BACHELOROPPGAVE

STUTTREIST – DIN LOKALE REISEGUIDE

FORFATTERE:

DENNIS A.Ø. GJERDINGEN
PÅL A. STORSVEEN
TRINE J. STORSVEEN

Dato: 15.05.2015

SAMMENDRAG

Tittel:	Stuttreist – Din lokale reiseguide	Dato: 15.05. 2015
Deltakere:	Dennis A. Ø. Gjerdingen Pål A. Storsveen Trine J. Storsveen	
Veileder:	Ivar Farup	
Evt. oppdragsgiver:	AndMark Software Development DA	
Stikkord/nøkkelord (3-5 stk)	Reiseguide, tur, severdighet, android, lokal	

Antall sider: 114	Antall vedlegg: 15	Tilgjengelighet (åpen/konfidensiell): Åpen

Kort beskrivelse av bacheloroppgaven:

Vi har laget en android applikasjon hvor brukere kan se severdigheter i sitt område og lage en rute av severdigheter de vil besøke. Brukerne kan også legge til egne severdigheter, sammen med bilder og lyd til disse severdighetene. Hver severdighet har en informasjonsside hvor brukerne kan lese mer informasjon om et sted, se bilder og høre lydopptak. Alle steder og ruter er åpne og tilgjengelige for alle innloggede brukere.

ABSTRACT

Title:	Stuttreist - Your local travelguide	Date: 15.05. 2015
Participants:	Dennis A. Ø. Gjerdingen Pål A. Storsveen Trine J. Storsveen	
Supervisor:	Ivar Farup	
Employer:	AndMark Software Development DA	
Keywords (3-5)	Travelguide, hiking, landmarks, android, local	

Number of pages: 114	Number of appendix: 15	Availability (open/confidential): Open

Short description of the bachelor thesis:

We have developed an android application that gives the user an overview over landmarks nearby. Users can add their own favourite places, along with pictures and even sound. Every landmark has a corresponding informationpage where users can get more information about this place, look at pictures and listen to sound. All places and routes are available for all logged in users.

Forord

Takk til Ivar Farup for god veiledning gjennom hele prosjektperioden.

Takk til Rune Hjelsvold for hjelp til utforming av database og SQL.

Vi vil også takke oppdragsgiver Anders Hagebakken og Markus Brovold i AndMark Software Development DA.

Innholdsfortegnelse

1. Innledning

AndMark Software Development AS, i samarbeid med Høgskolen i Gjøvik, ønsker å få utviklet en android applikasjon hvor konseptet er Guided Tour. AndMark Software Development ble grunnlagt i mai 2014 av Anders Hagebakken og Markus Brovold. Begge studerer Master i Applied Computer Science ved Høgskolen i Gjøvik. AndMark står bak Android og iOS -appen Stolpejakten, med tilhørende database og nettside, som de fikk stor suksess med etter sin Bacheloroppgave.
Tanken bak applikasjonen er at brukerne på en enkel måte introduseres for severdigheter i Gjøvik og omegn. Applikasjonen skal også kunne brukes over hele landet, og brukerne skal selv kunne lage ruter og opprette egne severdigheter.

1.1 Oppgaven

Oppgaven har gått ut på å utvikle en android applikasjon under konseptet Guided Tour. Den ferdige applikasjonen skal gjøre det mulig for brukerene å lage sine egne ruter og severdigheter i nærområdet. Brukerene skal også ha muligheten til å lese mer om stedene/severdighetene.

Vi bestemte oss tidlig for at stedene i applikasjonen skal være delt inn i forskjellige kategorier, slik at brukeren enkelt kan velge ruter og severdigheter ut fra interesseområde. For å fremheve gode ruter og severdigheter som brukerene er fornøyd med, ville vi ha med funksjonaliet som tillater at ruter og steder kan stemmes opp og ned. Vi har valgt å legge til en utvidelse hvor vi implementerer muligheten for å rapportere innhold som enten ikke passer inn i applikasjonen, eller som kanskje inneholder feil. Disse rapporteringene skal kunne vises for oppdragsgiver slik at steder med feil innhold kan rettes opp.

Vi ønsker at muligheten for å generere egne ruter skal være så enkel som mulig for brukerene, der brukerene kan velge alle punkter av interesse, for så å lage en tilpasset rute mellom disse punktene.

Et stykke inn i prosjektperioden ble det også uttrykt et ønske fra oppdragsgiver om at applikasjonen skal bruke bruke funksjonalitet fra Android 5 og følge designprinsippene i Material Design.

