerne litt større oppgaver som ble delt inn i *sub tasks*.
Sub task	En oppgave som er en del av en større og mer kompleks oppgave.
Task	En generell oppgave. Denne betegnelsen ble som regel benyttet da utviklerne i teamet selv opprettet oppgaver som måtte utføres i løpet av en sprint. Dette var gjerne oppgaver som ikke var direkte relatert til en *user story*.
Bug	Et kjent problem som må fikses.

Tabell 2: Beskrivelser av ulike typer issues som ble benyttet i Jira

2.3 Domenemodell

Figur 2 viser en oversikt over jegerens virkelighet og de ulike delene ved en rapport. Den beskriver jegerens attributter og innholdet i en rapport. Videre vises det til at et jaktområde har en rekke koordinater som utgjør et polygon. I tillegg kan en aktivitet også knyttes til koordinater.



Figur 2: Domenemodell for produktet

2.4 Operasjonelle krav

Gruppen utviklet en mobilapplikasjon med formål om å gjøre rapportering av småviltjakt en enklere oppgave for jegere. Det innebar at applikasjonen måtte ha stort fokus på brukervennlighet. Produkteier ønsket at appen skulle føles enkel å bruke og det var viktig at den kunne benyttes uavhengig av internettforbindelse. Dette lå til grunnlag for noen av de operasjonelle kravene som ble definert.

Teknisk

- Rapportering må kunne utføres i «offline» tilstand uten internettilkobling. Dette er fordi jakt som regel foregår ute i terrenget med begrenset tilgang til internett. Bruk av kart er et unntak og trenger ikke være tilgjengelig offline.
- Av Android-versjoner er det bestemt at gruppen skal forholde seg til dagens telefoner og ikke tidligere enn versjon 4.1 (API 16).
- Applikasjonen skal håndtere opptil 800 samtidige brukere. Dette mener gruppen er passende ettersom dette er 1% av antall aktive jegere i 2017 [1].
- Graphical User Interface (GUI)-responstid skal være på maksimalt 1 sekund. Responstid mellom klient og server skal være maksimalt 5 sekunder under normale omstendigheter.

Brukervennlighet

- For å sikre at registrerte data er mest mulig presise, skal det utføres sjekker som validerer om innhentet data er logisk i forhold til normale verdier.
- Brukervennligheten i applikasjonen skal være på et høyt nivå slik at veiledning og bruksanvisning ikke er nødvendig.
- Mobilapplikasjonen skal være intuitiv nok til at en jeger ikke bruker mer enn 1,5 minutt på å opprette en ny rapport. Dersom jegeren plotter fangst på kartet er det naturlig at lengre tid forekommer.
- Både mobilapplikasjonen og web-løsningen skal håndtere og validere alt av input slik at feilaktig data ikke leder til uønskede konsekvenser for brukeren.

Sikkerhet

- Det er kun brukere som har et registrert jegernummer som skal få tilgang til å logge seg inn i mobilapplikasjonen.
- Gruppen skal legge til rette for at sensitiv informasjon kan lagres og behandles på en sikker måte, selv om det benyttes «dummy-data» under utviklingsfasen.
- RESTful API-et som utvikles skal ha klare begrensninger på hvem som har tilgang til hvilken informasjon.
- Alt av bruker-generert input skal bli rensket og validert før dataene sendes via API-et og til databasen.
- Ved innlogging skal det legges til rette for å benytte en SMS-tjeneste for å identifisere en jeger på en sikker måte.
- Det skal oppdages og håndteres dersom en bruker «spammer» databasen med mobilapplikasjonen.
- Det skal legges til rette for at den nye personvernsloven (GDPR) kan støttes av produktet. GDPR skal også innføres i så høy grad det lar seg gjøre.

3 Utviklingsprosessen

Som nevnt i introduksjonen (se seksjon 1) var en av de definerte rammene til oppgaven at prosjektet skulle utvikles med smidige utviklingsmetoder (se seksjon 1.8). Dette kapittelet beskriver den smidige utviklingsmetodikken gruppen valgte, hvordan utviklingen ble gjennomført og hvorfor. I prosjektplanen (se vedlegg E) beskrives hvordan gruppen planla å gjennomføre utviklingen.

3.1 Utviklingsmetodikk

Gruppen valgte å benytte Scrum med aspekter fra KanBan som utviklingsmetodikk. Under planleggingsfasen ble elementer fra LEAN Software Development (LSD) benyttet.

Innenfor Scrum-delen av tilnærmingen valgte gruppen å følge møtene nokså strengt og slavisk. Det ble holdt *daily scrums*, *sprint planning*-møter, *sprint retrospective*-møter og *sprint review*-møter. Dette gjorde gruppen for å bevare oversikt over progresjonen i utviklingen og for å legge et godt grunnlag for kommende sprint. Videre arbeidet gruppen hovedsakelig i 2-ukers sprinter. Gruppen forholdt seg til denne lengden av flere grunner. Produkteier ytret blant annet at et ønske om å se ny funksjonalitet hyppig. Videre var det også et ønske gruppen selv hadde; det føltes mer motiverende å se ny og ferdig funksjonalitet annenhver uke. I tillegg er det også nokså standard å benytte 2-ukers sprint (dette er default i Scrum). Likevel hadde gruppen et par sprinter på én uke av forskjellige grunner. Disse er beskrevet under beskrivelse av de aktuelle sprintene i seksjon 3.2.5.

Innen KanBan valgte gruppen å begrense arbeidsmengden. Det ble fastsatt at det maksimalt skulle eksistere fire aktive oppgaver i arbeid og maksimalt tre oppgaver i *code review* (les mer om de ulike statusene i seksjon 3.2.3). Videre valgte gruppen å ikke benytte KanBan-metoden for *sprint backlog*-en. Selve backlog-en ble definert i samarbeid med produkteier og ble ført opp i Google Spreadsheets. Her ble hver *user story* definert med en spesifikk *business value* av produkteier. Deretter ble backlog-en sortert på prioritet i samarbeid med produkteier. Sorteringen ble utført ved å vurdere av hver *user story* sin *business value* og estimering av tidsbruk.

3.1.1 Argumentasjon

Ettersom det tidlig ble bestemt at utviklingsmodellen skulle være smidig ble modeller som Fossefall raskt ekskludert. I tillegg ekskluderes Rational Unified Process (RUP), da denne virker unødvendig kompleks med tanke på størrelse på utviklerteam og oppgave.

Gruppen valgte dermed å trekke frem fire utviklingsmodeller/-rammeverk som virket relevante for oppgaven: Scrum, eXtreme Programming (XP), KanBan og LSD.

Scrum

Scrum er ofte et natulig førstevalg av modell, da den er svært smidig og tilpasningsdyktig til de fleste prosjekter. Modellen er også et naturlig valg for gruppens prosjekt ettersom situasjonen passer godt med modellens bruksområde [7]:

- Modellen er smidig og har hyppige releaser (korte/egendefinerbare sprint-lengder). Dette gir rom for fortløpende endringer, noe som er relevant siden kravspesifikasjonen er ufullstendig.
- Idéell for små team.
- Roller som passer vår bemanning: produkteier, Scrum Master, Scrum Team.
- Gir utviklerteamet større frihet til å ta egne valg, som da øker effektiviteten (gunstig siden utviklingsperioden er svært begrenset).
- Arbeidsgiver får resultater raskt, og har aktiv deltakelse gjennom utviklingen.

Basert på disse betingelsene valgte gruppen Scrum som kjernemodell. Gjennom bruken av Scrum fikk gruppen mer autoritet til å ta egne valg som senere kunne avklares med produkteier. Dette fungerte godt ettersom gruppen og produkteier oppnådde en felles forståelse av applikasjonens funksjonalitet underveis i prosessen.

XP

Fra XP valgte gruppen å benytte *user stories* og *automatiserte test-rammeverk* [8]. User stories var ønskelig da produktet som ble utviklet ofte baserte seg på handlinger og oppgaver som brukeren skulle utføre. Videre ble automatiserte test-rammeverk også benyttet (spesielt på backend) da dette sikret at logikken fungerte som planlagt.

KanBan

KanBan har ett enkelt konsept: *begrense påbegynt arbeidsmengde* [9]. Ved å benytte dette konseptet måtte nye oppgaver vente til de påbegynte er ferdig. Konseptet sikret at gruppen ikke hadde mange uferdige oppgaver på én gang, og ga økt oversikt og kontroll. Det sikret også at «mindre fristende» oppgaver som dokumentasjon og code review ble gjennomført fortløpende. Basert på erfaring fra tidligere samarbeid ønsket gruppen å benytte KanBan under utviklingsprosessen, spesielt for å gjennomføre code review fortløpende.

LEAN Software Development

LSD er mer et rammeverk enn en modell, og gruppen besluttet å bruke prinsippene som retningslinjer under planleggingen og utviklingen [10]. Blant annet ble unødvendig arbeid begrenset ved å definere spesifikke arbeidstider og et estimat over tidsbruk i de forskjellige fasene. Disse metodene bidro til å redusere stress og kognitive påkjenninger [10]. Vi benyttet også Rapid Prototyping for å unngå store endringer senere i utviklingsfasen. Videre resulterte smidig utvikling i raske og fortløpende leveranser, som samsvarer med LEAN-konseptet.

Ett av kravene til produktet er at produkteier skal kunne ta over utviklingen etter prosjektperioden. Dermed ble det viktig at applikasjonen er «fleksibel, vedlikeholdbar, effektiv og responsiv» [10]. For å tilfredsstille disse kravene det innført unit-testing, jevnlig refaktorering og konsistent versjon-nummerering.

3.1.2 Estimering

Innenfor estimering ble «story points» benyttet. Gruppen valgte denne metoden ettersom det er en moderne og velkjent måte å utføre estimering på innenfor smidige prosjekter. Gruppen mente derfor at dette var en god mulighet til å lære å tenke annerledes. I tillegg hadde gruppen også erfaring med en del uenigheter rundt estimering da gruppen i andre prosjekter benyttet mer tradisjonelle metoder.

Story points er ulikt tradisjonell estimering ettersom story points ikke forholder seg til datoer eller bestemte tidsestimater (timer, dager, uker). I stedet er hvert story point relative i forhold til hverandre. En oppgave med tre story points skal være tre ganger så stor som en oppgave med ett story point. Et story point skal omfatte mengden arbeid som må utføres, kompleksitet i oppgaven og eventuelle risikoer [11]. Videre er det vanlig at story points-skalaen er basert på Fibonacci-tallene.

Under estimering benyttet gruppen skalaen 0, 0.5, 1, 2, 3, 5, 8, 13, 20, 40, 80, 100. Under prosjektet valgte gruppen å benytte den nedre delen av skalaen i stor grad. For å utføre selve estimeringen benyttet gruppen mobilapplikasjonen Scrum Time - Planning poker fra Endava [12].

3.2 Gjennomføring

I løpet av utviklingsperioden har gruppen hatt en rekke faste møter både internt og eksternt med produkteier og veileder. Sprintene har i stor grad vært to uker lange, men et par har vært på én uke. Uansett sprint har gruppen alltid startet en ny sprint med et *sprint planning*-møte og arbeidet til og med mandag to uker (eller én uke) senere.

3.2.1 Scrum-møter

Gruppen ønsket å lære så mye som mulig av å bruke Scrum som utviklingsmodell og valgte derfor å holde alle Scrum-relaterte møter nokså slavisk. Derfor ble hver sprint startet med et *sprint planning*- & *sprint review*-møte, det ble holdt daglige statusmøter (*daily scrum*) og sprinten ble avsluttet med et *sprint retrospective*-møte.

Sprint planning & sprint review

I starten av hver sprint (som regel tirsdag) ble det holdt et *sprint planning*- og *sprint review*-møte med proukteier. Gruppen valgte å slå sammen disse to møtene ettersom produkteier hadde en begrenset mengde med tilgjengelig møtetid. Derfor startet gruppen møtene med et *sprint review* der resultatene fra forrige sprint ble presentert for produkteier (gjerne med en demo). Deretter fikk gruppen tilbakemeldinger og forbederingspunkter. Etter denne delen av møtet hadde gruppen et bedre grunnlag for *sprint planning*-møtet. *Sprint planning*-møtet ble utført noe utradisjonelt fra det som er typisk innenfor Scrum. Gruppen valgte å gjøre estimeringer på forhånd av dette møtet. Deretter ble *sprint planning*-møte benyttet til å velge ut *user stories* til neste sprint i samarbeid med produkteier. Dette viste seg å være svært fordelaktig ettersom produkteier for hver sprint fikk mulighet til å omprioritere backlog-en basert på produktet som ble levert i forrige sprint.

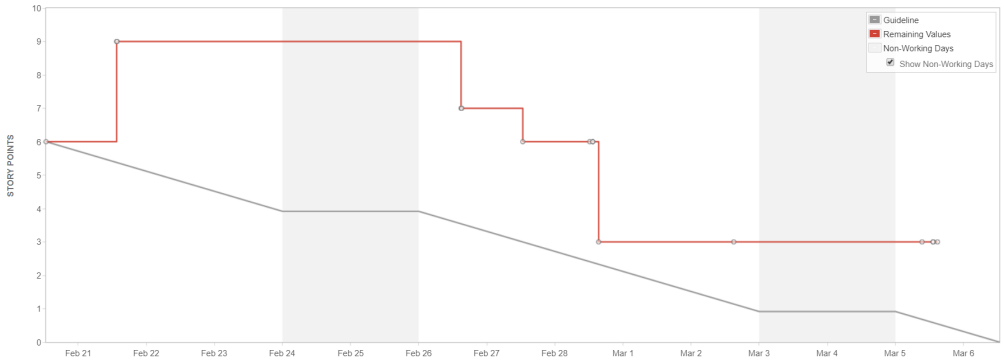
Daily Scrum

Gjennom utviklingsperioden har gruppen hatt daglige statusmøter (*daily scrum*-møter) der hvert medlem av gruppen har gjennomgått hva han/hun har jobbet med frem til

det tidspunktet. Ny funksjonalitet ble gjerne vist frem på disse møtene. Videre ble møtet benyttet til å oppdatere hverandre om hva hvert medlem hadde planer om å jobbe med resten av dagen. Dersom gruppen ikke møtte hverandre fysisk valgte gruppen likevel å oppdatere hverandre skriftlig.

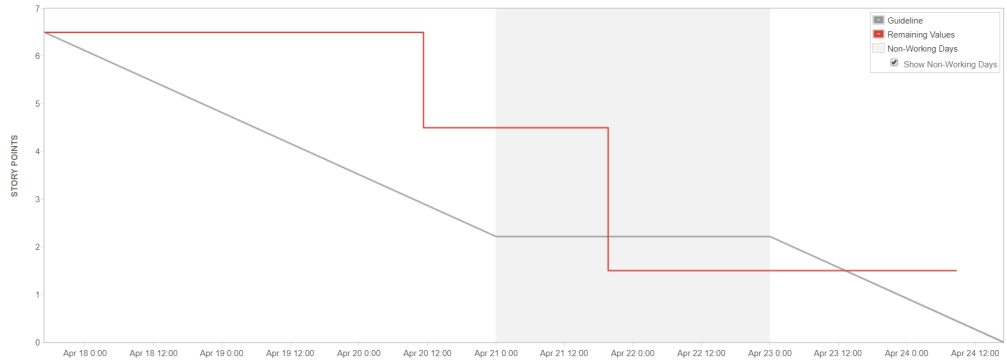
Sprint retrospective

Disse møtene ble holdt i slutten av hver sprint, som regel på en mandag på slutten av dagen (ettersom dette er rett før *sprint review* og *sprint planning*). Møtene ble benyttet til å gå gjennom den siste sprint-en og hvordan gruppen jobbet sammen. Derfra beskrev hvert gruppemedlem det de var fornøyde med i tillegg til forbedringspotensiale. Deretter ble dette diskutert i gruppen. På disse møtene ble gjerne Burndown Chart diskutert. Et slikt skjema er vist fra sprint 2 i figur 3.



Figur 3: Burndown chart fra sprint 2

Senere i prosessen sammenliknet gruppen tidligere Burndown Charts for å vurdere gruppens evne til å estimere. Figur 4 viser et Burndown Chart fra sprint 6. Den forteller oss at selv om gruppen ikke traff helt på estimatet for sprint 6 så ble gruppen bedre på å estimere i løpet av utviklingsperioden.



Figur 4: Burndown chart fra sprint 6

3.2.2 Møte med veileder

Videre hadde gruppen faste møter med veileder. I starten av prosessen ble disse møtene holdt ukentlig. Etter 3-4 uker bestemte gruppen seg for å trappe ned og fortsatte med regelmessige møter hver andre uke. Dette var mye på grunn av at sprintene ofte varte to uker. Dermed ble det mer naturlig å oppdatere veileder og be om råd etter en gjennomført sprint. Møtene ble benyttet til å konsuletere veileder om diverse saker som rapport, forventninger og temaer som burde undersøkes. Møtene ble holdt tirsdager 14.30.

3.2.3 Scrum board

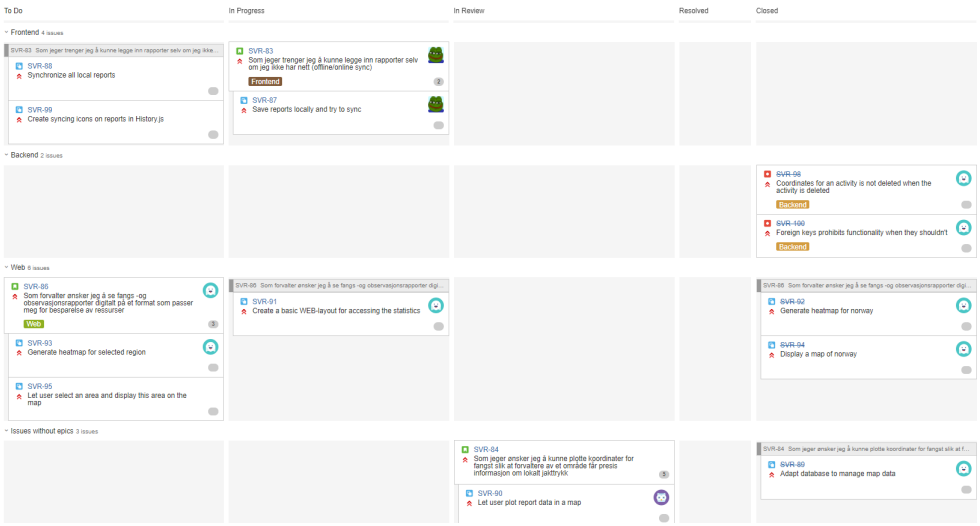
Verktøyet Jira ble benyttet for å støtte den smidige utviklingsprosessen. Tidlig overførte gruppen alle *user stories* fra Google Spreadsheets til Jira. Deretter ble aktuelle *user stories* flyttet inn i en ny sprint i hvert *sprint planning*-møte. Oppgavene i Jira var av forskjellige typer, som nevnt i seksjon 2.2.

Underveis i utviklingsprosessen har gruppen hatt stor nytte av Scrum Board-et som Jira tilbyr. Med dette verktøyet hadde alle utviklerne til en hver tid oversikt over hvor de andre var i prosessen med deres oppgaver. Videre hadde hver oppgave en egen status. I Scrum Board-et ble hver oppgave plassert i en kolonne basert på hvilken status den hadde. De ulike statusene er beskrevet i tabell 3.

Status	**Forklaring**
To Do	Alle oppgaver fikk denne statusen ved oppstart av en ny sprint. Alle oppgavene som skulle utføres i løpet av sprinten startet med denne statusen.
In Progress	Da en utvikler startet på en oppgave fikk denne statusen *In Progress*. Den statusen skulle oppgaven ha helt frem til utvikleren følte seg ferdig med oppgaven. Funksjonalitet skulle også testes av utvikleren selv i dette steget.
In Review	Da utvikleren mente han/hun var ferdig med en oppgave ble denne plassert i *In Review* og utvikleren opprettet en pull request. Oppgaver som hadde denne statusen beholdt statusen frem til de to andre utviklerne i gruppa hadde utført et code review på det som lå i pull request-en. Dersom de som vurderte koden mente at kvaliteten var god nok ble oppgaven godkjent. Ellers fikk utvikleren tilbakemeldinger på hva som kunne forbedres.
Resolved	Dersom en oppgave som lå i *In Review* ble godkjent skulle en av de som vurderte koden gi oppgaven status *Resolved*.
Closed	Når oppgaven som utvikleren hadde ansvar for fikk statusen *Resolved* var det den utviklerens ansvar å deretter lukke oppgaven ved å gi den en siste status *Closed*.

Tabell 3: Beskrivelse av statusene i Scrum Board-et

I figur 5 er et eksempel på hvordan Scrum Board-et har sett ut i løpet av utviklingsprosessen.



Figur 5: Scrum Board i Jira fra sprint 5

3.2.4 Arbeidsflyt

Under utvikling benyttet gruppen en arbeidsflyt som kalles «Git Feature Branch Workflow». Denne går ut på at all ny funksjonalitet lages i en egen branch, som så merges med hovedgrenen *master* gjennom en pull request. Dette sikrer at master alltid inneholder en kjørbar versjon av koden, som er svært gunstig ved bruk av Continous Integration (CI), og det blir enklere for utviklerne å utføre code review [13].

Gruppen brukte i tillegg en egen *dev*-branch som hovedgren gjennom sprintene, og merget denne med *master* etter endt sprint. Ved nye utviklingsoppgaver opprettet gruppen dedikerte *issue*-brancher som deretter ble merget med *dev* etter pull request.

3.2.5 Sprintoversikt

Under er en oversikt over alle sprintene som gruppen utførte i løpet av utviklingsperioden. Gruppen hadde totalt syv sprinter som hovedakelig ble dedikert til utvikling, men det ble også brukt tid på planlegging og rapportskriving.

Sprint 0 (10. jan - 4. feb)

Denne perioden ble benyttet til planlegging av prosjektet. Gruppen utformet en prosjektplan og startet på en kravspesifikasjon. Videre startet gruppen med testing av nye teknologier og satte opp de utviklingsmiljøene vi trengte. Et repository for mobilapplikasjonen og et repository for backend ble opprettet i Bitbucket og en server ble satt opp. I tillegg hadde gruppen mye fokus på utarbeiding av UI-prototype. Denne ble testet av en ekspert i interaksjonsdesign (les mer om dette i seksjon 8.1). Under testet fikk vi mange tilbakemeldinger som hjalp gruppen på veien videre. Utenom dette startet gruppen også å sette opp et API-rammeverk for backend.

Sprint 1 (5. feb - 19. feb)

I denne sprinten lå fokuset på å sette opp et skjelett og en struktur for mobilapplikasjonen og backend. Gruppen jobbet med enkel autentisering og startet på prosjektets kjerneoppgave; å la en jeger rapportere inn dagens jakt og observasjoner. I tillegg jobbet gruppen med å la en bruker definere et aktivt jaktområde for rapportering.

Sprint 2 (20. feb - 05. mars)

Her jobbet gruppen med å fullføre resten av oppgavene fra forrige sprint. Videre ble en ny story påbegynt; å se tidligere rapporter og mulighet til å endre disse. Disse oppgavene ble ferdigstilte. På backend ble det jobbet med redefinering av API og database.

Sprint 3 (06. mars - 13. mars)

Denne sprinten ble dedikert til bug-fiksing og stabilisering slik at mobilapplikasjonen kunne benyttes til en dedikert brukertest. Ettersom fokuset skulle være bug-fiksing ble det bestemt i samarbeid med produkteier at sprinten skulle vare én uke i stedet for to. I tillegg implementerte gruppen internasjonalisering for å sikre universell utforming. Å logge ut ble også implementert i tillegg til å håndtere Android sin tilbakeknapp. Videre brukte gruppen også tid på å utforme en plan til brukertestingen (se vedlegg C).

Sprint 4 (14. mars - 23. mars)

Da arbeidet med stabilisering ble satt i gang oppdaget gruppen at det var en mye større jobb enn forventet. Denne sprinten ble derfor også dedikert til stabilisering og varte litt over en uke. Her hadde gruppen fokus på å fullføre planen til brukertesting, opprette APK-fil for testing og sørge for at applikasjonen var stabil. Det ble også bestemt at brukertestingen skulle foregå over påske (26. mars - 28. mars). Derfor brukte gruppen de siste dagene av sprinten til refaktorering av mobilapplikasjonen og backend. Etter endt sprint opplevdes applikasjonen som svært stabil og pålitelig.

Sprint 5 (03. april - 16. april)

I denne sprinten ble fokuset rettet mot mer kompleks funksjonalitet. På mobilappliasjonen ble kart implementert og muligheten til å plassere markeringer der jegeren har skutt dyr. Det ble også implementert polygoner for de ulike jaktområdene. Disse ble markert på kartet. I tillegg henter appen brukerens gps-lokasjon. Videre ble offline-modus for applikasjonen nesten ferdig-implementert. Dermed ble det mulig for en jeger å opprette nye rapporter selv uten internettforbindelse. På backend ble API-et tilpasset de nye kartdataene. Utenom dette ble det også opprettet en enkel web-løsning for statistikk. I web-løsningen implementerte gruppen et heat map som illustrerer hvor det blir skutt mest dyr.

Sprint 6 (17. april - 23. april)

Gruppen valgte videre å ha en ny sprint på én uke. Dette var på grunn av flere årsaker: offline-modus ble ikke helt ferdig implementert sprinten før, og mobilapplikasjonen manglet å håndtere bestemte spesialtilfeller innenfor kart og redigering av markeringer. Produkteier ønsket at gruppen fikk dette på plass, og det ble estimert at gruppen ikke trengte mer enn en uke. Dermed ble denne sprinten dedikert til disse to oppgavene for mobilapplikasjonen. I tillegg hadde gruppen da et medlem uten tildelt arbeid, så gruppen ble enige om å implementere et filter for heat map-et i web-løsningen. Dette ble også

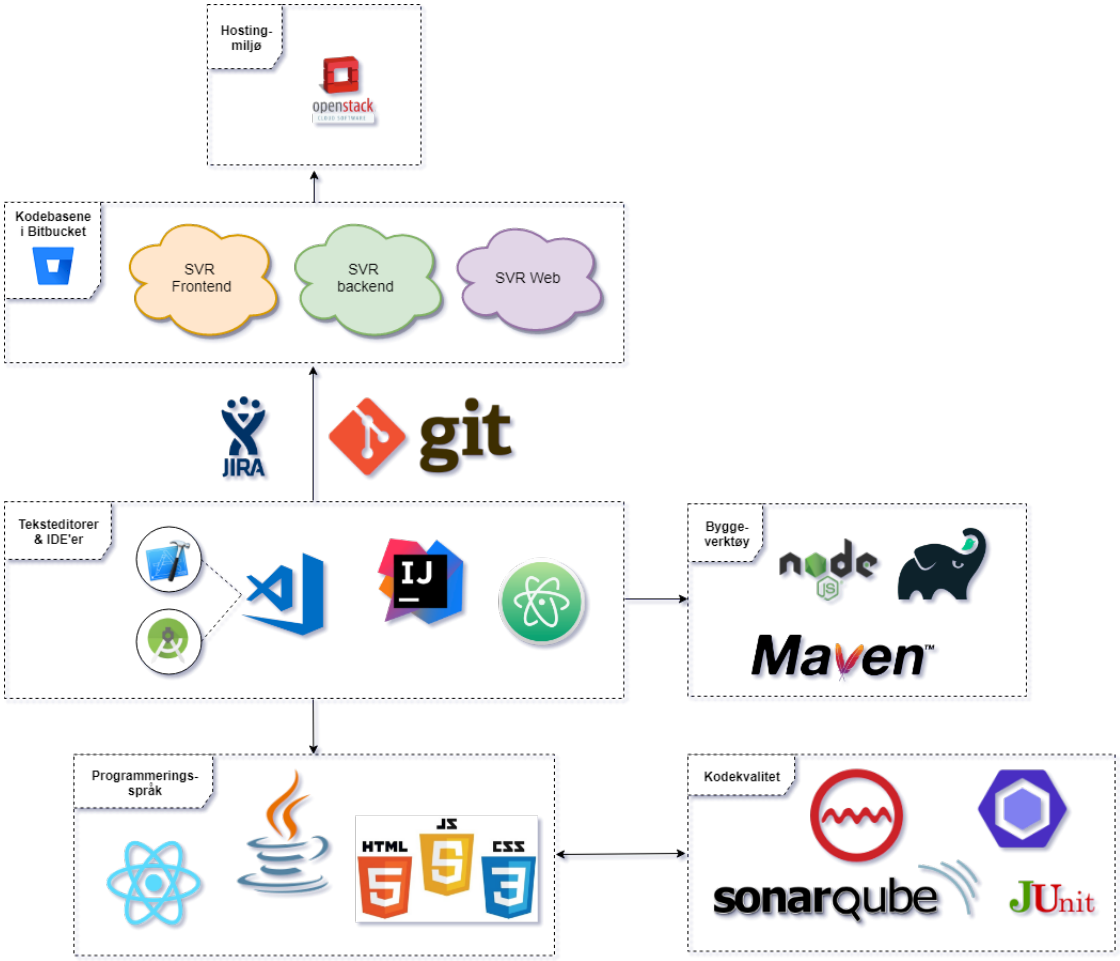
ferdig implementert i løpet av denne sprinten. Noe gruppen ikke rakk denne sprinten var å vise forskjellige ikoner per rapport i historikk (i mobilappplikasjonen). Det skulle vises ikoner som illustrerer om rapporten er synkronisert eller ikke.

Sprint 7 (24. april - 15. mai)

Den siste sprinten ble satt til tre uker ettersom dette var den gjenværende tiden av prosjektperioden. I denne sprinten tok ikke gruppen inn ny funksjonalitet. Innen utvikling arbeidet gruppen kun med kjente bugs og mer trivielle finjusteringer. Ellers hadde gruppen størst fokus på å skrive ferdig rapporten.

4 Teknologier

I oppgaven ble det benyttet en rekke ulike teknologier og verktøy for å oppnå et best mulig resultat. Noen av dem var gruppen kjent med fra tidligere i studiet og andre lærte gruppemedlemmene å bruke underveis i dette prosjektet. Figur 6 viser en oversikt over de teknologiene og verktøyene som ble benyttet. I dette kapittelet vil gruppen gå nærmere inn på hvilke programmeringsspråk/Integrated development environment (IDE)-er som ble benyttet og hvorfor. Videre fokuseres det også på utviklingsmiljøene som ble tatt i bruk.



Figur 6: Alle teknologier benyttet i oppgaven

4.1 Programmeringsspråk og rammeverk

4.1.1 Mobilapplikasjonen: React Native

React Native er et rammeverk som anvender React-strukturen på native Android og iOS, og var hovedsakelig et forslag fra produkteier. I oppgaveteksten var det nevnt at det er ønsket støtte for både Android- og iOS-enheter. Det kom frem i senere samtaler at produkteier har jobbet en del med React Native tidligere og ønsket veldig gjerne at gruppen benyttet dette rammeverket.

Gruppen hadde valget mellom å utvikle applikasjonen for både Android og for iOS separat eller benytte en kryssplattform-teknologi. React Native er en kryssplattform-teknologi og har en stor fordel ved at rammeverket tillater utviklere å lage en applikasjon én gang, men distribuere appen til flere plattformer; hovedsakelig Android og iOS. Vanligvis er Android- og iOS-utvikling to krevende og svært ulike oppgaver. Språkene benyttet i utviklingen er ulike; iOS benytter Objective-C/Swift og Android benytter Kotlin/Java for Android. Videre er de underliggende API-ene forskjellige; metodene for å hente ut data fra enhetens sensorer og hardware er ulike på de to plattformene. Det er her React Native skiller seg ut. React Native's underliggende rammeverk skiller ikke mellom de to plattformene [14]. Dette er fordi rammeverket bygger på de underliggende native API-ene til plattformene. Med andre ord reduserer React Native, i teorien, arbeidsmengden som behøves for å distribuere en kryssplattform-applikasjon for Android og iOS.

Videre undersøkte gruppen også ulempene ved å benytte React Native. En stor del av mobilutvikling i dag inkluderer bruken av Operativsystem (OS)-spesifikke API-er. Problemet er at det ikke alltid finnes en «React Native måte» å gjøre ting på; komponentene React Native tilbyr er ikke nødvendigvis tilstrekkelig. React Native har begrenset tilgang til «native»-modulene til telefonen [15]. Heldigvis finnes det flere tredjepartskilder som gir utviklere alternative løsninger. Et annet argument er at Android og iOS har ulike retningslinjer innenfor design av brukergrensesnitt. Dette er diskutert mer i detalj i seksjon 5.3. Dette innebærer at designet i React Native blir komplekst, og at det kreves mange forsikringer i koden og separering av kode for å oppnå ønsket design for alle enheter. Det kjente fenomenet med kryssplattform-utvikling gjelder her, og det lyder som følger: «Skriv en gang, debug overalt» [14].

Basert på de positive og negative aspektene ble det gjort en oppveiing og konkludert at React Native er et godt alternativ for vår applikasjon også uavhengig av produkteiers ønske. Dermed kunne gruppen utvikle én mobilapplikasjon med én kodebase for både Android og iOS.

4.1.2 Backend: Java

Det var ingen krav angående backend fra produkteier, annet enn at det skulle være et RESTful JSON API. Dermed stod gruppen fritt til å velge backend-teknologi selv. Som navnet JSON-API tilsier, benyttes JSON til å formidle data mellom klient og tjener. JSON var spesielt ønsket da det er implementert av de fleste språk og teknologier, og det gjør API-et svært fleksibelt med tanke på fremtidige klienter.

Innenfor backend-teknologier kjente gruppen til PHP og Java. Disse var derfor sterke kandidater. I tillegg ble gruppen anbefalt av produkteier å vurdere Node.js.

PHP og Java er begge «general purpose»-språk, noe som betyr at begge er laget for

å dekke de fleste bruksområder. PHP er laget spesielt for web, og har lenge vært en «de facto standard» for web-backend. Java er laget for å kjøre selvstendige applikasjoner på hvilken som helst plattform, men er i dag óg svært mye brukt som backend innen web [16].

Med gruppens studie som bakgrunn, har gruppen utviklet en preferanse for objektorienterte språk, noe som gjør at Java stiller sterkt. I tillegg viser Java til en kraftig ytelse i motsetning til PHP, der Java (i kombinasjon med Spring-rammeverket) utkonkurrerer PHP på alle områder [17]. Java krever derimot en noe mer avansert struktur, noe som kan bremse utviklingen noe. I motsetning til dette, har Java et bedre utvalg av IDE-er og muligheter for «step-debugging», som igjen vil øke produktiviteten.

Java gir også muligheten for å kunne kontrollere hele databasen gjennom Object Relation Mapping (ORM), noe som gjør det veldig enkelt å endre databaseteknologi, om det skulle bli ønskelig ved videre utvikling. I tillegg gir dette gruppen muligheten til å lære en ny teknologi i et kjent miljø, som også er svært relevant for gruppens fremtidige karriere innen programvareutvikling.

Valget falt på Java, med *Spring Boot* for å håndtere web-forespørsler, og *Katharsis* for å implementere JSON API-standarden. Både Katharsis og Spring Boot kan enkelt integreres, og de benytter *Hibernate ORM* for å kommunisere med databasen.

Node.js ble foreslått av produkteier, og stilte sterkt med tanke på ytelse. Etter en grundigere undersøkelse kom gruppen frem til at Node.js og Java er forholdsvis like [18], og spørsmålet ble mer om preferanser. Ettersom fokuset var på mobilapplikasjonen, som ble skrevet med ukjent teknologi, valgte gruppen å forholde seg til kjente omgivelser på backend. Java har også to store fordeler: bedre utviklingsmiljøer og et betydelig større utvalg i biblioteker [18]. Dette lå også til grunne for at gruppen valgte Java ovenfor Node.js for backend.

Valg av rammeverk

Jsonapi.org anbefaler tre open source rammeverk for implementasjon av RESTful JSON API i Java: [19]: *Katharsis*, *Elide* og *Crnk*. For å behandle web-forespørsler og integrasjon av database valgte gruppen å se på DropWizard og Spring Boot. Begge er populære og er biblioteker som er mye benyttet for å håndtere web-innhold i Java.

Crnk er et stort og velkjent bibliotek for implementasjon av JSON API i Java. Crnk er svært fleksibelt, da man selv kan implementere hvordan dataene skal behandles, lagres og returneres, i tillegg til at Crnk har en omfattende sikkerhetsmodul [20]. Ved en nærmere undersøkelse og sammenlikning mot de to andre bibliotekene (Katharsis og Elide, beskrevet nedenfor), viser det seg at repositories og funksjonene som Crnk må støtte må implementeres manuelt, noe både Katharsis og Elide behandler automatisk. Dette gjør at det blir mer unødvendig arbeid, og dermed ble Crnk til et mindre aktuelt bibliotek for vårt prosjekt.

Elide er tilpasset mobile enheter, med tanke på mengde data som sendes, og har en godt utviklet sikkerhetsmodul som enkelt lar utvikleren behandle tilgangsrettigheter til de ulike ressursene [21]. Elide så mest lovende ut, og virket som et godt valg for gruppens produkt. Derfor valgte gruppen å skrive en test-implementasjon av dette rammeverket for å se hvordan det oppførte seg. Gruppen oppdaget da at dokumentasjonen var utdatert, og alle eksempler som var beskrevet ikke fungerte. Etter noen dager med feilsøking, uten

hell, besluttet gruppen å teste kandidat nr. 2: Katharsis.

Katharsis har en relativt stor brukermasse, og ser ut til å være en de-facto-standard innen JSON API i Java. På mange områder er Katharsis lik Elide. Det som trekker Katharsis ned er utdatert dokumentasjon (på lik linje med Elide) og at rammeverket kun har en eksperimentell sikkerhetsmodul [22]. Til tross for utdatert dokumentasjon var det enklere å implementere dette da brukermassen er større, og flere «oppskrifter» var tilgjengelige fra tredjeparts-aktører.

Spring Boot og DropWizard er på mange måter like. Begge håndterer web-forespørsler og database-integrasjon, og har støtte for filtrering av forespørsler. Valget her ble ganske enkelt ettersom Katharsis har en modul for automatisk integrasjon med Spring Boot. I tillegg kan Spring Boot kjøres med bare én kodelinje, og er dermed utrolig rask å sette opp [23]. Ekstra moduler, for eksempel integrasjon mot databaser, var også ferdig konfigurert etter integrasjon gjennom Maven.

Spring er et rammeverk for å bygge web-applikasjoner i Java. Spring implementerer de underliggende funksjonene som ikke er spesifikke for applikasjonen. Et eksempel på dette er håndtering av inkommende og utgående forespørsler over HyperText Transfer Protocol (HTTP)-/HyperText Transfer Protocol Secure (HTTPS)-protokollen. *Spring Boot* er en utvidelse av Spring, som gjør at mye av konfigurasjonen er ferdig laget. Dette gjør at man kan starte en Spring-applikasjon på bare få minutter. Dette er ofte et naturlig utgangspunkt for de fleste applikasjoner [23].

Gruppen valgte å benytte Spring Boot og Katharsis som rammeverk for backend. Katharsis kan enkelt integreres med Spring Boot gjennom sin *Katharsis-SpringBoot*-modul, og den manglende sikkerhetsmodulen i Katharsis kan erstattes med Spring Boot sin sikkerhetsmodul. Gruppen mener at denne kombinasjonen førte til et skalerbart API med løse koblinger. Dermed kan løsningen endres og videreutvikles uten å påvirke implementasjonen av klientene.

4.1.3 Web-løsningen: JavaScript/jQuery

JavaScript er et script-språk for web som er støttet av alle store nettlesere. Dette språket legger tilrette for prosessering på klient-siden, som gjør at man avlaster serverne og applikasjonen oppleves raskere. Språket kan utvides med JavaScript-biblioteket jQuery, som forenkler mange av operasjonene i JavaScript, for eksempel Ajax. Ved å benytte JavaScript og jQuery kunne web-løsningen bli utviklet som en single page-applikasjon.

Web-grensesnittet presenterer statistikk av de dataene som mobil-applikasjonen registrerer. Siden backend til applikasjonen er et API basert på web trengte ikke denne applikasjonen en egen backend. Derfor valgte gruppen JavaScript (jQuery) for å hente og generere den statistikken som skal vises.

For å visualisere statistikken ble Google Maps API for JavaScript benyttet for å få en kartvisning. Dette API-et har også en modul for visualisering av data, som gjør at det kreves minimalt med kode for å kunne visualisere de dataene som er lagret i databasen på en oversiktlig måte.

4.2 Utviklingsmiljø

4.2.1 Mobilapplikasjonen

Visual Studio Code

Visual Studo Code er et tekstredigeringsverktøy utviklet av Microsoft [24]. Dette verktøyet ble benyttet for utvikling av mobilapplikasjonen. Det passet spesielt godt å bruke Visual Studio Code da gruppen utviklet i React Native ettersom programvaren kommer med innebygget støtte for JavaScript, TypeScript og Node.js. I tillegg er det mulig å legge til en rekke ulike tilleggpakker som ESlint og SonarLint. Slike pakker var med på å sikre god kodekvalitet. Les mer om gruppen tiltak for å sikre god kodekvalitet i kapittel 7.

Android Studio

Android Studio er en IDE som er utviklet spesielt for utvikling av Android-applikasjoner. Utviklingsmiljøet er laget av Google og JetBrains og er Android sin offisielle IDE [25]. I dette prosjektet har gruppen benyttet emulatoren som Android Studio tilbyr. Verktøyet lar utvikleren teste sine apper med en variasjon av Android-enheter. Android API, versjoner o.l kan velges spesielt for emulatoren.

Xcode

Xcode er en IDE spesielt utviklet for å lage applikasjoner for Apple-plattformer og er utviklet av Apple [26]. Også dette verktøyet ble benyttet på grunn av simulatoren som verktøyet tilbyr. Gruppen trengte Xcode for å forsikre seg at applikasjonen også fungerte for iOS-enheter. Simulatoren er rask og kan startes opp med flere instanser. Grunnen til at gruppen benyttet dette verktøyet var at ingen i gruppen eide et Apple-produkt (i stand til å kompilere kode) under utviklingsperioden.