I brainstorming-fasen var vi også innom tanken på å utvikle applikasjonen for wearables også, men siden hverken vi eller oppdragsgiver kom opp med noen nyttige og funksjonelle ideér rundt dette, ble ideén forkastet. Andre ideér vi har vært innom er at brukeren får push-meldinger når de nærmer seg et punkt. Denne funksjonaliteten har ikke vært høyt prioritert, og det blir derfor opp til oppdragsiver om de skulle ønske å videreutvikle appen. Vi snakket også om å kontakte historielag mm, for å få informasjon om steder som kan legges til. Dette har heller ikke blitt prioritert, mye grunnet at appen i størst mulig grad skal være brukerstyrt. I tillegg var vi åpne for å også utvikle applikasjonen for iOS, dersom vi fikk tid til det, men her prioriterte vi ekstra funksjonalitet i eksisterende Android-løsning i stedet.

1.2 Målgruppe

Hovedmålgruppen for applikasjonen er nyinnflyttede i Gjøvik-området som ønsker å bli bedre kjent med stedet. Applikasjonen kan også appellere til turglade personer fra lokalbefolkningen som ønsker å gå turer med litt ekstra innhold, og besøkende fra andre steder i landet.

Målgruppen for rapporten er personer fra Høgskolen i Gjøviks avdeling for informasjonsteknologi, spesielt rettet mot programvareutvikling. Vi forventer at leseren har generelt god teknologisk innsikt og har kjennskap til forskjellige programmeringskonsepter.

1.3 Formål

Formålet med dette prosjektet er i hovedsak å skape en plattform for turglade personer som ønsker en ny måte å gå tur på. I tillegg ønsker vi å bidra til å få flere mennesker ut på tur.

1.3.1 Effektmål

- Oppdragsgiver får økt brukergruppe.
- Mulighet for økt funksjonalitet i dagens Stolpejakt.

- 500 installasjoner i løpet av 2015.

1.3.2 Resultatmål

- Utvikle en Android-applikasjon under konseptet GuidedTour, med brukergenerert innhold.
- Applikasjonen lar brukere finne severdigheter i sitt nærområde og setter opp ruter/turer basert på disse.
- Brukeren skal også kunne benytte seg av predefinerte ruter.
- Applikasjonen skal la brukerne opprette egne brukere, og opprette severdigheter og ruter/turer.
- Applikasjonen kan brukes aktivt over hele landet.
- Prosjektet skal ha en tilhørende nettside for informasjon og/eller utvidet funksjonalitet.
- Deler av applikasjonen kan integreres i Stolpejakten av AndMark.
- Applikasjonen skal fungere for Android 4.0 og oppover (inkludert Android 5.0).

1.3.3 Læringsmål

- Få erfaring med å jobbe sammen med en oppdragsgiver.
- Bruke allerede tilegnet kunnskap i planlegging og gjennomføring av prosjektet.
- Få erfaring med å jobbe med et større prosjekt.
- Få kjennskap til arbeidsfordeling og det å ha forskjellige ansvarsroller.
- Få større faglig kompetanse innen mobilutvikling for Android.
- Få større faglig kompetanse innen brukervennlig mobildesign.
- Få større faglig kompetanse innen Scrum.

1.4 Avgrensning

Prosjektet er avgrenset til å kun gjelde utvikling for Android, fra API 15 og oppover, men vi har holdt åpent for iOS-utvikling.

1.5 Vår bakgrunn

I hovedsak har vi alle tre den samme akademiske bakgrunnen fra studiet i programvareutvikling. Vi har erfaring fra forskjellige programmeringsspråk, blant annet C++, Java, PHP, Bash, Powershell, Python og Javascript. Vi har også gjennom studiet jobbet en del med databaser og SQL. I et tidligere semester har vi vært innom mobilutvikling rettet mot Android-plattformen. I tillegg til dette er Trine utdannet Grafisk Ingeniør fra Høgskolen i Gjøvik.

I dette prosjektet har vi måttet utvide vår kunnskap om mobilutvikling en god del, spesielt rettet mot konsepter som er introdusert i Android 5, som kom sammen med Nexus 9 - lanseringen i november 2014. I tillegg har vært nødt til å sette oss godt inn i GoogleMaps API og funksjonalitet tilknyttet dette. Da oppdragsgiver også har satt krav til bruk av designkonseptet Material Design til Android og RESTful API, er dette noe vi har måtte sette oss inn i.

1.6 Prosjektrammer

1.6.1 Utviklingsmetodikk

Vi bestemte oss tidlig for å benytte oss av en agil utviklingsmetodikk for prosjektet, og valget falt på Scrum. Vi tok dette valget på bakgrunn av vår teoretiske kunnskap om agile utviklingsmetoder fra forskjellige emner vi har vært gjennom i studieløpet. Siden mye av funksjonaliteten i prosjektet er uavhengig av hverandre, er det mest hensiktsmessig å jobbe inkrementelt gjennom disse. Ved bruk av agil fremgangsmåte er vi bedre rustet for eventuelle forandringer oppdragsgiver måtte ha i forhold til ønsker og kravspesifikasjon.

Bruk av Scrum som utviklingsmetodikk er også godt utbredt i arbeidslivet, det at vi fikk erfaring med dette selv så vi på som veldig nyttig kunnskap.

Selv om vi valgte Scrum som utviklingsmetodikk, var det nødvendig for oss å tilpasse metodikken slik at den passet oss bedre. Siden gruppen ønsket å jobbe mer geografisk uavhengig av hverandre valgte vi å la Daily Scrum-møtene foregå gjennom en egen scrum-

chat. Her skulle alle tre oppdatere hverandre på hva som er blitt gjort siden sist, hva som har bydd på problemer, og hva de skulle gjøre til neste gang. Scrumboardet til prosjektet er gjort digitalt tigjengelig via trello.com, der vi i starten av utviklingsperioden fylte en backlog med funksjonaliteten vi skulle utvikle, og plukket oppgaver fra den.
Forholdet til Product Owner ble også litt anderledes enn hva som er vanlig i Scrum-prosjekter. Siden oppdragsgiver har mye av den samme akademiske bakgrunnen vi selv har, ble kravene stort sett satt til å dekke de tekniske aspektene av applikasjonen. Ut over det oppdragsiver har spesifisert i oppgaven, fikk vi mer eller mindre frie tøyler til å selv bestemme funksjonalitet i applikasjonen.

1.6.2 Prosjektorganisering

Alle gruppemedlemmene har pendlet til Gjøvik fra Lillehammer-området i studieperioden. Derfor var det naturlig for oss å jobbe i lokaler nærmere hjemmet. Etter at timeplaner var kryssjekket valgte vi å holde obligatoriske møter på tirsdager på et reservert grupperom på bibliotektet i Lillehammer, hvor vi jobbet sammen. I tillegg til disse møtene jobbet vi også sammen på Høgskolen i Gjøvik de dagene vi hadde møte med veileder og oppdragsgiver. Disse møtene ble satt til annenhver uke. Vi har også møttes utenom dette for å arbeide sammen, de gangene vi har jobbet med overlappende oppgaver, og da det var hensiktsmessig og nyttig. Ut over det jobbet vi mye hver for oss og holdt hverandre oppdatert digitalt via scrumchatten, scrumboardet på Trello, og et eget lukket forum hvor generell info om prosjektet ble delt.

For å holde kommunikasjon mot oppdragsgiver og veileder ryddig, bestemte vi at all kommunikasjon med disse skulle skje via prosjektleder. Rollen som prosektleder ble tildelt Trine uten motsigelser fra de andre gruppemedlemmene.

Gruppens veileder har vært professor Ivar Farup ved HiGs avdelig for informatikk og medieteknikk. Ivar har vært en viktig ressurs for gruppen. Han er kunnskapsrik innen fagområdet og har gitt oss nyttige tips og tilbakemeldinger gjennom utviklingsløpet.

1.6.3 Fremdriftsplan

Under forprosjektsperioden for prosektet satte vi sammen et Gantt-diagram for hvordan vi forventet at fremgangen i prosjektet skulle være (se forprosjektrapport, vedlegg O). Utviklingsløpet gikk stort sett etter planen i forhold til backend APIet, men når det kom til applikasjonen, fant vi etter hvert ut at utvikling av denne kom til å ta mye lenger tid enn først antatt. I utgangspunktet skulle vi i løpet av utviklingsperioden vurdere utvikling til iOS også, men dette ble aldri aktuelt fordi det var for mye arbeid som gjenstod på Android-applikasjonen, dersom vi skulle få med all den funksjonaliteten vi absolutt ville ha.