Node.js

Node.js er et åpent kryssplattform runtime-system for JavaScript som er bygget på «Chrome's V8 JavaScript Engine». Det er bygget for å være lett og effektivt [27]. Innenfor Node.js har gruppen underveis benyttet pakkebehandleren «npm» for å installere ønskede og nødvendige pakker og bibliotek for React Native. Videre har Node.js vært essensielt for å kjøre applikasjonen i en emulator/simulator. Node.js ble startet og kjørte i bakgrunnen hver gang gruppen bygget applikasjonen på nytt for Android eller iOS. Dette ble gjort ved å utføre følgende kommando for å starte applikasjonen i en Android-enhet:

```
$ react-native run-android
```

Eller med følgende kommando for å starte applikasjonen i en iOS-enhet:

```
$ react-native run-ios
```

Expo

Expo er et gratis og åpent kildekodeverktøy-sett bygget rundt React Native for å hjelpe til i prosessen ved utvikling av native iOS og Android projekter [28]. Dette verktøyet ble benyttet tidlig i prosessen da gruppen testet ut React Native for første gang. På dette tidspunktet hadde gruppen opprettet prosjektet ved å benytte «create-react-native-app»-pakken fra Node.js. Denne pakken ble benyttet for å initialisere prosjektet for første gang. Dette innebar at applikasjonen kunne testes på gruppemedlemmenes private mobil-enheter (både for Android og iOS), men ikke i en emulator eller simulator [29].

Etter litt testing ble gruppen raskt enige om at vi burde ha muligheten til å bygge prosjektet med «native code». Dermed hadde ikke gruppen lengre behov for Expo.

Testenheter

Under utviklingsperioden hadde gruppemedlemmene to private Android-enheter og én iOS-enhet tilgjengelig. Gjennom prosjektet var det enkelt å generere Android application package (APK) for å teste mobilapplikasjonen på en av Android-enhetene. Tilsvarende var ikke mulig for iOS-enheten ettersom det krever en Mac-enhet som har Xcode installert [30]. Ingen av gruppemedlemmene eide en Mac under utviklingsperioden, og derfor kunne ikke mobilapplikasjonen testes regelmessig i et iOS-miljø. Mot slutten av utviklingsperioden ønsket gruppen likevel å forsikre for at appen kunne kjøres på iOS. Da valgte gruppen å spørre om rettigheter til skolens Mac-lab. Dette fikk vi, og dermed fikk vi testet ut mobilapplikasjonen på en iOS-enhet. Les mer om de spesifikke konfigurasjonene som måtte defineres i seksjon 6.1.7.

4.2.2 Backend

IntelliJ Community

Intellij Community er en gratis, open source IDE for utvikling til Java- og Android-applikasjoner [31]. Valget stod mellom IntelliJ og Eclipse, men IntelliJ ble valgt på grunn av personlig preferanse, dets raske arbeidsflyt og muligheter for integrasjon med andre verktøy og rammeverk som gruppen bruker, f.eks. Git og Spring Boot. IntelliJ ble brukt til å utvikle backend.

Maven

Maven er et gratis verktøy for automatisert kompilering, pakking, testing og leveranse [32]. Maven ble primært benyttet på backend for å kompilere serveren, generere JavaDoc, kjøre unit-tester, statisk analyse og pakking, automatisk i en og samme operasjon:

```
$ mvn clean verify
```

Maven definerer også hvilke eksterne biblioteker som kreves for å kjøre applikasjonen, og hvilke versjoner av disse som skal brukes [32].

JavaDoc

JavaDoc er et verktøy utviklet av Sun Microsystems [33] for å automatisk generere utvikler-dokumentasjon. JavaDoc genererer et web-grensesnitt med dokumentasjonen, basert på kommentarene som er skrevet i koden.

4.2.3 Web-løsningen

Atom

Atom er en open source teksteditor for programmering, og er laget for å kunne spesialtilpasses hver enkelt bruker. Atom gir muligheter for integrasjon med mange ulike utviklerverktøy som Git, Terminal, og forskjellige Linters, i tillegg til autocomplete, prosjekthåndtering og innebygd pakke-manager [34]. Hele web-løsningen ble utviklet i Atom.

XAMPP

XAMPP er en open source utviklerplattform som gir enkel tilgang til en database, støtte for PHP og Pearl, og lokale web-servere for å kjøre web-løsningen under utvikling [35]. XAMPP ble aktivt benytttet under utvikling som test-miljø og oppsett av databasen.

Twitter Bootstrap 4

Bootstrap er et CSS-bibliotek som er utviklet av Twitter. Dette biblioteket sørger for at applikasjonen er responsiv. I tillegg benyttes Bootstrap for å få et uniformt grensesnitt med rent design.

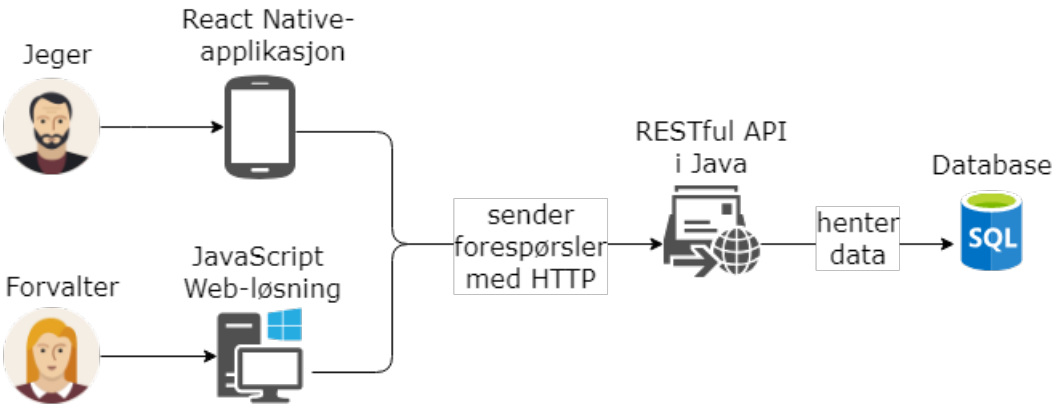
5 Programvaredesign

I dette kapittelet vil gruppen ta for seg produktets programvaredesign. Dette innebærer en diskusjon rundt valg av arkitektur og hvordan de ulike modulene i produktet er koblet sammen. Videre vil gruppen også diskutere grensesnittet til mobilapplikasjon i detalj i tillegg til grensesnittet i web-løsningen.

5.1 Arkitektur

Figur 7 viser et overordnet bilde av hvordan produktet ser ut og hvordan de ulike delene kommuniserer med hverandre. Både mobilapplikasjonen (skrevet i React Native) og web-løsningen (skrevet i JavaScript) henter data fra databasen gjennom det samme JSON API-et (skrevet i Java).

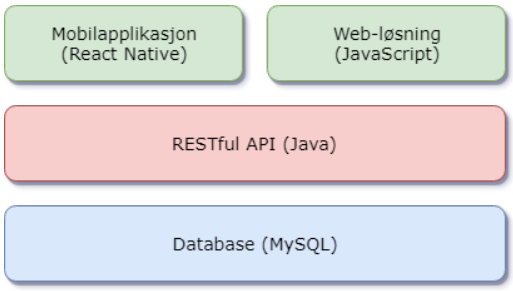
Tjenesten består med andre ord av to tynne klienter som henter alt av data gjennom et moderne RESTful API. API-et benytter JSON-formatering som standard. Gruppen valgte å utvikle et RESTful API av flere grunner. Den største årsaken var at dette var definert som et krav fra produkteier (se seksjon 1.8). Utenom dette opplevde også gruppen at det ville være en fordel ettersom produktet skulle bestå av to klienter som forholder seg til samme database i backend. Med et slikt API kunne to ulike klient-tjenester benytte samme database uten at API-et måtte endres eller tilpasses.



Figur 7: Produktets struktur og generell dataflyt

5.1.1 Lagdelingsmodellen

Arkitekturen til produktet har et tydelig preg av lagdelingsmodellen. Dette er illustrert i figur 8. Produktet følger denne modellen ettersom produktet består av lag der hvert lag har en egen og spesifikk rolle. Det vanligste er at applikasjoner har fire lag, men Mark Richards forteller også at det er vanlig for mindre applikasjoner (som gruppens produkt) å bestå av tre lag [36].



Figur 8: Oversikt over de ulike lagene i produktet

Produktet består av et *presentation layer* (mobilapplikasjon og web-løsning), et *persistence layer* (API-et) og et *database layer* (databasen). En annen grunn til at gruppen valgte å ikke definere mer enn tre lag var for å unngå anti-patternet *architecture sinkhole*. Patternet kan raskt bli et anti-pattern dersom det benyttes flere lag som ikke tilfører dedikert logikk til dataene som flyter gjennom [36].

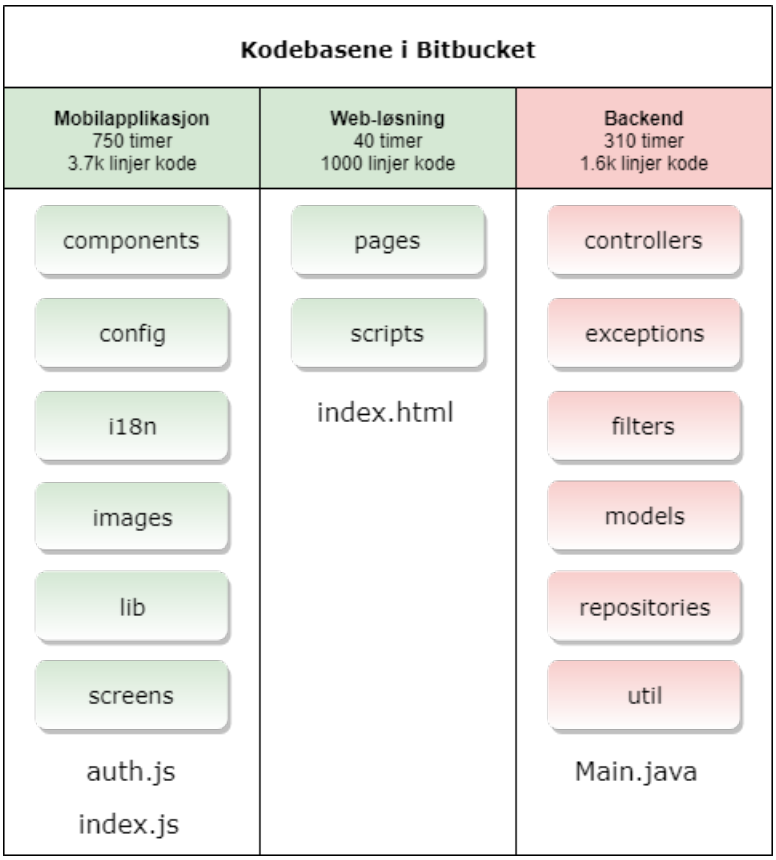
Hovedkonseptet med lagdelingsmodellen er at dataflyten kun skal flyte direkte mellom laget under og over. Slik fungerer også gruppens produkt; når en av klientene fra *presentation layer* trenger å hente informasjon fra databasen i *database layer* må denne forespørselen gå gjennom API-et i *persistence layer*. Dette foregår via spesifikke API-forespørsler. Disse er beskrevet i mer detalj i seksjon 6.2.1. En av de største fordelene med denne lagdelingen er at det isolerer hvert lag og funksjonaliteten. Dette innebærer at endringer eller oppdateringer i ett lag generelt ikke påvirker de andre lagene. Gruppen opplevde dette som et svært appellerende aspekt da arkitekturen skulle planlegges. Naturligvis viste det seg å være en stor fordel under utviklingsprosessen. Med denne arkitekturen kunne gruppemedlemmene arbeiede med hvert sitt «lag» uten å risikere at noe gikk galt i et av de andre lagene. For klientenes del kunne data enkelt mockes frem til API-et og databasen ble implementert. I tillegg var det enkelt å teste de ulike lagene.

Videre hadde gruppens arkitektur et unntak innenfor konseptet med isolering av lag. Produktet ble implementert på en slik måte at klientene ble påvirket dersom det ble utført betydelige endringer i API-et. Dette var et resultat av at gruppens *presentation layer* kommuniserte direkte med *persistence layer*. Richards beskriver også denne situasjonen og hvordan en applikasjon kan bli tett koplet sammen på denne måten [36]. På en annen side gjaldt dette kun de delene av klientene som sendte forespørsler til API-et. I tillegg er det kjent at det er svært vanlig for klienter å sende forespørsler direkte til et API. Videre hadde gruppen stort fokus på å arbeide på samme sted og til samme tider for å ha best mulig kommunikasjon under hele prosessen for å forebygge eventuelle feil. Les mer om utviklingsprosessen i kapittel 3.

5.2 Moduler

Figur 9 viser hvordan de forskjellige delene av produktet er bygd opp og hvordan kodebasene er strukturert. I figuren er fargen på hver kodebase assosiert med tilhørende lag fra lagdelingsmodellen i figur 8.

I produktet har mobilapplikasjonen den største kodebasen på ca. 3600 linjer med kode og har vært den mest tidskrevende delen av produktet (ca. 750 timer arbeid). Dette er blant annet på grunn av at gruppen har bygd opp hele applikasjonen og skrevet nesten hele kodebasen selv. Innenfor mobilapplikasjonen ble det satt av spesielt mye tid til implementasjon av kart og offline-funksjonalitet. Deretter har backend tatt opp nest mest tid med sin kodebase på ca. 1600 linjer (ca. 310 timer arbeid). Også backend ble utviklet med minimalt av eksterne ressurser. For backend gikk det mye tid til å undersøke og teste diverse rammeverk før gruppen fant en sammensetning som passet med produktets natur. Det kommer også frem i figur 9 at web-løsningen har vært minst krevende med sine ca. 1000 linjer med kode (ca. 40 timer arbeid). Grunnen til dette er, som nevnt innledningsvis i rapporten, at hovedfokuset skulle være på en mobilapplikasjon for jegerne. Dette har vært produkteiers største ønske og fokus gjennom hele utviklingsprosessen. Web-løsningen ble ikke implementert før den siste perioden av prosessen og ble implementert hovedsakelig fordi det, i følge produkteier, vil gi produktet en høyere verdi på markedet. Les mer om hvordan de ulike delene av produktet ble implementert i kapittel 6.



Figur 9: Strukturen og modulene til de ulike delene av produktet

5.3 Brukergrensesnitt i mobilapplikasjonen

Innenfor User interface (UI) har gruppen valgt å gjennomføre flere iterasjoner med utvikling av prototyper og brukertesting. Dette valgte gruppen å gjøre i god tid før oppstart av utviklingen. Hensikten med dette var å spare dyrbar tid. Ved å fokusere på dette i starten av utviklingsprosessen sikret gruppen seg et best mulig utgangspunkt. Ellers ville det vært svært sannsynlig at gruppen måtte brukt mye mer tid på å justerere grensesnittet senere i prosjektet slik at mobilapplikasjonen kan oppleves mer brukervennlig og intuitiv. Den endelige prototypen ble deretter implementert med hensyn til ulike konsepter fra Material Design [37].

Prototyper

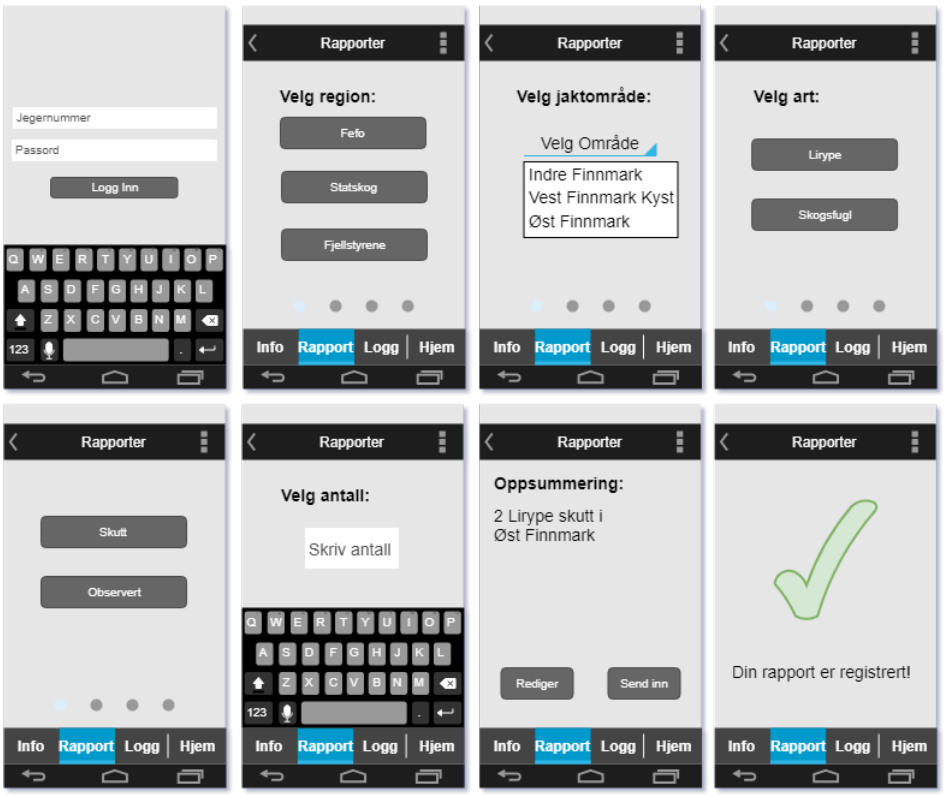
I hver prototype valgte gruppen å fokusere på mobilapplikasjonens hovedoppgave; å rapportere dagens jakt. Dette var på grunn av produkteiers sterke ønske om en enkel og intuitiv måte å utføre småvilrapportering. Under hele prosessen har jegeren vært i sentrum, og det viktigste for produkteier har vært at gruppen utvikler en god løsning med størst hensyn til jegeren.

Første prototype

Den første iterasjonen av prototypen ble bearbeidet etter første møte med produkteier. Den viser det første utkastet gruppen hadde til utforming av applikasjonens hovedoppgave. Bildene i figur 10 tar brukeren gjennom prosessen ved å logge inn, velge jaktområde og sende inn en rapport. På dette tidspunktet trodde gruppen at jegeren måtte registrere seg for å bruke applikasjonen. Derfor logger han/hun seg på med jegernummer og passord. Deretter velger brukeren en region og et spesifikt jaktområde. Da er brukeren på forsiden og kan velge å rapportere for en art som befinner seg i området. Deretter defineres det om arten er skutt eller sett og hvor mange fellinger/observasjoner som ble gjort. Til slutt får brukeren en oppsummering og bekreftelse på rapporten.

Allerede på dette tidspunktet vurderte gruppen også å la jegeren gjøre alt av rapportering på én skjerm. Gruppen valgte dette bort ettersom gruppen ville prøve å begrense mengden med informasjon vist på skjermen til jegeren. Tanken var at dette ville gjøre rapporteringsprosessen enklere og raskere.

Naturligvis ble den endelige prototypen svært ulik den første i figur 10. Spesielt måten å rapportere på i første prototype er tungvind og legger en del begrensninger på hvordan brukeren kan utføre rapporteringen. Dette er fordi brukeren ble tvunget til å rapportere for én art av gangen. Det er også lite effektivt og tidskrevende. Dette ble svært tydelig da gruppen viste prototypen til produkteier ved første møte etter jul. Produkteier minte oss på viktigheten av at rapportering skal være så enkelt som mulig, men også fleksibelt og intuitivt.

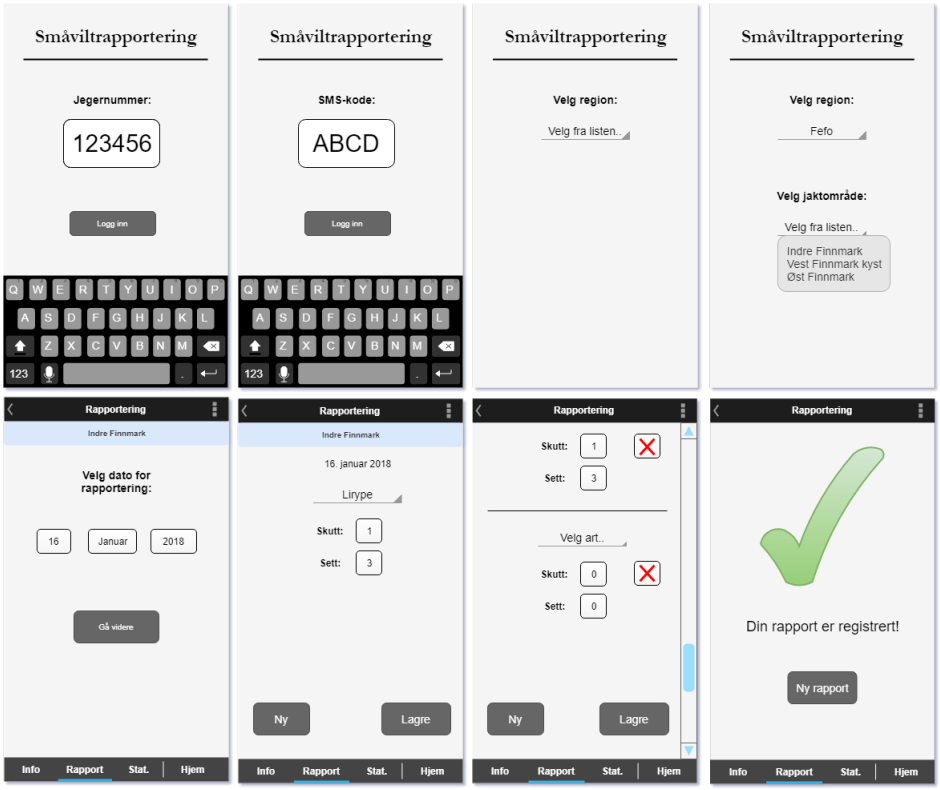


Figur 10: Første prototype

Andre prototype

Den andre prototypen (se figur 11) startet gruppen på raskt etter nevnt møte med arbeidsgiver. Her eksperimenterte gruppen mer med tanken på brukeridentifikasjon med hjelp av SMS-kode. Etter innlogging og valg av jaktområde må brukeren først velge en dato for rapportering. Gruppen ble enige om at det må være mulig å ikke bare rapportere for dagen i dag. Videre ble det gjort større endringer på selve rapporterings-oppgaven. Brukeren velger en art i en dropdown-meny og definerer antall skutt og sett for valgt art. Brukeren kan også legge til nye arter og fjerne dem før han/hun 'lagrer' rapporten. Det var denne prototypen gruppen benyttet da applikasjonen ble brukertestet av en ekspert i interaksjonsdesign (se resultatet i seksjon 8.1).

Alternativt vurderte gruppen også andre måter å hente valgt dato på. Gruppen diskuterte forskjellene mellom iOS og Android sine datovisninger, og kom frem til at gruppen foretrakk iOS sin typiske dato-input. Det ble ikke vist i prototypen, men tanken var også at hver input for sett og skutt skulle ha en «scroll wheel» effekt ved valg av input. Utenom dette ble også kalender nevnt, men gruppen tenkte på dette tidspunktet at en kalender ville ta opp for mye plass på skjermen.

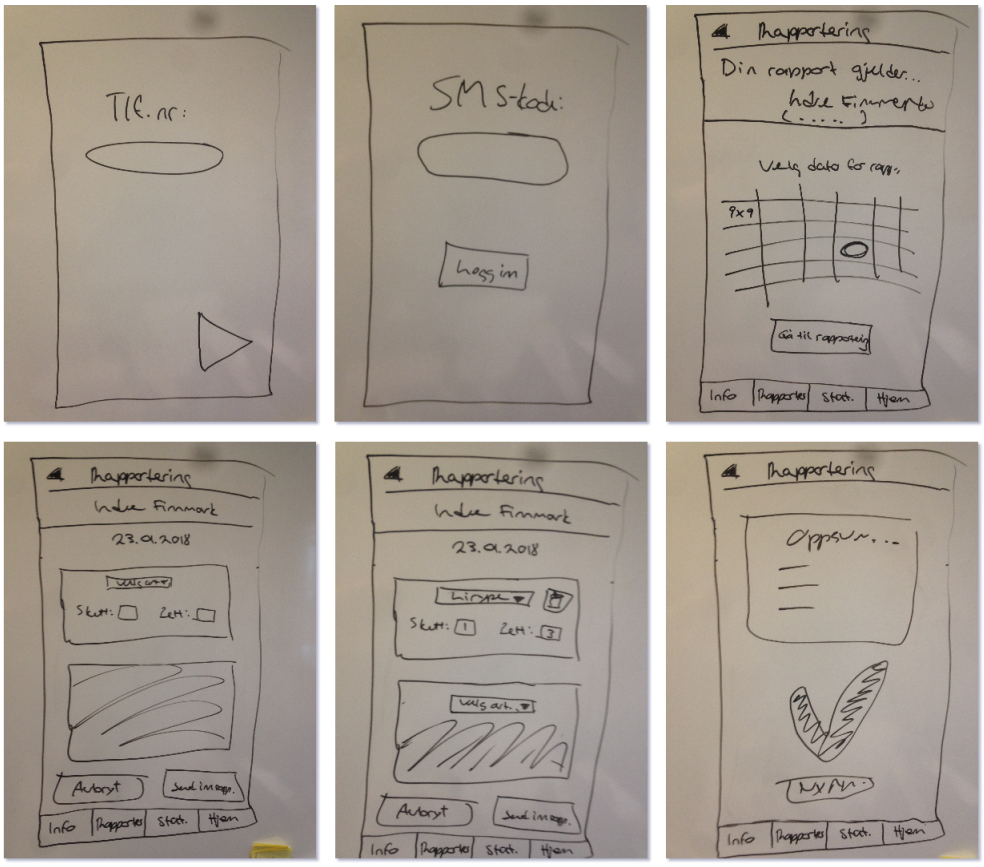


Figur 11: Andre prototype

Tredje prototype

Den tredje prototypen (se figur 12) skisserte gruppen raskt opp på et whiteboard etter brukertesting med nevnt ekspert. Her identifiserer jegeren seg med telefonnummer og SMS-kode. Videre er jaktområde allerede satt, men kan endres på «forsiden» til rapporteringsaktiviteten. Her velger brukeren dato med en kalender i stedet. Deretter kan brukeren legge til arter på tilnærmet måte som i prototype nr. 2, men hver art har fått et eget «kort». I tillegg kan brukeren fjerne arter fra rapporten. Når jegeren sender inn rapporten vises det en oppsummering av rapporten.

Basert på tilbakemeldinger fra eksperten og diskusjon i gruppen ble det klart at et spesifikt jaktområde burde være satt som aktivt for nye rapporter frem til brukeren selv endrer det. Brukeren burde velge jaktområdet én gang ved førstegangs-innlogging og ikke bli spurt om det siden. Videre ble gruppen også opplyst om at en kalender ikke ville ta opp «for mye» plass på skjermen og kunne være en svært god måte å hente valgt dato. Derfor valgte gruppen å benytte kalender i denne prototypen. I tillegg ble selve rapporteringen mye mer oversiktlig da hvert art og dens sett- og skutt-verdier ble skilt ut i egne «kort». Utenom dette var også en av tilbakemeldingene fra eksperten at det er svært viktig å gi brukeren god feedback etter opprettelse av en ny rapport. Derfor vises en oppsummering av rapporten i prototypen etter at jegeren har lagret en ny rapport.



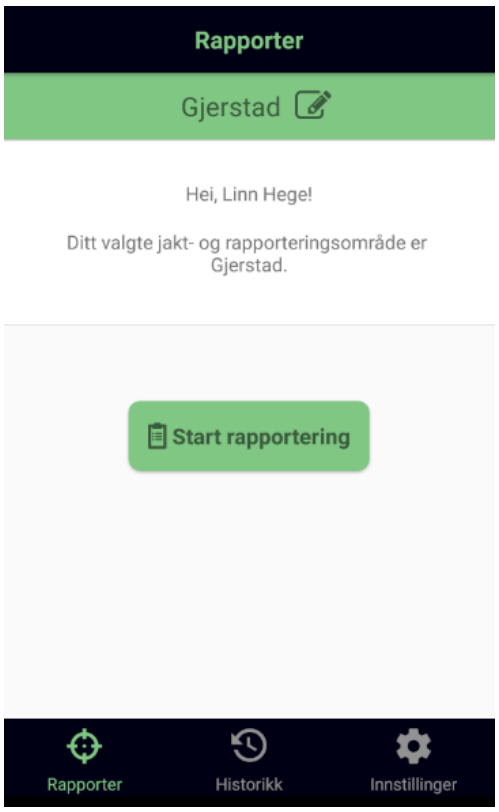
Figur 12: Tredje prototype

Layout

Innenfor layout og design valgte gruppen å forholde seg til Material Design og benytte diverse konsepter fra dem. I tillegg hadde produkteier et sterkt ønske om at applikasjonen skulle være enkel å bruke. For å oppnå dette besluttet gruppen at layouten måtte være preget av kjent design i stedet for noe mer unikt. En annen grunn til å velge Material Design er at det er utviklet av Google og er svært utbredt [37]. Derfor er det stor sannsynlighet for at jegerne vil kjenne igjen designet og oppleve færre utfordringer ved bruk.

Videre vurderte gruppen også Apple sine retningslinjer for design; *Human Interface Guidelines* [38]. En av grunnene til at gruppen valgte Material Design ovenfor Apple sine retningslinjer var det at gruppen også skulle definere et webgrensesnitt. Material Design viser til prinsipper innenfor både mobil og web, mens Apples retningslinjer kun gjelder grensesnitt for mobil. Gruppen ønsket at hele produktet ble preget av samme design, og derfor falt valget på Material Design.

Figur 13 viser mobilapplikasjonens grunnleggende layout. Nederst i applikasjonen er navigasjonsmenyen. Denne har fire funksjoner: å opprette en ny rapport, se oversikt over rapporter, endre en rapport og justere innstillinger (eventuelt logge ut). Bruk av kart og plotting av småvilt er tilgjengelig gjennom nevnt funksjonalitet.



Figur 13: Mobilapplikasjonens grunnleggende layout

Øverst i applikasjonen er en tittel (Rapporter) som korresponderer med valgt tab i navigasjonsmenyen. Den gir brukeren en ekstra bekreftelse på hva som kan gjøres med innlastet innhold.

Et annet element som går igjen i store deler av applikasjonen er det nest øverste feltet som sier Gjerstad i figur 13. Så lenge brukeren ikke er i innstillinger vises dette feltet som en bekreftelse og slags filter for applikasjonen. Dette er brukerens aktive jaktområde. Det er dette området en ny rapport blir assosiert med, og det er kun rapporter for valgt jaktområde som vises i historikken. Videre kan brukeren til enhver tid endre dette området i Rapporter-tabben.

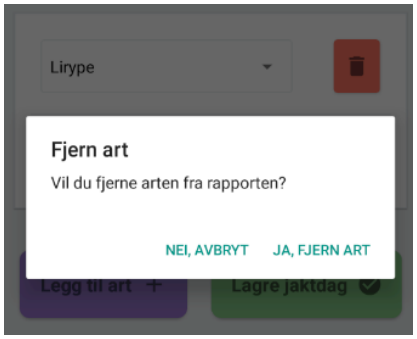
Fargepallett

Videre valgte gruppen ut mobilapplikasjonens hovedfarger med hensyn til både svaksynte og brukere med fargeblindhet. Ettersom gruppen ønsket å benytte konsepter fra Material Design valgte vi å bruke deres fargeverktøy for å definere applikasjonens farger: Color Tool [39]. Utvalgt fargepallett er vist i figur 14 og består av fargene hvit, grønn og lilla. Disse fargene er valgt ut basert på informasjon om fargeblindhet i en artikkel skrevet av Robyn Collinge [40]. I følge denne artikkelen er det stor sannsynlighet for at dette er en god fargekombinasjon. Likevel anbefales det sterkt å også benytte ulike mønster for hver farge i tillegg til ikoner. Av dette har gruppen ofte benyttet ikoner (spesielt på knapper) for å symbolisere hva slags handling som er mulig.

Figur 14: Fargepallett for mobilapplikasjonen

Feedback

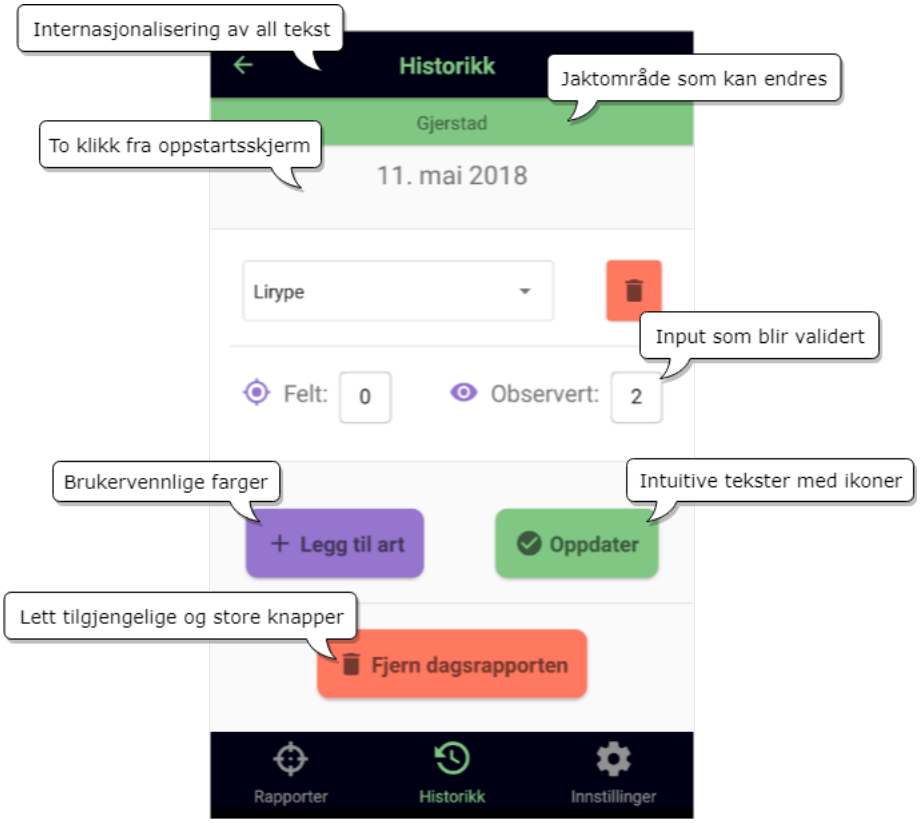
Videre sier Material Design noe om hvordan tekster bør formuleres for brukeren. Det er viktig å være konsis og direkte i tillegg til å bruke et enkelt språk. Tekst på knapper o.l. skal være intuitive og beskrivende. Dette var noe eksperten på interaksjonsdesign også la vekt på under brukertesten med han (les mer om resultatene i seksjon 8.1). Derfor har gruppen hatt mye fokus på dette. I figur 15 er det et eksempel fra applikasjonen som illustrerer feedback.



Figur 15: Eksempel på feedback i mobilapplikasjonen

Universell utforming

Som nevnt tidligere har det vært spesielt viktig for jegeren at klienten har et grensesnitt som er universellt utformet. Jegergruppen består av mennesker i flere ulike aldersgrupper, men det er velkjent at mange av dem er godt voksne. Derfor er det noen spesielle hensyn som mobilapplikasjonen måtte ivareta. Spesielt synsutfordringer og motoriske utfordringer var svært relevante å ta hensyn til. For å oppnå universell utforming viser Difi til syv ulike prinsipper [41]. I tabell 4 er en beskrivelse av de tiltakene gruppen har gjort for å støtte opp under disse prinsippene. Videre er noen av tiltakene også illustrert i figur 16. Figuren inneholder et skjermbilde fra redigering av en rapport.

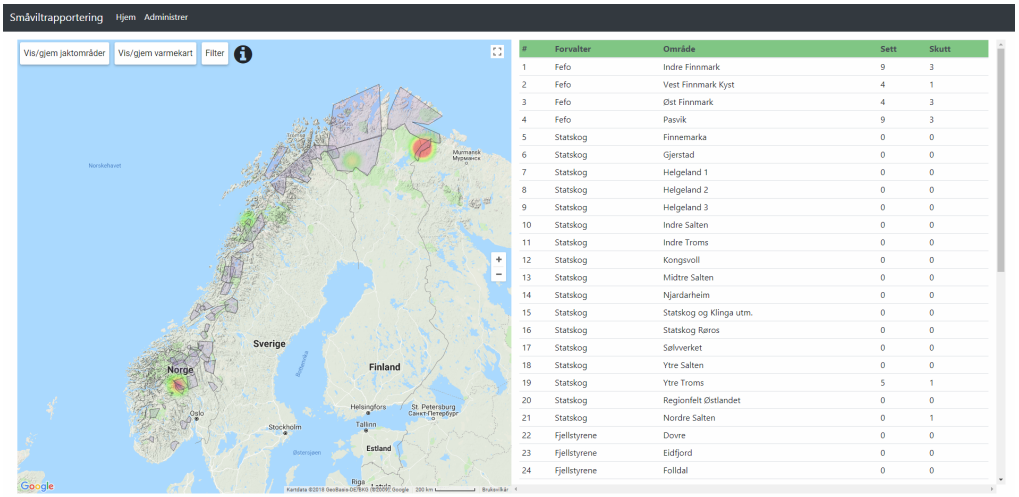


Figur 16: Universell utforming i mobilapplikasjonen

Prinsipp	Beskrivelse av tiltak
Like muligheter for bruk	Dette prinsippet innebærer at ulike mennesker skal ha samme muligheter ved bruk av applikasjonen. Dette har gruppen innført i mobilapplikasjonen ved å sørge for at knapper og tekst har passende størrelser. Dette fokuset sørger blant annet for at personer med motoriske utfordringer kan bruke applikasjonen. I tillegg har gruppen sørget for at de utvalgte fargene fungerer godt også for fargeblinde og svaksynte. Se fargepalletten i seksjon 5.3.
Fleksibel i bruk	For å forsikre at mobilapplikasjonen er så fleksibel som mulig innførte gruppen internasjonalisering. Slik sørget gruppen for at engelsktalende også skulle ha nytte av appen. I tillegg fikk brukeren mulighet til å spesifisere et aktivt jaktområde som til enhver til kan endres om ønskelig.
Enkel og intuitiv i bruk	Et av kravene gruppen satte innenfor brukervennlighet er sterkt knyttet til dette prinsippet (se seksjon 2.4). Dette er et aspekt gruppen har ivaretatt under hele utviklingsprosessen. For å forsikre at applikasjonen oppleves som enkel og intuitiv utførte gruppen flere iterasjoner med prototyper som beskrevet ovenfor. I tillegg utførte gruppen to brukertester med ulike testere for å få mest mulig tilbakemelding på brukervennlighet. Les mer om utfallet av disse testene i seksjon 8.1.
Forståelig informasjon	Gruppen innser at dette prinsippet har blitt ivaretatt nokså ubevisst i løpet av hele prosjektetperioden. Også Material Design har fokus på gode og effektive tilbakemeldinger [37]. Dermed ble det svært naturlig å sørge for at alle tekster og tilbakemeldinger i applikasjonen ble formulert på en mest mulig effektiv og forståelig måte. Et eksempel på dette er vist i figur 15.
Toleranse for feil	Det har vært svært naturlig for gruppen å validere data (for å tolerere feil) på klient-siden slik at brukeren ikke opplever uønskede konsekvenser. I tillegg er det også satt som et krav innen brukervennlighet at klientene skal tolerere feil (se seksjon 2.4). Dette ble innført ved å definere et sett med funksjoner hos klientene som validerer input før input-dataene eventuelt benyttes i API-kall.
Lav fysisk utfordring	Dette prinsippet innebærer at det skal oppleves som lettvint å benytte applikasjonen, og med minimum byrde. For å støtte dette prinsippet har gruppen sørget for at brukeren utfører færrest mulig klikk for å nå satt mål. Derfor er hovedfunksjonaliteten plassert i menyen nederst på skjermen. Videre har nevnte brukertester også vært til stor hjelp og har bidratt til at gruppen kunne forsikre at oppretting av en rapport er en enkel og effektivt oppgave.
Størrelse og plass for tilgang og bruk	Innenfor dette prinsippet er det største tiltaket at de fleste av knappene i mobilapplikasjonen er plassert nederst på skjermen. På denne måten kan nesten en hvilken som helst bruker navigere seg rundt i applikasjonen med én hånd.

Tabell 4: Beskrivelse av hvordan mobilapplikasjonen støtter universell utforming

5.4 Brukergrensesnitt i web-løsningen

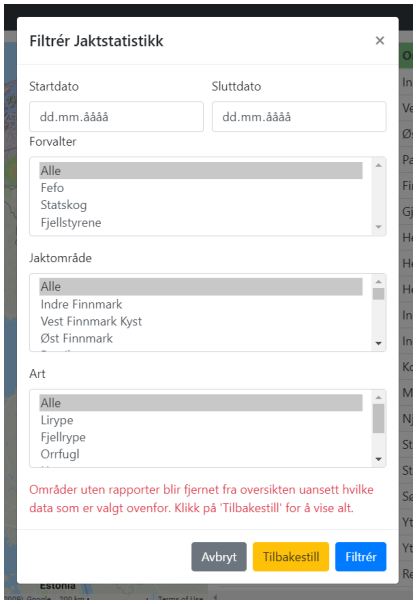


Figur 17: Grafisk brukergrensesnitt for fremvisning av statistikk i web-løsningen

Web-løsningen er designet for enkel tilgang til generell statistikk om dataene som brukere av applikasjonen genererer. Grensesnittet er minimalistisk og benytter samme fargepalett som i mobilapplikasjonen. Slik oppnår brukerne lettere en relasjon mellom dataene i web-løsningen og det som rapporteres i mobilapplikasjonen.

Videre benyttes Bootstrap 4 for å skape et rent og responsivt design, slik at grensesnittet fungerer like bra på mobil eller nettbrett som på en datamaskin. For å ytterligere forbedre brukeropplevelsen fikk tabellen på høyre side i grensesnittet (figur 17) en «scrollbar». Dette var nødvendig på grunn av den store mengden med jaktområder. Med denne layout-en vil ikke grensesnittet bevege seg, og brukeren kan se kartet til enhver tid selv om man blar nedover i tabellen.

For å vise kart benyttes Google Maps API-et. Dette gjør at man blir presentert med et kart-grensesnitt de aller fleste er kjent med. I tillegg gir det gode muligheter for å presentere informasjonen gruppen vil vise, siden API-et inneholder en visualiserings-modul.