1.7 Videre i rapporten

I kapittel 2 i rapporten kommer vi til å gå inn på hvilke krav vi har forholdt oss til i dette prosjektet, både krav fra oppdragsgiver, men også krav vi har satt selv. Vi går inn på use caser og beskriver mange av funksjonene som var planlagt å utvikle i prosjektet.
I kapittel 3 går vi inn på teori og teknologi vi har brukt under utviklingsløpet. Her går vi inn på verktøybruk og noe teori vi har benyttet oss av. I kapittel 4 går vi gjennom brukergrensesnittet og hvilke elementer vi har måttet ta hensyn til i forhold til material design. I kapittel 5 skriver vi om hvordan backendløsningen er designet med database- og api-arkitektur. Her har vi diagrammer som viser hvordan databasen og APIet bygd opp. Kapittel 6 handler om selve implementeringen av applikasjonen. Her har vi diagrammer som beskriver design, skjermbilder som viser gjennomgang av applikasjonen, beskrivelser av hvordan noen funksjoner er implementert og kodesnutter som eksempler. Kapittel 7 handler om hvordan backendløsningen er implementert med dypdykk i hvordan koden fungerer i forskjellige script og konfigurasjonsfiler. Her har vi prøvd å dekke mange sider av hvordan APIet er bygd opp og fungerer.

2. Spesifikasjoner

2.1 Interessenters krav

Etter samtale med oppdragsgiver AndMark kom vi frem til noen få krav, selv om vi i stor grad fikk frie tøyler til å være kreative og komme med våre egne ideer. Oppgaveteksten setter også noen underliggende krav om funksjonalitet:

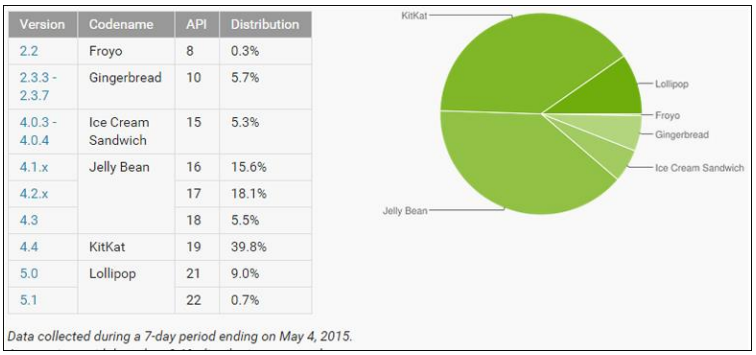
- Bruker skal kunne legge til sine egne steder
- Bruker skal kunne lage sine egne ruter
- Programvaren skal kunne brukes hvor som helst i landet
- Applikasjonen skal utvikles med designkonseptet Material Design og Android 5.0
- Det skal lages et enkelt RESTful API for henting av data.
- Applikasjonen skal utvikles under MIT-lisens

2.2 Tilleggskrav

2.2.1 Plattform

Etter samtaler med oppdragsgiver ble det bestemt at plattformen skal være Android. Vi la opp utviklingsløpet slik at utvikling til iOS også kunne vurderes etter hvert.

Oppdragsiver ønsket en applikasjon som var ny og moderne og ville derfor at vi utviklet for den nyeste versjonen av Android; Lollipop. For å nå en størst mulig brukergruppe innenfor denne plattformen valgte vi selv å sette bred avgrensning til bakoverkompabilitet for applikasjonen. Basert på dagens distribusjon av Android-versjoner [1] bestemte vi oss for å utvikle en applikasjon som er kompatibel med versjoner helt tilbake til API 15, noe som gjør at applikasjonen vil være tilgjengelig for 94% av alle Android-brukere. Se figur 2.1.



Version	Codename	API	Distribution
2.2	Froyo	8	0.3%
2.3.3 - 2.3.7	Gingerbread	10	5.7%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	5.3%
4.1.x	Jelly Bean	16	15.6%
4.2.x		17	18.1%
4.3		18	5.5%
4.4	KitKat	19	39.8%
5.0	Lollipop	21	9.0%
5.1		22	0.7%

Figur 2.1: Statistikk for distribusjon av Android-versjoner

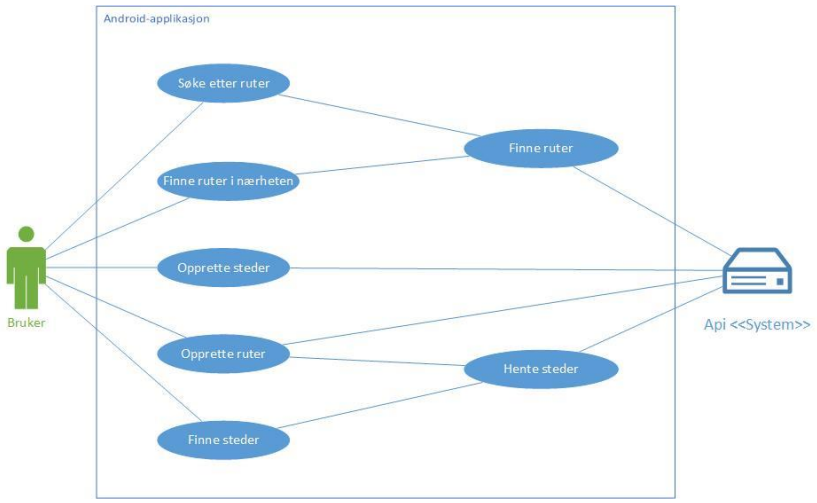
2.2.2 Internasjonalisering

Selv om applikasjonen skal utvikles for det norske markedet, er det en god vane å legge inn språk og tekst som en ekstern ressurs, fremfor å skrive tekst direkte inn i koden. Ved å gjøre dette vil det bli en enklere oppgave å gjøre applikasjonen tilgjengelig i andre markeder i fremtiden om dette skulle bli aktuelt, eller rett og slett å legge til nynorsk og samisk. Under utviklingsprosessen satte vi standardspråket til å være engelsk og oversatte denne språkpakken til bokmål til slutt.

2.3 Use Case diagram

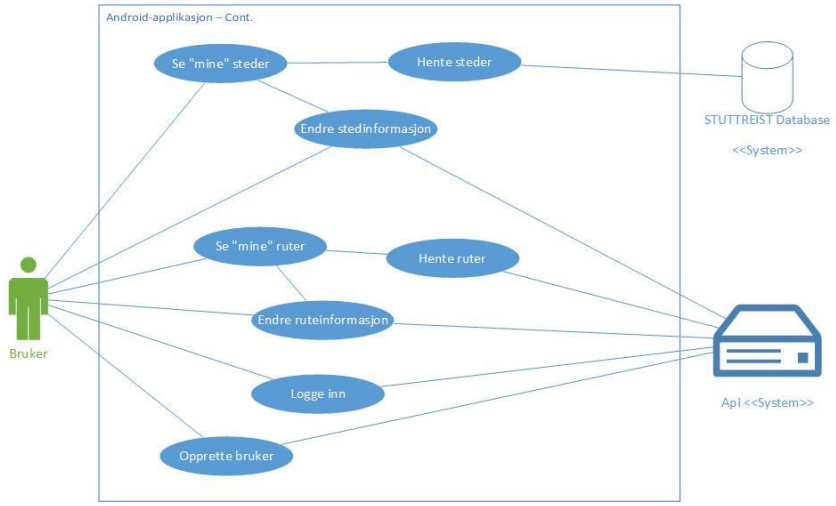
I use case diagrammene ser vi en oversikt over hovedfunksjonaliteten i løsningen. I første Use Case diagram, se figur 2.2, ønsker brukeren å kunne søke etter ruter og å finne ruter i nærheten av seg. Begge deler benytter seg av den samme operasjonen, men vi filtrerer ut aktuelle ruter på server-delen. I begge tilfeller går oppgaven ut på å finne ruter, noe som gjøres ved at det blir sendt en forespørsel til serveren om ruter.

Brukeren ønsker også å finne steder, opprette steder og opprette ruter. Dette er også en del av kjernefunksjonaliteten i løsningen. Oppretting av steder blir sendt som en direkte forespørsel til APIet, altså trenger vi ikke hente noen informasjon for å kunne opprette steder. Men for å finne steder, må vi først hente stedene fra APIet. Disse vises på et kart som brukeren kan navigere i for å finne steder. Det samme gjøres når brukeren skal opprette ruter, som består av en rekke steder brukeren har valgt ut.



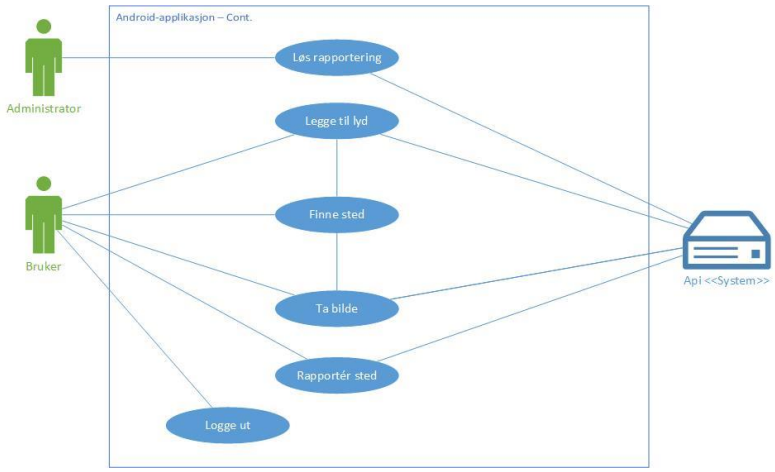
Figur 2.2: Use Case diagram for finne ruter, finne steder, opprette steder.