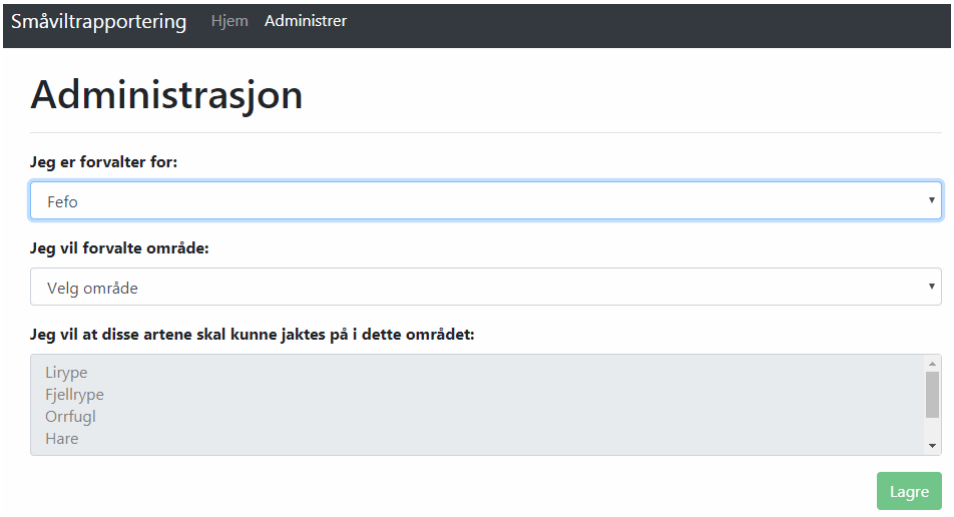
Figur 18: Grafisk brukergrensesnitt for filter i web-løsningen

Innenfor kartet vises polygoner i lilla farge. Disse illustrerer de jaktområdene som har jeger-genererte rapporter. Rapportene og dataene i disse er vist på kartet i et heat map. Videre har brukeren mulighet til å vise/gjemme jaktområdene (polygonene), vise/gjemme varmekart og filtrere infor-

masjonen som vises i kartet. Les mer om hvordan kartet, heat map og polygoner ble implementert i seksjon 6.3.1.

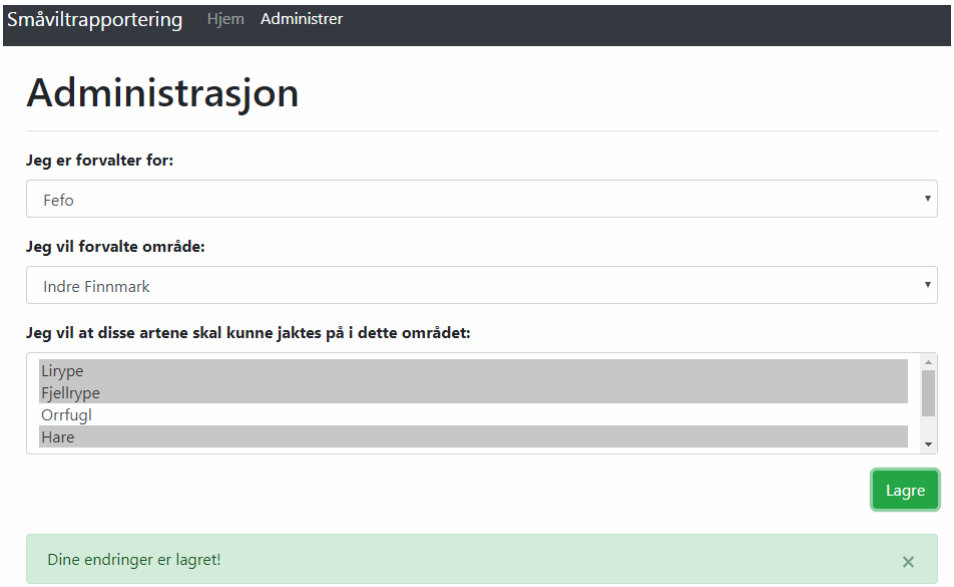
Grensesnittet til filteret er vist i figur 18. Filteret gir forvalteren mulighet til å velge start- og sluttdato, forvaltere, jaktområder og art. På denne måten kan forvalter velge å se informasjon om kun sine områder, noe produkteier mener vil gi produktet stor markedsverdi.

I tillegg valgte gruppen å implementere et grensesnitt for administrasjon av jaktområder. Denne funksjonen gruppen lar en administrator velge hvikle arter det skal rapporteres på i et valgt jaktområde. Dette grensesnittet er vist i figur 19.



Figur 19: Grafisk brukergrensesnitt for administrasjon i web-løsningen, Trinn 2

Det er benyttet en trinnvis tilnærming for å gjøre grensesnittet så intuitivt som mulig. Det vil si at administrator må fullføre «trinn 1» før han/hun kan starte på «trinn 2». Dette ble gjort ved å deaktivere de påfølgende feltene, og først når en verdi blir valgt vil det neste feltet bli aktivert, som vist i figur 19. Valgmulighetene blir dynamisk oppdatert basert på tidligere valg for å minimere gjentagende handlinger. Tidligere definerte arter vises ved at de velges som standard som illustrert i figur 20. Administrator kan så velge eller fjerne arter som ønsket ved å klikke på dem. Deretter lagres endringene ved å klikke «Lagre». Administrator vil da bli presentert med en tilbakemelding som forteller om endringene ble lagret eller ikke. Dette er også vist i figur 20. Hvordan dette ble implementert er beskrevet i seksjon 6.3.



Figur 20: Grafisk brukergrensesnitt for administrasjon i web-løsningen, Trinn 4

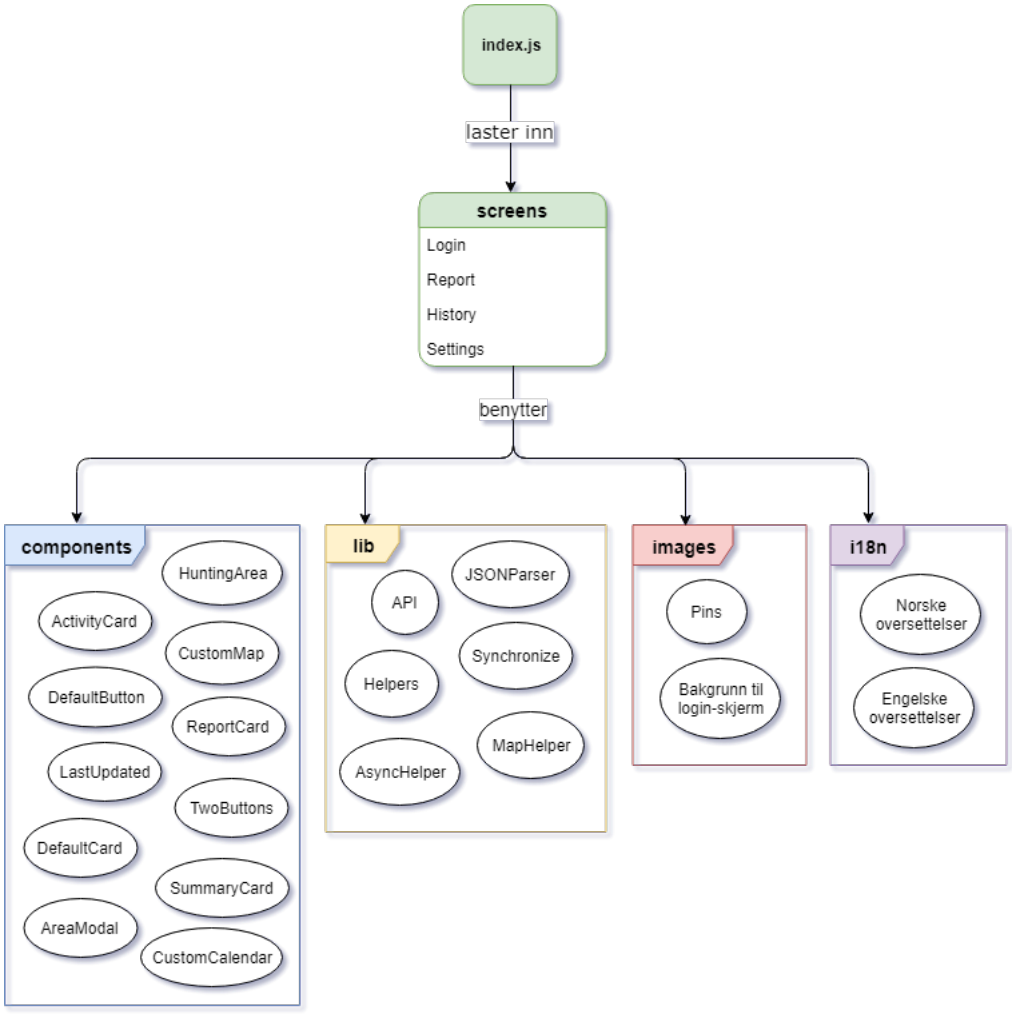
6 Implementering

Her vil gruppen diskutere hvordan de ulike delene av produktet ble strukturert og implementert. Underveis hadde gruppen mye fokus på å følge retningslinjene til de ulike teknologiene som ble benyttet. Gruppen så blant annet på hvordan et React Native-prosjekt burde implementeres og hvordan det anbefales å strukturere et RESTful API.

6.1 Mobilapplikasjonen

På mobilapplikasjonen ble strukturen definert basert på et sett med eksempler funnet på nett [42] [43]. Gruppen valgte å undersøke en del React Native eksempel-applikasjoner og så på hvordan disse var strukturert. Mobilapplikasjonens struktur er illustrert i figur 21. Strukturen er i stor grad lik mange av eksemplene gruppen fant. Slik er mobilapplikasjonens filstruktur:

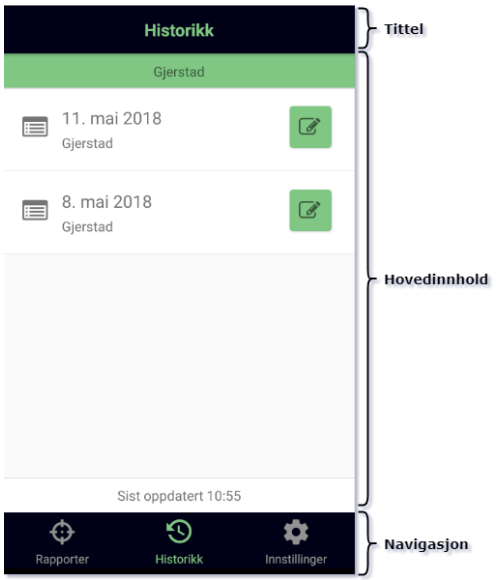
- **index.js** er applikasjonens entrypoint. Den laster inn hovedinnholdet basert på om brukeren er logget inn fra før eller ikke. Dersom brukeren har logget inn en gang tidligere blir han/hun logget inn automatisk neste gang.
- **auth.js** håndterer innlogging og autentisering. Det er denne filen som lagrer data om brukeren i AsyncStorage dersom innlogging er vellykket. Filen er ikke illustrert i figur 21.
- **screens** inneholder alle klasser som definerer hovedinnholdet i applikasjonen. F.eks. «Report»-mappen i screens definerer et sett med klasser som lar brukeren opprette en ny rapport. Hver klasse definerer hvordan ett spesifikt skjermbilde skal se ut og en rekke funksjoner som blant annet håndterer bruker-input.
- **components** inneholder alle egendefinerte komponenter som er benyttet i applikasjonen. Dette er typisk elementer som benyttes i flere screens i applikasjonen. Les mer om komponenter i seksjon 6.1.2.
- **lib** inneholder de forskjellige bibliotekene som gruppen har definert selv. Ett av disse er et bibliotek som håndterer alle forespørsler til API-et, kalt *API.js*. Les mer om disse bibliotekene i seksjon 6.1.3.
- **images** er en mappe som holder på alle bilder benyttet i applikasjonen. Et av disse er bakgrunnsbildet som vises på login-skjermen.
- **i18n** er internasjonalisering for applikasjonen og inneholder oversettelser for alle strenger.
- **config** er ikke illustrert i figur 21, men inneholder en del filer som definerer konfigurasjon for hele applikasjonen. Dette er blant annet globale fonter, farger og oppsett av navigasjon. Les mer om navigasjonen i seksjon 6.1.1.



Figur 21: Strukturen i mobilapplikasjonen

6.1.1 Navigasjon

For å implementere navigasjon har gruppen benyttet en tredjeparts-pakke kalt react-navigation [44]. Fra denne pakken ble komponentene *StackNavigator* og *TabNavigator* benyttet for å implementere navigasjon.



Figur 22: Mobilapplikasjonens navigasjon

Mobile applikasjoner består ofte av flere screens (skjermer med ulikt innhold). Screens fungerer vanligvis på samme måte som en stack der en screen legges på toppen av nåværende screen. I React Native utfører komponenten *StackNavigation* denne funksjonaliteten. *StackNavigation* gjør det mulig å navigere seg mellom ulike screens og håndtere rekkefølgen i stack-en. For å navigere mellom ulike screens benyttes hovedsakelig en *TabNavigator*-komponent. Denne komponenten består av tre *StackNavigator*-komponenter som tilsvarer hver tab i menyen (se figur 22). Dette sørger for at hver tab i navigasjons-menyen kan inneholde et komplekst sett med screens.

I *TabNavigator*-komponenten er det standard at tittelen og navigasjons-menyen er knyttet til hverandre. I figur 22 er det synlig at tittelen «Rapporter» korresponderer med valgt tab. Videre har tittelen en tilbakepil som fjerner øverste screen i stack-en og viser dermed forrige screen. Basert på valg av tab oppdateres hovedinnholdet (også markert i figur 22) i applikasjonen.

Ettersom React Native er et kryssplattform-rammeverk er det nødvendig å håndtere spesialtilfeller innen navigasjon. Navigasjon i Android er noe annerledes fra navigasjon i iOS; Android-enheter har gjerne en «fysisk» tilbakeknapp som iOS-enheter ikke har. Et resultat av dette er at Android-enheter fritt kan navigere seg nedover i stack-en. Av den grunn valgte gruppen å overskrive funksjonen *goBack()* som er spesifikk for Android. Funksjonaliteten som ble implementert forsikrer at brukeren ikke kan navigere seg tilbake til screens i situasjoner der dette ikke er ønskelig. Et eksempel på en slik situasjon er ved lagring av en ny rapport. Etter at brukeren har trykket «Lagre» navigeres brukeren til neste screen som viser en oppsummering. I dette tilfellet er det ikke ønskelig at brukeren kan navigere seg tilbake til forrige screen og lagre en eksisterende rapport på nytt. Dette ble løst ved å implementere at screen stack-en fjernes i *goBack()* og at hjemsiden legges på toppen av stack-en.

6.1.2 Komponenter

En av de største fordelene med React Native er rammeverkets bruk av komponenter. En utvikler kan definere egne komponenter og/eller benytte andre personers ferdiglagde komponenter. Utenom dette er også hele rammeverket bygd på et sett med definerte komponenter.

Det er gunstig å benytte komponenter av flere grunner; applikasjonen oppnår en svært løs kopling ettersom det er lettvint å bytte ut komponenter. I tillegg blir koden enklere å lese både underveis i prosessen, men også for utviklere som eventuelt videreutvikler applikasjonen. Utenom dette forenkles også en en del av utviklingsprosessen. Dersom det finnes elementer som gjentas flere steder kan en egendefinert komponent (som defineres én gang) benyttes og gjenbrukes. Dette er svært tidsbesparende for utvikleren dersom man tidlig definerer noen komponenter man vet man vil få bruk for.

I mobilapplikasjonen ble det utviklet et sett med egendefinerte komponenter som støttet opp under god kodekvalitet og leslighet av koden. Et eksempel på dette er en midtstilt knapp som gruppen trengte flere steder i koden, men med ulik tekst og eventuelt et unikt ikon. Dette var et typisk tilfelle der det var naturlig å benytte en egendefinert komponent. Listing 6.1 viser hvordan koden for en knapp så ut før det ble benyttet en egen komponent.

```
1  <TouchableOpacity
2      onPress={this._startReport}
3      style={styles.nextButton}>
4
5      <View style={styles.nextArea}>
6          <Text style={fonts.buttonText}>
7              {I18n.t('reportFor')} {"\n"} { this._printDate() }
8          </Text>
9      </View>
10 </TouchableOpacity>
```

Listing 6.1: En knapp før implementasjon av DefaultButton

Listing 6.2 viser hvordan koden ble etter at gruppen implementerte en egen komponent for en knapp kalt *DefaultButton*. Her er det tydelige forbedringer: antall linjer med kode ble mer enn halvert og lesligheten ble økt kraftig. I tillegg ble *DefaultButton* implementert med en ekstra egenskap kalt *icon*. Dersom gruppen trengte trengte et ikon til knappen kunne utvikleren inkludere feltet *icon* og sende med ønsket ikon-komponent som parameter.

```
1  <DefaultButton
2      text={I18n.t('reportFor') + "\n" + this._printDate()}
3      color={colors.active}
4      onPress={this._checkReportDate} />
```

Listing 6.2: En knapp etter implementasjon av DefaultButton

Utenom komponenten *DefaultButton* definerte gruppen 10 andre komponenter (illustrert i figur 21) som forbedret koden betraktelig. En annen komponent som ble implementert er *CustomMap*. Denne komponenten reduserte også antall kodelinjer (for visning av kart) med nesten 50 %. Koden var originalt på 27 linjer, men ble redusert til 14 linjer med kode. Kartet ble definert som en egen komponent ettersom det ble benyttet to

steder i koden; ved visning kart ved oppretting av en ny rapport og visning av kart ved redigering av en rapport.

6.1.3 Biblioteker

Som nevnt innledningsvis i dette kapittelet definerte gruppen et sett med hjelpebiblioteker. Disse ble definert for å gjøre koden mer oversiktlig, bedre strukturert og mer leslig. Tabell 5 viser en oversikt over de bibliotekene (klassene) gruppen implementerte og hvordan de ble benyttet.

Bibliotek	**Ansvarsområde**
API	Biblioteket håndterer alle kall til API-et. I tillegg definerer klassen et sett med funksjoner som beskriver og returnerer strukturen til *body* i API-forespørslene. Disse strukturene var ofte på 15-20 linjer med kode. Derfor ble koden mye mer leslig etter at dette ble skilt ut i en egen klasse.
AsyncHelper	Denne klassen håndterer alle interaksjoner med AsyncStorage. Klassen består av funksjoner som henter ut, oppdaterer og erstatter elementer som lagres lokalt (primært rapporter).
Helpers	Dette er et bibliotek som definerer generelle funksjoner som er nyttige flere steder i applikasjonen. Eksempler på funksjoner i denne klassen er funksjoner som returnerer en gitt dato på et annet format.
JSONParser	Ettersom applikasjonen håndterer en del JSON-data implementerte gruppen et bibliotek med relevante hjelpefunksjoner. Klassen definerer et sett med funksjoner som søker gjennom mottatt JSON-data og returnerer et resultat. Et eksempel er en funksjon som finner og returnerer id-en til en art når art-navnet sendes med som parameter.
MapHelper	Under utviklingen av kart ble det raskt tydelig at gruppen trengte et hjelpebibliotek. Det var store mengder med kode som kun håndterte kartdata, og derfor ble koden mer oversiktlig da dette ble skilt ut i en egen klasse. I tillegg var det en fordel ettersom mange av de samme funksjonene trengtes flere steder i koden (ved ny rapport òg redigering av rapport). Funksjonene i *MapHelper* henter blant annet ut aktuelt polygon til kartet og markeringer.
Synchronize	Denne klassen inneholder et sett med funksjoner som håndterer alt av synkronisering i applikasjonen; hovedsakelig synkronisering av rapporter. Dette var nødvendig ettersom det skulle være mulig å benytte applikasjonen offline. Les mer om hvordan dette ble implementert i seksjon 6.1.5.

Tabell 5: Forklaring av implementerte biblioteker i mobilapplikasjonen

6.1.4 Kart

I mobilapplikasjonen implementerte gruppen også muligheten til å markere dagens fangst på et kart. Dette gjør brukeren først og fremst under rapporteringen. Disse markeringene gjelder felte dyr ettersom det er disse dataene som er verdifulle for produkteier. Dersom jegeren rapporterer at han/hun har skutt ett eller flere dyr (og brukeren har internett) vil kartet vises. Dermed får jegeren mulighet til å markere hvor på kartet dyret/dyrene ble skutt. Dette er vist i figur 23. I tillegg kan også en bruker redigere disse posisjonene i etterkant ved redigering av en tidligere rapport.

For å implementere kart valgte gruppen å benytte en tredjeparts-pakke kalt react-native-maps [45]. Dette er en velkjent pakke som ofte benyttes i React Native-applikasjoner som implementerer kart. Pakken gir utvikleren muligheten til å definere om alle enheter skal benytte Google Maps eller om iOS-enheter skal benytte MapKit. Gruppen valgte å sette Google Maps som leverandør for alle enheter for å forsikre seg en større grad av forutsigbarhet. Videre er det helt gratis å benytte Google Maps API-et for både Android- òg iOS-plattformene [46]. Utenom dette er det verdt å nevne at prisene endres etter 11. juni 2018, men det vil fremdeles være gratis å bruke kart for Android og iOS [47].

Figur 23: Markeringer plottet på kartet under rapportering

MapView

Hovedkomponenten til react-native-maps-pakken er MapView. Det var denne komponenten som tillater en klient å se et kart fra Google Maps. I denne komponenten måtte gruppen definere hva som skulle settes som kartets senter og en leverandør. Videre var det i første omgang naturlig å sette valgt jaktområdet sitt ca. senter til å være kartets senter.

Tile Overlay

Videre er det viktig at jegeren klarer å kjenne seg igjen når han/hun skal sette ut sine markeringer på kartet. Derfor ba produkteier om at kartet viste lokale stedsnavn, veinavn o.l. Dette ble implementert ved å benytte react-native-maps sin «tile overlay»-komponent. Ved å bruke denne komponenten kunne gruppen legge et annet kart på toppen av det som Google Maps viser som standard. Gruppen fikk tips fra produkteier om å benytte Kartverket sine kart. Dermed ble «Grunnkart»-typen valgt ut fra Kartverket sine sider [48]. Figur 23 viser et norskt grunnkart over Gjerstad.

Polygon

En utfordring gruppen hadde med kart var å sørge for at en markør ble plassert på innsiden av det valgte jaktområdet. For å løse utfordringen ble gruppen enig om å implementere polygoner for hvert jaktområde. Dette var en mindre jobb for backend og en større jobb for frontend. For å implementere polygoner brukte gruppen «Polygon»-komponenten fra react-native-maps. Komponenten trengte først og fremst en liste med bredde- og høydegrader for å definere polygonet. Videre måtte også fyll-fargen defineres og grad av gjennomsiktighet på polygonet.

Da selve polygonene for hvert jaktområde ble implementert var neste steg å definere en algoritme som skulle sjekke en markerings posisjon opp mot polygonet. Gruppen tenkte på dette tidspunktet at det var viktig å ikke bruke tid på å «finne opp hjulet på nytt». Derfor så vi på eksisterende løsninger der det ble sjekket om et punkt lå på innsiden av et gitt polygon eller ikke. Løsningen som ble valgt er vist i listing 6.3. Funksjonen er hentet fra en React Native-pakke kalt «point-in-polygon» [49]. Personen bak denne pakken har definert algoritmen basert på W. Randolph Franklin sin forklaring på hvordan man finner ut om et punkt ligger på innsiden av et polygon [50].

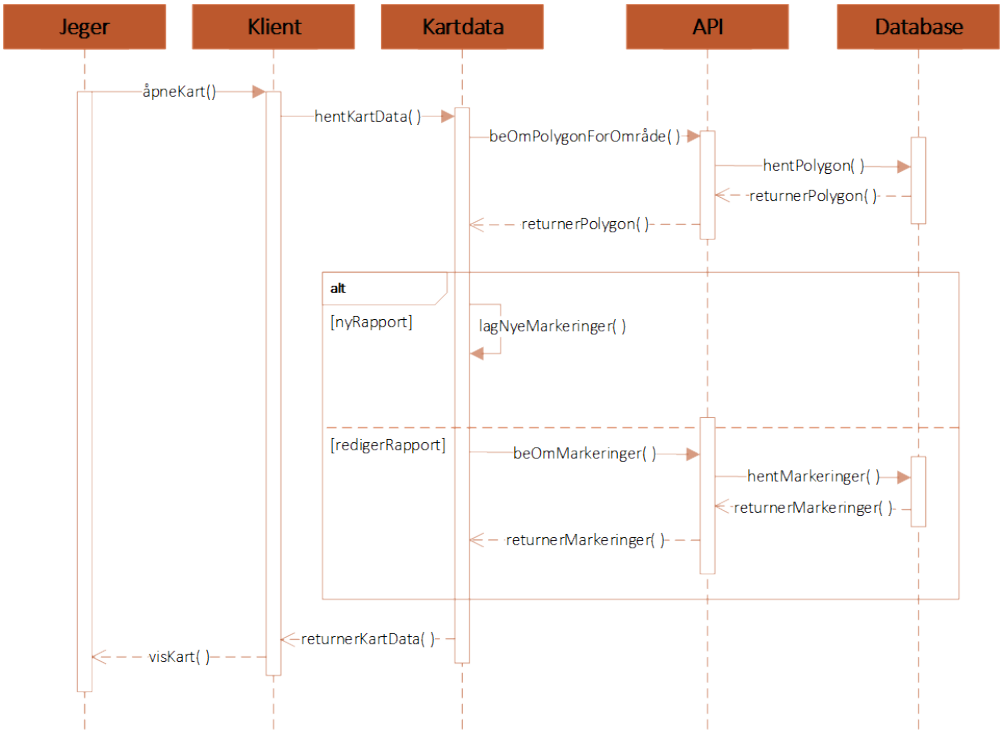
```
1  static markerIsInPolygon(point, polygon) {
2      let x = point.coordinate.latitude;
3      let y = point.coordinate.longitude;
4      let inside = false;
5
6      for (let i = 0, j = polygon.length - 1; i < polygon.length; j = i++) {
7          let xi = polygon[i].latitude, yi = polygon[i].longitude;
8          let xj = polygon[j].latitude, yj = polygon[j].longitude;
9
10         let intersect = ((yi > y) != (yj > y)) && (x < (xj - xi) * (y - yi) / (
               yj - yi) + xi);
11
12         if (intersect) {
13             inside = !inside;
14         }
15     }
16
17     return inside;
18 }
```

Listing 6.3: Sjekker om et punkt ligger på innsiden av et polygon.

Etter implementasjon av denne funksjonen kunne gruppen sørge for at brukeren fikk en advarsel om han/hun plasserte en markering på utsiden av polygonet. I tillegg ble det også lagt inn en sjekk på alle markeringers posisjon som kjøres hver gang brukeren prøver å lagre markeringene. Dermed får ikke brukeren lov til å lagre ugyldige data.

Dataflyt

Dataflyten som oppstår ved innlasting av kartet er illustrert i figur 24. Sekvensdiagrammet viser at en jeger prøver å åpne kartet. Deretter prøver applikasjonen å hente ut et objekt med kartdata. Ettersom bruk av kart kun er tillatt når brukeren har internettforbindelse er dette brukt som utgangspunkt for sekvensdiagrammet. Videre ber Kartdata-objektet API-et om å hente det relevante polygonet. API-et henter så dataene fra databasen før dataene blir returnert til Kartdata-objektet. Videre gjøres det en sjekk på om forespørselen er relatert til en ny rapport eller redigering av en eksisterende rapport. Dersom rapporten er ny utføres et lokalt funksjonskall som lager nye markeringer. Dersom brukeren vil redigere en rapport sendes en ny forespørsel til databasen via API-et om å hente markeringene for rapporten. Til slutt returneres et kartdata-objekt til klienten med all nødvendig informasjon. Kartet kan lastes inn og vises for brukeren.



Figur 24: Dataflyten ved innlasting av kart i mobilapplikasjonen

Jegerens posisjon

Da polygoner ble implementert var det i første omgang naturlig å benytte polygonets senter som kartets senter under rapportering. Videre tenkte gruppen på eget initiativ at det ville være fordelaktig å bruke jegerens posisjon som kartets senter. Dette ville være lønnsomt dersom jegeren fremdeles befant seg i jaktområdet under rapporteringen. Derfor implementerte gruppen en sjekk på jegerens posisjon ved innlasting av kartet. Dersom jegeren tillater å oppgi hans/hennes posisjon vil denne posisjonen sjekkes opp mot jaktområdets polygon ved hjelp av nevnt algoritme. Dersom jegeren befinner seg på innsiden av polygonet benyttes jegerens posisjon som senter. Ellers benyttes polygonets senter.

Listing 6.4 viser hvordan posisjonen hentes ved å benytte et globalt element i React Native kalt «navigator.geolocation». Koden viser at utvikleren ber dette elementet om nåværende posisjon. Dersom brukeren tillater applikasjonen å hente GPS-posisjon så lagres koordinatene i et egendefinert objekt kalt *gpsLocation*. Deretter sjekkes det om denne posisjonen er på innsiden av polygonet. Dersom posisjonen er på innsiden skal markeringene initialiseres på denne posisjonen i stedet. Ellers er polygonets senter standard senter for kartet og nye markeringer.

```
navigator.geolocation.getCurrentPosition(
    (position) => {
        let gpsLocation = {
            coordinate: {
                latitude: position.coords.latitude,
                longitude: position.coords.longitude,
            }
        };

        if (MapHelper.markerIsInPolygon(gpsLocation, this.state.polygon)) {
            this._updateMarkerCoordinates(gpsLocation);
        }
    },
    () => {
        Helpers.simpleFeedback(I18n.t('failTitle'), I18n.t('gpsFailDesc'));
    },
    {
        enableHighAccuracy: true,
        timeout: 20000,
        maximumAge: 1000
    },
);
```

Listing 6.4: Henter ut klientens GPS-lokasjon i mobilapplikasjonen

Fullstendig kode

Komponentene som viser selve kartet, polygon, tile overlay og markeringer er ikke på mer enn 25 linjer med kode. Disse er vist i listing 6.5. Som nevnt er det MapView-komponenten som viser selve kartet. Rett under denne komponenten defineres blant annet Google Maps som leverandør og verdiene for kartets senter blir satt i *initialRegion*. På innsiden av MapView-komponenten defineres det først en UrlTile-komponent. Denne definerer nevnt *tile overlay*. I denne komponenten oppgis det en url til ønsket overlay. I tillegg defineres *zIndex* til å være -3. Dette gjøres for å være sikker på at polygonet og markeringene legges på toppen av *tile overlay*. Under UrlTile defineres MapView.Polygon. Dette er komponenten som definerer polygonet til det valgte jaktområdet. Der settes polygonet, fargen på polygonet og polygonets ramme. Til slutt skrives alle markeringer ut i kartet. Markeringene sendes med som parameter ettersom disse varierer basert på om brukeren rapporterer for første gang eller redigerer en eksisterende rapport.

```
<MapView
    style={styles.map}
    initialRegion={{
        latitude: Number(centerLatitude),
        longitude: Number(centerLongitude),
        latitudeDelta: 0.0922,
        longitudeDelta: 0.0421,
    }}
    loadingEnabled={true}
```

```
10      provider={"google"}>
11
12      <UrlTile
13          urlTemplate={settings.TILE_OVERLAY_URL}
14          zIndex={-3}
15      />
16
17      <MapView.Polygon
18          coordinates={polygon}
19          strokeColor="#F00"
20          fillColor="rgba(255,0,0,0.1)"
21          strokeWidth={1}/>
22
23      { markers }
24
25  </MapView>
```

Listing 6.5: Alle involverte kart-komponenter i mobilapplikasjonen

Redigering av markeringer

Som nevnt implementerte gruppen også muligheten til å redigere markeringer på tidligere rapporter. I dette scenariet var det en del ulike utfordringer og hensyn som måtte tas. Tabell 6 viser hvilke scenarier gruppen måtte ta hensyn til og hvordan det ble løst. Konteksten til alle scenariene er at en bruker redigerer en eksisterende rapport.

Scenario	Løsning
Brukeren har registrert felte dyr og markerte disse på kartet under rapportering.	Denne utfordringen ble løst ved å hente inn alle registrerte markeringer for en rapport inn i kartet. De aktuelle markeringene ble navngitt med den arten de tilhørte. Deretter kunne brukeren flytte på markeringene som ønsket og oppdatere disse.
Brukeren har registrert felte dyr, men plasserte ingen markeringer under rapportering.	På grunn av dette tilfellet ble det inn en sjekk i koden på hvor mange markeringer som lå i databasen i forhold til antall felte dyr for én art. Dersom det ikke lå noen registrerte markeringer i databasen ble nye markeringer lagt ut for den/de aktuelle arten(e). Disse ble navngitt med ordet «Ny» foran artsnavnet i kartet. Deretter kunne brukeren endre på tidligere markeringer i tillegg til å legge til nye.
Brukeren fjerner en art fra rapporten der han/hun har registrert markeringer på kartet.	Løsningen på dette var svært enkel. Det ble løst ved at det ble satt opp en fremmednøkkel mellom en aktivitet (bestående av en art og antall skutt og sett) og markeringer for gitt aktivitet i API-et. Da dette ble implementert ble aktuelle markeringer fjernet automatisk fra databasen dersom brukeren fjernet en eksisterende aktivitet.
Brukeren legger til en ny art til rapporten som han/hun har felt.	Løsningen på dette scenariet er identisk til scenariet der brukeren har felt dyr, men registrerte ingen markeringer på kartet. Dersom det ikke finnes markeringer i databasen opprettes nye markeringer for den/de aktuelle arten(e).
Brukeren endrer på antallet felte dyr på en registrert art i rapporten	Dette var et av de mest utfordrende scenariene som måtte håndteres. Følgende ble implementert: dersom brukeren endrer på antall felte dyr vil alle markeringer for denne arten fjernes og brukeren blir opplyst om dette. Gruppen mente at dette var en passende løsning dersom brukeren satte antall skutt til å være færre enn originalt. I dette tilfellet ville det være umulig å vite hvilken markering som skulle fjernes. Dersom brukeren heller øker antall skutt er dette en mindre god løsning. Det føles uheldig at brukeren må legge ut alle markeringene på nytt bare fordi han/hun skal legge til en ekstra. Derfor ville det vært mer gunstig dersom dette ble implementert på en annen måte. En mulighet hadde vært å differensiere mellom økning og minking av antall skutt. Dersom det økes kunne heller den/de nye arten(e) initialiserer på midten av kartet med nøkkelordet «Ny» foran artsnavnet.

Tabell 6: Ulike aspekter ved redigering av markeringer på kart

6.1.5 Online- og offlinemodus

Applikasjonen hadde et krav om å kunne bli benyttet uten tilgang til internett. Dette er på grunn av at jakt ofte foregår ute i felt, hvor tilgang på internett og generell mobildekning kan være begrenset. Muligheten til å benytte applikasjonen med og uten internett øker fleksibiliteten for jegeren fordi applikasjonen kan benyttes til enhver tid. For at rapportering skal fungere uten internett så må all bruker-generert data lagres lokalt på enheten når internett ikke er tilgjengelig, og må senere synkroniseres med databasen i backend så raskt enheten får internettforbindelse. Denne oppgaven viste seg å være langt større enn det gruppen forutså.

Omfanget av støttet offline-funksjonalitet var hovedsakelig å lage nye rapporter i tillegg til å endre og oppdatere eksisterende rapporter. Ikke alle deler av applikasjonen fungerer offline. Brukeren har ikke muligheten til å logge inn i applikasjonen uten en internettforbindelse, og derfor forutsettes det at brukeren har logget inn før han/hun kan benytte applikasjonen. Det er ikke mulig å benytte kart-funksjonalitet offline fordi Google sitt regelverk vedrørende Google Maps forteller at nedlasting og/eller caching av deres kartdata ikke er lovlig utenfor deres egne tjenester [51].

For å lagre data lokalt på enheten benyttet gruppen en komponent fra React Native kalt AsyncStorage. All data i AsyncStorage lagres i nøkkel/verdi-par som tekststrenger. Alle rapporter i applikasjonen håndteres og lagres som JSON-objekter. For å kunne lagre JSON-objekter i AsyncStorage måtte JavaScript sin innebygde *JSON.stringify(parameter)*-funksjon benyttes. Ettersom nøkkel/verdi-par kun tillater én verdi per nøkkel så måtte alle rapport-objektene lagres på én nøkkel. Gruppen opprettet derfor en liste som inneholder alle rapportene som lagres på nøkkelen «svrallreports».

Ideen er at en rapport inneholder all data for en gitt jaktdag og at en rapport består av flere aktiviteter. En aktivitet inneholder data om hvilken art som er skutt og/eller observert, antall og posisjonsdata.

For å holde styr på dataene som er synkronisert med databasen valgte gruppen å legge til et status-attributt på rapport- og på aktivitet-objekter. De ulike statusene er: *NEW*, *UPDATED*, *ORIGINAL* og *DELETED*. Disse brukes som indikatorer på en nøvendig handling. Tabell 7 viser en oversikt over hva de ulike statusene indikerer.

Status	Forklaring
NEW	Dette betyr for en rapport at den er nylig opprettet og at den ikke finnes i databasen enda. Det innebærer at ingen synkronisering har blitt utført. For en aktivitet betyr dette at det nylig har blitt opprettet en ny aktivitet (gjelder uansett om rapporten er NEW, UPDATED eller ORIGINAL).
UPDATED	Dette betyr at rapporten allerede ekisterer på backend, men at deler av rapporten har blitt oppdatert. Enten så har eksisterende data blitt endret, nye aktiviteter har blitt lagt til eller så har data blitt slettet. En aktivitet har kun denne statusen dersom selve aktiviteten har blitt endret.
ORIGINAL	Dette betyr at det ikke har skjedd noen lokale endringer med rapporten og at den er synkronisert. Det samme gjelder for en aktivitet.
DELETED	Dette betyr at brukeren har bedt om å få denne rapporten slettet, men den er ikke slettet fra databasen enda. Det samme gjelder for en aktivitet.

Tabell 7: Forklaring av statuser i rapport- og aktivitet-objekter i mobilapplikasjonen

Strukturen til et rapport-objekt

Listing 6.6 viser strukteren til en rapport i JSON. I tillegg er den beskrevet under:

- Første *key* på rapport er et tall som inkrementeres med en for hver nye rapport, og det samme gjelder *key* for en aktivitet. Disse *key*-ene blir oppdatert med en ny id gitt fra API-et når dataene blir synkronisert.
- *date*-attributtet inneholder datoen for den aktuelle rapporten.
- *area* inneholder jaktområdet rapporten gjelder.
- *status* inneholder informasjon om synkroniseringsstatusen til rapporten. Det samme gjelder *status* for en aktivitet.
- *activities*, en liste, inneholder alle aktivitetene koblet til rapporten.
- *selected* forteller hvilken art aktiviteten gjelder.
- *shot* og *seen* er antall skutt og sett av den aktuelle arten.

```
let report = {
        key: "reportKey",
        date: "reportDate",
        area: "reportArea",
        status: "NEW / UPDATED / ORIGINAL / DELETED",
        activities: [
            {
                key: "someKey",
                selected: "someSpecies",
                shot: "shotCount",
                seen: "seenCount",
                status: "NEW / UPDATED / DELETED / ORIGINAL"
            }
        ]
    }
```

Listing 6.6: Strukturen til et rapport-objekt

Status-kombinasjoner

Det er en del ulike scenarier innen status-kombinasjoner som kan oppstå (totalt 16). Disse er illustrert i tabell 8. For å holde kontroll på synkroniseringsfunksjonalitet opprettet gruppen en egen klasse for hele synkronisering-prosessen mellom AsyncStorage og databasen kalt *Synchronize*. Gruppen implementerte en funksjon i Synchronize.js som henter alle rapporter fra AsyncStorage. Funksjonen sjekker statusen til en rapport og utfører korresponderende funksjonalitet. Hver funksjon i Synchronize.js tar imot et rapport- eller aktivitet-objekt. Deretter forsøker funksjonen å sende forespørsel om oppdatering til API-et og oppdaterer det lokale objektet dersom forespørselen til API-et var vellykket. Ved slutten av hver funksjon så returneres det oppdaterte objektet; dette gjør at hovedfunksjonen mottar alle oppdaterte objekter og kan oppdatere AsyncStorage med en oppdatert liste av rapportobjekter.