Brukeren ønsker også å se en oversikt over steder han/hun har opprettet kalt "mine steder", se figur 2.3. Her må brukeren også ha mulighet til å endre informasjon om stedet, som navn, beskrivelse osv. For å gjøre dette må vi også hente steder fra APIet, men vi filtrerer ut steder med brukerens ID. Det samme gjelder for ruter med at brukeren ønsker å se en oversikt over over "sine" ruter og kunne redigere disse. For å kunne gjøre dette må brukeren kunne opprette bruker og kunne logge inn, som vi også ser i samme Use Case diagram.



Figur 2.3: Use Case diagram for se/endre ruter/steder, innlogging, ny bruker.

I siste Use Case diagram, figur 2.4, ser vi at brukeren har mulighet til å legge til lyd og bilde til et sted. Her må brukeren først finne et sted som han/hun ønsker å legge til. Lyd kan bare legges til av eieren av stedet, maks én lyd per sted, mens alle kan legge til bilder. Det er ingen grense for antall bilder som er knyttet til et sted. Brukeren må også kunne logge ut fra applikasjonen. I figuren kan man og se at bruker skal ha mulighet til å rapportere steder, mens aktøren administrator har mulighet til å løse disse rapportene.



Figur 2.4: Use Case diagram for ta bilde, lydopptak, utlogging

2.4 Lavnivå use case

Vi har plukket ut tre av use casene vi har jobbet med, og vil gå gradvis mer detaljert gjennom hvordan disse er implementert i applikasjonen.

Use Case ID:	SR001
Use Case navn:	Legge til sted
Aktører:	Bruker
Beskrivelse:	Legge inn nytt sted/severdighet i applikasjonen
Utløses av:	Bruker velger "Legg til sted" fra sidemenyen til venstre
Forhåndsbetingelser:	1. Bruker er logget inn.
Postbetingelser:	1. Brukerens sted er lagt til i databasen. 2. Kartet oppdateres. 3. Bruker kan se det nye stedet på kartet.
Vanlig flyt:	1. Bruker trykker "Legg til sted". 2. "Legg til sted"-aktiviteten startes. 3. Bruker velger koordinater på kart. 4. Bruker fyller ut nødvendig informasjon. 5. Bruker trykker "Lagre". 6. Stedet lastes opp. 7. Bruker blir ført tilbake til startskjerm mens kart oppdateres. 8. Bruker åpner kart og kan se det nye stedet på kartet.
Alternativ flyt:	1. Trinn 1 og 2 over. 2. Bruker glemmer å finne koordinater eller fylle inn nødvendig informasjon. 3. Bruker trykker "Lagre". 4. Bruker blir bedt om å fylle inn all data. 5. Trinn 3 og 4 til nødvendig data er lagt inn. 6. Trinn 6-8 over.
Unntak:	1. Bruker har mistet internettforbindelsen.
Spesielle krav:	Ingen spesielle krav.
Forutsetninger:	Bruker må ha tilgang til internett og GPS på mobilen.
Notater og problemer:	Ingen kjente problemer hittil.

Use Case ID:	SR002
Use Case navn:	Opprette rute
Aktører:	Bruker
Beskrivelse:	Legge inn nytt sted/severdighet i applikasjonen.
Utløses av:	Bruker trykker på rute-ikonet i toolbaren.
Forhåndsbetingelser:	1. Bruker er logget inn. 2. Bruker har åpnet "se alle steder".
Postbetingelser:	1. Brukerens rute er lagt til i databasen. 2. Bruker kan se den nye ruten under menyvalget "mine ruter" eller ved å hente inn ruter.
Vanlig flyt:	1. Bruker trykker på en stedsmarkør. 2. "Vis mer"-aktiviteten startes. 3. Informasjonsboks vises nederst på skjermen. 4. Bruker trykker på ikonet for å legge til sted i rute. 5. Rute-ikonet i toolbaren oppdateres med nytt tall. 6. En toast gir tilbakemelding på at stedet er lagt til. 7. Repetér trinn 1-6. 8. Bruker trykker på rute-ikonet i toolbaren. 9. "vis ruteliste"-aktiviteten startes. 10. Kart med opptegnet rute og liste over steder i ruten vises. 11. Bruker trykker på "lagre". 12. "Lag ny rute"-aktiviteten startes. 13. Bruker fyller inn feltene. 14. Bruker trykker på "lagre". 15. Ruten lagres.
Alternativ flyt 1:	1. Trinn 1 - 13 over. 2. Bruker har ikke fyllt inn nødvendig informasjon. 3. Bruker trykker "Lagre". 4. Bruker blir bedt om å fylle inn all data. 5. Trinn 15 over.
Alternativ flyt 2:	1. Trinn 1 - 11 over. 2. Bruker trykker på krysset i toolbaren. 3. Bruker blir sendt tilbake til "alle steder".
Alternativ flyt 3:	1. Trinn 1 - 13 over. 2. Bruker trykker på krysset i toolbaren. 3. Bruker blir sendt tilbake til "alle steder".
Unntak:	1. Bruker har mistet internettforbindelsen.