Statuser: (N)EW (U)PDATED (O)RIGINAL (D)ELETED
Objekter: (R)EPORT (A)CTIVITY
API forespørsler: (PO)ST (PA)TCH (DE)LETE

Tabell 8: Statuskombinasjoner innen synkronisering av rapporter & aktiviteter og tiltak

Statuskombinasjoner		Tiltak	
Status	**Handling**	**Status**	**Handling**
N R N A	-> Legg til i AsyncStorage	N R N A	-> PO R, PO A
N R U A	-> Legg til i AsyncStorage	N R U A	-> PO R, PO A
N R O A	-> Umulig	N R O A	-> Umulig
N R D A	-> Ikke legg til i AsyncStorage	N R D A	-> Umulig
U R N A	-> Legg til i AsyncStorage	U R N A	-> PO A
U R U A	-> Legg til i AsyncStorage	U R U A	-> PA A
U R O A	-> Legg til i AsyncStorage	U R O A	-> Umulig
U R D A	-> Legg til i AsyncStorage	U R D A	-> DE A
O R N A	-> Umulig	O R N A	-> Umulig
O R U A	-> Umulig	O R U A	-> Umulig
O R O A	-> Legg til i AsyncStorage	O R O A	-> Legg til i AsyncStorage
O R D A	-> Umulig	O R D A	-> Umulig
D R N A	-> Umulig	D R N A	-> DE R
D R U A	-> Umulig	D R U A	-> DE R
D R O A	-> Umulig	D R O A	-> DE R
D R D A	-> Umulig	D R D A	-> DE R

Kode fra Synchronize.js

Listing 6.7 viser switch-statementet (valgkontrollmekanisme i programmering) i hovedfunksjonen til Synchronize.js. Her kjøres funksjonalitet tilsvarende rapportens status. *syncedReport* settes lik funksjonskall fordi alle funksjoner returnerer et oppdatert objekt dersom synkronisering med backend var vellykket. Dersom rapporten har status *ORIGINAL* så vet gruppen at rapporten er uendret og kan legges direkte tilbake i listen av rapporter.

```
switch(report.status) {

    /* Call create report function */
    case globals.NEW: {
        syncedReport = await this._createReport(userId, report);
        break;
    }

    /* Call update report function */
    case globals.UPDATED: {
        syncedReport = await this._updateReport(report);
        break;
    }

    /* Add report directly, unmodified. */
    case globals.ORIGINAL: {
        syncedReport = report;
        break;
    }

    /* Call delete report function */
    case globals.DELETED: {
        syncedReport = await this._deleteReport(report);
        break;
    }

    /* if for some reason status is broken/ not set to correct value earlier */
    default:
        syncedReport = report;
        break;

}   // end switch.
```

Listing 6.7: Synkronisering av rapporter

Pattern

Gruppen brukte patternet *Read/Write data one-to-many* av OutSystems for online-offline synkronisering, men valgte å avvike noe fra patternet for å gjøre det mer tilpasset mobil-applikasjonen. Ideen bak patternet er å: «synkronisere data fra mobile apper som benytter seg av entiteter som bruker en-til-mange relasjoner og som ikke kommer til å ha flere slutt-brukere som modifiserer samme data» [52]. Patternet fungerer på følgende måte:

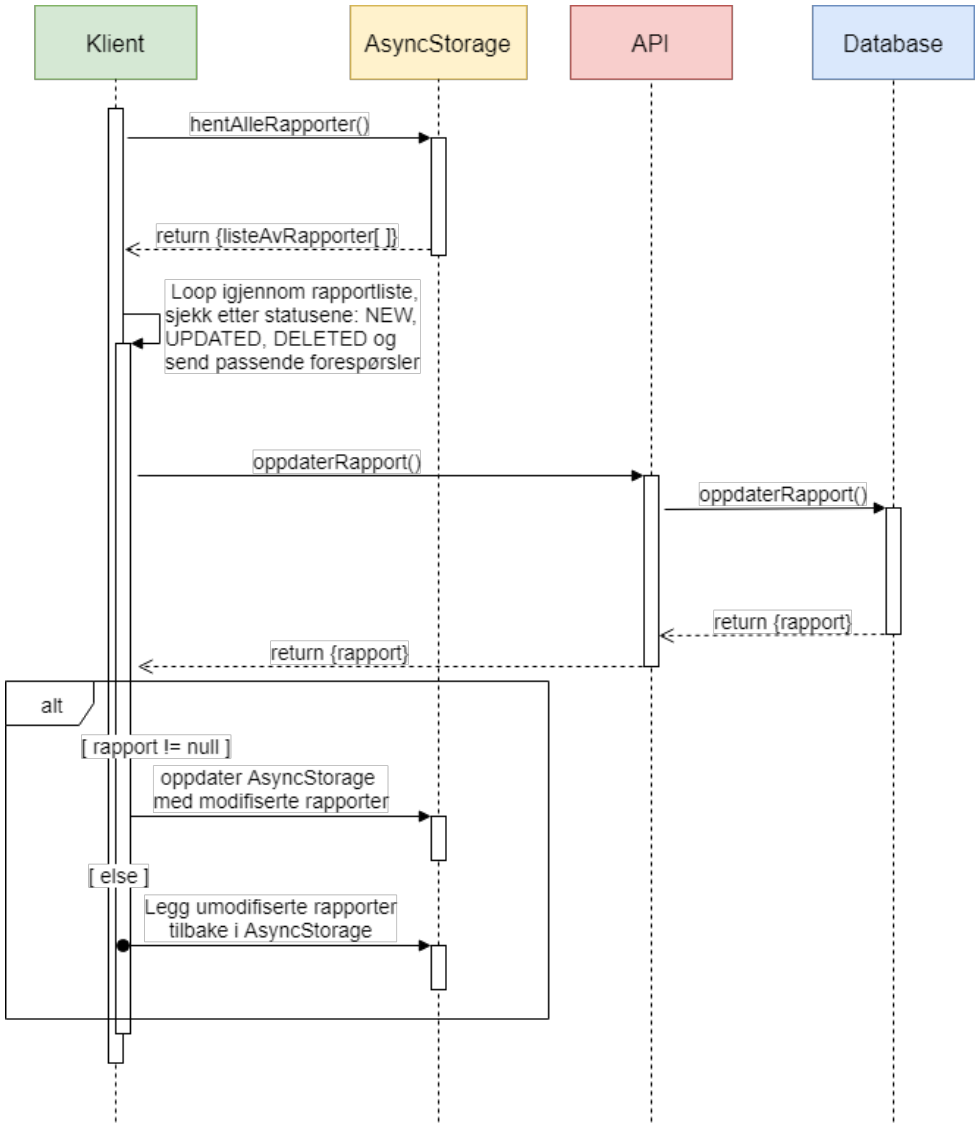
1. Klient sender lokal data modifisert i applikasjonen til server.
2. Server oppdaterer databasedataen med dataene som er tilsendt fra enheten, og tar de nødvendige stegene for å opprettholde forholdet mellom oppføringene på master og detalj-entitetene. For eksempel: et nøkkel/verdi-par kan brukes til å holde styr på ID-ene til de lokale oppføringene og ID-ene til de korresponderende oppføringene på serveren.
3. Server sender oppdatert databasedata.
4. Klient sletter og gjenskaper dataen lagret i lokal lagring med dataen mottatt fra serveren.

Det ble tatt et valg om å avvike fra bruken av nøkkel/verdi-par for å holde styr på forholdet mellom lokal og ekstern rapport-ID. Gruppen endret i stedet rapportene og aktivitetenes *key* så raskt de ble synkronisert med svaret fra API-et som inneholder en unik nøkkel.

API-responsen inneholder *http status ok* og en ny ID som kan skrives til det lokale

objektet om det ikke allerede har en legitim ID fra API-et. Videre fikk de lokale dataene status-attributt i stedet for verdiene *IsFromServer, IsModified, IsActive* som patternet benytter. Dette tillot gruppen å sjekke ett attributt istedenfor tre. Sekvensdiagrammet i figur 25 viser dataflyten til offline/online-funksjonaliteten. Gruppen benyttet patternet på følgene måte:

1. Listen som inneholder alle de lagrede rapportene på den mobile enheten ble hentet ut fra AsyncStorage uavhengig av statusen deres (NEW, UPDATED, ORIGINAL, DELETED). Dette er likt med OutSystems' pattern.
2. Deretter ble hovedfunksjonen i Synchronize.js kallet på. Denne looper gjennom alle rapporter og rapportens aktiviter og sender aktuelle forespørsler til API-et. Disse forespørslene kan være: POST, PATCH og DELETE. Dette er ulikt med OutSystems' pattern ettersom de sender all data til serversiden og lar server sjekke om det skal utføres oppdateringer på eksisterende data i databasen.
3. Dersom forespørslen var vellykket så ble databasedataen oppdatert. I tillegg ble det sendt en godkjenning fra API-et i form av hele rapportobjektet med en ny servergenerert ID for rapport og aktivitet(er). Dette er ganske likt med OutSystem's pattern, men at den servergenererte ID-en sendes med er unikt for mobilapplikasjonen.
4. Deretter oppdateres listen på den mobile enheten til å benytte de potensielt nye ID-ene og setter statusen på objektene til ORIGINAL dersom forespørslen(e) var vellykket. Dersom noe gikk galt under synkroniseringsprosessen ble rapporten satt i AsyncStorage uten en oppdatert ID og status. Dette er identisk med OutSystems sitt pattern med unntak av oppdateringen av *keys*.



Figur 25: Synkronisering mellom mobilapplikasjonen og databasen

6.1.6 Kommunikasjon

I seksjon 6.1.3 ble biblioteket *API.js* beskrevet. Som nevnt er det denne klassen som håndterer alt av API-forespørsler som definerer klientens kommunikasjon med backend. For å kommunisere med backend benyttes *Fetch API* fra React Native [53]. Dette er et grensesnitt fra JavaScript som tillater asynkron kommunikasjon over et nettverk.

Listing 6.8 viser en funksjon fra biblioteket *API.js*. Denne funksjonen ble benyttet der klienten trengte å kommunisere med API-et. Funksjonen tar imot tre parametere: *url*, *data* og *method*. *url* definerer den lenken som klienten skal sende en forespørsel til. *data* definerer de parameterne som skal sendes til databasen. For eksempel når en ny rapport opprettes, sender klienten en forespørsel der *data* inneholder informasjon om den nye rapporten. Det siste parameteret *method* definerer hva slags forespørsel som skal sendes. *method* kan være *POST* for å legge til noe nytt, *PATCH* for å oppdatere eller *DELETE* for å slette. Disse metodene er beskrevet i mer detalj i seksjon 6.2.1.

Innenfor *Fetch API* valgte gruppen å benytte ES7-standarden ovenfor eldre standarder ettersom denne syntaksen gjorde det enklere for gruppen å håndtere kommunikasjon i koden [54]. Fordelen med å benytte ES7 var at det tillot gruppen å vente på responsen fra API-et. På grunn av dette kunne gruppen definere funksjoner slik som i listing 6.8 der resultatet returneres. I ES6 derimot håndteres resultatet asynkront, noe som ikke tillater å returnere et resultat på samme måte.

```
1 static async request(url, data, method) {
2     const response = await fetch(url, {
3         method: method,
4         headers: headers,
5         body: JSON.stringify(data)
6     });
7
8     return response;
9 }
```

Listing 6.8: Hvordan mobilapplikasjonen kommuniserer med API-et

6.1.7 Spesifikt for iOS og Android

Under utviklingsperioden benyttet gruppen tre eksterne React Native-pakker for å oppnå ønsket funksjonalitet og design. I disse tilfellene måtte gruppen definere spesifikke konfigurasjoner for iOS og Android. Dette innebar hovedsakelig konfigurasjoner i Android sine .gradle-filer og iOS sin Info.plist-fil og/eller Podfile. Uten om dette fungerte alle komponenter definert av React Native for begge plattformene. De eksterne pakkene med spesifikk konfigurasjon er følgende:

- **react-native-vector-icons** er en pakke som gir utvikleren et stort utvalg av skalerbare ikoner [55]. Denne ble benyttet for å vise alle ønskede ikoner i mobilapplikasjonen.
- **react-native-i18n** ble benyttet for å implementere internasjonalisering. Pakken gjorde det mulig å definere oversettelser til ønskede språk [56].
- **react-native-maps** tillater utvikleren å benytte kart fra Google Maps og Mapkit [57]. Pakken ble benyttet for å implementere kartet som beskrevet i seksjon 6.1.4.

Under hele prosjektet har gruppemedlemmene testet applikasjonen på en Android-emulator. Derfor ble alle konfigurasjoner for Android definert underveis i utviklingspro-

sessen. I slutten av prosjektperioden forsøkte gruppen å konfigurere nevnte pakker for iOS ved å benytte en lånt PC med macOS (der Xcode var installert). Vi påbegynte arbeidet, men det ble ikke ferdigstilt før prosjektavslutning.

6.1.8 Sikkerhet

Gruppen har tatt en rekke tiltak for å støtte kravene satt innen sikkerhet (se seksjon 2.4). Et av disse kravene var at jegeren skal logge seg inn med jegernummer og identifisere seg ved hjelp av en tilsendt SMS-kode. Funksjonaliteten ble kun mocket i koden, men det er lagt til rette for implementasjon av en SMS-tjeneste. Grunnen til at implementasjonen ble mocket var at produkteier ikke ønsket at gruppen brukte tid og energi på å implementere «logg inn»-funksjonalitet. Han ønsket at gruppen prioriterte kjernefunksjonalitet som rapportering, kart og offline-modus. Videre er innloggingen illustert i listing 6.9. I koden hentes *data* som skal sendes med API-forespørselen. Deretter sendes en forespørsel om innlogging. Dersom jegernummeret ble funnet returneres en SMS-kode (som også skal ha blitt sendt til brukeren). Deretter kan klienten sjekke returnert SMS-kode mot koden som jegeren skriver inn i applikasjonen.

```
1  const data = API.getLoginData(this.state.uid);
2
3  try {
4      const response = await API.post(settings.LOGIN_URL, data);
5
6      if (response.ok) {
7          const json = await response.json();
8          let smsKey = json.data.key;
9
10         this.setState({
11             smsCode: smsKey,
12             smsInputVisible: true,
13             loginButtonText: I18n.t('login'),
14         });
15     }
16
17 } catch (error) {
18     Helpers.simpleFeedback(I18n.t('failTitle'), I18n.t('loginFailDesc'));
19 }
```

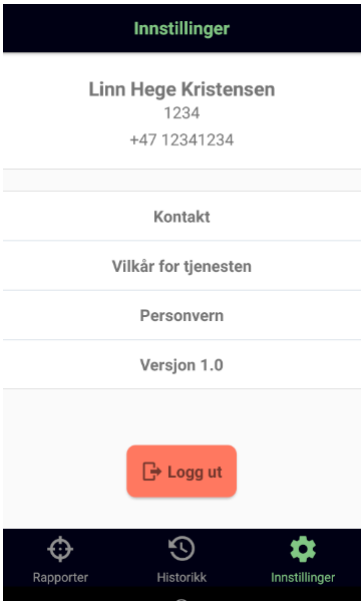
Listing 6.9: Kode for innlogging i klienten

Utenom innlogging har gruppen definert funksjoner i applikasjonen som rensker og validerer alt av bruker-generert input. Dette har vært essensielt ettersom disse dataene skal sendes via API-et og samhandle med databasen. Funksjonene er hovedsakelig benyttet der brukeren oppretter en ny rapport eller gjør endringer på en eksisterende rapport.

Videre ble det tidligere i oppgaven definert Misuse Cases som klienten måtte håndtere (se seksjon 2.1). Det første tilfellet som angår sikker innlogging ble håndtert som beskrevet over. Det andre tilfellet som omhandlet å oppdage og håndtere «spamming» av databasen ble også implementert. Problemstillingen ble løst ved å implementere funksjoner som sjekker alle bruker-genererte verdier opp mot logiske verdier. Gruppen fikk blant annet vite fra produkteier at en jeger svært sjeldent feller mer enn 5 dyr på én jaktdag. Derfor kunne vi implementere en funksjon som sjekket alle felt-verdier og som ikke tillater brukeren å registrere mer enn en bestemt mengde felte dyr av gangen. I tillegg satte gruppen en restriksjon på maks én rapport per dag. Dette bidrar også til å forhindre spamming av databasen.

En svakhet ved implementeringen er at brukeren for øyeblikket kan rapportere hver eneste dag med logiske (men uekte) data. Det ville styrket applikasjonens sikkerhet dersom det ble implementert en sjekk på totalt antall skutt for sesongen.

Et annet forberingspotensiale applikasjonen har er å innføre en mer sikker samhandling med API-et. For videre utvikling ville gruppen implementert API-keys. Les mer om dette i seksjon 6.4.



Figur 26: Personvern i mobilapplikasjonen

Personvern

For å tilfredsstille kravet om støtte for den nye personvernsloven (GDPR) [58] har gruppen gjort noen tiltak og tiltrettelagt for andre. Et tydelig tiltak som har blitt gjort er å la brukeren vite hvilken informasjon som systemet har om brukeren. Dette er illustrert i figur 26 der brukerens fulle navn er øverst, jegernummer i midten og lagret telefonnummer nederst i øverste boks. Informasjon om hvorfor disse opplysningene er nødvendige burde dokumenteres og beskrives i «Vilkår for tjenesten». Dette har ikke gruppen gjort ettersom vi ikke eier produktet selv og ikke bestemmer hvordan opplysningene skal benyttes. Videre kan det implementeres en meldingskanal der brukeren kan kontakte administrator dersom opplysningene ikke er riktige. «Kontakt» kan benyttes til dette. I tillegg er det også viktig at brukeren til enhver tid har mulighet til å protestere på applikasjonens bruk av persondata. Også dette må implementeres ved en senere anledning.

6.2 Backend

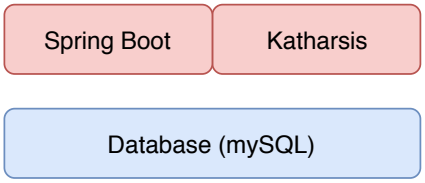
Serveren har i all hovedsak ansvaret for å behandle forespørsler som omhandler innlogging, rapportering og lagring av historikk/statistikk. Kommunikasjonen mellom klient og server foregikk gjennom et API, slik at databehandlingen ble standardisert og sentralisert.

Serveren består av to deler: en MySQL-database og et RESTful JSON API. Serveren ble primært skrevet i Java og benyttet rammeverket Katharsis for å implementere JSON API-standarden og støtte for Hypermedia As The Engine Of Application State (HATEOAS). Dette sørget for at alle klienter kun trengte å forholde seg til HTTP-/HTTPS-protokollen for å kommunisere med serveren, og at all data ble presentert på en uniform måte. I tillegg ble rammeverket Spring Boot benyttet for å håndtere web-forespørsler til og fra klientene.

6.2.1 Struktur

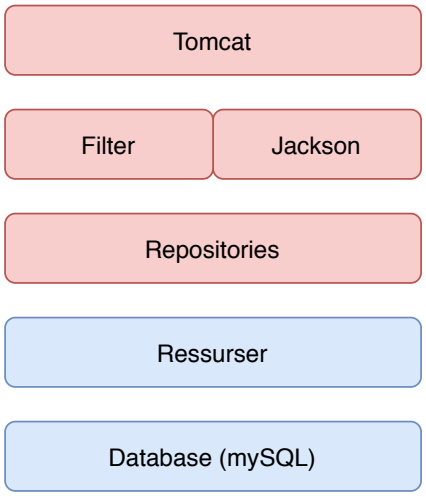
Lagdeling

Serveren følger en lagdelt struktur (se figur 27), der Spring Boot håndterer forespørsler, presentasjon av web-innhold, adgangskontroll, kobling mot database og offentliggjøring av endepunkter. De forespørslene og dataene som Spring Boot håndterer fanges så opp av Katharsis som omformer ressursene slik at de stemmer over ens med JSON API-standarden.

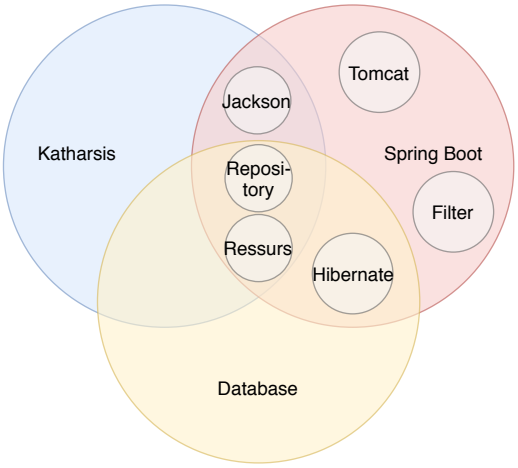


Figur 27: Overordnet oversikt over lagdelingen på serversiden

Både Spring Boot og Katharsis benytter en rekke biblioteker, rammeverk og design-patterns som gruppen måtte ta hensyn til under implementasjonen. Dette førte til en mer detaljert lagdeling, som er illustrert i figur 28.



(a) Utvidet oversikt over lagdelingen på serversiden, etter at transitive biblioteker er tatt i betraktning



(b) Oversikt bibliotekers ansvarsområder og overlapping mellom andre biblioteker

Figur 28: Lagdeling og ansvarsområder på serversiden

Spring Boot benytter *Embedded Tomcat* som web-server. Denne har ansvar for å sende/motta web-forespørsler og presentere web-innhold. Videre benyttes *Jackson* for å omforme Java-objekter (Ressurser) til og fra JSON og *Hibernate ORM* for å skape ett felles grensesnitt (Repository) mot forskjellige databaser.

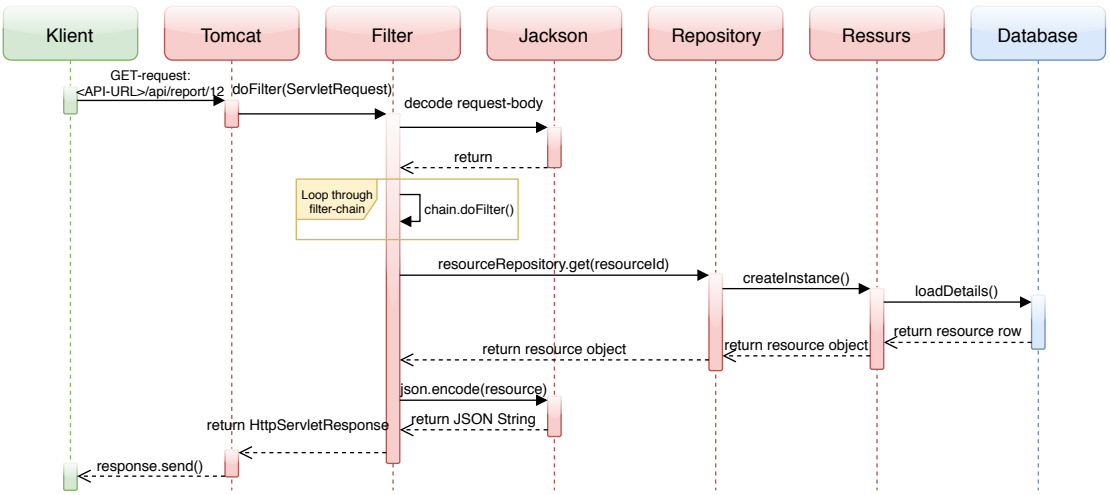
Spring Boot inkluderer et API for *filter*-implementasjon. Et filter oppdager forespørsler før de blir prosessert, og tillater dermed et ekstra kontoll-lag. Et eksempel på et bruksområde er å kontrollere brukerens tilgangsrettigheter til den ressursen som blir etterspurt, eller sende feilmeldinger dersom en ressurs ikke ble funnet.

Katharsis har en egen modul som heter «katharsis-spring-boot». Denne modulen overstyrer konfigurassjonen som Spring Boot bruker for både ressurser, repositories og Jackson. Dette gjør at Katharsis kan forsikre seg om at JSON-formatet som Jackson genererer, er i tråd med JSON API-standarden.

Kommunikasjon

Kommunikasjonen mellom lagene er visualisert i figur 29. Sekvensdiagrammet har tatt utganspunkt i et scenario der en klient vil ha informasjon om en rapport som er registrert i databasen.

Klienten sender først en GET-forespørsel for å hente ut rapporten med ID 12: `<API-URL>/api/report/12`. Denne forespørselen mottas av Tomcat, som gjør forespørselen om til ett `HttpServletRequest`-objekt, slik at den kan behandles videre. Den sendes så gjennom en Filter-Chain, som først dekoder eventuell data ved hjelp av Jackson (ingen data i dette tilfellet). Deretter kjøres resten av filtrene, som blant annet kontrollerer at klienter har tilstrekkelig tilgang, og definerer hvilken ressurs som skal hentes ut; i dette tilfellet en `Report`. Denne hentes ved å kalle på repository-et til den ressursen: `ReportRepository`. Repository-et er koblet til Hibernate som oppretter en ny instans av `Report`-objektet, og populerer det med data fra `report`-tabellen i databasen. `Report`-objektet returneres så til filteret som behandler forespørselen og gjør objektet om til en JSON-streng (via Jackson). Videre legges dataene til i et `HttpServletResponse`-objekt, som Tomcat tar i mot og sender tilbake til klienten.



Figur 29: Kommunikasjonen mellom arkitekturlagene på serversiden

Spring Boot og Hibernate ORM tilføyer mye «magi», som gjør det svært enkelt å implementere nye ressurser. Eksempler på dette er vist i listing 6.10 og 6.11. I disse kodesnuttene er implementasjonen for `Report` og `ReportRepository` illustrert, som benyttet i figur 29. Kommentarer og get-/set-metoder er utelukket i kode-eksemplene.

```
1  @Entity
2  @JsonApiResource(type = "report")
3  public class Report {
4
5   @Id
6   @GeneratedValue(strategy = GenerationType.AUTO)
7   @JsonApiId
8   private int id;
9
10
11  private Date date;
```

```
12
13
14   @ManyToOne(targetEntity = Area.class)
15   @JsonApiRelation(lookUp = LookupIncludeBehavior.AUTOMATICALLY_WHEN_NULL,
16           serialize = SerializeType.EAGER,
17        opposite = "id")
18   private Area area;
19
20
21   @ManyToOne(targetEntity = Person.class)
22   @JsonApiRelation(lookUp = LookupIncludeBehavior.AUTOMATICALLY_WHEN_NULL,
23        serialize = SerializeType.EAGER,
24        opposite = "id")
25   private Person person;
26
27
28   @JsonApiRelation(lookUp = LookupIncludeBehavior.AUTOMATICALLY_WHEN_NULL,
29        serialize = SerializeType.EAGER,
30        opposite = "report")
31   @OneToMany(mappedBy = "report",
32              cascade = CascadeType.REMOVE,
33              orphanRemoval = true)
34   private Set<Activity> activities;
35
36 }
```

Listing 6.10: Implementasjon av entiteten Report i Hibernate/Spring Boot

```
1 public interface ReportRepository extends CrudRepository<Report, Integer> {
2
3 }
```

Listing 6.11: Implementasjon av repositoryet ReportRepository i Hibernate/Spring Boot

Ressurser, repositories, JPA og ORM

Hibernate ORM benytter Java Persistence API (JPA) for å annotere hvilke klasser som er database-entiteter og repositories. Dette er en implementasjon av repository-patternet. Hibernate gjorde det i tillegg mulig å endre database-type uten at implementasjonen må endres. Dette gjorde applikasjonen svært fleksibel med tanke på integrering, videreutvikling og drifting av serveren.

JPA benyttes for at Hibernate og Katharsis skal forstå hvordan dataene skal behandles, lagres og presenteres. Koden som vist i listing 6.10 og 6.11 er en vanlig implementasjon for en tabell i en database ved bruk av ORM. Annoteringene er beskrevet kort i tabell 9.

Design-patterns

For å enklere kunne lese og skrive JSON-strenger ble det laget en `Builder`-klasse (`JsonApiJsonBuilder`) for JSON-Objekter. Formålet med denne klassen var å bygge JSON-objekter på JSON API-format siden dette formatet er standardisert, og blir ofte gjentatt ved manuell håndtering av responser. Denne klassen inneholder tre `JSONObject`-objekter, som representerer de tre seksjonene i en JSON API-respons: *data*, *attributes* (Attributter) og *relations* (Relasjoner). Ettersom det bare skal være én instans av disse per objekt, ble patternet `Singleton` benyttet for å kontrollere det. Dette ble implementert ved å referere til objektene gjennom funksjoner som kontrollerer om en instans er opprettet eller ikke, i stedet for å benytte objektet direkte. Se listing 6.12 for eksempel. Constructoren initialiserer instansene til `NULL`, slik at instansene ikke blir opprettet med mindre de er nødvendige. I tillegg er det mulig å spesifiere en verdi å kontrollere mot.

Annotering	Beskrivelse
@Entity	Forteller Hibernate at denne klassen er en tabell i en database.
@JsonApiResource	Forteller Katharsis at denne klassen er en ressurs, og navnet på den.
@Id	Forteller Hibernate at dette attributtet er en primærnøkkel.
@GeneratedValue	Forteller Hibernate hvordan denne verdien skal auto-genereres.
@JsonApiId	Forteller Katharsis at dette attributtet skal brukes som id.
@ManyToOne/@OneToMany	Forteller Hibernate at attributtet er en relasjon til en annen ressurs.
@JsonApiRelation	Forteller Katharsis at dette er en relasjon.

Tabell 9: Overordnet beskrivelse av alle JPA-annoteringer som brukes ved implementasjonen av en ny ressurs

```
1  private JSONObject data;
2
3  private JSONObject data(){
4      if (this.data == null) {
5          this.data = new JSONObject();
6      }
7
8      return this.data;
9  }
```

Listing 6.12: Eksempel på implementasjon av patternet Singleton

Builder-patternet er implementert ved å spesifisere set-/add-funksjoner som returnerer `this` (en referanse til det objektet funksjonene er kalt mot). Dette innebærer at man kan bygge og initialisere en instans av objektet i samme operasjon. Build-operasjonen avsluttes med en konvensjonell `build()`-funksjon, som pakker alle dataene i ett JSON-objekt, og legger til «data»-seksjonen, som er påkrevd i følge JSON API-standarden. Et eksempel på denne funksjonaliteten er vist i listing 6.13.

```
1  public class JsonApiJsonBuilder {
2
3      private JSONObject data;
4
5      private JSONObject attributes;
6
7      private JSONObject relations;
8
9      // Constructor og singleton-metoder er ikke tatt med
10
11     public JsonApiJsonBuilder setResourceId(int id) {
12         data().put("id", id);
13         return this;
14     }
15
16     public <T> JsonApiJsonBuilder addResourceAttribute(String key, T value){
17         attributes().put(key, value);
18         return this;
19     }
20
```

```
21
22      public String build(){
23          JSONObject dataSection = new JSONObject();
24
25          dataSection.put("data", data());
26
27          if (this.attributes != null) {
28              data().put("attributes", attributes());
29          }
30
31          if (this.relations != null) {
32              data().put("relationships", relations());
33          }
34
35          return dataSection.toJSONString();
36      }
37  }
```

Listing 6.13: Eksempel på implementasjon av patternet Builder

Forespørsler

Etter en kartleggelse av hvilke data som skulle lagres i databasen (se seksjon 6.2.4), ble det satt opp en tabell over hvilke operasjoner og kall som skulle støttes av API-et. Kallene er oppsummert i tabell 10.

API-ressurser

#	Ressurs	URL	GET	POST	PATCH	DELETE
1	person	/api/person				
2	person	/api/person/:id	X			
3	report	/api/report	X	X	X	X
4	activity	/api/activity	X	X	X	X
5	activityCoordinate	/api/activityCoordinate	X	X	X	X
6	species	/api/species	X			
7	area	/api/area	X			
8	areaCoordinate	/api/areaCoordinate	X			
9	region	/api/region	X			
10	municipality	/municipality	X			
Andre tilgjengelige ressurser						
11	login	/login		X		
12	autentisering	/authenticate		X		
13	klient APK	/apk	X			
14	API-JavaDoc	/doc	X			

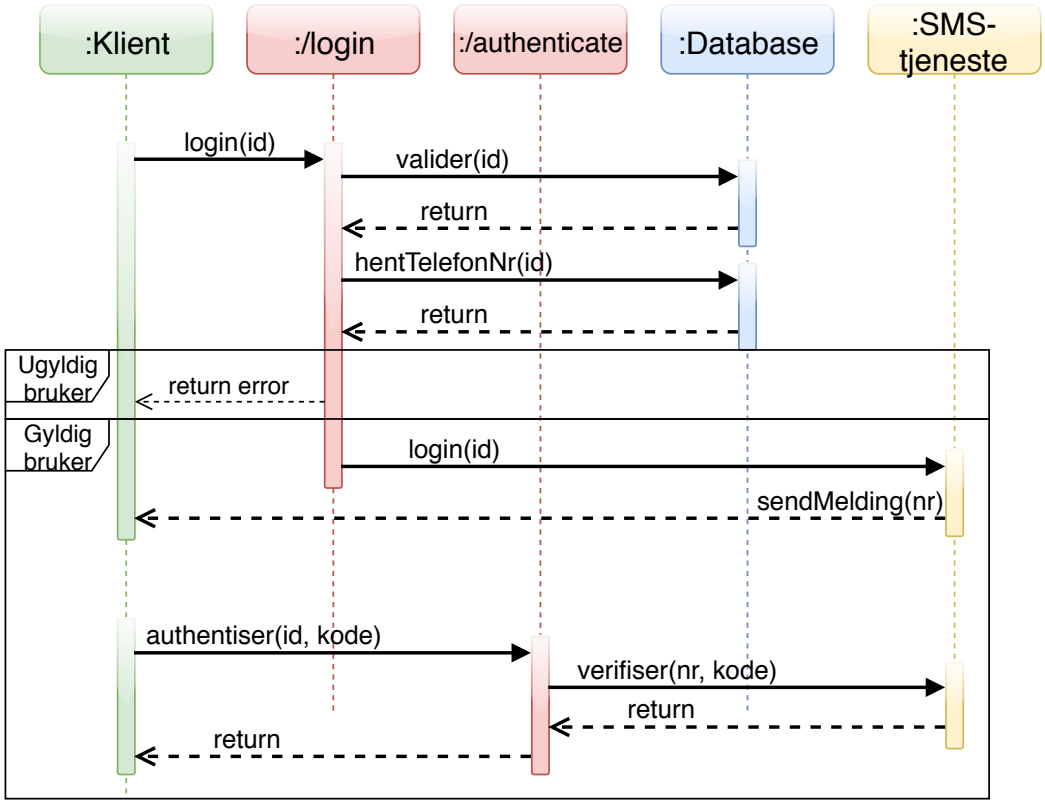
Tabell 10: Alle endepunkter som er tilgjengelige gjennom API-et

De definerte ressursene kobles til en korresponderende Uniform Resource Locator (URL), og operasjonene som kan utføres på ressursen defineres av typen forespørsel som blir benyttet. JSON API-spesifikasjonen begrenser hvilke forespørsler som kan benyttes til *GET*, *POST*, *PATCH* og *DELETE*. Respektivt benyttes disse til å *hente*, *opprette*, *endre/oppdatere* og *slette* ressurser.

6.2.2 Autentisering

En bruker av mobilapplikasjonen må være en verifisert jeger for å kunne benytte applikasjonen. For å få til dette, benyttes en to-trinns-verifisering der brukeren skriver inn sitt jegernummer. Videre verifiseres nummeret mot databasen (kall nr. 11 i tabell 10). Deret-

ter benyttes en ekstern SMS-tjeneste (simulert) for å sende en tilfeldig generert kode til brukeren (telefonnummer lagret i databasen). Brukeren må da skrive inn denne koden i applikasjonen, som da kontrolleres mot den koden som ble generert (kall nr. 12). Dette er illustrert i figur 30. For å holde kostnadene nede, og forenkle utviklingen, ble den eksterne SMS-tjenesten simulert ved å lokalt generere koden og sende den i respons på login-forespørselen (kall nr. 11). Jegernummerene er heller ikke verifisert som faktiske jegernumre, som resultat av avgrensing i seksjon 1.2.



Figur 30: Dataflyt under innlogging og autentisering av en bruker

6.2.3 Feilhåndtering

Feilmeldinger genereres og håndteres i form av Exceptions, noe som Katharsis håndterer ved hjelp av en `ExceptionMapper`. Dette er et grensesnitt i Java som lar Katharsis plukke opp feilmeldingen, og returnere denne som JSON til klienten. Gruppen har også definert noen spesifikke feilmeldinger/exceptions som behandler feil angående tilgang til ressurser. Siden gruppen «manuelt» behandler tilgangen til ressurser måtte gruppen implementere egne tilbakemeldinger når en klient ikke har tilgang til den forespurte ressursen. Dette ble gjort ved å utvide den eksisterende klassen `RuntimeException` fra Java, og koble denne til `ExceptionMapper`-en i Katharsis.

6.2.4 Databasen

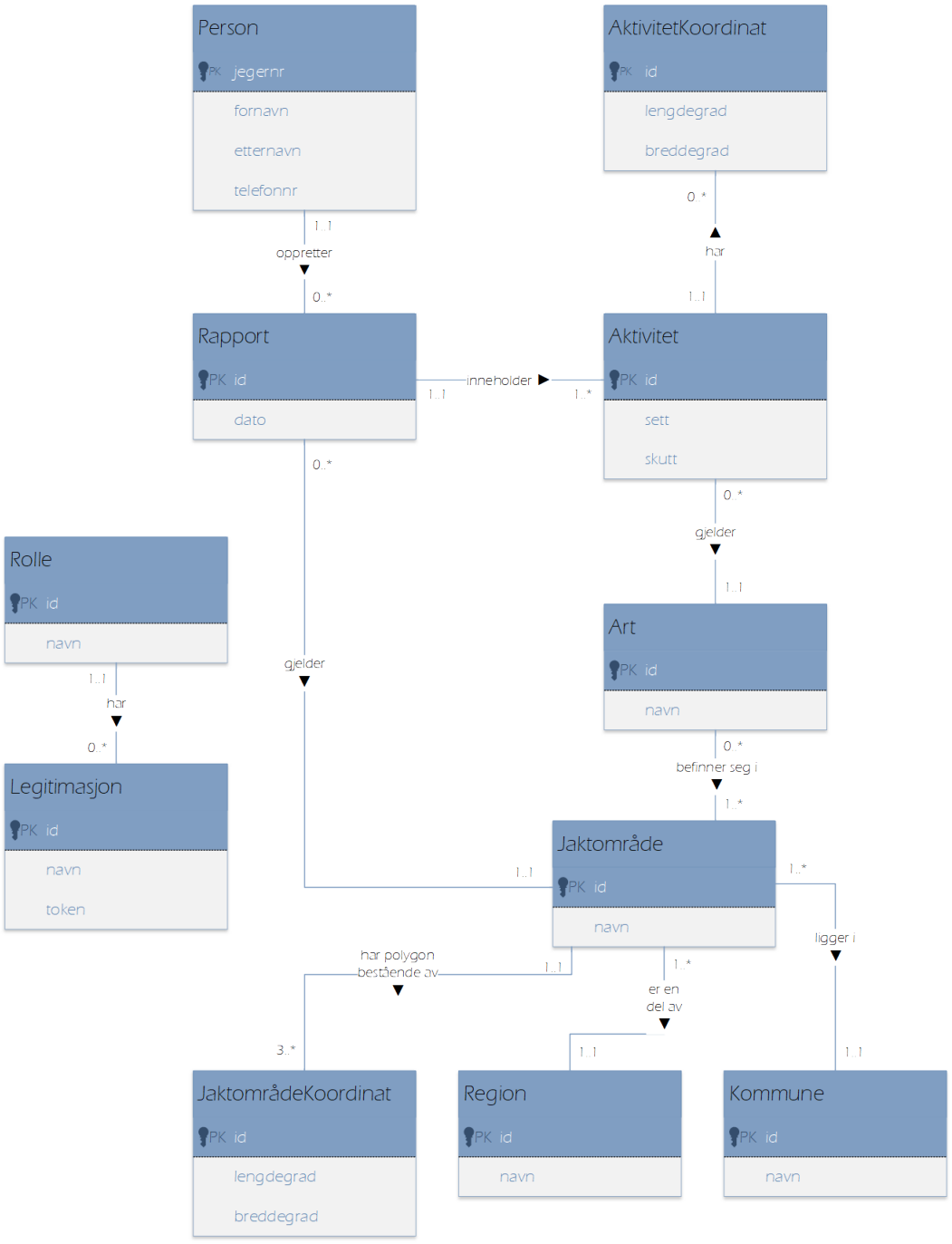
Valg av teknologi

Gruppen valgte å benytte MySQL versjon 5.6 for databasen. For databasemotoren benyttes InnoDB og denne kjøres på en server med Ubuntu Server 17.04. Gruppen falt på

dette valget av flere grunner. En av dem er at gruppen ble godt kjent med MySQL gjennom studiet. Videre er det en av verdens største databaser som er open source. De har blitt ledende spesielt på web-applikasjoner, men de gir mange gode muligheter også for mobil [59]. MySQL har også en enkel og lite krevende installasjonsprosess, som gjør at den raskt kan integreres og settes i produksjon. I tillegg er databaselaget abstrahert bort fra selve server-applikasjonen ved å benytte Hibernate ORM. Dette gjør at den underliggende databaseteknologien vil være uavhengig av implementasjonen, og kan dermed enkelt skiftes ut dersom produkteier ønsker en annen databaseteknologi ved senere utvikling.

EER-diagram

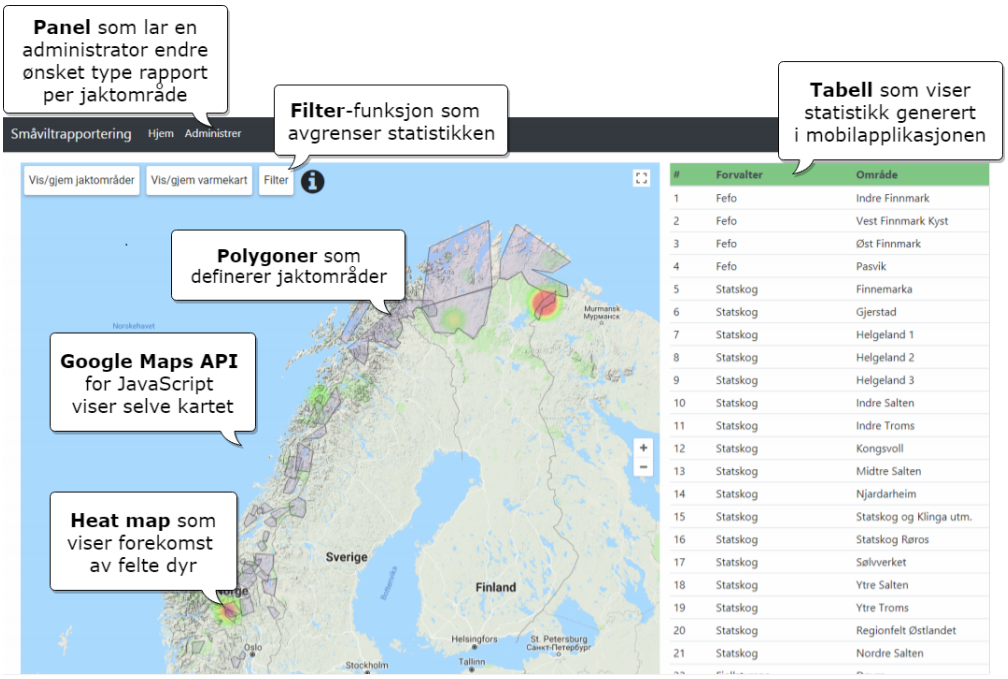
Før gruppen implementerte databasen ble det utformet et Enhanced entity–relationship (EER)-diagram for å illustrere de ulike dataene som produktet skulle håndtere. Diagrammet er vist i figur 31. Dette ga gruppen et godt overblikk som gjorde det enklere å se hvordan databasen burde implementeres. I tillegg ble det lettere å se helheten i databasen og alle dataene. Utenom dette var det en stor fordel å ha diagrammet tilgjengelig gjennom utviklingsprosessen ettersom gruppen hadde behov for å gjøre kall til databasen gjennom API-et. Videre definerte gruppen et relasjonsskjema som beskriver relasjonene mellom alle entitetene. Relasjonsskjemaet er lagt ved i vedlegg F.



Figur 31: Databasedesignet illustrert i et EER-diagram

6.3 Web-løsningen

Web-løsningen er primært skrevet i Javascript og jQuery, og benytter samme backend som beskrevet i seksjon 6.2. Løsningen er implementert som en single page-applikasjon for å minimalisere overhead og responstid. I tillegg benyttes Google Maps API for JavaScript ved implementasjon av kart. Videre viser figur 32 en oversikt over implementert funksjonalitet.



Figur 32: Funksjonalitet som ble implementert i web-løsningen

6.3.1 Kart og statistikk

JQuery Ajax benyttes til å hente data fra serveren dynamisk. Dataene lagres i et `DataStructure`-objekt som omformer dataene til det formatet som kreves av Google Maps. Dette objektet lastes inn når applikasjonen lastes første gang, eller ved en oppdatering av siden (oppdateres ikke i sanntid). Dette er gjort for å spare overhead, og i tillegg endres ikke dataene ofte nok til at dette er nødvendig.

Når datastrukturen er lastet inn opprettes et `Map`-objekt. Dette objektet håndterer alle interaksjoner med kartet. Videre definerer objektet også funksjoner som gjør det enklere å dynamisk oppdatere det som vises på kartet, spesielt med tanke på filtrering.

Et filter er implementert for å øke nytteverdien av applikasjonen for forvaltere. Forvaltere kan velge å se samlet statistikk for sine områder, et spesifikt område eller en spesifikk art i en definert tidsperiode. Brukeren kan også velge flere forvaltere, områder og arter samtidig.

Når applikasjonen laster inn første gang vises alle rapporter som standard. Ved filtrering kjøres en ny spørring mot datasettet, og de rapportene som passer filterets kriterier hentes ut. Deretter oppdateres kartet og tabellen (figur 17) med de dataene som samsvarer med filteret.

Google Maps API

Kartløsningen er en implementasjon av Google Maps API. API-et tilbyr også en modul for visualisering, der det spesifikt benyttes heat map for visualisering av registrerte rapporter.

I følge Google Maps API sine sider har Google Maps for JavaScript visse restriksjoner. Det er gratis å benytte dette API-et så lenge kartet lastes inn maksimalt 25 000 ganger per døgn fra samme applikasjon [46]. Kart-løsningen vil primært benyttes av forvaltere, og derfor mener gruppen at trafikken vil være lav nok til at denne kvoten ikke overskrides. I tillegg vil prisene som nevnt tidligere justeres etter 11. juni 2018. Fra juni vil kun «Street View»-visninger bli fakturert ved bruk av Google Maps JavaScript API [47]. «Street View» benyttes ikke av denne applikasjonen, og vil dermed ikke påføre produkteier ekstra kostnad.

Heat map

Heat map genereres ved å opprette et nytt `HeatMapLayer`-objekt, og bruke det som et overlegg over det eksisterende kartet. Dataene som visualiseres består av lengde- og breddegrader, som tilsvarer de punktene jegerne registrerer på kartet i mobilapplikasjonen.

Polygoner

Polygoner ble benyttet for å illustrere jaktområder på kartetet. Polygonene består av sett med lengde- og breddegrader som lastes inn fra databasen. Disse punktene er de samme punktene som benyttes i mobilapplikasjonen.

En viktig del av polygoner er å tegne dem i korrekt rekkefølge, slik at de vises som et omriss av et område (i motsetning til tilfeldige linjer mellom punktene). Dette er løst ved å sortere alle punktene etter en spesifisert rekkefølge, definert i databasen, før de tegnes på kartet.

Tabellen

Tabellen blir dynamisk oppdatert basert på de preferansene brukeren setter med hjelp av et filter. Uten spesifisert filter vises samlet statistikk for alle jaktområder, inkludert områder uten rapporter. Dersom brukeren ønsker å filtrere dataene, vises bare de valgte områdene. Områder uten noen rappporter ekskluderes.

Filter

Ved første innlasting av applikasjonen hentes alle rapporter fra datasettet. Deretter reduseres dette datasettet ved å fjerne de rapportene som ikke er innenfor det spesifiserte dato-intervallet. Videre kan en forvalter filtrere data på forvalter, område og art. Da fjernes informasjonen som allerede er visualisert på kartet og i tabellen, og disse oppdateres med det nye datasettet. Koden som gjør dette er vist i listing 6.14.

```
1  var filteredActivities = [];
2
3      // Find all activities that matches the filter
4      for(var i in dataset.activity) {
5        var activity = dataset.activity[i];
6
7        if((dateStart === 0 || dateStart <= activity.report.date) &&
8           (dateEnd === 0 || activity.report.date <= dateEnd) &&
9           containsItem(activity.report.area.region, selectedRegions) &&
```

```
10          containsItem(activity.report.area, selectedAreas) &&
11          containsItem(activity.species, selectedSpecies)){
12
13            // Activity passes filter
14            filteredActivities.push(dataset.activity[i]);
15        }
16
17      }
18
19      // Update the map and table with filtered information
20      updateMap(filteredActivities);
```

Listing 6.14: Filtrerer datasettet i web-løsningen og oppdaterer kartet med de filtrerte dataene

Administrasjonspanel

Administrasjonspanelet består av tre dropdown-elementer (felter) som dynamisk oppdateres (se figur 20). Først registreres det en «onChangeListener» på hvert felt, og en «onClickListener» på lagre-knappen. Disse kjøres når administrator gjør en endring, og har som oppgave å aktivere og oppdatere innholdet i neste felt. For eksempel når administrator velger forvalter, hentes alle områder under den organisasjonen fra backend og blir deretter lagt inn i neste felt, som til slutt aktiveres. Koden for dette er vist i listing 6.15.

```
1  function onRegionChanged(){
2    // Fetch the id of the selected region
3    let selectedRegion = $(this).val();
4
5    // Fetch all areas under the selected region, and register/run callback
6    getAPIResource('region/' + selectedRegion + "/relationships/areas", (data) =>
         {
7
8      // Remove all existing elements, except first (default/no selection)
9      $('#selectedArea option').not("option:first").remove();
10
11     // Create <option>-elements and add to <select>
12     for(var i in data.data) {
13
14       let option = $('<option></option>', {  // Create element
15         value: data.data[i].id,              // Region ID
16         text: data.data[i].attributes.name  // Region Name
17       })
18
19       // Add the element
20       $('#selectedArea').append(option);
21     }
22
23     // Enable select and set hint as default selection
24     $('#selectedArea').prop('disabled', false);
25     $('#selectedArea option:first').select();
26   });
27 }
```

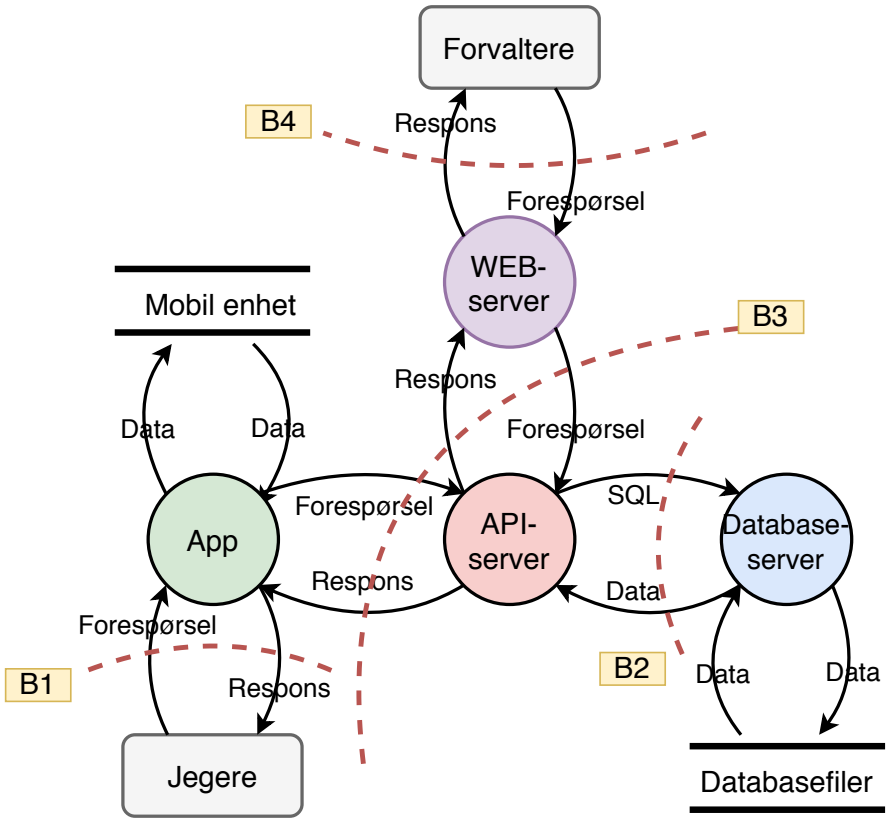
Listing 6.15: OnChangeListener som oppdaterer innhold i neste dropdow basert på valgt forvalter

Når lagre-knappen klikkes, hentes alle verdiene som er valgt, og det sendes en PATCH-request til backend som oppdaterer relasjonen mellom valgt jaktområde og alle de valgte artene. Deretter vises en tilbakemelding til forvalter basert på resultatet av spørringen. Koden for dette er vist i listing 6.16.

```
function onSave(){
    let selectedArea = $('#selectedArea').val();
    let data = [];      // IDs of selected species

    // Omitted code: Find all selected species and store in data[]

    $.ajax({
         method: 'PATCH',
         accept: 'application/vnd.api+json',
         contentType:'application/vnd.api+json',
         url: API_URL + 'area/' + selectedArea + '/relationships/species',

         data: JSON.stringify({data: data}),

         success: function(data) {
           $('#successMessage').fadeIn(400);
         },

         error: function(data){
           $('#errorMessage').fadeIn(400);
         }

   });
}
```

Listing 6.16: OnClickListener som oppdaterer backend med arter for valgt område

6.4 Sikkerhet



Figur 33: Dataflyt-diagram for trusselvurdering

I figur 33 vises en oversikt over systemets hovedkomponenter. I figuren er det definert

4 boundaries som angir hvilke kommunikasjonskanaler som må sikres mot angrep. Under diskuteres de ulike sikkerhetstiltakene som er implementert, og hvilke sikkerhetstiltak gruppen anbefaler for videre utvikling.

B1: Jeger mot mobilapplikasjon

Det første punktet som må sikres er mobilapplikasjonen, siden input-felter er et naturlig punkt å angripe for sabotører (se Misuse Case i figur 1). Feltene er sikret ved å kjøre input-validering som kontrollerer at det kun skrives inn nummer, som beskrevet i seksjon 6.1.8. Alle tekstlige verdier er hentet fra backend, og kan ikke endres i mobilapplikasjonen.

I tillegg lagres ressurser (se tabell 10) lokalt på enhetene, ved hjelp av AsyncStorage. AsyncStorage lagrer dataene globalt fra applikasjonens synspunkt, men siden applikasjonen har sitt eget navnerom vil dataene være utilgjengelige fra andre applikasjoner [60]. Gruppen mener derfor at dataene er tilstrekkelig sikret siden de ikke inneholder sensitive data. Et mulig tiltak vil være å kryptere dataene før de lagres.

B2: API mot database

Neste punkt er mellom database-serveren og API-serveren. Dette er et kritisk punkt som kan medføre svært alvolige konsekvenser dersom det angripes. Gjennom utviklingsprosessen kjørte databasen lokalt på samme nettverk som API-serveren, og all ekstern kommunikasjon kunne derfor blokkeres. I tillegg ble all interaksjon med serveren kryptert ved hjelp av Secure SHell (SSH), og brukere måtte autentiseres med Rivest–Shamir–Adleman (RSA)-nøkler. I tillegg ble det opprettet en egen bruker for databasen som benyttes av API-et, og denne har kun tilgang til de operasjonene som kreves av Hibernate.

Andre mulige tiltak er å kryptere kommunikasjonen mellom API-server og database ved hjelp av HTTPS eller Virtual Private Network (VPN).

B3: API mot web-løsning og mobilapplikasjon

Dette punktet omhandler all kommunikasjon til og fra backend. Dette er mest sannsynelig det punktet som er mest utsatt, da det er direkte eksponert mot internett. Det er ikke implementert noen spesifikke sikkerhetstiltak her, utenom å begrense tilgang til spesifikke ressurser (se filter i listing 7.3), da serveren ikke er satt i produksjon.

Tiltak for å sikre kommunikasjonen er å benytte HTTPS, i tillegg til en form for autentisering. Autentisering kan gjøres på mange måter, men OWASP Foundation anbefaler JSON Web Tokens (JWT), som regnes for å være en av dagens beste løsninger for autentisering med tokens mot RESTful JSON API [61].

B4: Forvalter mot web-løsning

På lik linje med mobilapplikasjonen må alle input-felter valideres. I tillegg hentes dataene fra backend ved hjelp av Ajax, og derfor bør domenet som benyttes valideres på backend. Dette kan gjøres ved hjelp av Cross-Origin Resource Sharing (CORS), som anbefalt av OWASP Foundation [61]. En annen mulighet er at API-et genererer den statistikken som skal vises, slik at man minimaliserer hvilke data som trenger å eksponeres gjennom API-et.

Implementasjon av token

Mot slutten av prosjektperioden startet gruppen med å implementere autentisering med token. Modulen *Spring-Boot-Security* ble benyttet for å implementere dette. Entitetene *Legitimasjon* og *Rolle* ble opprettet i databasen (se figur 31). I tabellen *Legitimasjon* ble det lagret navn og token for *klienter* som skulle benytte API-et. Både web- og mobil-applikasjonen fikk utdelt hvert sitt token, slik at alle kallene til «/api/*» (se tabell 10) kunne bli autentisert. Klientene kunne dermed benytte *Basic HTTP Authentication* (HTTP-transaksjon) for å sende denne legitimasjonen til backend. Denne implementasjonen ble dessverre ikke fullført innen prosjektavslutning.

Denne implementeringen har mange forbedringspotensialer. *Basic HTTP Authentication* har flere sikkerhetshull, som for eksempel sending av passord uten kryptering/hashing, og som i tillegg sendes på nytt for hver forepørsel. Dette øker angrepsflaten, og gruppen anbefaler derfor HTTPS i tillegg til autentisering med OAuth 2.0 og JWT ved senere utvikling, da dette er regnet for å være en industri-standard [62].

7 Kodekvalitet

I dette kapittelet beskrives de metodene og verktøyene som ble benyttet for å sikre god kodekvalitet. Et av konseptene som ble benyttet er code reviews. Videre ble også SonarQube benyttet for alle kodebasene i tillegg til diverse Linter. Utenom dette har gruppen benyttet seg av pair programming-metoden fra XP i de tilfellene det har oppstått spesielt store utfordringer. Pair programming har foregått enten ved å gjennomgå psuedo-kode på en tavle eller ved å sitte side ved side og utvikle i fellesskap.

7.1 Code review

Det er mange grunner til å utføre code reviews. En av de største grunnene er for å forebygge eventuelle feil (bugs eller sårbarheter). I tillegg sikrer det at koden blir implementert på best mulig måte. Code review sørger også for at kompetanse blir fordelt på flere av gruppemedlemmene. For gruppen la metoden til rette for kunnskapsdeling på tvers av kodebasene i Bitbucket.

Professional Programming

Ved siden av dette prosjektet hadde gruppen samtidig et emne kalt Professional Programming. Simon McCallum var ansvarlig for emnet. Gjennom emnet fikk gruppen mulighet til å utføre code review med både lærer og klassen samlet. Den 6. april hadde gruppen et slikt code review. Gruppen valgte å vise React Native-kode fra mobilapplikasjonen. Dette er hovedpunktene fra vurderingen:

- Variabler skal aldri dupliseres. Dersom det er et behov bør det benyttes hjelpefunksjoner.
- Funksjoner, parametere, strenger fra I18N o.l skal alltid ha beskrivende og intuitive navn.
- Komplekse funksjoner burde benytte hjelpefunksjoner.
- For bedre leslighet bør JavaScript-funksjoner benyttes dersom det er mulig (i stedet for å definere dem selv).
- Kode burde aldri inneholde mer enn én «statement» på én linje med kode.
- I mer komplekse funksjoner kan det være en fordel å håndtere alt av feilsjekking først og utføre funksjonalitet etter.
- Lokale variabler bør ikke bli initialisert før man vet helt sikkert at man har bruk for dem.

Pull requests

Under utviklingen valgte gruppen å benytte pull requests. Hensikten med pull request er å la en utvikler signalisere at det finnes ferdig utviklet kode som kan integreres med kodebasen. Pull requests forsikrer at ny og endret kode må godkjennes før den kan integreres. Gruppen valgte å gjøre pull requests obligatorisk i Bitbucket. Ved å gjøre dette obligatorisk forsikret gruppen at det ble utført et code review ved vurdering av en pull request. Dette forsikret en høyere grad av kodekvalitet og at applikasjonen på developement-

branchen alltid var kjørbar. Pull requests og code reviews har hovedsakelig foregått i BitBucket-repositoriet. Ved en pull request har et annet medlem fra gruppen lagt inn kommentarer på gode og dårlige elementer ved koden. I tillegg ble det også kommentert hva som måtte gjøres før koden kunne bli godkjent.

7.2 SonarQube

For å få en oversikt over status på kodekvaliteten og dekking av tester, valgte gruppen å sette opp og kjøre en SonarQube-server (versjon 6.7.1) som utfører statisk analyse. SonarQube er open source og gir en god oversikt over blant annet code smells, kjente bugs, dekking av tester og sikkerhetshull.

Gruppen satte opp tre prosjekter; ett for web-løsningen, ett for backend-serveren og ett for mobilapplikasjonen. Alle på gruppen fikk tildelt en egen bruker slik at alle kunne utføre en skanning av prosjektene. Skanning ble utført regelmessig, og på backend ble scanning automatisert gjennom Maven.

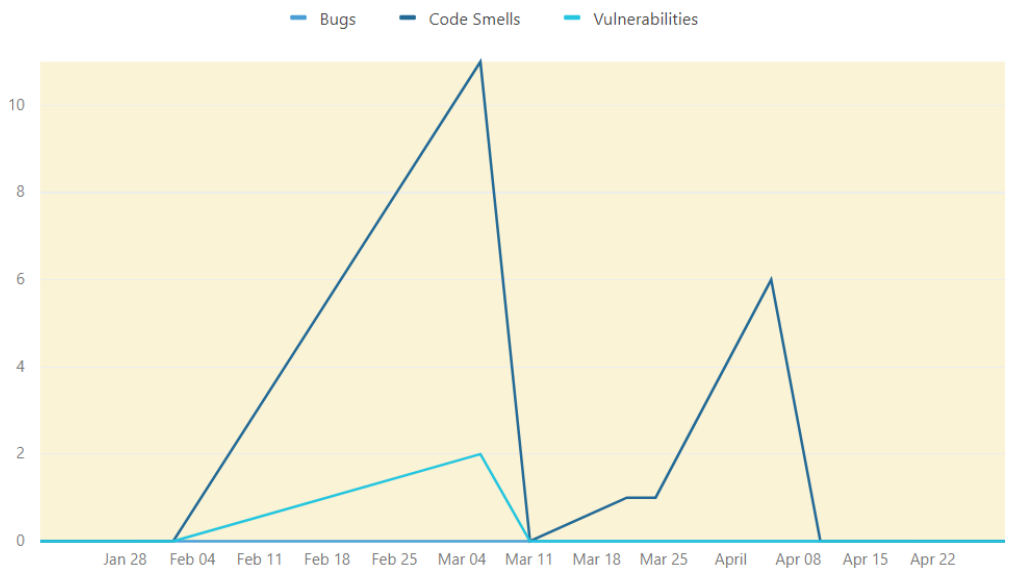
7.3 Mobilapplikasjonen

Under implementasjon av mobilapplikasjonen benyttet gruppen en rekke verktøy og metoder for å forsikre god kodekvalitet. Skanner av prosjektet med SonarQube ble utført regelmessig for å ha oversikt over bugs og sårbarheter. I tillegg var det en fordel at analysen også inkluderte grad av duplisering. Duplisering var en utfordring ettersom mye av koden var felles for oppretting av en ny rapport og endring av en som allerede eksisterer. Videre benyttet gruppen både SonarLint og ESLint i Visual Studio Code for ytterligere tilbakemeldinger på kodekvaliteten. Utenom dette ble det også satt av en del tid til refaktorering av koden.

SonarQube-analyse

Som nevnt ble SonarQube benyttet regelmessig for å holde oversikt over nåværende kodekvalitet og generell status. Utenom informasjon om ren kodekvalitet fikk gruppen også verifisert totalt antall linjer med kode og hvor stor del av koden som var duplisert. Disse resultatene ga gruppen stor motivasjon til å refaktorere koden. I figur 34 er en oversikt over alle bugs og sårbarheter som ble oppdaget av SonarQube i mobilapplikasjonen i løpet av utviklingsprosessen. Som grafen også viser har gruppen i perioder hatt mye fokus på ny funksjonalitet. Deretter kommer det perioder der grafen går nedover ettersom gruppen har satt av tid tid til å fikse bugs og refaktorere koden.

Det er tydelig at en stor grad av ny funksjonalitet ble implementert fra starten av februar til slutten av februar. Dette vises i figur 34 der grafen bratt går oppover i nevnt periode. Dette stemmer godt overens med funksjonaliteten som ble implementert i denne perioden, forklart i seksjon 3.2.5. Videre går grafen svært brått ned i starten av mars. Dette er perioden der gruppen gjennomførte sprint 3 som ble dedikert til å fikse bugs og sårbarheter (for å klargjøre applikasjonen til brukertest). Videre stiger grafen igjen fra mars til tidlig i april. Økningen av bugs kommer igjen av at ny funksjonalitet ble implementert i sprint 5 der kart og offline-modus ble implementert. I tillegg ble det utført en ny etappe med refaktorering i starten av april før nevnt sprint var omme. Videre har gruppen fremdeles utviklet i perioden etter, men hatt større fokus på å fikse bugs underveis. Derfor er grafen svært lav resten av utviklingsperioden.



Figur 34: Antall bugs og sårbarheter som SonarQube registrerte i mobilapplikasjonen i løpet av utviklingsperioden

SonarLint

I tillegg til SonarQube ble SonarLint benyttet i Visual Studio Code. Dette er en utvidelse som enkelt ble installert i teksteditoren. Deretter fungerer SonarLint ved å automatisk lokalisere installert JRE [63]. Når linten er installert oppdager den automatisk feil underveis når koden utvikles. En av de største fordelene med denne linten var at den tillot gruppen å se eventuelle feil før en pull request ble opprettet. Et resultat av dette var at pull requests allerede hadde fått en form for validering av kodekvalitet før et annet gruppemedlem utførte code review.

ESLint

En annen lint gruppen benyttet i mobilutviklingen var ESLint. Denne linten registrerte mange av de samme feilene som SonarLint, men det var andre fordeler ved å benytte ESLint. Med ESLint kunne gruppen blant annet definere at linten skulle oppdage bugs i et ES6-miljø [64]. Med denne innstillingen fikk gruppen ytterligere validering av kodekvaliteten som var mer rettet mot React Native-syntaks. Videre hadde bruken av denne linten den samme fordelen som SonarLint i forhold til validering av kode før pull request.

Refaktorering

Som bekreftet i figur 34 refaktorerte gruppen koden i mobilapplikasjonen i to større etapper (mars og april). Ved refaktorering hadde gruppen spesielt mye fokus på følgende områder:

- Definere komponenter som kan gjenbrukes og øke kodens leslighet.
- Opprette hjelpebiblioteker som skiller ut aktuell kode. Dette var for å gi applikasjonen en bedre struktur og øke lesbarhet.
- Redusere syklomatisk kompleksitet for å øke lesbarhet, redusere kompleksitet og minimere risiko for uønsket funksjonalitet.

- Redusere duplisering for å forebygge og unngå fremtidige feil.

Ett av disse fokusområdene er illustrert i listing 7.1 og listing 7.2. Den førstnevnte kodesnutten viser hvordan koden så ut før refaktorering. Dette er kode for å lagre en ny aktivitet (bestående av en art i tillegg til skutt- og sett-verdier) i en rapport i databasen via en API-forespørsel. I listing 7.1 benyttes *Fetch API* for å sende en forespørsel til API-et. Forespørselen er definert med ES6-syntaks. Videre sendes det med en *body* som tar opp mye plass. Til slutt håndteres resultatene.

```
1  _sendActivity(activity, speciesId, reportId) {
2
3          fetch(settings.API_ACTIVITY, {
4              method: 'POST',
5              headers: {
6                  'Accept': 'application/vnd.api+json',
7                  'Content-Type': 'application/vnd.api+json',
8              },
9              body: JSON.stringify({
10                 "data": {
11                     "type": "activity",
12                     "attributes": {
13                         "seen": activity.seen,
14                         "shot": activity.shot,
15                     },
16                     "relationships": {
17                         "species": {
18                             "data": {
19                                 "id": speciesId,
20                                 "type": "species",
21                             }
22                         },
23                         "report": {
24                             "data": {
25                                 "id": reportId,
26                                 "type": "report",
27                             }
28                         }
29                     }
30                 }
31             }),
32         })
33         .then((response) => response.json())
34         .then((response) => {
35
36             // Request was successful.
37             if (response.data) {
38                 return true;
39             } else {
40                 return false;
41             }
42
43         })
44         .catch((error) => {
45           alert("Aktiviteten sendes inn: \n" + error);
46         });
47     }
```

Listing 7.1: Opprette ny aktivitet før refaktorering av mobilapplikasjonen

For å refaktorere koden i listing 7.1 valgte gruppen å implementere et eget bibliotek for API-forespørsler. Dette er det samme biblioteket som ble omtalt i seksjon 6.1.3. I tillegg ble også forespørslene til API-et implementert på samme vis som illustrert i seksjon 6.1.6. Dermed ble den originale koden på 47 linjer redusert til 11 linjer som

vist i listing 7.2. Antall linjer med kode ble med andre ord redusert med 80 %. I listing 7.2 henter koden først ut datastrukturen i riktig format med hjelp av funksjonen *API.getActivityData(...)*. Deretter prøver funksjonen å hente responsen ved å vente på svaret fra API-et. Her benyttes en funksjon kalt *API.post(...)* som ber API-et om å legge til en ny aktivitet. Til slutt kan responsen sin status returneres.

```
1     async _sendActivity(activity, speciesId, reportId) {
2         const data = API.getActivityData(activity.seen, activity.shot, speciesId
              , reportId);
3
4         try {
5             const response = await API.post(settings.API_ACTIVITY, data);
6             return response.ok;
7
8         } catch (error) {
9             Helpers.simpleFeedback(I18n.t('failTitle'), I18n.t('sendReportFail')
                  + error);
10        }
11    }
```

Listing 7.2: Opprette ny aktivitet etter refaktorering av mobilapplikasjonen

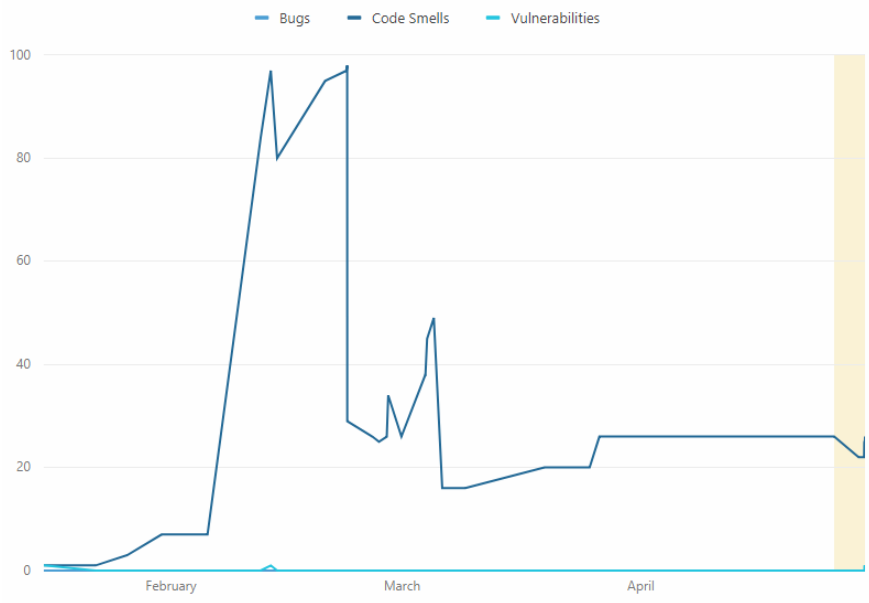
7.4 Backend

Kodekvaliteten på backend ble sikret ved å utføre automatiserte unit-tester, code review, statisk analyse og jevnlig refaktorering. I tillegg ble kommentarene og beskrivelsene til klasser og funksjoner skrevet før koden ble skrevet, slik at utvikler fikk et bedre innblikk i hva kodens oppgave var.

SonarQube-analyse og refaktorering

Det ble kjørt en statisk analyse hver gang alle unit-testene gikk gjennom, etterfulgt av en refaktorering av bugs og code smells (les mer om den automatiske testingen i seksjon 8.2). En forbedring i responstid ble spesielt merkbar etter en sprint dedikert til refaktorering. Dette kommer frem i figur 35; i starten av mars er det en kraftig nedgang i code smells, etterfulgt av et stabilt nivå for den resterende utviklingsprosessen. Dette er code smells som gruppen ikke kan fikse, da de er resultat av implementeringer som kreves av bibliotekene.

I figur 35 er det en relativt stor oppgang i code smells, etterfulgt av en tilsvarende nedgang. På dette stadiet i utviklingen, ble det innført en implementering av en OAuth 2.0-provider og JWT. Tanken bak dette var å innføre en «state-of-the-art»-løsning for autorisering av API-klientene [62]. Etter en test-implementasjon av denne, medførte det mange code smells. Etter implementasjonen viste det seg at endepunkts som Katharsis automatisk eksponerer, ikke ble lagt under sikkerhetslaget selv om dette ble spesifisert i konfigurasjonen til SpringBoot. Etter flere interne samtaler, og konsultasjon med produkteier, ble det vedtatt at implementasjonen av API-autentisering skulle utsettes, og muligens ikke tas i betraktning før etter prototype-stadiet. Sikkerhetsmodulen ble dermed fjernet fra kodebasen, som er grunnlaget for den drastiske nedgangen i code smells noen dager senere.



Figur 35: Antall bugs og sårbarheter på backend i løpet av utviklingsperioden

Refaktorering ble utført gjevnlig på eget initativ eller som resultat av et code review, spesielt under sprinter der fokuset var refaktorering. Ettersom gruppen benyttet mange biblioteker med ulike måter å løse oppgaver på, ble fokuset for refaktoreringen å påse at den beste løsningen ble benyttet. I tillegg ble funksjoner og dataflyt optimalisert for raskest mulig behandling. Et eksempel på dette er implementasjonen av filter. Gruppen benyttet filter for å definere lese- og skrivetilgang til ressurser, der det først ble benyttet ett filter per ressurs. Dette ble samlet til ett filter med en switch-statement (listing 7.3), noe som gjorde kallene mer effektive og reduserte kodemassen betraktelig: Før refaktorering var alle switch-case-ene en egen implementering av `Filter`-klassen.

```
1  switch (resourceName) {
2      case "activity":
3          chain.doFilter(request, response);
4          break;
5      case "area":
6          allowGetOnly(request, response, chain);
7          break;
8      case "municipality":
9          allowGetOnly(request, response, chain);
10         break;
11     case "person":
12         if (req.getMethod().equals("GET") && urlElements.length > 3)
13             chain.doFilter(request, response);
14         else
15             throw new ForbiddenException();
16         break;
17     case "region":
18         allowGetOnly(request, response, chain);
19         break;
20     case "report":
21         chain.doFilter(request, response);
22         break;
23     case "species":
24         allowGetOnly(request, response, chain);
25         break;
```

```
26      case "activityCoordinate":
27          chain.doFilter(request, response);
28          break;
29      case "areaCoordinate":
30          allowGetOnly(request, response, chain);
31          break;
32      default:
33          throw new NotFoundException();
34  }
```

Listing 7.3: Filter for lese-/skrive-tilgang etter refaktorering

7.5 Web-løsningen

Kodebasen til web-løsningen er relativt liten, så gruppen ble enige om at code review var tilstrekkelig for å sikre god kodekvalitet. Dette ble utført på tilsvarende måte som for de andre kodebasene.

8 Testing

8.1 Brukertesting av mobilapplikasjonen

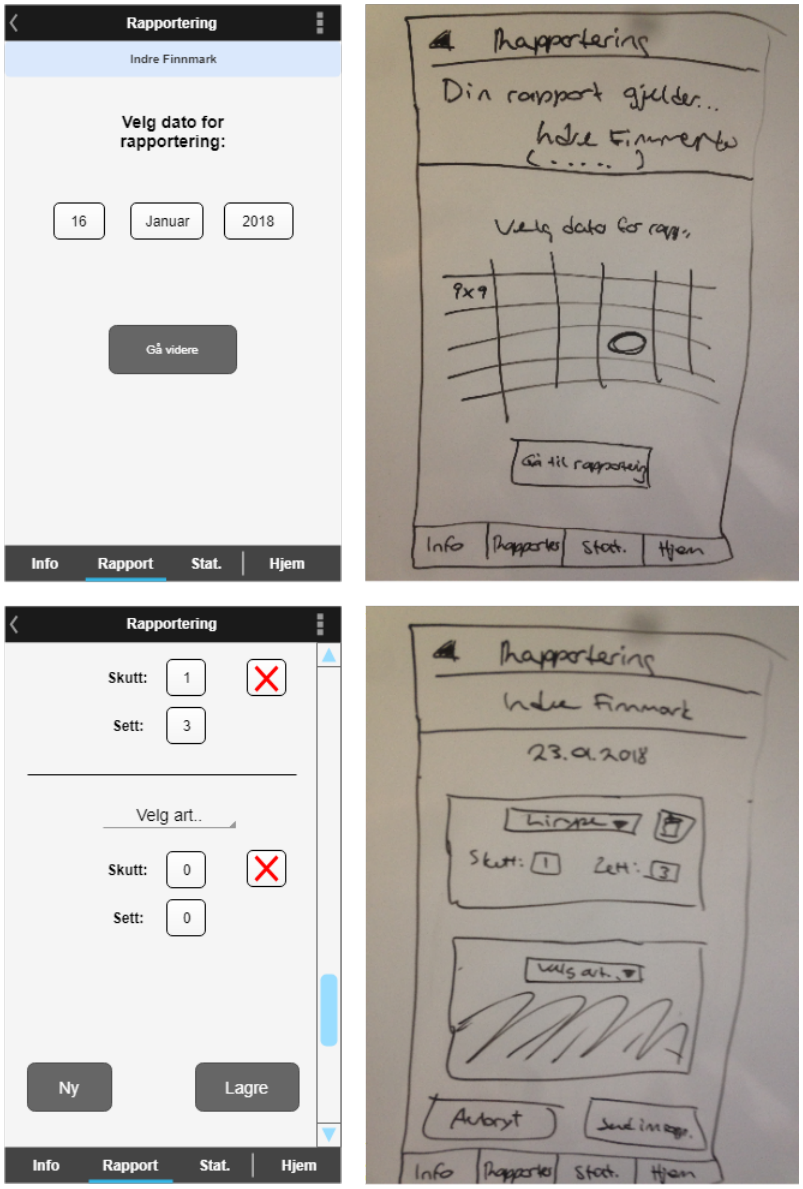
Gruppen hadde mye fokus på brukertesting i løpet av utviklingsprosessen. Det ble utført både formelle og uformelle brukertester. De største brukertestene utførte vi med Eivind Arnstein Johansen (interaksjonsdesign-ekspert) og med en dedikert gruppe med jegere. Utenom dette ble det også utført et par uformelle brukertester med nære venner og familie.

Interaksjonsdesign-ekspert

Den første brukertestingen gruppen foretok seg var med interaksjonsdesign-eksperten med prototype nr. 2 (se figur 11). Brukertestingen fant sted på vedkommendes kontor 23.01.2018 (se vedlegg B for grove notater). Brukertesten ble holdt ved at eksperten fikk låne en dedikert PC med mobilapplikasjonen installert på en emulator. Deretter skulle han prøve å opprette en rapport uten hjelp. Vi observerte hvordan eksperten prøvde å løse oppgaven samtidig som han tenkte høyt. Dette er hovedpunktene fra testens resultat:

- Innlogging kunne vært mer personlig enn jegernummer. Det er ikke en selvfølge at jegeren husker et nummer på opp til 8 siffer. Dersom jegernummer uansett benyttes er det aktuelt å ha en «Glemt jegernummer»-knapp.
- Tilbakemeldinger og informasjon burde være så intuitivt som mulig. Det er viktig at brukeren forstår nøyaktig hva som skjer dersom han/hun klikker på en knapp eller «går videre».
- Gruppen burde undersøke mer om hvilken informasjon som egentlig er tilgjengelig, og bruke dette for å gjøre brukeropplevelsen raskere og enklere.
- Gruppen burde gjøre seg sikre på at begreper i applikasjonen gjenkjennes av jegeren og at f.eks. «region» ikke forveksles med Norges regioner i stedet for en forvalters region.
- Det må være mulig å velge et annet jaktområde etter innlogging. Før rapporten startes burde også valgt område bekreftes.
- Et alternativ til kalender kan være å ta med ukedag i «scroll wheel» ved valg av dato. Indikasjonen på valgt dato må være tydelig. De nærmeste dagene kan gjerne beskrives som 'i dag' og 'i går' o.l.
- Dersom man har satt i gang en prosess er det viktig at man har muligheten til å gå tilbake eller avbryte.
- Sett- og skuttverdiene burde settes ved siden av hverandre for å spare plass. Dersom et nytt felt skal vises før det forrige er «ferdig» kan dette vises som inaktivt under.
- Jegeren burde få en oppsummering av rapporten ved lagring. I tillegg bør brukeren bli bedt om å bekrefte handlingen først dersom handlingen er destruktiv (slette art, slette rapport).
- Dersom en kalender blir tatt i bruk bør hver dagsrute være på minst 9x9 millimeter.

Etter denne brukertesten hadde gruppen en lengre diskusjon der designet og brukervennlighet ble diskutert i detalj. Gruppen utarbeidet deretter den tredje prototypen. Noen av endringene som ble gjort ble er illustrert i figur 36. Les og se mer på de endringene som ble gjort etter denne brukertesten i seksjon 5.3.



Figur 36: Prototype før og etter brukertest med interaksjonsdesign-ekspert

Jegergruppe

Videre fikk gruppen mulighet til å brukerteste en reell brukergruppe; en gruppe bestående av fire jegere. Dette var jegere som produkteier kjente til og jaktet sammen med på egen fritid. I tillegg deltok produkteier i brukertesten som en femte tester.

Det ble bestemt at testen skulle utføres ved å gi jegerne minst mulig hjelp. I forkant av testen definerte gruppen et skriv som forklare de ulike oppgavene testerne skulle utføre. Det var tre oppgaver; å opprette en rapport, å endre en rapport og slette en rapport fra et annet jaktområde. Hele skrivet finnes i vedlegg C. Omstendighetene for brukertesten var en hytte som jegerne benyttet under jakt. Produkteier installerte mobilapplikasjonen på en dedikert Android-telefon og tok med denne og gruppens skriv til brukertesten med til hytta. Der utførte jegerne brukertesten uten oppsyn. Grunnen til at gruppen utførte brukertesten på denne måten var at det var vanskelig å samle jegerne på et felles sted på en annen måte. I tillegg viste det seg å være en stor utfordring å finne et passende, og gratis, verktøy til å støtte brukertesten. Gruppen undersøkte et stort antall remote brukertest-verktøy uten hell. Blant disse undersøkte gruppen spesielt verktøy som støttet opptak og video av brukertester.

Etter at testerne hadde gjennomført brukertesten ble alle bedt om å besvare et spørreskjema gruppen hadde definert i Google Forms (se link nederst i vedlegg C). Spørreskjemaet ble definert som en kvalitativ spørreundersøkelse, og hvert spørsmål var nøye formulert for å unngå ledende spørsmål. Tre av fem besvarte skjemaet, og resultatene fra disse tre er beskrevet i tabell 11.

Tema	**Tilbakemelinger**
Tekst	Én jeger hadde utfordringer med å forstå «Statskog» da vedkommende skulle velge aktivt jaktområde. Jegeren mente at dette kunne minne om statsallmenning og at «Fjellstyrene» er et ord man sjelden forholder seg til. De andre jegerne opplevde ingen problemer med teksten i applikasjonen og forstod hvilke handlinger som kunne utføres.
Knapper	Alle jegerne var fornøyde med størrelsen på knappene i applikasjonen. De var uten tvil store nok.
Ny rapport	Ved oppretting av en ny rapport svarte jegerne at det var enkelt og forståelig. Dette gikk uten problemer.
Endre rapport	To av jegerne skrev at endring av rapport gikk greit og uten problemer. Den tredje nevnte at det var noe uklart hva som lå bak de ulike valgene (gruppen tror denne jegeren refererte til teksten i navigasjonsmenyen).
Slette rapport	Sletting av en rapport innebar at jegerne måtte bytte jaktområde først. To av dem synes dette gikk helt fint, men en tredje tester nevnte også her at de ulike valgene var noe uklart.
Forbedringer	Ingen av jegerne hadde forslag til forbedringer utenom det som allerede var nevnt.

Tabell 11: Tilbakemeldinger fra jegergruppen som deltok i en brukertest

Nære venner og familie

Utenom de mer formelle brukertestene ble også mobilapplikasjonen brukertestet av gruppemedlemmenes nære venner og familie. Til tross for det uformelle forholdet til testerne valgte hvert gruppemedlem å utføre brukertesten ved å gi testerne minimalt med hjelp og veiledning. Fordelen med disse testene var at det ga gruppen muligheten til å observere det testerne gjorde underveis i brukertesten. Dette ga overraskende resultater som gruppen ikke hadde vurdert tidligere og som heller ikke ble avdekket av de mer formelle brukertestene. Under er en liste over verdifulle tilbakemeldinger som gruppen fikk gjennom disse brukertestene:

- Noen av Android-enhetene klarte ikke å laste inn bakgrunnsbildet i innloggingsskjermen.
- Det er nødvendig å ha et laste-symbol (spesielt ved innlasting av historikk) i applikasjonen for å øke brukerens tålmodighet.
- Navigasjonen med tilbakeknapp (både i applikasjonen óg på Android-enheter) fungerte ikke alltid som ønsket.
- Hele «kortet» for en rapport i historikk burde være klikkbart; mange av testerne trykket bare på kortet og ikke på rediger-ikonet.
- Det burde være mulig å benytte en «sveipebevegelse» for å bla gjennom månedene i kalenderen.