Spesielle krav:	Ingen spesielle krav.
Forutsetninger:	Bruker må ha tilgang til internett og GPS på mobilen.
Notater og problemer:	Ingen kjente problemer hittil.

Use Case ID:	SR003
Use Case Navn:	Legge inn bilde
Aktører:	Bruker
Beskrivelse:	Legge inn et bilde på et sted brukeren har besøkt.
Utløses av:	Bruker klikker på kamera-ikonet som vises når bruker har klikket på et sted.
Forhåndsbetingelser:	1. Bruker er logget inn.
Postbetingelser:	1. Brukerens bilde er lagt til i databasen. 2. Bildet vises når brukeren går inn på detaljert beskrivelse av sted.
Vanlig flyt:	1. Bruker trykker på et sted på kartet. 2. Kortinfo om stedet kommer opp. 3. Bruker trykker på kamera-ikonet. 4. Bruker velger kamera-applikasjon. 5. Bruker tar bilde. 6. Bruker trykker “lagre”. 7. Bildet lastes opp i bakgrunn. 8. Bruker blir ført tilbake til kartet. 9. Bruker får tilbakemelding når bildet er lastet opp.
Unntak:	1. Bruker har mistet internettforbindelsen.
Spesielle krav:	Ingen spesielle krav.
Forutsetninger:	Bruker må ha tilgang til internett.
Notater og problemer:	Ingen kjente problemer hittil.

2.5 User stories

Her er noen av de mest sentrale user storyene vi har hatt i scrumboardet på Trello.

Navn	User Story	Operasjoner	Prioritet	Status
Se kart	Jeg som bruker ønsker å se et kart over alle stedene/severdighetene i nærområdet slik at jeg kan planlegge en tur eller lese mer om steder	Vise kart	Høy	Ferdig
		Legge inn punkter	Høy	Ferdig
		Vise stedsinformasjon	Høy	Ferdig

Navn	User Story	Operasjoner	Prioritet	Status
Finne ruter	Jeg som bruker ønsker å finne ruter i nærområdet slik at jeg kan gå denne turen senere	Velge radius det skal søkes i	Høy	Ferdig
		Finne ruter i gitt radius	Høy	Ferdig
		Vise ruter i liste og på kart	Høy	Ferdig
		Tekstbasert søk	Lav	Ferdig

Navn	User Story	Operasjoner	Prioritet	Status
Legg til sted	Jeg som bruker ønsker å legge til steder/severdigheter i applikasjonen slik at andre kan besøke disse.	Finne koordinater på kart	Høy	Ferdig
		Skrive inn stedsinformasjon	Høy	Ferdig
		Lagre stedet	Høy	Ferdig

Navn	User Story	Operasjoner	Prioritet	Status
Lag rute	Jeg som bruker vil sette opp en rute slik at jeg kan planlegge hvilke severdigheter jeg vil besøke neste gang jeg skal gå tur eller så andre kan gå denne ruten.	Velge steder i ruten	Høy	Ferdig
		Fylle inn informasjon om ruten	Høy	Ferdig
		Lagre rute	Høy	Ferdig

		Gjøre ruten offentlig	Middels	Ferdig
		Gjøre ruten privat	Lav	Åpen

Navn	**User Story**	**Operasjoner**	**Prioritet**	**Status**
Redigere sted	Jeg som bruker ønsker å endre informasjon om et sted fordi den er feil eller lite utfyllende.	Finne koordinater på kart	Høy	Ferdig
		Skrive inn stedsinformasjon	Høy	Ferdig
		Lagre stedet	Høy	Ferdig