8.2 Backend

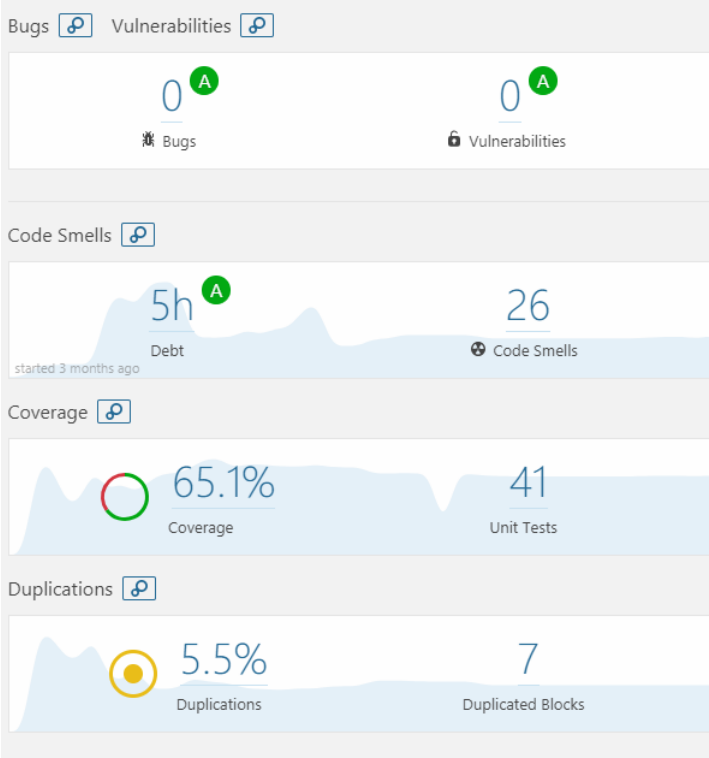
Backend ble grundig testet gjennom automatiserte unit-tester, statisk analyse og faktisk bruk under utviklingen. Alle testene ble kjørt hver dag det foregikk utvikling. Det ble benyttet flere biblioteker som spesialiserer seg på API-testing og JSON-validering. Disse bibliotekene er oppsummert i tabell 12.

Rammeverk	**Beskrivelse**
JUnit	Et mye brukt og velkjent rammeverk for å skrive standard unit-tester.
RESTAssured	Et rammeverk for å teste RESTful API-er.
SpringBootTest	Et rammeverk for å teste Spring Boot applikasjoner.
RESTAssured JSON Schema Validator	Et rammeverk for å validere JSON. Er en del av *RESTAssured*.
Hamcrest	Et rammeverk for å sammenlikne forventede og faktiske verdier i tester.

Tabell 12: Oppsummering av rammeverk benyttet for testing av backend

Unit-tester

JUnit ble benyttet for å teste spesifikk logikk som ikke var direkte koblet til et endepunkt. I alt 41 unit-tester ble skrevet for backend, som dekker 65.1% av kodebasen. Dette er illustrert i figur 37. Videre ble SpringBootTest benyttet for å opprette en SpringBoot-server i et virtuelt test-miljø. Dette biblioteket oppretter et lokalt nettverk og en ny test-database, slik at det kan kjøres automatiserte tester som faktiske forespørsler. Disse forespørslene ble sendt ved hjelp av *RESTAssured*, som også tok imot responsen. Responsen ble så validert gjennom et JSON Schema som validerer JSON-responsen mot JSON API-



Figur 37: Rapport fra Sonar over status på backend etter utvikling

spesifikasjonen. Dette ble utført av *JSON Schema Validator*, som er en del av *RESTAssured*. Deretter ble responskodene og verdiene verifisert, der dette var nødvendig. Ett eksempel på en slik test er vist i listing 8.1, der det testes at klienten ikke kan opprette nye brukere gjennom API-et. Her testes det også at responskoden og respons-dataene er korrekt.

Når alle testene passerte ble det utført en statisk analyse i SonarQube, som beskrevet i seksjon 7.4.

```
1  @Test
2   public void d_createPersonTest() {
3    String body = new JsonApiJsonBuilder()
4      .setResourceType("person")
5      .build();
6
7    given()
8     .body(body)
9     .header(accept)
10    .header(contentType)
11    .post("/api/person")
12     .then()
13     .assertThat()
14     .statusCode(is(HttpResponseCodes.FORBIDDEN.toInt()))
15      .and()
16      .assertThat()
17      .body("status", is(HttpResponseCodes.FORBIDDEN.toInt()));
18  }
```

Listing 8.1: En unit-test skrevet for backend

9 Veien til produksjon

I dette kapittelet beskrives den prosessen som gjenstår før alle delene av løsningen kan produktsettes. Her diskuteres de ulike aspektene ved å tilgjengeliggjøre mobilapplikasjon i App Store og Play Store. I tillegg beskrives nødvendige tiltak for å produksjonssette backend og web-løsningen.

9.1 Mobilapplikasjonen

For å lansere mobilapplikasjonen på Apple og Google sine app-butikker må de aktuelle kravene være oppfylt. Tidligere utførte Apple og Google validering av applikasjoner svært ulikt; Apple kjørte en manuell tilnærming mens Google benyttet seg av en automatisk gjennomgang ved hjelp av algoritmer. Siden 2015 har Google også benyttet seg av manuell validering, men dette har fungert som et ekstra lag etter den automatiske gjennomgangen [65]. Dette betyr at valideringen har ulik prosesseringstid; Apple skriver at gjennomsnittlig 50 % av alle apper blir validert innen 24 timer og at 90 % blir validert innen 48 timer [66]. Google har ikke spesifisert en konkret prosesseringstid, men i følge et innlegg på *androidb* er prosesseringstiden som regel på under 2 timer [65].

Apple

Apple har i hovedsak 5 seksjoner de deler kravene sine inn i [67]. Tabell 13 viser en oversikt over tiltak gruppen tok for å støtte de mest essensielle retningslinjene innen hver seksjon.

Krav	Tiltak
Sikkerhet	Applikasjonen viser ikke urovekkende eller støtende innhold.
Ytelse	Applikasjonen er omfattende testet og det eksisterer en «demo bruker» som kan benyttes til testing. Backend-tjenesten må kjøre under selve revideringen for at enkelte funksjonaliteter skal fungere på riktig måte.
Business	Applikasjonen har ingen definert business-plan.
Design	Applikasjonen har spesifikk funksjonalitet, den er ikke avhengig av en annen applikasjon og den spammer ikke brukeren.
Juridisk	Det foregår ikke og oppfordres ikke til kriminelle handlinger i applikasjonen. Fullstendig eierrett går til produkteier. Appen informerer brukeren om hvilken informasjon som lagres.

Tabell 13: Tiltak for å lansere applikasjonen i App Store

Utenom dette forventes det at det inkluderes en liste over de ikke-åpenbare funksjonene til appen og at appen generelt følger veiledningen som er gitt angående programmeringsstandarder (benyttelse av API-er o.l), designstandarder og markedsføringsstandarder.

Google

Google deler også sine krav inn i seksjoner [68]. Tiltak for å støtte Google sine retningslinjer er beskrevet i tabell 14.

Krav	Tiltak
Begrenset innhold	Applikasjonen innholder ikke upassende innhold som pornografi, vold eller oppfordringer til ulovlige aktiviteter.
Falsk identitet og åndsverk	Applikasjonen benytter ikke villedende titler eller ikoner, og den er original. Fullstendig eierrett faller går til produkteier.
Personvern, sikkerhet og villedelse	Applikasjonen er åpen om hva slags informasjon som lagres. Videre så følges lovverk og data stjeles ikke fra brukeren.
Inntektsgenerering og annonser	Applikasjonen inneholder ikke innhold som krever betaling eller annonser.
Butikkoppføring og markedsføring	Applikasjonen har ingen markedsføring. Alle varsler i appen benytter standarder for varsling.
Nettsøppel og minimumsfunksjon	Applikasjonen har spesifikk funksjonalitet og det er ingen forekomst av nettsøppel.
Familier og COPPA	Applikasjonen har ikke familier eller barn som målgruppe eller brukergruppe. Målgruppen er jegere.
Håndhevelse	Applikasjonen følger de retningslinjene Google oppgir.
Oppdateringer og andre ressurser	Applikasjonen oppfylte kravene da den var under gruppens ansvar. Dette ansvaret vil produkteier få når han overtar produktet.

Tabell 14: Tiltak for å lansere applikasjonen i Play Store

Utenom dette kreves det en plan for samtidig utrulling av utgaver, at det er definert hvilke enheter applikasjonen er kompitabel med, at land appen skal distribueres i er definert og at pris er satt.

Tiltak

Under er en liste med punkter gruppen mener er nødvendig å utarbeide for å lansere applikasjonen i Play Store óg App Store:

- Legge til metadata.
- Bestemme navn på applikasjonen, definiere business-plan og prisdetaljer, og spesifisere distribueringsland.
- Se mer på muligheten for å kjøre applikasjonen på andre iOS-enheter (spesielt iPad).
- Tydeliggjøre hvordan brukerens data benyttes og til hvilke formål.
- Vurdere om jakt kan regnes som mulig faktor for potensiell fysisk skade.

Programvarekravene under ytelse nevner kravet: «Applikasjonen må kun benytte de

offentlige API-ene og være kjørbar på nyeste versjon av OS»[67]. Applikasjonen gruppen har utviklet er begrenset av hvorvidt React Native-rammeverket er oppdatert og hvilke API-er det benyttes i bakgrunnen av rammeverket.

9.2 Backend

For å sette backend-serveren i produksjon kreves en installert versjon av Maven 3, Java 8 og en kjørende database-server som er støttet av Hibernate ORM. Deretter må serveren konfigureres ved å endre filen `src/main/resources/application.properties`. Her må man endre `spring.datasource.url` til å peke på database-serveren, og angi hvilken bruker som skal benyttes for å endre databasen. Deretter kan serveren pakkes ved å kjøre kommandoen `mvn verify` og startes ved å kjøre følgende kommando:

```
java -jar target/svr-backend-x.y.z.jar
```

9.3 Web-løsningen

For å sette web-løsningen i produksjon kreves det at backend-serveren er satt opp og kjører. I tillegg må repository-et klones til en mappe på serveren som er eksponert gjennom en web-server. Under utviklingen ble det benyttet en apache2-server for Ubuntu server 17.04. I filen `Global.js` må variabelen `API_URL` settes til backend-serverens IP/domene.

10 Utfall

I dette kapittelet drøftes utfallet av prosjektet. De ulike resultatene beskrives og gruppen diskuterer videre arbeid (les om de ulike prosjektmålene i vedlegg E). Videre utfører gruppen en evaluering av arbeidet og gir kritikk. I tillegg defineres en konklusjon for prosjektet.

10.1 Resultater

I løpet av prosjektperioden har gruppen utviklet en mobilapplikasjon med kryssplattform-teknologien React Native. Applikasjonen har ingen direkte kobling mot statlige organer, men det ble utviklet et RESTful API som kan benyttes av eksterne systemer (slik som Brønnøysundregistrene og SSB). I tillegg har gruppen utviklet en ferdig web-løsning for forvaltere som viser statistikk basert på data generert i mobilapplikasjonen. Med disse resultatene mener gruppen at resultatmålet om en klar prototype er oppnådd.

Effektmålet som angår effektivisering innenfor behandling av rapporter mener gruppen vil bli oppnådd etter integrasjon med offentlige organer. Dette er et resultat av at rapportering blir digitalisert. Behandlingen vil bli mer effektiv ettersom alle brukergrupper kan forholde seg til ett system. Videre estimerer gruppen at presisjon av data vil økes med minst 10 % ettersom plotting av småvilt på et kart ble implementert. I tillegg vil rapporteringstiden for jegeren reduseres betraktelig av samme grunn som behandling av rapporter blir effektivisert. Jegeren kan forholde seg til én plattform for rapportering og unngår eventuelt papirarbeid.

Utenom dette mener gruppen at alle læringsmålene som ble satt har blitt oppnådd. Gjennom prosjektarbeidet har gruppen tilegnet seg kunnskap om hvordan man utvikler mobilapplikasjoner med kryssplattform-teknologien React Native. Videre har gruppen også blitt kjent med RESTful API og har satt opp et slikt selv med JSON-standard. Arkikteturen ble planlagt og lagdelingsmodellen ble valgt ut som et passende arkitekturpattern. Dette ble beskrevet i seksjon 5.1. Videre var det stor fokus på brukervennlighet i mobilapplikasjonen. I denne sammenhengen lærte gruppen mye om universell utforming og hvilke tiltak som burde benyttes. Disse tiltakene ble beskrevet i seksjon 5.3. I tillegg har vi utviklet med hjelp av den smidige arbeidsmetodikken Scrum gjennom hele utviklingsperioden på en profesjonell måte. Dette ble beskrevet i kapittel 3.

10.1.1 Alternativer

I løpet av utviklingsprosessen tok gruppen en rekke valg i forhold til arkitektur, teknologier og verktøy. Likevel kan det diskuteres at de største valgene som ble tatt omhandlet språk og rammeverk for mobilapplikasjonen, hvordan API-et skulle implementeres (diskutert i seksjon 4.1.2) og valg av teknologier i web-løsningen.

Cross platform eller native

Et ønske fra produkteier var at applikasjonen skulle utvikles for både Android og iOS. Likevel satte ikke produkteier et bestemt krav for valg av teknologi for mobilapplikasjonen.

Det førte til et stort spørsmål: skulle mobilapplikasjonen utvikles som to native apper (én for iOS med Swift og én for Android med Java), eller som én kryssplattformapplikasjon? Under er en liste med punkter som gruppen diskuterte:

- Gruppens kompetanse:
  - Alle gruppemedlemmene hadde tidligere hatt et emne i Java. Gruppen visste også at emnet Mobile Developement skulle holdes samtidig med bachelor-oppgaven. I dette emnet kom gruppen til å lage Android-applikasjoner. Videre var ingen av gruppemedlemmene kjent med verken React Native eller Swift. React Native skrives med Javascript-syntaks, noe alle gruppemedlemmene var kjent med. Uavhengig at valg måtte gruppen lære å utvikle for en ukjent plattform.
- Apple- og Google-spesifikke elementer:
  - React Native er laget av Facebook, og er ikke støttet av hverken Apple eller Google. Dette gjør at all ny funksjonalitet i deres OS deres kan forårsake at React Native ikke lenger kan benyttes før OS-et oppdateres. [14].
  - Både Apple og Google har laget sine egne retningslinjer innenfor design. *Material Design* fra Google, og *Human Interface Guidelines* fra Apple. Det kunne bli komplisert å opprettholde begge standardene på en kryssplattformapplikasjon.
- Generelle fordeler og ulemper:
  - Dersom gruppen utviklet applikasjonen med kryssplattform så åpnet det for et tettere samarbeid mellom medlemmene i gruppen. Det kunne tillate gruppen å fokusere på færre kodebaser og å løse problemene sentralt. Dersom gruppen utviklet for to plattformer samtidig ville vi sannsynligvis møtt de samme problemene, men i ulike språk.
  - Navigasjon på Android og iOS-enheter utføres noe ulikt; Android-enheter har en «fysisk» tilbakeknapp som iOS ikke har. Dette kan medføre ytterlige spesialtilfeller som må håndteres på en kryssplattform-app.
  - Ytelsen til kryssplattformapplikasjoner regnes som tregere enn standard native applikasjoner [69].

Gruppen valgte å benytte React Native hovedsakelig fordi rammeverket er raskere å implementere. I tillegg er det raskere å produksjonssette en kryssplattformapplikasjon enn å utvikle to isolerte applikasjoner. Kryssplattformapplikasjoner regnes ofte som tregere enn native apper, og de egnes ikke for komplekse applikasjoner som krever høy ytelse, men det passer bra for moderat komplekse applikasjonen [70]. Det ble estimert at ytelsen til en kryssplattformapplikasjon ville være tilstrekkelig for vårt produkt på grunn av mobilapplikasjonens omfang og kompleksitet.

JavaScript eller React

For webløsningen valgte gruppen å benytte JavaScript i kombinasjon med jQuery for å skape en enkel og skalerbar løsning som ikke krever noen omfattende installasjonsprosess. Videre ble *React.js* vurdert som et mulig rammeverk. React.js er laget for å bygge grafiske grensesnitt [71], og ble derfor mindre aktuelt da grensesnittet på web-løsningen

er minimalt. I tillegg ble mengden konfigurasjon i forhold til utbytte ved bruk av React.js en avgjørende faktor for valget, ettersom JavaScript ikke har behov for konfigurasjon. I tillegg må React.js kompileres og kjøres gjennom en spesialisert server. For å få en bedre kontroll over grensesnittet, valgte gruppen å benytte *Twitter Bootstrap 4* for å håndtere responsivitet og design.

10.2 Kritikk

Videre vil gruppen gi seg selv og arbeidsprosessen kritikk. Et tema som gruppen ville fokusert mer på er estimering. Under utviklingsperioden hadde gruppen en dårlig tendens til å kun estimere *user story*-oppgaver. Det oppstod ofte flere oppgaver som var relatert til en *user story*, men disse ble sjeldent estimert. Dermed hadde gruppen lett for å ta seg «vann over hodet». Et annet resultat av denne arbeidsmetoden var at Burndown Chart ikke ble like verdifulle.

I tillegg ville gruppen startet raskere med utviklingen. Dette var noe gruppen kjente på da sprint 3 og 4 (to uker) ble benyttet til stabilisering av mobilapplikasjonen og gjøre den klar til brukertesten med den dedikerte jegergruppen (se seksjon 8.1). Dersom utviklingen av grunnleggende funksjonalitet ble påbegynt tidligere hadde gruppen fått tid til å utvikle ekstra funksjonalitet eller bedre kvaliteten ytterligere innen kart og offline/online-modus.

Utenom dette ville gruppen hatt mye større fokus på sikkerhet fra starten. Dette var et aspekt som gjerne ble tatt hensyn til underveis i prosessen. En bedre metode ville vært å planlegge for sikkerhet i første delen av utviklingsperioden.

10.3 Videre arbeid

Både gruppen og produkteier mener at dette produktet har et stort potensiale og høy markedsverdi. Videre kan både mobilapplikasjonen og web-løsningen utvides med en rekke interessante funksjoner. Utenom potensialet for ny funksjonalitet er det også verdt å nevne at kjente bugs også må håndteres av utviklere som overtar prosjektet. De kjente feilene og manglene er beskrevet i vedlegg A.

Integrasjon med offentlige organer

Innenfor videre arbeid kan det diskuteres at integrasjon med offentlige organer er det mest essensielle og interessante. Ønsket til produkteier er at produktet gruppen har utviklet kan benyttes av staten. Arbeidet vil derfor gå ut på å integrere tjenesten med SSB sin rapporterings-kanal. I tillegg er også målet å integrere tjenesten med kanalene som forvaltere benytter (for eksempel inatur.no). Utenom dette kan det også være relevant å integrere tjenesten med Brønnøysundregistrene. Dette er fordi de tilbyr jegere å se relevante jaktdata for den innloggede brukeren.

For SSB sin del vil det være gunstig å benytte gruppen sitt RESTful API for å hente ut dataene som nå samles inn gjennom deres kanal. Deres kanal kan dermed elimineres, og deres datasystemer kan justeres til å hente ut ønsket informasjon gjennom gruppens API. I tillegg kan forvaltere benytte samme API for å hente ut dataene de trenger. I dette tilfellet er det essensielt at API-et har klare begrensninger på hvem som har tilgang til hvilke datasett. Det samme gjelder ved eventuell integrasjon med Brønnøysundregistrene.

Ny personopplysningslov

Et annet tema som må håndteres i større grad er forslaget til den nye personopplysningsloven (også kjent som GDPR) [58]. Med den nye loven er det flere aspekter av produktet som må forbedres innenfor personvern. I gruppens løsning lagres en jegers jegernummer, fullt navn og telefonnummer. Av disse dataene må det gjøres en vurdering på om alle dataene egentlig trengs eller ikke. For å ta hensyn til den nye loven må blant annet disse punktene oppfylles og støttes av applikasjonen:

- Brukeren må gjøres bevisst på hvilke persondata som applikasjonen har og trenger om brukeren (implementert).
- Det må opplyses om hvorfor opplysningene trengs og til hvilke formål.
- Brukeren må alltid ha mulighet til å rette eller endre persondataene som finnes i systemet.
- Brukeren også må få ha mulighet til å protestere på applikasjonens bruk av persondata.

Utenom persondataene som lagres i databasen må også de samme prinsippene støttes av applikasjonen ved bruk av kart. Som beskrevet under implementasjon av kart (se seksjon 6.1.4) vil applikasjonen prøve å hente jegerens nåværende posisjon. Dette må også tilrettelegges for den nye loven.

Vingeprøver

Muligheten til å ta vingeprøver en svært interessant idé som gruppen ikke fikk tid til å implementere. Dette innebærer å gi jegeren mulighet til å ta et bilde av vingene til en felt fugl. Deretter bør applikasjonen benytte en algoritme som kjenner igjen om fuglen er ung eller voksen. I dag sendes disse vingeprøvene inn gjennom post, og derfor er det svært nyttig å digitalisere denne obligatoriske oppgaven.

Lagre GPS-posisjon

Videre kan det være aktuelt å lagre jegerens GPS-posisjon ved rapportering dersom brukeren ikke har internettforbindelse. I dette tilfellet kan ikke brukeren benytte kart og plotting av småvilt. Derfor kan det være relevant å lagre GPS-posisjonen dersom enheten ikke har internettforbindelse. Dersom dette blir implementert kan den lagrede posisjonen (typisk brukerens jakthytte) benyttes som senter for kartet ved redigering av en rapport. Videre burde lagret GPS-posisjon kun benyttes så lenge posisjonen er på innsiden av jaktområdets polygon.

10.4 Evaluering av arbeidet

Gruppens tanker er at arbeidet har blitt planlagt og strukturert på en profesjonell måte. Selv er gruppen svært fornøyd med resultatene fra utviklingen.

Underveis har gruppen hatt svært mye fokus på å arbeide smidig og følge Scrum-prinsippene. Dette har vært til stor hjelp og støttet utviklingen på en god og profesjonell måte. Videre ble selve arbeidet fordelt på gruppemedlemmene på en fornuftig måte. To av utviklerne hadde hovedansvar for mobilapplikasjonen ettersom dette var ukjent terreng som ville kreve mye ressurser. Dermed fikk den tredje utvikleren hovedansvar for backend og web-løsningen. Denne fordelingen har fungert svært godt, og gruppen har

hatt mye fokus på å fordele kompetansen ved å vise og gå gjennom kode i fellesskap.

Videre er det verdt å nevne at gruppen liker prosjekt som arbeidsmetode svært godt. Vi mener at prosjektet har gitt oss gode muligheter til å lære mye av hverandre (óg på egenhånd) ettersom det gjør det mulig å arbeide med større og mer komplekse oppgaver.

10.5 Konklusjon

Etter denne prosjektperioden sitter gruppemedlemmene igjen med mye verdifull kunnskap og kompetanse om smidig utvikling, interessante teknologier og planlegging. Vi opplever at vi har blitt mye tryggere på vårt potensiale og evne til å løse store og komplekse oppgaver. Derfor vil vi gjerne takke hverandre for den støtten og kompetansen som har blitt delt gjennom hele utviklingsperioden.

Bibliografi

[1] Statistisk sentralbyrå. 2017. Aktive jegere. [Hentet: 24.01.2018]. URL: https://www.ssb.no/jord-skog-jakt-og-fiskeri/statistikker/jeja.

[2] Miljødirektoratet. 2018. Jakt i norge. [Hentet: 13.04.2018]. URL: http://www.miljodirektoratet.no/no/Tema/Jakt-og-fiske/Jakt-og-fangst/Jakt-i-Norge/.

[3] Brønnøysundregistrene. 2018. Jegerregisteret. [Hentet: 13.04.2018]. URL: https://www.brreg.no/person/jegerregisteret/.

[4] Statistisk sentralbyrå. 2018. Velkommen til statistisk sentralbyrås elektroniske rapporteringssystem. [Hentet: 12.04.2018]. URL: https://jakt.ssb.no/.

[5] Facebook Inc. 2018. React native - github repository. [Hentet: 17.01.2018]. URL: https://github.com/facebook/react-native.

[6] JSON API. 2015. Json api. [Hentet: 17.01.2018]. URL: http://jsonapi.org.

[7] Wikipedia. 2017. Scrum. [Hentet: 12.01.2018]. URL: https://no.wikipedia.org/w/index.php?title=Scrum&oldid=18026299.

[8] Wikipedia. 2018. Extreme programming. [Hentet: 12.01.2018]. URL: https://en.wikipedia.org/w/index.php?title=Extreme_programming&oldid=819859831.

[9] Wikipedia. 2016. Kanban. [Hentet: 12.01.2018]. URL: https://no.wikipedia.org/w/index.php?title=Kanban&oldid=16669179.

[10] Wikipedia. 2017. Lean software development. [Hentet: 11.01.2018]. URL: https://en.wikipedia.org/w/index.php?title=Lean_software_development&oldid=813094758.

[11] Cohn, M. 2018. What are story points? [Hentet: 16.04.2018]. URL: https://www.mountaingoatsoftware.com/blog/what-are-story-points.

[12] Endava. 2018. Scrum time - planning poker. [Hentet: 16.04.2018]. URL: https://play.google.com/store/apps/details?id=rs.pstech.scrumtimeplanningpoker.

[13] Bitbucket, A. 2018. Git feature branch workflow. [Hentet: 15.05.2018]. URL: https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow.

[14] Narayan, A. 2017. React native vs native ios/android. [Hentet: 06.04.2018]. URL: https://www.coursereport.com/blog/so-you-want-to-build-a-mobile-app-react-native-vs-native-mobile.

[15] Oleksander Knyga, S. H. 2017. React native vs real native apps. [Hentet: 06.04.2018]. URL: https://codeburst.io/react-native-vs-real-native-apps-ad890986f1f.

[16] Marsh, J. 2018. Php vs. java. [Hentet: 10.04.2018]. URL: https://www.upwork.com/hiring/development/php-vs-java/.

[17] Cracan, M. 2017. Web rest api benchmark on a real life application. [Hentet: 10.04.2018]. URL: https://medium.com/@mihaigeorge.c/web-rest-api-benchmark-on-a-real-life-application-ebb743a5d7a3.

[18] InfoWorld. 2017. Peter wayner. [Hentet: 07.05.2018]. URL: https://www.infoworld.com/article/2883328/node-js/java-vs-nodejs-an-epic-battle-for-developer-mindshare.html.

[19] JSON API. 2015. Implementations. [Hentet: 29.01.2018]. URL: http://jsonapi.org/implementations/#server-libraries-java.

[20] Crnk. 2017. Crnk: Json api library. [Hentet: 29.01.2018]. URL: http://www.crnk.io/.

[21] Elide. 2018. Elide. [Hentet: 29.01.2018]. URL: http://elide.io/.

[22] Katharsis. 2016. Katharsis. [Hentet: 29.01.2018]. URL: http://katharsis.io/.

[23] Pivotal Software Inc. 2018. Spring: The source for modern java. [Hentet: 10.04.2018]. URL: https://spring.io/.

[24] Microsoft. 2018. Visual studio code. [Hentet: 08.04.2018]. URL: https://code.visualstudio.com/.

[25] Google. 2018. Everything you need to build on android. [Hentet: 08.04.2018]. URL: https://developer.android.com/studio/features.html.

[26] Apple Inc. 2018. What's new in xcode 9. [Hentet: 08.04.2018]. URL: https://developer.apple.com/xcode/.

[27] Node.js Foundation. 2018. Nodejs homepage. [Hentet: 24.01.2018]. URL: https://nodejs.org/en/.

[28] Expo.io. 2018. Expo homepage. [Hentet: 24.01.2018]. URL: https://expo.io/.

[29] Facebook Inc. 2018. React native getting started. [Hentet: 24.01.2018]. URL: https://facebook.github.io/react-native/docs/getting-started.html.

[30] Apple Inc. 2018. Distribute to registered devices (ios, tvos, watchos). [Hentet: 13.05.2018]. URL: https://help.apple.com/xcode/mac/current/#/dev7ccaf4d3c.

[31] JetBrains s.r.c. 2018. Intellij idea - choose your edition. [Hentet: 30.04.2018]. URL: https://www.jetbrains.com/idea/.

[32] The Apache Software Foundation. 2018. Apache maven project. [Hentet: 30.04.2018]. URL: https://maven.apache.org/.

[33] Wikipedia. 2018. Javadoc. [Hentet: 30.04.2018]. URL: https://no.wikipedia.org/w/index.php?title=Javadoc&oldid=17651382.

[34] Atom. 2018. Atom - a hackable text editor for the 21st century. [Hentet: 30.04.2018]. URL: https://atom.io/.

[35] Apache Friends. 2018. What is xampp? [Hentet: 30.04.2018]. URL: https://www.apachefriends.org/index.html.

[36] Richards, M. 2015. *Software Architecture Patterns*. O'Reilly Media, Inc., United States of America.

[37] Google. 2018. Material design. [Hentet: 16.02.2018]. URL: https://material.io/.

[38] Apple Inc. 2018. Human interface guidelines. [Hentet: 02.05.2018]. URL: https://developer.apple.com/ios/human-interface-guidelines/overview/themes/.

[39] Google. 2018. Color tool. [Hentet: 30.04.2018]. URL: https://material.io/color/#!/?view.left=0&view.right=0.

[40] Collinge, R. 2017. How to design for color blindness. [Hentet: 30.04.2018]. URL: http://blog.usabilla.com/how-to-design-for-color-blindness/.

[41] Difi. 2018. Kva er universell utforming? [Hentet: 30.04.2018]. URL: https://uu.difi.no/kva-er-universell-utforming.

[42] GitHub, Inc. 2018. Boostnote for ios and android. [Hentet: 08.05.2018]. URL: https://github.com/BoostIO/boostnote-mobile/tree/master/app.

[43] GitHub, Inc. 2018. Mobile app which calculate gasoline/oil ratio for 2 stroke engines. [Hentet: 08.05.2018]. URL: https://github.com/filippofilip95/gas-oil-mixture-mobile/tree/master/src.

[44] React Navigation Contributors. 2016. React navigation. [Hentet: 04.05.2018]. URL: https://reactnavigation.org/.

[45] Github user «JulienR2». 2017. react-native-maps. [Hentet: 28.04.2018]. URL: https://github.com/react-community/react-native-maps/blob/master/README.md.

[46] Google Developers. 2018. Pricing and plans. [Hentet: 29.04.2018]. URL: https://developers.google.com/maps/pricing-and-plans/#details.

[47] Google. 2018. Pricing and billing changes. [Hentet: 09.05.2018]. URL: https://cloud.google.com/maps-platform/user-guide/pricing-changes/.

[48] Kartverket. 2018. Lage kart på nett. [Hentet: 28.04.2018]. URL: https://www.kartverket.no/data/lage-kart-pa-nett/.

[49] Github user «substack» (James Halliday). 2018. point-in-polygon. [Hentet: 28.04.2018]. URL: https://github.com/substack/point-in-polygon/blob/master/index.js.

[50] Franklin, W. R. 2017. Pnpoly - point inclusion in polygon test. [Hentet: 28.04.2018]. URL: https://wrf.ecse.rpi.edu//Research/Short_Notes/pnpoly.html.

[51] Google. 2018. Google maps apis terms of service. [Hentet: 09.05.2018]. URL: https://developers.google.com/maps/terms.

[52] OutSystems. 2018. Read/write data one-to-many. [Hentet: 30.04.2018]. URL: https://success.outsystems.com/Documentation/10/Developing_an_Application/Use_Data/Offline/Offline_Data_Sync_Patterns/Read%2F%2FWrite_Data_One-to-Many.

[53] Mozilla and individual contributors. 2018. Using fetch. [Hentet: 03.05.2018]. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch.

[54] Facebook Inc. 2018. Networking. [Hentet: 03.05.2018]. URL: https://facebook.github.io/react-native/docs/network.html.

[55] Github user «oblador». 2018. react-native-vector-icons. [Hentet: 14.05.2018]. URL: https://github.com/oblador/react-native-vector-icons/blob/master/README.md.

[56] Github user «AlexanderZaytsev». 2018. react-native-i18n. [Hentet: 14.05.2018]. URL: https://github.com/AlexanderZaytsev/react-native-i18n/blob/master/README.md.

[57] Github user «react-community». 2018. react-native-i18n. [Hentet: 14.05.2018]. URL: https://github.com/react-community/react-native-maps/blob/master/docs/installation.md.

[58] Regjeringen.no. 2018. Forslag til ny personopplysningslov. [Hentet: 05.05.2018]. URL: https://www.regjeringen.no/no/aktuelt/forslag-til-ny-personopplysningslov/id2594896/.

[59] Oracle Corporation and/or its affiliates. 2018. About mysql. [Hentet: 24.01.2018]. URL: https://www.mysql.com/about/.

[60] Inc., F. 2018. Asynstorage. [Hentet: 14.05.2018]. URL: https://facebook.github.io/react-native/docs/asyncstorage.html.

[61] Oftedal, E., van der Stock, A., Chih, T. H. H., Peeters, J., Wolff, J., & Gränitz, R. 2018. Rest security cheat sheet. [Hentet: 14.05.2018]. URL: https://www.owasp.org/index.php?title=REST_Security_Cheat_Sheet&oldid=240363.

[62] Parecki, A. 2018. Oauth 2.0. [Hentet: 14.05.2018]. URL: https://oauth.net/2/.

[63] SonarSource S.A. 2018. How to get the most out of sonarlint? [Hentet: 03.05.2018]. URL: https://www.sonarlint.org/vscode/howto.html#analyze-your-code.

[64] JS Foundation and other contributors. 2018. About. [Hentet: 03.05.2018]. URL: https://eslint.org/docs/about/.

[65] SG, C. 2017. App review times: How fast will your app be published. [Hentet: 06.05.2018]. URL: https://www.androidb.com/2016/08/app-review-times-for-google-play-aka-how-fast-will-your-app-be-published/.

[66] Apple Inc. 2018. App review. [Hentet: 06.05.2018]. URL: https://developer.apple.com/support/app-review/.

[67] Apple Inc. 2018. App store review guidelines. [Hentet: 06.05.2018]. URL: https://developer.apple.com/app-store/review/guidelines/.

[68] Google Play. 2018. Retningslinjesenteret for utviklere. [Hentet: 06.05.2018]. URL: https://play.google.com/about/developer-content-policy/.

[69] DA-14. 2018. React native vs native. how to choose the best platform for mobile app development? [Hentet: 07.05.2018]. URL: https://da-14.com/blog/react-native-vs-native-how-choose-best-platform-mobile-app-development.

[70] Matyunina, J. 2017. React native vs native. how to choose the best platform for mobile app development. [Hentet: 07.05.2018]. URL: https://codetiburon.com/react-native-vs-native-choose-best-platform-mobile-app-development/.

[71] Facebook, I. 2018. React. [Hentet: 07.05.2018]. URL: https://reactjs.org/.

[72] Statistisk sentralbyrå. 2018. Jaktstatistikk. [Hentet: 17.01.2018]. URL: https://jakt.ssb.no/.

A Kjente bugs

Under er en liste over bugs som gruppen kjenner til innenfor de forskjellige delene av produktet. Listen er gitt slik at videreutvikling blir enklere for Escio som overtar produkt og utvikling.

Mobilapplikasjonen

- Teksten som lastes inn i historikk-tabben dersom det ikke finnes rapporter vises ikke.
- Navigasjonen må fikses fra redigering av en rapport til historikk. Dersom brukeren kun har gjort endringer i rapporten og sletter denne (gjelder både pil tilbake i tittel óg Android-knapp) går dette fint og brukeren blir navigert til historikk. Dersom brukeren har benyttet kart ved redigering av rapport og deretter sletter rapporten fungerer ikke funksjonaliteten for navigering til historikk (siden kartet inkluderes i stacken).
- Applikasjonen krasjer første gang Androids «back button» trykkes på i redigering av en rapport.
- En jeger kan fjerne alle aktiviteter i en rapport og lagre en tom rapport.
- Forbedring: funksjonalitet for å synkronisere jaktområder, regioner og arter finnes ikke enda. På dette tidspunktet finnes disse dataene som lokale JSON-filer som er kopiert fra API-spørringer.
- Forbedring: historikk-tabben burde vise synkroniseringsstatusen til hver rapport. Er rapporten synkronisert enda eller ikke?
- Mobilapplikasjonen må konfigureres for iOS for pakkene «react-native-vector-icons», «react-native-i18n» og «react-native-maps».

Backend

- Feilmeldinger mappes ikke alltid etter JSON API-standarden, men følger heller et standardformat generert av Spring Boot.

Web-løsningen

Ingen kjente bugs.

B Møtereferater

Her finnes referater av alle møter som er holdt så langt i forbindelse med bachelor-oppgaven. Møtene som refereres til er: møter med veileder, møter med produkteier og gruppemøter/diskusjoner som var avgjørende for oppgaven.

Gruppemøte 10.01.2018

- Fastsatte timeplan:
  - Man - Ons: 08:00 - 16:00 (8t)
  - Tor: evt. 2t etter forelesning
  - Fre: 08:00 - 12:00 (4t)
- Avtalte fast møtetidspunkt med veileder (Tom Røise) hver tirsdag kl. 14:30 til 15:00
- Definerte en grunnleggende/innledende plan for planlegging av prosjektarbeidet. (Prosjektplan)

Gruppemøte 12.01.2018

- Diskuterte Gantt-diagram
  - Argumenterte for å kjøre UX-prototype i planleggingsfasen, for å slippe store omveltninger i test-fasen (April/Mai)

Møte med arbeidsgiver 16.01.2018

- Spørsmål fra gruppa:
  - Mål og rammer: Effektmål:
    - Bedre datakvalitet for eiere av jaktområder
    - Bedre presisjon av data og statistikk
    - Spare tid for jegeren
    - Spare tid for databehandler
  - Rammer til applikasjonen
    - Android versjoner som skal støttes?
      - Mest utbredte versjon ila. siste to år
      - Kortest mulig tid tilbake
      - S7 til i dag
  - Språk:
    - Bruk jsonapi.org for å definere RESTful API mellom backend/frontend
    - React native i frontend
  - Android og/eller iOS?
    - Spiller ikke stor rolle for arbeidsgiver
    - Anbefaler Android

- Avgrensinger for oppgaven
  - Jobb mot en klient
  - Fokuser på mvp
- Offline/lagring
  - Definere hvordan man skal synkronisere

- Prototyp:
  - Parametre i jaktterrenget:
    - Ønsker fra jaktstedeier
    - Krav til Skutt/Observert rapportering forskjellig i forskjellige områder
  - Hardkode verdier fra hønsefulportalen
  - Inspill ang. design
    1. Spør om jegernummer
       - siffer
       - 6 tegn
    2. Autorisering (Senere)
    3. Velg område:
       - Sett dette som standard for bruker
       - Knapp for enkel måte å bytte sted
    4. Rapportere
       - Jegerinfo
       - Område (Hva skal rapporteres her defineres av en admin)
         - Sett og/eller skutt (optional for sett)
         - Art + antall (velges separat for skutt/observert)
       - Kunne rapportere hele jakten i en rapport?
       - Ta stilling til
         - Dato, men ikke klokkeslett (Må vere satt før 2)
         - Matrise (forslag):

```
                ART
           |art1|art2|...|art N|
skutt      |0      0    1    0
sett       |10     2    2    0
```

         - Få rapportert både skutt og sett i samme rapport
         - Tenk dag-for-dag rapportering
         - Endre tidligere rapporter
         - FK: jegernummer + dato + område Evt. rapportID
  - Datoer for jakt skal kunne defineres (ikke nødvendig vis dagens dato)
    - Første skjerm kalender: Dagens dato som standard

    - Arter
        1. Lirype
        2. Fjellrype
        3. Orrfugl
        4. Hare
        5. Rev
        6. Storfugl
- Kommentarer til scrum

TASK	Viktighet (Business Value)	Storypoints
1. Onboarding	100	40
2. Rapportering	80	80

    - storypoint skala: 0, 0.5, 1, 2, 3, 5, 8, 13, 20, 40, 80, 100
    - Velocity: Antall storypoints pr. sprint
    - Bruk referanseoppgave for å estimere tasker
    - Ikke bruk oppgaver som er større enn 20
    - Mest mulig verdi for minst mulig innsats
    - 50%estimat : 50% sjangse for at det tar lengre tid
    - 90%estimat : 10% sjangse for at oppgaven tar lengre tid
    - Akkumuler prioritert liste for å kunne planlegge sprint
    - Burndown-chart
    - Sprint-retrospective: internt, hvordan effektivisere neste sprint
    - Store oppgaver i backlog: Opprett sub-tasks på disse
    - Reservere Fredager til møter
    - Dummyverdier i DB, men må implementeres i API og klient
- Backlog
    - Bør skrives i Google-spreadsheet
    - Format: Som <rolle> trenger jeg <funksjonalitet> fordi <begrunnelse>
        · Legg inn alle stories i spreadsheet
        · product-owner legger inn BV
        · Estimer?
        · Legg inn i JIRA
    - Oppgaver: Referer til håndskrevet dokument
- Rapport: Vurder perspektiver rundt personvern og lagring/eierskap av brukerdata.
- Generelt
    - Enkelt og god ytelse
    - En del anti-patterns i React Native (Lukes ut av Sonar)
    - 1. sprint-planning meeting: Torsdag, 01.02.2018 kl. 9-10 (NTNU)
    - Omformuler MVP til minste lanserbare produkt (ikke skall)
- Underskrevet kontrakt

Møte med Veileder 16.01.2018

- Inkluder Eksterne systemer i UseCase
- Sprint-opplegg kan endres underveis.
- Kan diskutere estimeringsprosess med product-owner utover i prosessen.
- Code review: Fontent -> backend, backend -> frontend
- Krysskompatibel kunnskap: Frontend/backend, interne kurs.
- Avgjøre spesialisering innad i gruppen
- I rapport: Vis prosessen fra idé -> Prototyp -> produkt
- Diskutere svakheter/metodikk rundt brukertesting
- Bruk testgruppen godt. (Prøv først med medstudenter)
- Diskuter whinle design-prinsipper som skal følges i design, avgjørelser basert på omgivelser.
- Diskuter UI-patterns
- Når ferdig med prosjektplan: Kom raskt i gang med utvikling, skriv rapport
- Kom fort i gang med programmering
- Bruk perioden før 1. sprint til å utføre noen test-caser som er del av kravspec.
- Bruk ressurspersoner i fagmiljø: Bruk konkrete spørsmål.
- Argumenter for valg av verktøy med gode argumenter.
- Skaff kilder på enkelhet rund proting av backend
- Kart/GPS-løsning:
- Forventet diskusjon rundt I18N, stemmestyring, personsensitivitet/personvern, støtte for svaksynte, ol.
- Lytt til arbeidsgiver, men ikke gå for minst mulig arbeid.
- Oppdater kravspesifikasjon etter sentrale endringer, diskuter disse i rapport. Gjerne før/etter «bilder»
- «Sånn var det, sånn ble det, Hvorfor avvik»
- Releaser/Brukertesting under sprinter?
- 4 milepæler mer enn nok for vår prosjektlengde.
- Legg inn nøkkelpunkter og kilder i rapporten. Full analyse og diskusjoner tas senere.
- Kan bruke User-story i staden for high-level usecase beskrivelse.
- Ta snapshot av JIRA/andre verktøy regelmessig.
- Rapport: Skriv om planleggingsprosesen: Det å oppnå en felles forståelse av oppgaven.
- Skriv rapport jevnlig. Bruk mer tid mot slutten.
- Ordliste: Ja, for spesifikk terminologi innen teknologi/jakt, etc.
- Vi, og teamet i rapporten går fint. Forsiktig med Jeg. Nåtid eller fortid, vær konsistent
- vi-skal i prosjektplan, vi-har i rapport
- Risiko: Ta med det som påvirker hvordan vi jobber/hva vi lagrer.
- Diskuter risikovurderinger i rapporten.

Møte med Eivind om GUI 23.01.2018

- Benytte jegernummer eller noe mer personlig for å logge inn?
- Endre «logg inn» tekst på knapp til «motta melding» eller lignende.

- Eventuelt endre knapp? En «Logg inn» knapp etter mottat melding?
- Eventuelt en «Glemt jegernummer» knapp
- Beskjed om å logge på når man har dekning?
- Er man registrert som jeger et spesifikt sted? Hvilken informasjon er koblet til jegernr. Mobilnummer istedenfor jegernr? Smarte defaults.
- Skippe valg av sted helt - hente fra tidligere valg, eventuelt endre i innstillinger og første gang bruker benytter appen.
- Føles ikke som region, vi må passe på at brukerne forstår ordene/stedene. (Fefo, fjellstyrene og ...)
- Lage en bar for å velge sted, over dato?
- Se tall på scrollbaren til dato, se tall over og under for å indikere at det er scrollwheel.
- Ta med ukedag, ta med «idag», «imorgen». Indikere hvilken dag datoen er. Hvilken dato det er i dag. Ikke bruk dato på de nærmeste dagene (bruk heller igår, idag, forgårs).
- Har man satt igang en prosess? Mulighet til å gå tilbakke.
- Benytt aktiv form på meny (aktive ord, verb, «Rapportere»)
- Ny og lagre er confusing. Ikke naturlig å trykke på ny, føles ut som rapport slettes og en ny startes. Fjerne denne, gjøre at nye felt kommer opp automatisk.
- Vise felt som inaktive frem til forrige felt er innfylt. Dropdown først, Sett og Skutt ved siden av hverandre under?
- Viktig med feedback.
- Avbryte knapp.
- Bekreftelse er viktig. Gjerne kvittering.
- Du rapporterer for «...» er viktig å vise.
- Benytte «X» eller søppelkasse for å slette elementer fra rapport er OK, men det burde poppe opp en bekreftelses melding før alle destruktive handlinger.
- Et kort for hver art? Som kan sveipes ut til venstre for å slettes.
- Kalender? «Gå til rapportering» på knapp. 9x9 millimeter per rute i kalenderen.
- Synlig er et poeng.

Møte med Veileder 23.01.2018

- Prosjektplan:
    - Fjerne teori: Rekne med at leser kan teorien, og legg til mer diskusjon/tyngre argumentasjon.
    - Korriger: Spesifiser at Android er i fokus: Virker innkonsistent.
    - Dersom det er lite arbeid å porte til iOS: Vurder å gjøre det i neste sprint.
    - Prioriter android i spesialtilfeller.
    - Sjekk tilgang/integrasjonsmuligheter med statlige organ, prøv å få til dette om mulig.
    - Gantt: Lite arbeid i sprint-perioder: Flytte noe fra planlegging til sprint-periode.
    - Mulig release etter 3 sprinter, og «brukbart produkt» i følgende sprint-review.
    - Litt lite spesifikke grupperegler.
    - Målbare effektmål.
    - Hva menes med «fungerende prototype»?

    - Resultatmål: mer spesifikt, men ikke gjenta (grove mål)
    - Mer detaljert oppgavebeskrivelse (bruksområde, aktører/roller, dyr, osv.)
    - Legg til «testere» i ansvarsforhold og roller
    - Litt mye teori i valg av utviklingsmodell
    - Estimering: Kommenter referanse-oppgave og hvordan skalaen brukes.
    - Risikovurdering: Ikke ferdig til deadline: sansynlig. Bruker mye ytter ytterdeler av skala. Tiltak ved usannsynlige risikoer??? Fokuser mer på prosjektsetting, ikke prosjektsammenheng.
    - Fokuser på de mest kritiske risikoene (Trenger ikke tiltak på så mange)
    - Kategori av risiko?
    - Tiltak til risiko: spesifiser tiltak og/eller konsekvens for de mest kritiske risikoene
    - Dypere diskusjon av ferre ting (risikotiltak)
    - Gantt: Må ha med tekst-del. Forskyv enkeltoppgaver fra start til senere i utviklingen
    -

- Kravspesifikasjon:
    - Legg med andre/offentlige aktører til use-case
    - Splitt opp i use-case til use-tasks for de største oppgavene
    - Domenemodell: Tenker litt for mye DB
    - aggregering: «har/inngår i»
    - composisjon: (sterk aggregering) eksisterer ikke uten...

Møte med veileder 30.01.2018

- Skriv lite notat om valg av rammeverk for API
- Følg regningslinjer angående design/layout/ikonbruk og skriv litt om det.
- Bruk tidlige utkast i rapport (ta vare på mockups/screenshots)
- Push litt på møtetidspunkter.
- Seksjon om verktøy i rapporten: Ingen stor seksjon med detaljert analyse, men en oversikt er greit. Kommenter valg ovenfor andre. Integrer verktøybruk/erfaringer generelt i rapporten. (Se på tidligere rapporter)
- Diskuter de patterne som er essensielle. Ikke vis alle, men ett detaljert eksempel.
- Generelt: I staden for å diskutere alt, velg en eller to som er essensielle.
- Utvide case: Lit lite i pr. dags dato.
- Lever MVP ASAP. Deretter strekk case. Kunne bevise at vi kan implementere forskjellige problemstilligner.
- Dokumenter møtene og valg som ble tatt under de forskjellige møtene.
- Pakk issues inn i sprint0?

Møte med Veileder 06.02.2018

- Oppdatering av status i prosjektet
- Huske sprint-review: Vise funksjonalitet
- Typisk mye refaktorering av første kode
- Smart at vi bruker ressursene vi har tilgjengelig
- Over/undervurderinger er del av læringsprosessen

Man har ikke lov til å ta på seg flere storypoints en man klarte i forrige sprint

- Sjekke opp tilgjengelige APIer, skaper god diskusjon
- Universell utforming: Talestyring? etc.
- Profesjonell tankegang: Tenk på sikkerhet og universell utforming.
- Kunne slå av generelle funksjoner i telefonen fra appen? lydløs?
- Neste møte avlyses siden det er midt i en sprint

Møte med arbeidsgiver 06.02.2018, 1. sprint-planning meeting

- Litt oppsummering ang. status i prosjektet.
- Legg til ideer i backlog: Fint for senere dersom man vil avsnitt i rapport om "videre utvikling"
- SVR-5: lett å undervurdere størrelse: SVR-7 sansynlig vis mindre enn SVR-5
- Revurder hvor godt vi traff på estimering under sprint review/retrospective
- Hvor mange storypoints klarer vi pr. sprint? : ——
- Har etablert godt utgangspunkt.
- Jegernr: 6 siffer, men padding på 2. (00——)
- Enkel innlogging, men gi tilbakemeldinger
  Trenger ikke bruke meldinger under utvikling (travelt som utvikler)
  Den faktiske innloggingen kan lages senere, flyten må prioriteres.
  Innlogging ikke viktig, men skal se ut som det er på plass.
- Reestimere med ny definition-of-done og spesifiseringer
- Forvalter:
  Hardkodes foreløpig
  Vurder muligheter for videreutvikling
  Helt annen situasjon en jeger
- "Hva er definisjonen av ’Done”"
  "Angi Aktivt Jaktområde"
  - Jeger -> setter aktivt område -> kall mot backend -> forvalter -> Område -> jaktområde/kommune -> sett som aktivt (synlig) -> må kunne bytte på enkel måte
  - To/flere aktive områder pr. sesong, men kun 1 i gangen.
  - Aktivt område alltid synlig/tydelig
  - område/kommuner kan overlappe.
  - kommuner registreres mot område i bakgrunnen
    Kreves av SSB
  - Registreringen må testes: Er vesentlig for om folk faktisk kommer i gang med rapporteringen.
  - DoD: Velge område, satt som aktivt by defaultneste gang.

  Rapportere dagens observasjoner"
  - Start enkelt med den mockupen vi har
  - DoD: Rapportere Sett/skutt den dagen. (til backend/database)
    Må kunne gjøres offline (Subtask)
- Eksportere jaktrapportering"
  XML/JSON/CSV.

kommer an på hva eksterne intresenter vil ha
- 1 Sprint: SVR-3, SVR-4 og SVR-5
- Prøve å få til MVP innen sprint 3
- Nytt videomøte/sprint retrospective Fredag/Mandag om 2 uker.

Møte med Arbeidsgiver 20.02.2018, 1. Sprint-review + 2. sprint-planning

- Gjennomgang av applikasjonen/brukertest:
  Login: Skill på knapper og input-felt
  kalender på egen skjerm, litt mer subtil? (Dagens dato valgt som standard, mulighet for å endre)
  Valg av område på egen skjerm?
  Ikke «send rapport» men «Lagre» eller «lagre jaktdag» på send-knapp.
  Gjør +-knapp på rapporteringssiden mer tydelig: bruk tekst
  Prøve å komprimere UI litt, mtp. liste over arter og kalender
  Bruk heller flere cards, enn mye på en skjerm.
  Ikke kunne velge arter som er valgt/rapportert tidligere (i samme rapport)
  Kunne «fjerne» observer-del på de områder som ikke krever dette
  Kunne velge «Favoritt»-områder
  Første skjerm etter innlogging: Oversikt over funksjonalitet: knapper
- Tips: Sjekk ut redux for å kunne bruke «global state» i react-native
- Lage forside som oppsummerer funksjonalitet: Gjengivelse av menubar
- må kunne avslutte jaktsesong, etter hvert.
- SVR-3 og SVR-4 blir med videre, men må reestimeres (mye mindre arbeid som gjenstår)
- SVR-6
  Listevisning
  Vis alle rapporter i inneværende år (Aktiv sesong) sortert på dato, nyest først
  Mulighet for å redigere
  Så enkelt som mulig
  Beskrivende
- Alt skjer i kontekst av jaktområde: Denne må vises og kunne endre i uansett card.
- sjekk mulighet for å kunne kjøre app virtuelt i forbindelse med brukertesting i Sprint 3.
- GDPR bør vurderes og diskuteres i rapporten.

1. Sprint retrospective-meeting 20.02.2018

- Jaran
  Teknisk vanskelig, men ellers bra.
  Skriv engelsk i Jira
  Committe oftere
- Linn
  Flinkere på å bli ferdig med en oppgave
- Lars
  Bli flinkere på å bruke feature-branch
  Bli flinkere på å ajourføre Jira
  Dele opp stories i mindre subtasks og skrive en mer utfyllende beskrivelse

Møte med veileder 20.02.2018

- Oppdatering siden sist:
  Sprint review-meeting
- Trenger ikke å levere noe ekstra", men kan virke positivt at man har prøvd.
- Bruekrtesting i Sprint 3;
  Tenk gjennom hva som skal testes
  Snakk med Eivind om gjennomføring av brukertest (???)
- Code review på tvers av roller
- Lynkurs 2:
  Snakk med noen som var der om å få notater
  Mye går igjen i tidligere rapporter
  Virker som vi har god kontroll på rapport-skriving i forhold til mange andre
- Tom kan lese gjennom når vi har dokumentasjon, Men si fra noen dager før.
- Ta vare på detaljer om user stories.
- Diskuter endringer i GUI; beskriv hvorfor de ble gjort.
- Viten i Senter: Delen om brukertesting er noe svak, vi kan fokusere mer på det.
- I rapporten:
  Ha noen totalbilder av applikasjonen og ha ett dypdykk i noen emner
  Man må bevise hvorfor man gjør som man gjør.

Møte med Håvard 05.03.2018

- UI

  Brukerne vil si ifra om de finner fram. Viktigst er at brukerne får gjort hva de skal.

  Smart å ha en knapp med oppdateringssymbol + tekst på historikk skjermen.

  Default swipe ned for å refreshe i alle skjermbilder?

  Slett alt denne dag?på endre historikk. Nå lurer jeg på om jeg sletter alt på Pasvik". (hvis jeg later som jeg er en dum bruker).

  Første gang for å velge sted burde være NULL. Vis modalen med en gang. Mer tydelig at man skal trykke her for å endre lokasjon. Legg til endre". Ta med knappen fra historikk over til valg av sted.

  Endre sted fra hvilket som helst sted i appen? Kun på forsiden, byttes ikke så ofte. Viktigere å komme til test. Beskjed om at sted han endres senere. Må vise brukeren velkommen og la de velge sted for første gang på en hyggelig måte. Selecte datoen i kalender.
- Generelt

  Fortest mulig få vekk bugs. Så brukerteste, så mer vanskelige ting.

  Behøver ikke internasjonaliseres, veldig hvite menn som pusher 50 type ting å dra på småviltjakt. Lage språk config, sånn at det står på norsk men alle strenger bruker keys.

  Teste remote via public server? : Test on"verktøy, skybasert test.

  Kan hive inn mer dersom det er mye tid til overs, eventuelt kjørt en kortere sprint.

  Bug'ene som tasks, hvor mange story points tar de? Jobbe litt sammen dersom samme personen har sitti fast lenge.

Kan bli litt "Hummer og kanarisprint.

Noen må ta ansvar for testen, hva skal gjøres, hvordan skal det gjennomgåes, snakke med de, fylle ut skjema? osv osv.

Få jegernr og legge det inn i db før test.

Teamviewer, flere instancer, openstack?

Sikkerhet med jegernr? ha med navn?

Lite panel med navn/jegernr. En liten personlig touch.

Teste å hente ut og akkumulere data fra rapporten.

Småvilt felten mulighet for mer fyll data.

Avslutte sprinten midten av neste uke.

Implementere log out. Dersom tid.

Testing av offline neste runde. Litt på pause. Viktigere å sjekke hvor brukervennlig appen er.

Prioriter å stabilisere appen.

Gi testerne skriftlig info, så samle inn i etterkant. Tenke analytics inni appen, for å sjekke hva de gjør. "Hot jarsjekker hva de gjør i sesjon.

Onsdag neste uke.

- Etter bruktertesting

  Etter brukertestinga må vi gjøre mer gøye ting:

  GPS-tracking, kart og tegne hvor du har gått.

  Bildedokumentasjon.

  Velge fugl ved bilde.

  Stress test. Hva gjør appen med internett/uten nett. Teste synkronisering.

  Bruke testen til å gjøre endringer på UI i kjernen.

  Jaktkort.

  Hva er veien videre.

  Mest prioritert er å mappe fangsten, hvor nøyaktig det er skutt. Drømmescenario er når jeg har skutt 5 dyr, så får jeg 5 pins jeg kan plassere på kartet.

  Objekt og bildegjenkjenning.

  Eksport på slutten av sesongen.

- Avsluttende

  Ikke viktig å kunne legge til ekstra arter. Kan bare legge inn de parameterne statisk i koden.

  Skalering, analytics - voldsomt trykk i jaktperioden. Ta med i rapporten.

Møte med veileder 13.03.2018

- Oppdatering av status

  Hør med UI-lab for testing av iOS-kompatibilitet

  Begrensinger ved iOS?

- Brukertesting

  Kanskje litt mye tekst?

  Diskuter funn basert på hvem som utfører brukertesten.

  Ta høyde for funn ved evt. neste brukertest.

  Informer om hva som samles inn av data (!!!)

  Fortell om settingen (ikke reelle data som skal legges inn)

  Fortell hvorfor vi tester og hva vi tester

- Spørreskjema

  Vær sikker på at spørreskjemaet gir verdifull informasjon, ikke bare «Funka fint» (Hva? Hvorfor?).

  Litt mye «Ja/nei»: Pass på at det ikke blir for mange spørsmål
- Bra vi har kommet til dette nivået (Brukertest)
- Samle inn data om bruk dersom mulig: Men tenk over etikken rundt data som samles inn.
- Diskuter litt rundt hva som funket og ikke med brukertesten, hva ville vi ha gjort annerledes neste gang?
- Deler av rapporten må være lett å forstå for de fleste (Spesielt innledning). Mer teknisk i hovuddel og avslutning.

  Ingen innføring i teori.
- Ikke skriv «at», men beskriv hvorfor og hvordan. Diskuter erfaringer osv.
- Bruk av «vi» i rapporten:

  Individuelt/subjektivt: Ingen promer med «vi», men ikke bruk «jeg»

  Les gode eksempler for referanse
- Sikkerhet

  Implementasjon ikke nødvending (men bra). I det minste må det diskuteres hva som er gjort, hvorfor og hvordan.

  Ikke si «Det var ikke ønske fra arbeidsgiver»

  Spesielt viktig rundt Kartdata.

  Analyser Applikasjonen med tanke på sikkerhet (ProgSec).

Møte med Håvard 14.03.2018

- Brukertest

  Samsung har en emulator sak, google sin firebase? Alternativer.

  App fila må kunne lastes inn på emulatoren. Eventuelt sette en maskin som folk kan logge seg på over nettet.

  Overføring av hva brukern gjør på sin telefon kontra nettside. Emulator i nettleseren.

  Recorde hva brukeren gjør, så spille av det etterpå? (Slik som Hot jar)

  Håvard kan være en pretester av brukertesten. Sende han app fila, så kan han teste hele biten.

  En liten test før påske hadde vært fint. Rigge noe selv, sette opp en maskin som er tilgjengelig er fallbacken.

  Om vi eller Håvard har en VM som står og kjører. Da blir det eventuelt bare 1 og 1 om gangen.

  Trimme ned brukerteksten til absolutt minimum tekst.

  Si noe om hva vi samler inn av data.

  Spørreundersøkelse til slutt er bra.

  Brukertest på fredag? Rent hypotetisk, er det mulig?

  "Jeg tar gjerne en build, en applikasjonsbuild, så kan jeg starte å teste".

  Skriv ferdig introduksjonen, finne en måte å sette opp testmiljøet på.

  Håvard kan teste dev, han har tilgang til bitbucket.

  Brukertesten bør skje neste uke.

  På fredag bør vi ha fikset alle bugs og satt opp et testmiljø.

Enda en minisprint? "Å få gjennomført test".

Jegernr kommer fortløpende, 3 har kommet, muligens 3 til. Muligens 1 kollega til.

Dry run", kjøre igjennom hele løpet med Håvard før vi sender ut til resten.

Litt hyppigere dialog på denne inspurten frem til påske.

Møte med veileder 20.03.2018

- Brukertesting

  Lurt med både opptak av skjerm og ansikt

  Først og fremst jaktlaget, men kan utvide

  Kontrollere rettigheter av opptakene som lagres på serveren til lookback

  Jobbet mye med refaktorering

  Informer brukerne om hva som lagres og hvordan det brukes. Tydelig på dette.

  Testerne bør ha ett valg om å ikke måtte delta.

  Kjøre en Pre-brukertest på medstudenter?

  Legge til fritekst på spørreundersjøkelse: «Forslag til ideer/ny funksjonalitet?»

  2 uker virker mye heller 4 dager.

  Informer brukeren om at appen må avinstalleres etter testen.
- App tilbakemelding fra Tom:

  Tilbakemelding til bruker på innlogging: «validering av input»

  Legge til kjønn på dyr?

  Se/endre: Litt ulogisk ved hjelp av knapper

  «lagre område»: Høres ut som jeg lagrer ett nytt område. «Sett nytt aktivt jaktområde»

  Se/endre: Gruppere på måneder f.eks?

  Legge inn tidspunkt?

  Rapporter bør vektlegges mer fremover. start med å lage figurer. Les tidligere rapporter! (Viten i senter + 1 valgfri)
- Neste steg

  Snakke med Håvard ang. Kart og GPS (Statens kartverk mer nøyaktig for norge enn google?)

  Kan føre til en del endringer

  Ha en bevissthet rund påskeferie og arbeid som må gjøres før/etter påske

  Avklare forventninger til hverandre i påsken.

  En setter seg inn i heatmap, en annen i kart. f.eks. Hvor skal det lagres?

Møte med arbeidsgiver 03.04.2018

- Brukertest: Bruk det for hva det er"

  Fekk utdelt en android enhet med APK installert: Minimalt med instrukser

  Får ikke bilde på innlogging på eldre enheter
- Planen videre

  Nye stories basert på brukertest?

  Funker bra nå, kan gå videre på annen funksjonalitet

  Logg inn: Generer kode (ikke nødvendig vis sikker)

  Forvalter-stories: Skriv om i rapporten, ikke lag funksjonaliteten

  Eksporter til eksternt format: Knapp på forsiden

Ryddig JSON eller CSV
Størst bussiness value: Kart, plotting av felt
Må kunne spesifisere i etterkant
Ikke offline støtte.
Ikke obligatorisk (Bonus-feature)
Nøyaktighet: Mest hvor i området (ca.)
Polygon for plott av områder
Må diskutere hvor i rapporterings-prosessen dette skal plottes
Forslag: Web-grensesnitt
Forvalter-view: Heatmap + annen statistikk
Stort + for applikasjonen (for å få med forvalterne på løsningen)
2 Sprinter
Stopper utviklingen 1. mai.
Trapper ned utvikling utover
Sprint 5:
- Fokus: Online/offline + kart
- Forvalter-view på backend (Definerer denne selv, trenger ikke være ferdig denne sprinten)
Sprint 6:
- Forvalter-view på backend
- Eksport til JSON/CSV

Møte med veileder 03.04.2018

- Update:
  Oppdatering ang. sprint review
  Viktig at man kommer i gang med mer avansert funksjonalitet
- brukertest:
  Purre litt men kan ikke vente for lenge.
  Kommenter erfaringer rundt resultatet av brukertesten
  Dybdeintervju av en/flere?
  Diskuter hva som gjekk bra/dårlig og hva som burde gjøres annerledes
- Bør ha en mer glidende overgang mellom rapport/utvikling
  Push utviklingen, men 4t på fredager er for lite
  Skriv rapport underveis under utviklingen
  Skriv diskusjon rundt kartløsningen nå/underveis.
- Kompleksitet i oppgaven:
  Grunnfunksjonaliteten er triviell, men man kan strekke i caset til å få en god karakter.
  Bør dra inn mer
- man må få vist hva som er bra i prodokutet og hva som er gjort.
- Figurer tar en del tid.
- Man kan gjerne involvere flere i rapportskrivingen for å få en bedre forståelse for hva som er bra/må forbedres.
- Pattern-bruk, Diskusjoner, viser en systematisk fremgangsmåte, antall timer brukt, etc.
- diskuter hvorfor appen ikke er satt i drift (inkluderer offentlige systemer)

- Rapporten skal være et godt nok dokument slik at man slipper å lese koden for å forstå.
- Diskuter hvorfor appen bare er tilgjengelig for Android.
  Markedsbehov for X-plattform

Møte med arbeidsgiver 17.04.2018

- Få polygon til å avgrense mer? Type dersom markør flyttes ut, flytt markør tilbake til forrige posisjon.
- Screenshot av kart (markør og polygon) er bra å ta med i rapport.
- En bug i edit? Noe som ikke ble håndtert: å legge til en ny aktivitet med skutt-verdi.
- Fikse edge cases i kart (redigering av rapport) i neste sprint.
- Kjøperne vil få stjerner i øynene av kartet. (Svært høy business verdi).
- Web gjør ett punkt på kartet til flere når det zoomes in, det er bra.
- Felt-test av offline/online burde utføres. Det går som analyse for rapporten.
- Mange jegere kjører bil til dit de skal jakte, fort litt inn og ut av polygoner.
- Ikke legg energi i innlogging på web, men tall (på statistikk) kan være smart. Klikke på kart: zoome inn og vise antall. ((Art og antall)/tid)). Med andre ord: antall skutt for art over tid. Uke, måned, sesong.
- Hvordan normaliserer vi fargeskalaen når det er et par 100 000 dyr. Prøve å legge inn 1000 og se hvordan sirkelen blir. Henger sirkelstørrelse sammen med antall skutt eller størrelse på området?
- Rødt: ofte bruk for å indikere fare. Kan være området med mest skutt, men er ikke fare før man sammenligner med antall det finnes av arten.
- «Taksert mengde grønt hvis bra»?
- Hvis det er skutt mer enn det burde ha vært -> en egen farge?
- Lokale rapporter: «security by obscurity», bruk jegernummer for å finne rapporter. All lokal data må ha den lokale id'en.
- Rutiner: 1 gang i året. «Knappen rapporter til SSB» kan eksportere alt av data innenværende sesong. Alt til SSB. Kun for område til forvalter.
- «Eksporter alt», «Pr område» - knapp.
- Usannsynlig at en jeger er innom mer enn 3 områder i løpet av en sesong.
- Frister for innrapportering til forvalter er ulikt, hender det skjer på epost eller via brev.
- Håvard må rapportere til 3 steder. SSb, inatur ... og for både skutt og sett.
- Se på alle behovene vi har dekket.
- 80-90 prosent av dyrene blir felt første uka i sesongen. Å sjekke statisikk «live» og sammenligne med forventninger er gull. Da kan forvaltere vite hvor mange flere jaktkort de skal selge o.l.
- Godt med på å utvide web litt mer.
- Vi er ikke Storviltrapportering. Der er kommune, art og antall skutt relevant (+ jaktlag og jaktens varighet). Men det trenger ikke vi tenke på.
- Tips: opparbeide domene-kunnskap er viktig. Mange prosjekter feiler fordi de ikke forstår hva behovet er godt nok.
- Hender man har med egne domene-eksperter med på teamet. (Håvard vår ekspert?)
- Brreg og SSB snakker sammen, antageligvis samme database.

- Forvaltere på Hønsefuglportalen er offentlige. Det finnes også private forvaltere (grunneierlag), men de gjør som de vil. (i forhold til rapporterting)
- Fjern «hensyn til grunneiere» i rapport.
- Inatur er en «e-commerce», forvaltere **kan** selge jaktkort der, men de også styre det selv. Det er valgfritt. Flere og flere velger å gå digitale.
- Vingeprøver benyttes for å finne ut om dyret er «klekket» i år eller om dyret har overlevd en vinter. Dette kan være et særtilfelle for Håvards jaktområde.
- Kart: hvilken markør fjernes dersom antall skutt reduseres? Endre antall -> plotte alle på nytt?
- Tabeller eller grafer på web, se backlog.
- Ta med statistikk observert?
- Jaktkort i appen? Må fakes så mye så blir unødvendig.
- Håvard har skutt maks 4 på en dag.
- Facebook blir bråk. For politisk.
- Ta heller med jaktlag funksjonalitet, selv om det finnes mange apper som allerede gjør dette.
- Send siste versjon av apk.

Møte med Veileder 17.04.2018

- Oppdatering om sprint:

  Oppdatering om kart

  Legg til rette for at kodebasen rundt kart etc. lett kan endres/funksjonalitet kan legges til

  Diskuter kostnader rundt google-maps og kommersielle lisenser

  Statens kartverk er gratis.

  Observer pattern for oppdatering av data på alle steder (klient)

  Tenk sikkerhet/sabotasje rundt den funksjonaliteten som er på plass

  Lage mis-usecase?

  Tenk sikkerhetsproblematikk rundt caset

  «Guttorms indre»

  Sikkerhet må håndteres: iallefall kartlegges (Viser en systematisk gjennomgang av sikkerhet)

  SMS: Kan spore de som missbruker applikasjonen.

  Feilestimering ang. kart vs. offline

  Lage flowchart over alle cases under offline og diskuter dette i rapport

  Få til iOS, hvis ikke så er det mismatch med å bruke et cross plattform verktøy.
- Rapport

  Les Viten i senter, autoklave og Drismo rapportene

  Trekk frem skjermbilder av grensesnitt eller små kodesnutter for å synliggjøre produkt

  Fortell mer om Hovudstruktur for applikasjonen

  Fortell om løsninger som er spesifikke for denne løsningen.

  «Vis hvordan vi har gjort dette»

  Vi har stort fokus på brukergrensesnitt: Fortell mer om hovudstruktur, arkitektur-/design-patterns, og spesifikke løsninger/kodesnutter

  Strukturer rapporten med figur i Sommerville: s. 56 i bakhodet. Spesielt fokus

på designmessige argumenter og valg

Lag tidlig utkast av sammendrag

Bakgrunn: Kun om jakt. Si mer om oppdragsgiver og oss som gruppe. Fortell mer om Jakt under fagområde. Trenger ikke nødvendigvis flytte det som er skrevet

Fortell mer under oppgavebeskrivelse: Inkluder de ferskeste punktene rundt sikkerhet og kart

Litt for detaljert om hvert enkelt fag og om OOP-spesielt

UseCase: Vis hvilke useCase i figuren som ikke skal implementeres

Mer utdypende om hvert usecase

Lett inn heatmap og andre nye oppgaver

Fortell om feilsituasjoner i hvert enkelt useCase: Hva skjer med GUI, data etc. Gjerne flere feilsituasjoner

Domenemodell: Mer finmasket gruppering av arter

Brukervennlighet: Litt lite målbart

Scrum-board: Dra inn eksempel

Design og UI: Design er mye mer enn GUI: Legg in arkitektur-design. Hovudkapittel: Programvare-design, underkapitler: GUI, arkitektur, moduler, klient/tjener etc.

Bestem en struktur i programvaren: Finn den strukturen som er brukt under utvikling.

Struktur: Mer detaljert: Hvor ligger data og når. etc.

Lit mange figurer under Navigasjon. Mer diskusjon om hvorfor vi valgte hva: Mye figurer, men for lite diskusjon. Kan evt. plukke ut noen essensielle figurer og forklare mer konseptuelt rundt dette.

Material Design: «Diverse konsepter»: Forklar spesifikt hvilke konsepter. Oppdater rundt Android/iOS

Web løsningen: Vurder alternativer rundt kart-løsningen. trenger ikke alternatver. Sjekk økonomisk løsning rungt google maps API

Teknologi: Ligg for generel beskrivelse.

Nevne Hibernate og hvordan det går inn i vår app.

Hvorfor benytter vi xcode?

Lagdelig, få frem figuer.

Ikke si «Vi har lært, derfor valgte vi det» bruk mer tekniske begrunnelser.

Teste om at det å endre backend fungerer?

Må komme frem hvordan nøkler og relasjoner fungerer. Hvordan har vi brukt ORM. Ta med dokumentasjon.

Fokus på bachelor. Trenger ikke koble inn professional programming, men fortell hvordan vi har sikret kvalitet på koden.

Si at «Vi fikk 60 prosent eller 3/5 svar på brukertest».

Få med de nylig implementerte tingene i rapporten.

Hold kode på listings.

Møte med arbeidsgiver: 24.04.2018

- Testet app på mobil: virker førnøyd
- Kart: Litt lite detaljer: Anbefaler Statens Kartverk for mer detaljert data i standardkart

- Fått gjort det meste, med unntak av offline/online
  Burde ha gjort dette tidligere: Ble mye jobb nå i etterkant
- SVR-103
  Funker, men tar lang tid: ca. 15sek.
- SVR-104
  Vis hvilket filter som er valgt: Felt over tabellen
  Vise informasjon i filter: Fjern områder som ikke er valgt fra tabell i liste:
  Lagre filter i cookie?
  Generelt svært fornøyd
- Neste sprint:
  Vårt ønske: Ikke ta inn no ny funksjonalitet:
  Det er i henhold til planen
- Rapport:
  Kan skrive om issues i forslagskasse:
  Kunne rapportere til offentlige instanser er svært viktig. Spesielt automatikken rundt dette, datakvalitet og
  Jaktkort blir en mye mindre sak, sammenliknet med den over: Sync med QR?
  Bilder og gjenkjenning er også viktig, men er ikke like nyttig i alle områder.
  Kunne koble regler og retningslinjer til ett spesielt område, slik at områdene kan forvaltes individuelt.
  Språk: Ikke viktig
  Kunne se rapporter for tidligere sesonger: Ikke viktig.
  Kunne koble art til område: Viktig for forvalter
  Drøfte sosiale funksjoner:
  Ble nedprioritert av arbeidsgiver, siden man har mange applikasjoner som tar seg av det sosiale aspektet.
  Drøfte Utvidelser: Storvilt
  Mye mer komplisert: Må nok leie inn konsulenter.
  Kunne ta i bruk GPS og plotte hvor man har gått (rute)
  Enklere å plotte felt vilt langs en linje som sier hvor man har gått
  Ikke så viktig, men viktige vurderinger i rapport. (med på å øke verdien på dataene og presisjon)
- Fremtidig:
  Om appen slår an, kan den utfases til ett eget firma/AS
- Offentliggjørelse av rapport:
  Sharing is caring
  Må tenke over konfidensialitet rundt innholdet i rapporten
- Overtakelser:
  Legge inn installasjonsinnstrukser i README
- Tilbakemelding fra Håvard:
  Vi er flinke til å si det som det er, og ikke feie feil under teppet"

Møte med veileder: 24.04.2018

- Si hvilke seksjon vi vil ha tilbakemelding på i fremtiden
- Henvis til juridiske dokumenter (Google's retningslinjer) angående bruk av google

API

Redegjør for begrunnelser: Brukes applikasjonen lite nokk til at APIet kan brukes fritt?

- Sikkerhet MÅ fortsatt HÅNDTERES i større grad enn nå
- Synkronisering: Fortsatt litt som gjenstår
- Klargjør applikasjonen for eksport/overføring til andre systemer

  «Partner (SSB/brreg) er ikke klar: Vi gjer derfor slik for å legge til rette for overdragelse/overføring til andre systemer»
- Konfidensialitet rundt rapport burde hvert avklart tidligere.
- Rapport:

  Førsteutkast:

  Nøkkelpunker om vår løsning, ikke jegeren. Få frem teknologiperspektivet

  TOC:

  Trenger ikke ett tredje nivå under 1.4

  Litt lite spesifikt i seksjon 1.4.3

  1.6: Forbedring

  1.8: Litt mye i detalj om tidligere fag.

  UseCase: Legg til ekstern aktør: SMS-/kart-tjeneste

  stiplet linje fra jeger til logg inn, vanlig pil fra log inn til SMS-tjeneste

  Legg til kart

  For lite spesifikt under detaljert useCase beskrivelse

  Ikkje gjør bare det som Tom sier, men tenk rundt caset om det er mer å hente fra den tilbakemeldingen?

  Jobb generelt rundt sikkerhet rundt hele caset

  Referer når det prates om SCRUM

  Si spesifikt hvilke statuser som defineres når det snakkes om utviklingsprosessen

  Ikke referer til Håvard med fornavn, bruk produkteier. Formelt språk

  3.1.1 ekstra figur

  Sprint planning & sprint review: Litt gjentakelse, men godt dekt ellers

  Scrumming the scrum eller scrum of scrums?

  Figur 5: Bruk figuren mer aktivt i diskusjon: Gi ett helhetsbilde av situasjonen. Evt. bruk logoen i tekst som en referanse til figur

  5.1: Litt for muntlig fremstilling. Bra start, men fortell mer om hvorfor dette er en god løsning for oss

  Figur 6: Forklar nermere i tekst

  Generelt litt mye fornavn

  Listing 6.1 og 6.2: For langt eksempel + forklar hva som skjer og hvorfor det er en positiv utvikling

  Beskrivelse først, og en mer visuell fremstilling senere

  Større eksempel kan legges ves som vedlegg (som ett ekstra eksempel)

  6.1.4 Finn ett juridisk pliktende dokument som sier noe om hvordan vi kan bruke google maps APIet

  6.2.4: Bruk gjerne flytdoagram: Koble gjerne til krav

  EER må være konsistend med domene-modell

  Generelle mangler av kommentarer rundt listings

referer til personers rolle i stedet for navn
Til slutt: Litt for detaljert
Veldig bra fom. påske.

- Sensur:
  Ser etter det gode
  Enklere nå å påpeke det som er litt mangler
- Smart å fintune på viktige løsninger
- Argumenter for faglige og fornuftige valg

Møte med veileder 02.05.2018

- Nytt utkast på mandag
- Gjennomgang av nye endringer:
  Sammendrag: Kommenter utfordringer som er spesielt vektlagt, Kommenter stikkord, online/offline
  UseCase: Viktig å skille på sluttbruker og systemer som aktører. Avklar at misusecase inngår i samme diagram. Skill på det som er implementert og det som er tilrettelagt for
  Domenemodell: Merkelig bruk av komposisjon: For sterkt modellert. Fjerne metoder. EER er heller en direkte DB-gjengivning. Litt rart at modellen ikke har en dyreklasse
  Kommenter endring av aktivitet
  2.4: «ytelsen.... dvalemodus» Hva menes med dette? Sett spesifikt krav
  Sikkerhet: Bedre, men mangler fortsatt litt.
  Har litt igjen på språkbruk. «i motsetning» kontra «for å kompensere for»
  node.js: Mer sammenlikning.
  Web-løsningen: Litt tynnere enn resten. (Flytte prosessering fra backend til frontend) pro/con i contekst av applikasjonen.
  Kommenter hvor omfattende prosjektene er (Kodelinjer, antall biblioteker, timebruk etc. «Hva har vi egentlig gjort?»)
  Moduler: Splitte opp lagdelingsmodell for hele prosjektet.
  5.3.2: «til slutt»: bruk heller «vi valgte å forholde oss til». overgang til iOS.
  «Lager vi en struktur som er særegen, eller svært kjent for de fleste brukere»
  Tabell 1: Litt muntlig struktur, koble til figurer?
  Polygoner: Litt for teknisk i teksten: **sekvensdiagram**/Komponentdiagram/deploymentdiagram? Mangler «Oversiktsbilde»
  Tabell 2: «Tiden er ikke knapp», dårlig argument. Litt vanskelig å følge tekst for leser.
  Generelt: Oversikt før detaljer
  online/offline: Litt muntlig tekst. (Sikkerhet rundt manipulering av usynkronsiert data?)
  Tabell 4: Ikke bruk gjentakelsestegn
  Sekvensdiagram Fig 18: Litt mer detaljert: + sikkerhet
  HTTP kontra HTTPS
  6.2: For langt tilbake i tid

- Vær generelt flinkere av å skryte av det vi har gjort: og heller påpek det som kunne

vert bedre. (offansiv, ikke defansiv)

Møte med veileder 08.05.2018

- Vi ligger godt an, men alltid noe som kan forbedres (ingen kvile enda)
- Sammendrag:
  Det som «flagges» er det viktigste i rapporten, og må diskuteres ekstra godt i selve rapporten.
- Litt mye linker: blå skrift
- SSB (etc.) ikke en brukergruppe, men ekstern aktør. Lage eget punkt for eksterne systemer
  Addresser hvem som må håndtere feil, etc,
- Lagdelingsmodell (5.1.1)
  Vær åpne på avvik fra teori, og begrunn dette
  Spesifiser hvordan dataene opprettes: Finnes admin-panel? hvorfor/hvorfor ikke?
  Evt. implementer og ignorer login
- «Mobilapplikasjonen»: Litt vanskelig å vite om dette er iOS óg Android, eller begge..?
  Vær mer konsistente, og spesifiser hva det snakkes om
- Generelt litt mange tabeller
- Edge-case: kjent? ordliste?
- listing. Går greit med engelsk.
- Under listing 6.2 ikke skriv «flere»
- Diskuter litt om hvor innsatsen ligger: Hvilken kode krever mer arbeid enn andre?
- Er appen kjørbar på iOS? Diskuter hvorfor/hvorfor ikke, og evt. forskjellene i kodebasen. Hva må til for å få den kjørbar.
- Posisjon: Lese posisjon fra GPS dersom det blir rapportert fra felt og lagre til de skal plottes på kart
- Ikke tabell for gjengivelse av teori
- Sikkerhet:
  Spesifiser hvorfor SMS-tjeneste ble mocket.
  Diskuter misusecase
  GDPR og personvern-loven
- Feilhåndtering:
  litt tynt. Elementer fra proprog?
  Spesifiser hvordan vi har løst dette
  hvem har ansvar for feilhåndtering? klient eller server?
- Kontroller figurer etter rapporten er så å si ferdig.
- trenger ikke å direkte forklare alt som står i figurene, forklar hva modellen viser, og hvorfor vi har valgt den strukturen
- Figurtekster: Konsistente
  Trenger ikke å spesifisere verktøy i figurtekst
- «Har du en følelse av at man har hentet mye fra en kilde»
- DFD-modell
  Referer til ProgSec
- Flere figurer når det diskuteres verktøy

og analyser figurene

- Test: Backend
  Mer: Hvor mange tester, hvor ofte ble de kjørt etc.
- Tabell 15: Mer spesifikt hvilke krav vår applikasjon tilfredsstiller
  Spesifiser hva vår app dekker/ikke dekker
- Tiltak:
  Mer og dypere
- generelt mer om sikkerhet
- Valg av rammeverk for backend: Litt mange detaljer/begreper. supplere med figur? evt. referere til tidligere figur?
- GDPR må diskuteres/nevnes tidligere i rapporten
  «farlig» å si at man skal «se bort i fra dette».
  Hvilken del av løsningen vår vil påvirkes av dette lovverket?
  Blir litt for tilfeldig nevnt

C Plan for brukertest

Velkommen til brukertesting av vår applikasjon, Småviltrapportering!

Vi vil først takke deg for at du setter av tid til å gjennomføre denne brukertesten. Dine tilbakemeldinger er svært verdifulle for oss. Med din hjelp kan vi sørge for at applikasjonen blir så brukervennlig og intuitiv som mulig. I brukertesten kan du bruke ditt eget jegernummer for å logge inn i applikasjonen. All annen informasjon du legger inn kan være fiktivt.

Om applikasjonen

Applikasjonens hovedoppgave er å la deg som jeger rapportere inn arter som er felt og observert så raskt og enkelt som mulig på mobiltelefonen. I tillegg vil applikasjonen gi deg mulighet til å se historikk over dine tidligere rapporter. Du får også mulighet til å endre på tidligere rapporter. Dette er funksjonene vi ønsker å teste.

Oppgaver

Under beskrives tre oppgaver som er en del av brukertestingen. Vennligst noter deg ting du opplever som utfordrende, spesielt bra og liknende. Når du har utført oppgavene kan du lukke applikasjonen.

Rapportering

Du har vært på jakt i dag og nå ønsker du å lage en rapport. Du skal rapportere de dyrene du har skutt og sett i løpet av dagen. Videre har du jaktet i det området du vanligvis jakter i.

NB: Benytt ditt personlige jegernummer for å logge inn.

Endre en rapport

Du har allerede laget en rapport for dagen i dag. Nå husker du at du la inn feil informasjon på en eller flere arter. Gjør de endringene du ønsker i rapporten du lagret og oppdater rapporten.

Slette rapport

For et par måneder siden lagret du en rapport da du jaktet i Pasvik (Fefo region). Nå husker du at rapporten ble registrert på feil dato. Du må slette rapporten.

Vurdering

Til slutt trenger vi tilbakemelding fra deg om hvor lett eller vanskelig du opplevde oppgavene og hvorfor. Denne tilbakemeldingen er som nevnt svært verdifull for oss, så vi ønsker at du gir en ærlig (og gjerne detaljert) tilbakemelding.

Følg denne lenken for å gi oss din tilbakemelding: https://goo.gl/forms/Hre9PPMLfmRX6TMl1

Dersom du bekreftet at vi også kan få tilbakemelding over telefon vil vi ta kontakt

med deg en ukedag i uke 14 mellom 08.00 og 16.00.

Tusen takk for din deltagelse.

D Prosjektavtale

NTNU

Norges teknisk-naturvitenskapelige universitet

Vår dato Vår referanse

Prosjektavtale

mellom NTNU Fakultet for informasjonsteknologi og elektroteknikk (IE) på Gjøvik (utdanningsinstitusjon), og

HÅVARD NARVESEN (ESCIO AS)

(oppdragsgiver), og

LINN-HEGE KRISTENSEN, LARS JOHAN NYBØ OG JARAN GODEJORD.

(student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 10.01.18 til 16.05.18.

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der NTNU IE på Gjøvik yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra NTNU å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon/fax, reiser og nødvendig overnatting på steder langt fra NTNU på Gjøvik. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. NTNU IE på Gjøvik står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av intern og ekstern sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv hvis de har skriftlig karakter A, B eller C.

Tilgjengeliggjøring i det åpne arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og instituttleder/fagenhetsleder om de i løpet av prosjektet endrer syn på slik offentliggjøring.

Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av NTNU til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU og/eller studenter har interesser.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i NTNUs elektroniske eksamenssystem. I tillegg leveres ett eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med ett eksemplar til hver av partene. På vegne av NTNU, IE er det instituttleder/faggruppeleder som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og NTNU som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten NTNU som partner.

10. Når NTNU også opptrer som oppdragsgiver, trer NTNU inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene imellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): Ian Reide

Oppdragsgivers kontaktperson (navn): Håvard Vaverem

Student(er) (signatur): Vinteg Kristensen dato 16.01.18

Lars Johan Nyhus dato 16.01.18

Jaran Godejord dato 16.01.18

_____ dato _____

Oppdragsgiver (signatur): Håvard Vaverem dato 16.01.18

Signert avtale leveres digitalt i Blackboard, rom for bacheloroppgaven.
Godkjennes digitalt av instituttleder/faggruppeleder.

Om papirversjon med signatur er ønskelig, må papirversjon leveres til instituttet i tillegg.
Plass for evt sign:

Instituttleder/faggruppeleder (signatur): _____ dato _____

E Prosjektplan

E.1 Mål og rammer

Bakgrunn

Bakgrunnen for oppgaven er å forenkle jegerens hverdag i felt. Hovedsakelig innebærer dette digitalisering av småviltrapportering. I dag utføres dette gjerne analogt og det finnes flere tjenester jegerne benytter som er lite samkjørt. Dette er en stor ulempe ettersom det i 2017 fantes over 84 000 aktive jegere [1] som måtte forholde seg til dette.

I dag må hver jeger fylle ut et skjema med diverse informasjon fra jakten. I tillegg må hver jeger etter endt jaktsesong rapportere inn til Brønnøysundregisteret. Utenom dette kan også en jeger kjøpe jaktkort fra inatur.no. Jegeren må også sende inn rapporter på denne portalen.

Videre er det en fordel at dersom rapporteringen gjøres enklere for jegeren, så vil dataene som sendes inn ha økt presisjon. Dette vil gjøre statistikken over felte og observerte dyr mer korrekt. Videre sparer digitaliseringen tid for både jeger og forvalter. Som en bonus vil applikasjonen også spare miljøet ved å unngå papirbruk.

Prosjektmål

Under har vi definert ulike mål for prosjektet vårt. Vi har definert resultatmål for produktet, effektmål og læringsmål for gruppen.

Resultatmål

Resultatmålet for prosjektet er å utvikle en prototype for mobilplattformene iOS og Android. Dette innebærer at vi har utviklet et ferdig produkt, men ikke nødvendigvis med tilkobling til statlige organer. Målet er også at prototypen blir klar til å settes i drift hos Escio AS.

Effektmål

- Effektivisere behandling av rapporter.
- Øke presisjon av data med minst 10%.
- Redusere rapporteringstid for jeger med 50%.

Læringsmål

- Lære React Native.
- Lære å definere REST API.
- Lære å jobbe profesjonelt og systematisk i et Scrum-team.
- Lære å lage applikasjoner på mobile plattformer.
- Lære å benytte hensiktsmessige patterns, blant annet for arkitektur-design.
- Lære å ta hensyn til universell utforming og enkelhet.
- Lære å benytte tilegnet kunnskap fra studiet på en relevant måte.

Rammer

Her er de følgende rammene vi må forholde oss til:

- Frontend lages i React Native etter sterkt ønske fra arbeidsgiver.
- Oppdragsgiver ønsker at applikasjonen støtter de mest utbredte versjonene i løpet av de to siste årene og generelt nyere mobiler (Samsung Galaxy S7 og nyere). React Native støtter Android versjon 4.1 (API 16) og oppover. I tillegg støttes også iOS versjon 8.0 og oppover [5].
- Oppdragsgiver ønsker at vi benytter et RESTful API for kommunikasjon mellom frontend og backend. Det kreves at standaren beskrevet på jsonapi.org [6] benyttes under utforming av API-et.
- Applikasjonen skal brukertestes (gjerne flere ganger) i løpet av utviklingsperioden.
- Applikasjonen skal utvikles ved hjelp av smidige arbeidsmetoder.

E.2 Omfang

Fagområde

Rapportering av småvilt er lovpålagt [72] innen jakt. Det er krav om at det skal innrapporteres hva som er observert og felt, og lokasjon hvor aktivitetene har foregått.

Intensjonen med applikasjonen vi skal utvikle er å lage en god og enkel mobilplattform hvor jegere kan rapportere jakten sin. Hovedfokuset vårt vil være på oppbygningen av løsningen.

Programmeringsspråk og verktøyvalg

Oppdragsgiver ønsker at React Native skal benyttes som hovedspråk under utviklingen. På backend står vi frie til å velge programmeringsspråk selv uten anbefalinger. Under er en liste over språk og verktøy vi skal benytte i prosjektperioden:

Språk og rammeverk

- React Native (frontend)
- Java (backend)

Verktøy

Navn	Type	Bruksområde
MySQL/PHPMyAdmin	Databasesystem	Lagring av data
Visual Studio Code	Tekstredigeringsprogram	Skrive React Native-kode
Eclipse	Java IDE	Skrive Java-kode
SonarQube	Plattform for kodevalidering	Kodeinspeksjon
Git/Bitbucket	Versjonskontroll plattform og hosting-tjeneste for kildekode etter Git-standarden	Versjonskontroll
Jira	Digitalt scrum board og issue tracking	Prosjektstyring
ShareLaTeX	Nettbasert redigeringsverktøy og kompilator for LaTeX-dokumenter	Prosjektrapport
Google Drive	Nettsky	Lagring av figurer
Android Studio	Offisiell IDE for Android-utvikling	Android applikasjon
Google Spreadsheet	Nettbasert regneark	Loggføring av timer
draw.io	Nettbasert diagramverktøy	Organisasjonskart
Microsoft Project 2016	Prosjekthåndterings-program	Gantt-diagram
Trello	En form for scrum board	Prosjektstyring/ Liste over ekstra ideer
JavaDoc	Dokumentasjonsverktøy for Java	Kildekode dokumentering i Java
JUnit	Unit test verktøy for Java	Testing av kode i Java

Liste over alle verktøy som skal benyttes

Avgrensning

- Applikasjonen skal ikke omhandle de sosiale aspektene ved jakt, da det finnes andre applikasjoner som fokuserer på dette.
- Applikasjonen skal foreløpig ikke kobles opp mot offentlige tjenester, da behandlinsperioden for dette er for lang.

Oppgavebeskrivelse

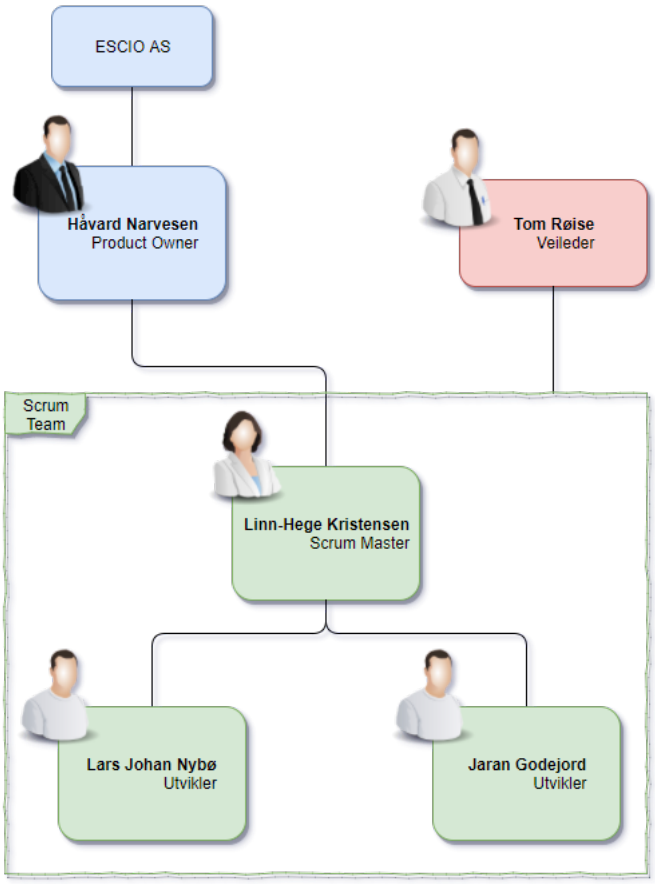
Prosjektet vårt innebærer at vi skal utvikle en mobilapplikasjon som digitaliserer rapportering av småvilt. Informasjon som må rapporteres inn er blant annet jaktområde, småvilt observert og småvilt som ble skutt. Her må art og antall også defineres. Ettersom dette er nokså tungvint ønsker Escio AS å utvikle en kryssplattform mobilapplikasjon som digitaliserer denne aktiviteten. Vi har fått i oppgave å sørge for at mobilapplikasjonen kjører på Android-enheter. Dersom det er tilstrekkelig med tid vil vi også konfigurere for iOS. Applikasjonen skal inneholde følgende funksjoner:

- Identifisere en jeger ved hjelp av jegernummer.
- La en jeger velge jaktområde.
- La en jeger rapportere jakt-aktiviteter.

- La en jeger plotte småvilt på et kart.
- Applikasjonen skal kunne benyttes offline.
- Eksportere lagrede data til generelle format som f. eks. JSON og CSV..

E.3 Prosjektorganisering

Organisasjonskart



Organisasjonskart

Ansvarsforhold og roller

Rolle	Beskrivelse
Product Owner	Prosjektets produkteier er Håvard Narvesen. Han er vårt bindeledd til Escio. Han er tilgjengelig for brukertester og er vår hovedkontakt fra oppdragsgiver.
Scrum Master	Scrum master er Linn-Hege Kristensen. Hun har ansvaret for å være bindeledd mellom Scrum-teamet og Håvard. Hun har også ansvar for å sørge for at teamet overholder interne og eksterne tidsfrister. Videre skal hun lede møtene som gruppen foretar seg. Hun fungerer også som utvikler.
Utviklere	Lars Johan Nybø, Jaran Godejord og Linn-Hege Kristensen har som hovedansvar å utvikle. De skal følge kode- og dokumentasjonsstandard som er satt i gruppen.
Referent	Lars Johan Nybø er ansvarlig for å skrive ned møtereferater.
Testere	Arbeidsgiver stiller med testgruppe (bestående av jegere) som kan brukerteste applikasjonen.

Beskrivelse av ansvarsforhold og roller

Rutiner og grupperegler

Generelt

- Møte til avtalt tid hver dag. Du er pliktig til å informere de andre medlemme tidligst mulig dersom å møte fysisk ikke lar seg gjøre.
- Timer skal loggføres.
- Alle medlemmer må håndtere sensitive data i henhold til den norske lov.
- Medlemmer er pliktig til å informere andre gruppemedlemmer tidlig dersom de mener at arbeidsinnsatsen til vedkommende ikke er tilstrekkelig.
- Det forventes at alle medlemmer har gjort sine oppgaver til avtalt tid.
- Ved større uenigheter konsulteres veileder.

Arbeidsregler

- Når gruppen møtes skal det fokuseres på prosjektet og personlige interesser skal ikke prioriteres.
- Gruppen skal følge en felles kodestandard og dokumentasjonsstil.
- Det skal skrives referat av alle møter.
- Alle gruppemedlemmer skal arbeide minimum 30 timer med oppgaven hver uke.

Rutiner ved brudd på regelverket

- Et mindre brudd på reglene skal følges av en advarsel fra gruppen.
- Et alvorlig brudd på reglene skal følges av et møte med veileder for å avklare konsekvenser og redegjøre erstatning for tapt arbeid.

E.4 Planlegging, oppfølging og rapportering

Hovedinndeling av prosjektet

Prosjektet består av en mobilapplikasjon for to plattformer; iOS og Android. Applikasjonen vil bestå av en frontend, backend og et RESTful API. Dersom det er nok tid vil det også implementeres en enkel web-løsning som viser diverse statistikk basert på data i mobilapplikasjonen database.

Valg av utviklingsmodell

Etter anbefaling og sterkt ønske fra arbeidsgiver, har vi besluttet å gå for en smidig utviklingsmodell. Dermed ekskluderes modeller som *Fossefall*. I tillegg ekskluderes *Rational Unified Process (RUP)*, da denne virker unødvendig kompleks med tanke på størrelse på utviklerteam og oppgave.

Vi velger dermed å trekke frem fire utviklingsmodeller/-rammeverk som vi har kjennskap til, og som virker relevante for oppgaven: *Scrum*, *eXtreme Programming (XP)*, *KanBan* og *LEAN Software Development (LSD)*.

Scrum [7]

Scrum er ofte et natulig førstevalg av modell, da den er svært smidig og tilpassningsdyktig til de fleste prosjekter. Modellen er også et naturlig valg for oss, siden vår situasjon passer godt med modellens bruksområde:

- Modellen er smidig og har hyppige releaser (korte/egendefinerbare sprintlengder). Dette gir rom for fortløpende endringer, noe som er relevant siden kravspesifikasjonen er ufullstendig.
- Idéell for små team
- Roller som passer vår bemanning: Product-owner, Scrum Master, Scrum Team.
- Gir utviklerteamet større frihet/evne til å gjøre egne valg, som da øker effektiviteten (Gunstig siden utviklingsperioden er svært begrenset).
- Arbeidsgiver får resultater raskt, og har aktiv deltakelse gjennom utviklingen.

Basert på disse betingelsene velger vi Scrum som vår kjernemodell.

XP [8]

Fra XP ønsker vi å benytte *User-stories* og *Automatiserte Test-rammeverk*. User-stories er ønskelig da applikasjonen vi skal utvikle ofte baserer seg på handlinger og oppgaver som brukeren skal utføre.

Automatiserte test-reammeverk skal benyttes da dette sikrer at logikken fungerer som planlagt. React-Native har også lagt opp til TDD (Test-Driven Development) som standard, og dette er derfor enkelt å praktisere.

KanBan [9]

KanBan har ett enkelt konsept: *Begrense mengde arbeid som er påbegynt*. Dette vil føre til at nye oppgaver må vente til de påbegynte er ferdig. Dette sikrer at vi ikke har mange uferdige oppgaver på en gang, og vil gi økt oversikt og kontroll. Det sikrer også at «mindre artige» oppgaver som dokumentasjon og code-review blir gjennomført fortløpende. Basert på tidligere samarbeid med gruppen ønsker vi å benytte KanBan i våre sprinter, spesielt for å få gjennomført code-review fortløpende.

LSD [10]

LSD er mer ett rammeverk enn en modell, og vi har besluttet å bruke prinsippene som retningslinjer under planleggingen. Bl.a. begrenser vi unødvendig arbeid ved å definere spesifikke arbeidstider, og ett estimat over tidsbruk i de forskjellige fasene. Dette vil redusere stress og kognitive påkjenninger (m.fl. som LSD definerer [10]). Dette vil igjen redusere småfeil/bugs og logiske feil. Vi benytter også rapid-prototyping for å unngå store endringer senere i utviklingsfasen.

Vi utvider teamets kunnskap ved å benytte teknologier vi ikke er kjent med, som *React Native* og *RESTful API*. Vi setter også av en egen fase for opplæring og utforskning av disse teknologiene, før den faktiske utviklingen starter.

Svært få krav er fastsatt før utviklingen starter, og siden utviklingen er smidig, vil både teamet og arbeidsgiver få en bedre forståelse av hvilke krav til funksjonalitet som skal inngå etter hvert i utviklingen. Smidig utvikling vil også resultere i raske og fortløpende leveranser, som samsvarer med LEAN-konseptet.

Gjennom å bruke Scrum, får teamet mer autoritet til å ta egne valg, som da senere kan avklares med arbeidsgiver. Dette fungerer siden utviklerne og arbeidsgiver, underveis, oppnår en felles forståelse av hva applikasjonen skal gjøre.

Ett av kravene til applikasjonen er at arbeidsgiver skal kunne ta over utviklingen etter prosjektperioden. Dermed er det viktig for oss at applikasjonen er «Fleksibel, vedlikeholdbar, effektiv og responsiv» [10] Der vi legger spesielt vekt på *vedlikeholdbar*. For å oppnå disse punktene vil vi benytte bl.a. unit-testing, jevnlig refaktorering og konsistent versjon-nummerering.

Gjennom planleggingsfasen vil vi opprette use-case-diagram, klassediagram, o.l. for å se helheten og oppnå en god forståelse for problemstillingen.

Valg av metode og tilnærming

Vi vil benytte «vanilla scrum» med KanBan, og under planleggingsfasen benytter vi elementer fra LEAN, som beskrevet i seksjon E.4.

Innenfor Scrum-delen av tilnærmingen vår velger vi å følge møtene ganske strengt. Vi skal holde daily scrums, sprint planning-møter, sprint retrospective-møter og sprint review-møter. Dette gjør vi for å bevare oversikt over progresjonen i utviklingen og for å legge et godt grunnlag for hver neste sprint. Videre har vi valgt å jobbe etter 2-ukers sprinter. Vi falt på dette valget av flere grunner. Oppdragsgiver har blant annet ytret at de ønsker å se ny funksjonalitet hyppig. Videre er det også et ønske vi har selv innad i gruppen. Det kan også virke mer motiverende å se ny og ferdig funksjonalitet annenhver uke. I tillegg er det også nokså vanlig å benytte 2-ukers sprinter (dette er default i Scrum). Vi velger 2-ukers sprinter overfor 1-uke fordi vi tror dette er mest passende basert på størrelsen på våre user stories. Vi tror ikke 1 uke vil være tilstrekkelig med tid til å implementere og revidere ny funksjonalitet. Innen Kanban velger vi å begrense arbeidsmengden. Til å starte med ønsker vi å prøve ut å ha maksimalt fire aktive oppgaver i arbeid og maksimalt tre oppgaver i 'code review'. Videre ønsker vi ikke å benytte denne Kanban-metoden for sprint backlog-en. Når det gjelder backlog er tanken at denne opprettes med hjelp fra produkteier. Deretter vil vi sørge for at backlog-en er sortert på prioritet i samarbeid med produkteier. Dette utføres ved en vurdering av hver use case

sin business value"og estimering av tidsbruk. Innenfor estimering skal vi benytte story points. Her skal vi ta i bruk skalaen 0, 0.5, 1, 2, 3, 5, 8, 13, 20, 40, 80, 100. I bacheloroppgaven skal vi å benytte den nedre delen av skalaen. Vi planlegger å splitte alle tasks med story points på 8 og høyere inn i mindre subtasks.

Plan for statusmøter og beslutningspunkter i perioden

Gruppen møtes daglig (med noen unntak av torsdager) for å jobbe sammen. Disse dagene utføres det først en daily scrum. Videre følger vi Scrum-møtene nokså strengt. Det skal holdes et sprint planning-møte før hver sprint og et sprint retrospective-møte etter hver sprint. Videre holdes det også et sprint review-møte med Håvard for å få tilbakemeldinger på utført arbeid. Disse skal holdes fredag i slutten av hver sprint. Utenom dette har vi faste møter med veileder tirsdag 14:30-15:00.

E.5 Organisering av kvalitetssikring

Dokumentasjon, standardbruk og kildekode

I dette prosjektet vil vi ha stor fokus på god kodestandard & kvalitet, testing og dokumentasjon på et profesjonelt nivå. Vi vil til ytterste evne benytte de kodestandardene som er anbefalt for hvert programmeringsspråk vi benytter. Under er en liste over relevante verktøy og standarder vi skal benytte under prosjektperioden.

JavaDoc

Dette er et naturlig dokumentasjonsverktøy som vi planlegger å benytte, spesielt for den Java-koden vi utvikler for applikasjonen. Ved å benytte dette verktøyet blir det enklere for oss å holde god oversikt over koden og for teamet å jobbe på tvers av applikasjonen.

SonarQube

SonarQube er et verktøy som skanner kode for bugs, sårbarheter og diverse feil. Det er et verktøy vi planlegger å benytte for å kvalitetssikre koden. Vi tenker det er spesielt relevant å benytte SonarQube etter hver fullførte task for å kvalitetssikre koden i tillegg til etter hver endte sprint.

Internasjonalisering

Vi tenker også at det er relevant å sørge for at applikasjonen er internasjonalisert. Som et minimum ønsker vi å ha norsk og engelsk som språkvalg og følger standardene ved implementasjon av dette.

Testing

I prosjektet planlegger vi å benytte JUnit som enhetstester for Java-koden vi skriver (i Eclipse). Unit-testing er et konsept som tillater automatisert testing. Dette gjør at det er raskere å lage god og riktig kode ved å få bekreftet at koden er funksjonibel. Vi tenker å unit teste mest mulig, men kun hovedfokus på kritiske elementer og deretter kun på de trivielle dersom det er tilstrekkelig tid. Vi har avklart med oppdragsgiver at en funksjonibel prototype kan brukertestes av en gruppe jegere han er bekjent med.

Dokumenter

For å skrive rapporten og diverse vedlegg benytter vi for det meste LaTeX. Utenom dette loggfører vi timene våre i Google Spreadsheet. Videre legger vi diverse oppgaver (som går utenfor selve utviklingen) i et eget Trello board. Utenom dette lagres all kode i prosjektet i egne repositories.

Konfigurasjonsstyring

Generelt

- Det skal utføres en backup minst tre ganger i uken av rapporten og dokumentasjon.
- All ny kode skal kommenteres før den commites.
- Commit-meldinger skal være intuitive.
- Hver commit skal også inneholde ID fra task-en i Jira.
- Ny funksjonalitet skal utvikles på en egen branch for tasks. Når koden er klar skal den merges med developer-branchen. I slutten av hver sprint skal developer-branchen merges med master og det legges til en tag som identifiserer versjonen (Se seksjon E.5 for mer info).
- For estimering skal det story points-skalaen benyttes.

Arbeidsrutine

1. Ta ansvar for én task (sett 'In progress' i Jira).
2. Iterer: kommentere, utvikle, enhetsteste og debug kode.
3. Utfør en statisk analyse med SonarQube av task-en og revider.
4. Sett oppgaven 'In code review' i Jira og la noen andre vurdere kommentarer og logikk.
5. Eventuelt revider arbeid og sett til 'Done' i Jira.

Versjonskontroll

Applikasjonen skal få versjonsnummer på følgende format: `<releaseType>.<subRelease>.<devRelease>`.

releaseType sier noe om hvilket stadie utviklingen er på, og er definert slik:

Versjon	Stadie
1	ALPHA
2	BETA
3	RELEASE

Definisjon av versjonsnummer for generelt stadie i utviklingen.

subRelease økes med 1 når man merger *dev-branch* med master. Dette vil signalisere en ny release.

devRelease økes med 1 når man merger en *feature-branch* med *dev*. Denne tilbakestilles til 0 når *subRelease* øker.

Versjonene kobles til kildekoden gjennom *tag*-systemet i Git.

Risikoanalyse

Under følger en tabell over ulike risikoer ved prosjektet. Sannsynligheter delt opp i usannsynlig, sannsynlig og svært sannsynlig. Konsekvensen er enten uproblematisk, problematisk eller kritisk.

Siden vi vet at risikoanalysen kun skal angå vår tid med prosjektet, så velger vi å benytte «dummy-data» i prosjektet. Dermed unngår vi risikoene koblet til konfidensielle data. Vi kan si at tap av data og konfidensielle data på avveie ikke er en risiko for oss fordi det ikke er lagret noe konfidensiell data. Vi vet at disse risikoene må vurderes ved en eventuell kobling mot statlige organer og bruk av personlige data.

#	**Risiko**	**Sannsynlighet**	**Konsekvens**	**Tiltak**
1	Prosjektet er ikke ferdig til deadline.	Sannsynlig	Problematisk	Nei
2	Et gruppemedlem blir sykmeldt over lengre tid eller slutter.	Usannsynlig	Kritisk	Nei
3	Oppdragsgiver trekker seg og bryter kontrakten.	Usannsynlig	Kritisk	Nei
4	Tap av data (backup, rapport eller kildekode).	Sannsynlig	Problematisk	Ja
5	Endringer i kravspesifikasjon av oppdragsgiver.	Svært sannsynlig	Uproblematisk	Nei
6	Statlige organer gir oss ikke tilgang til nødvendige data innen jakt.	Sannsynlig	Problematisk	Ja
7	Stor konflikt i utviklergruppen.	Usannsynlig	Problematisk	Nei
8	Mangel på universell utforming/enkelhet.	Sannsynlig	Kritisk	Ja
9	Andre firmaer utvikler en applikasjon som utfører tilsvarende funksjonalitet.	Usannsynlig	Problematisk	Nei
10	Lover/reglerverk om personvern og håndtering av personlige opplysninger endres slik at applikasjonen strider mot disse.	Usannsynlig	Problematisk	Nei

Risikoanalyse av prosjektet

Håndtering av risiko

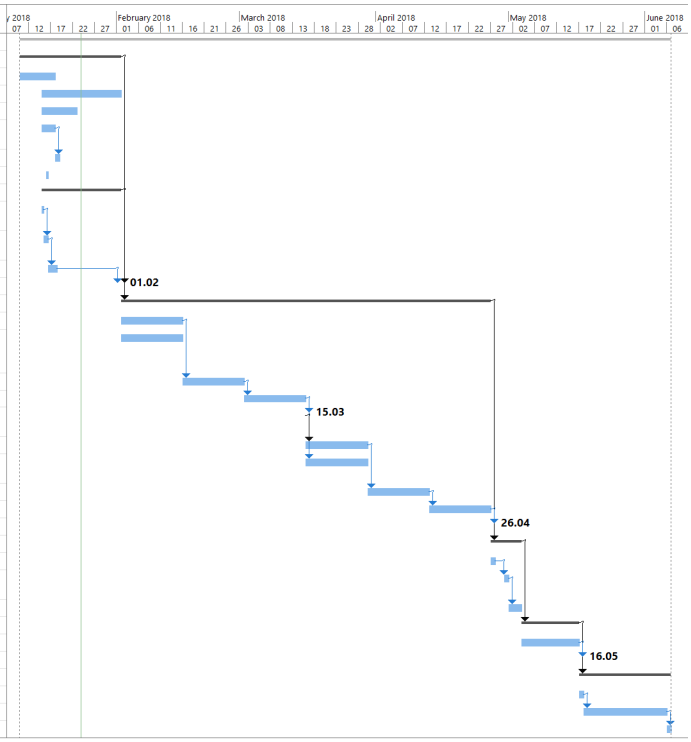
#	**Tiltak**
4	For å forhindre dette vil gruppen lagre backup av alle data på flere steder/maskiner. I tillegg er det som nevnt bestemt at gruppen skal kjøre backup minimum tre ganger i uken for å sikre dataene. Dersom risikoen likevel inntreffer vil gruppen først prøve å kjøre en gjennopretting. Dersom dette ikke lar seg gjøre vil teamet fortsette fra siste tilgjengelige versjon.
6	Dersom de statlige organene (som ssb og brreg) ikke tillater oss å hente ut data som f.eks. jegernummer, så vil vi benytte dummy-datafor å implementere nødvendig/ønsket funksjonalitet i applikasjonen.
8	For å sikre at applikasjonen får en universell utforming vil vi følge standardprinsipper ved utforming av GUI. Videre velger vi å utvikle flere UX-protoyper for brukertesting i forkant av utviklingen. Dette gjør vi for å sikre at applikasjonen føles enkel å bruke. I tillegg vil vi utføre flere brukertester underveis i utviklingen. Dersom en av aktørene opplever at applikasjonen ikke møter krav til enkelhet vil gruppen ha fokus på refaktorering.

Liste over tiltak mot risikoer

E.6 Plan for gjennomføring

Gantt-diagram

ID	Task Name	Duration	Start	Finish	Predecessors
0	**Bachelor**	**106 days**	**Wed 10.01.**	**Wed 06.06.**	
1	**Planlegging**	**17 days**	**Wed 10.01.1**	**Thu 01.02.18**	
2	Prosjektplan	6 days	Wed 10.01.1	Wed 17.01.1	
3	Teste teknologi	14 days	Mon 15.01.1	Thu 01.02.18	
4	Kravspesifikasjo	6 days	Mon 15.01.1	Mon 22.01.1	
5	Utarbeide UX-prototype	3 days	Mon 15.01.18	Wed 17.01.18	
6	Brukertesting	1 day	Thu 18.01.18	Thu 18.01.18	5
7	Kontrakt	1 hr	Tue 16.01.18	Tue 16.01.18	
8	**Oppsett**	**14 days**	**Mon 15.01.1**	**Thu 01.02.18**	
9	Sette opp repository	1 hr	Mon 15.01.18	Mon 15.01.18	
10	Sette opp SonarQube	1 day	Mon 15.01.18	Tue 16.01.18	9
11	Sette opp server	2 days	Tue 16.01.18	Thu 18.01.18	10
12	Klar til utvikling	0 hrs	Thu 01.02.18	Thu 01.02.18	1;11
13	**Utvikling**	**60 days**	**Fri 02.02.18**	**Thu 26.04.18**	**8;1**
14	Sprint 1	2 wks	Fri 02.02.18	Thu 15.02.18	
15	Undersøke integrasjon mot statlige organer	2 wks	Fri 02.02.18	Thu 15.02.18	
16	Sprint 2	2 wks	Fri 16.02.18	Thu 01.03.18	14
17	Sprint 3	2 wks	Fri 02.03.18	Thu 15.03.18	16
18	Funksjonibel prototype	0 days	Thu 15.03.18	Thu 15.03.18	17
19	Sprint 4	2 wks	Fri 16.03.18	Thu 29.03.18	18
20	Brukertesting med	2 wks	Fri 16.03.18	Thu 29.03.18	17
21	Sprint 5	2 wks	Fri 30.03.18	Thu 12.04.18	19
22	Sprint 6	2 wks	Fri 13.04.18	Thu 26.04.18	21
23	Ferdig prototype	0 wks	Thu 26.04.18	Thu 26.04.18	22
24	**Testing**	**5 days**	**Fri 27.04.18**	**Thu 03.05.18**	**13**
25	Brukertest - jakt	1 day	Fri 27.04.18	Fri 27.04.18	
26	Brukertest - ansatte	1 day	Mon 30.04.18	Mon 30.04.18	25
27	Refaktorering	3 days	Tue 01.05.18	Thu 03.05.18	26
28	**Rapport**	**9 days**	**Fri 04.05.18**	**Wed 16.05.1**	**24**
29	Skrive rapport	9 days	Fri 04.05.18	Wed 16.05.1	
30	Ferdig med rapp	0 days	Wed 16.05.1	Wed 16.05.1	29
31	**Fremføring**	**15 days**	**Thu 17.05.18**	**Wed 06.06.1**	**28**
32	Planlegge	1 day	Thu 17.05.18	Thu 17.05.18	
33	Øve	13 days	Fri 18.05.18	Tue 05.06.18	32
34	Fremføre	1 day	Wed 06.06.1	Wed 06.06.1	33



Gantt-diagram

Tekstversjon av gantt-diagrammet

#	Navn	Varighet	Forgjengere
T	**Bachelor**	126 d	
T1	**Planlegging**	17 d	
T2	Prosjektplan	6 d	
T3	Teste teknologi	14 d	
T4	Kravspesifikasjon	6 d	
T5	Utarbeide UX-prototype	3 d	
T6	Brukertesting	1 d	T5
T7	Kontrakt	1 h	
T8	**Oppsett**	14 d	
T9	Sette opp repository	1 h	
T10	Sette opp SonarQube	1 d	T9
T11	Sette opp server	2 d	T10
T12	Klar til utvikling	0	1, 11 (M1)
T13	**Utvikling**	60 d	
T14	Sprint 1	2 wks	
T15	Undersøke integrasjon mot statlige organer	2 wks	
T16	Sprint 2	2 wks	T14
T17	Sprint 3	2 wks	T16
T18	Funksjonibel prototype	0	T17 (M2)
T19	Sprint 4	2 wks	T18
T20	Brukertesting med jegergruppe	2 wks	T17
T21	Sprint 5	2 wks	T19
T22	Sprint 6	2 wks	T21
T23	Ferdig prototype	0	T22 (M3)
T24	**Testing**	5 d	T13
T25	Brukertest - jakt	1 d	
T26	Brukertest - ansatte	1 d	T25
T27	Refaktorering	3 d	T26
T28	**Rapport**	9 d	T24
T29	Skrive rapport	9 d	
T30	Ferdig med rapport	0	T29 (M4)
T31	**Fremføring**	15 d	T28
T32	Planlegge	1 d	
T33	Øve	13 d	T32
T34	Fremføre	1 d	T33

Gantt-diagram i tekst

Milepæler og beslutningspunkter

Under følger en liste over milepælene vi har satt for prosjektet.

1. **(01.02) Klar til utvikling**: på dette tidspunktet skal prosjektplan, kravspesifikasjon og design være grunnleggende utformet. Alt det tekniske skal også være satt opp (server o.l).
2. **(15.03) Funksjonibel prototype**: innen denne datoen vil gruppen prøve å ha klar en prototype som er klikkbar og kan testes på en mobilplattform. Innlogging og jaktrapportering-delen skal være på plass.
3. **(26.04) Ferdig prototype**: dette er datoen prototypen skal være ferdig utviklet og skal møte oppdragsgiverens krav.
4. **(16.05) Ferdig med rapport**: fristen for innlevering av rapporten.

F Relasjonsskjema

Vedlegget viser definisjon av relasjonsskjema for databasen. Å definere et relasjonsskjema ga oss også muligheten til å se eventuelle feil ved tidligere database-design. Gjennom utforming av skjemaet gjorde teamet flere revideringer på EER-diagrammet for å bedre designet. Ellers var det også heldig å ha relasjonsskjemaet tilgjengelig under utviklingen ved kommunikasjon med databasen. Skjemaet ga gruppen ytterligere forståelse om hvordan alle dataene henger sammen.

Person (jegernr, fornavn, etternavn, telefonnr)
PK jegernr

Rapport (id, dato, omraadeId, jegernr)
PK id
FK omraadeId **REF** Jaktomraade (id)
FK jegernr **REF** Person (jegernr)

Aktivitet (id, type, artId, antall, rapportId)
PK id
FK artId **REF** Art (id)
FK rapportId **REF** Rapport (id)

Art (id, navn)
PK id
UNIQUE navn

Forekomst (art, omraadeId)
PK (art, omraadeId)
FK art **REF** Art (navn)
FK omraadeId **REF** Jaktomraade (id)

Jaktomraade (id, navn, regionId, kommuneId)
PK id
UNIQUE navn
FK regionId **REF** Region (id)
FK kommuneId **REF** Kommune (id)

Region (id, navn)
PK id
UNIQUE navn

AktivitetKoordinat (id, lengdegrad, breddegrad, aktivitetId)
PK id
FK aktivitetId **REF** Aktivitet (id)

JaktomraadeKoordinat (id, lengdegrad, breddegrad, jaktomraadeId, rekkefolge)
PK id
FK jaktomraadeId **REF** Jaktomraade (id)

Kommune (id, navn)
PK id
UNIQUE navn

Rolle (id, navn)
PK id

Legitimasjon (id, navn, token, rolleId)
PK id
FK rolleId **REF** Rolle (id)

G Spørsmål til SSB angående tilgang til data

Fra: linn-hege@live.no [mailto:linn-hege@live.no]
Sendt: torsdag 12. april 2018 20:23
Til: Postmottak <Postmottak@ssb.no>
Emne: Forespørsel SSB - Generell henvendelse

Språk: no
Navn: Linn-Hege Kristensen
Epost: linn-hege@live.no

Spørsmål: Hei! Jeg studerer programvareutvikling ved NTNU i Gjøvik og skriver nå min bacheloroppgave med to andre. I denne sammenhengen utvikler vi en mobilapplikasjon for småviltrapportering. Målet er at jegeren skal kunne fylle ut den årlige rapporteringen med hjelp av en app på mobilen. Vi lurer på hva som skal til for at vår applikasjon skal få tilgang til data om registrerte jegere. Det hadde vært svært verdifullt for rapporten vår å beskrive en eventuell fremgangsmåte for de som overtar applikasjonen, protokoller o.l. Dersom applikasjonen i fremtiden får tilgang til disse dataene vil en jeger kunne logge seg direkte inn i applikasjonen uten å registrere seg på forhånd.

SA

Steinset, Trond Amund <trond.amund.steinset@ssb.no>
fr. 13.04.2018, 10:04
Du; Informasjon;

Hei

SSB har ikke lov til å utlevere opplysninger om hvem som er registrert som jeger, jf. statistikkloven.

Jegerregisteret i Brønnøysund leverer heller ikke ut personopplysninger til andre. De oppgir også at bare jegeren selv har tilgang til informasjon i registeret som gjelder dem selv. Tror derfor det blir vanskelig for dere å få tak i et slik register.

Vi har også ønsket at vi kunne utviklet en app for småviltrapporteringen, men vi skal fra neste år bruke Altinn.no som rapporteringskanal, og er avhengig av at de legger til rette for det.

Med vennlig hilsen

Trond Amund Steinset
Seniorrådgiver
Seksjon for eiendoms-, areal- og primærnæringsstatistikk
Statistisk sentralbyrå

E-poster som ble sendt mellom et gruppemedlem og en ansatt i SSB.

H Timelogg

Timelogg uke 2

Person	Timer
Linn-Hege Kristensen	16
Lars Johan Nybø	17.30
Jaran Godejord	16
Totalt	**49.30**

Timelogg uke 3

Person	Timer
Linn-Hege Kristensen	33.30
Lars Johan Nybø	34.30
Jaran Godejord	32
Totalt	**100**

Timelogg uke 4

Person	Timer
Linn-Hege Kristensen	26
Lars Johan Nybø	23
Jaran Godejord	25.30
Totalt	**74.30**

Timelogg uke 5

Person	**Timer**
Linn-Hege Kristensen	29.30
Lars Johan Nybø	34.30
Jaran Godejord	30
Totalt	**94**

Timelogg uke 6

Person	**Timer**
Linn-Hege Kristensen	20.30
Lars Johan Nybø	33.30
Jaran Godejord	22
Totalt	**76**

Timelogg uke 7

Person	**Timer**
Linn-Hege Kristensen	27
Lars Johan Nybø	24.30
Jaran Godejord	28.30
Totalt	**80**

Timelogg uke 8

Person	**Timer**
Linn-Hege Kristensen	27
Lars Johan Nybø	23
Jaran Godejord	29
Totalt	**79**

Timelogg uke 9

Person	**Timer**
Linn-Hege Kristensen	30.30
Lars Johan Nybø	26.30
Jaran Godejord	8 (sykdom)
Totalt	**65**

Timelogg uke 10

Person	**Timer**
Linn-Hege Kristensen	29.30
Lars Johan Nybø	20
Jaran Godejord	24
Totalt	**73.30**

Timelogg uke 11

Person	**Timer**
Linn-Hege Kristensen	24
Lars Johan Nybø	18.30
Jaran Godejord	20.30
Totalt	**63**

Timelogg uke 12

Person	Timer
Linn-Hege Kristensen	32.30
Lars Johan Nybø	14.30
Jaran Godejord	30
Totalt	**77**

Timelogg uke 13 (påske)

Person	Timer
Linn-Hege Kristensen	16
Lars Johan Nybø	0
Jaran Godejord	0
Totalt	**16**

Timelogg uke 14

Person	Timer
Linn-Hege Kristensen	38
Lars Johan Nybø	17
Jaran Godejord	24
Totalt	**79**

Timelogg uke 15

Person	Timer
Linn-Hege Kristensen	38
Lars Johan Nybø	24.30
Jaran Godejord	28
Totalt	**91.30**

Timelogg uke 16

Person	Timer
Linn-Hege Kristensen	39
Lars Johan Nybø	38
Jaran Godejord	47.30
Totalt	**124.30**

Timelogg uke 17 (gruppen hadde et prosjekt i et annet emne)

Person	Timer
Linn-Hege Kristensen	9
Lars Johan Nybø	1.30
Jaran Godejord	11.30
Totalt	**22**

Timelogg uke 18

Person	Timer
Linn-Hege Kristensen	52
Lars Johan Nybø	31.30
Jaran Godejord	25
Totalt	**108.30**

Timelogg uke 19

Person	**Timer**
Linn-Hege Kristensen	55.30
Lars Johan Nybø	65
Jaran Godejord	62.30
Totalt	**183**

Timelogg uke 20

Person	**Timer**
Linn-Hege Kristensen	15.30
Lars Johan Nybø	19
Jaran Godejord	18
Totalt	**52.30**