Navn	**User Story**	**Operasjoner**	**Prioritet**	**Status**
Legge inn bilde	Jeg som bruker ønsker å legge inn bilder fra steder jeg besøker slik at andre kan se at jeg har vært der.	Sende bruker til kamera-applikasjon	Høy	Ferdig
		Hente bildet	Høy	Ferdig
		Laste opp bildet	Høy	Ferdig

Navn	**User Story**	**Operasjoner**	**Prioritet**	**Status**
Rapportere sted	Jeg som bruker ønsker å rapportere et sted fordi innholdet er feil eller bør fjernes fra applikasjonen slik at innholdet blir bedre.	Velge rapporteringsårsak	Høy	Ferdig
		Sende rapportering	Høy	Ferdig

Navn	**User Story**	**Operasjoner**	**Prioritet**	**Status**
Løse rapporteringer	Som administrator ønsker jeg en oversikt over alle rapporteringer og muligheten til å løse disse slik at innholdet i applikasjonen har bedre kvalitet.	Vise liste over rapporteringer	Høy	Ferdig
		Funksjonalitet for å løse	Høy	Ferdig

		rapporteringer		

Navn	**User Story**	**Operasjoner**	**Prioritet**	**Status**
Ta opptak	Som bruker ønsker jeg å lytte til informasjonen om stedet når jeg er der, slik at det blir som en guide på et museum.	Opptaksfunksjon	Medium	Ferdig
		Avspillingsfunksjon	Medium	Ferdig

3. Teori og Teknologi

3.1 Verktøy

All kode er samlet i to repositorier på BitBucket - en repository for backend og en repository for mobilapplikasjonen. Trello.com er brukt som scrumboard og til organisering av arbeidsflyten. Markører, ikoner og illustrasjoner er laget i Adobe Photoshop.
Vi opprettet en skjult gruppe og en egen scrumChat på Facebook til oppdateringer/beskjeder til hele gruppen. Nettsiden for bachelor-prosjektet er laget ved hjelp av WordPress. Modeller og diagrammer er laget i Microsoft Visio. Applikasjonen er utviklet i Android Studio. Backend APIet er utviklet i Notepad++ og PHPStorm, mens APIet ligger live på et privat domene fra One.com (storsveen.rocks/guide/).

3.2 Rest

Et av kravene fra oppdragsgiver var å lage et RESTful API for innsetting og henting av data. Vi har valgt å implementere dette veldig enkelt, men holde oss innenfor rammene til Rest. Rest står for Representational State Transfer. Lei Gao, Chunhong Zhang og Li Sun har skrevet en artikkel[2] om RESTful API i forhold til sensorer, og ut i fra denne artikkelen kan vi kort sagt si at det er en teori om en standard arkitektur om hvordan man skal designe et web-API. Poenget med et RESTful API er at man lager et felles grensesnitt for alle enheter, Windows, UNIX, Android osv, som behandler alle enheter likt og man lagrer ikke noe data om klienten[2]. Klienten skal enkelt kunne hente eller sette inn data uavhengig av hvilken klient det er. I et RESTful API bruker man HTTP-protokollen med GET, POST, UPDATE og DELETE.

Chin-Liang Tsai, Hsiao-Wen Chen, Jiun-Long Huang, Chih-Lin Hu har også skrevet en artikkel[3] om "Transmission Reduction" mellom Mobil-enheter og RESTful API. Her skriver forfatterene om hvordan man bør bruke GET, POST, UPDATE og DELETE. Tabell 3.1 er en gjenskapning fra en tabell som er et sammendrag fra kapittelet om hvordan bruke HTTP i RESTful API. Som vi ser skal GET brukes til SELECT-uttrykk, mens POST skal brukes til INSERT-uttrykk. I vår løsning går dette ofte om hverandre. Vi ønsker gjerne å hente data etter vi har satt inn osv, vi har derfor valgt å bruke POST på alt for å få konsistens.

Action	AQL	HTTP	Query Format
Create	Insert	POST	INSERT INTO [API].[Method] ([Parameter Key(s)]) VALUES ([Parameter Value(s)])
Read	Select	GET	SELECT [Field]//[Attribute] FROM [API].[Method] WHERE A or B A: [Key Value Parameters] B: [Key] IN ([SubSelect])
Update	Update	PUT	UPDATE [API].[Method] SET [New Value] WHERE [Key Value Parameters]
Delete	Delete	DELETE	DELETE FROM [API].[Method] WHERE [Key Value Parameters]

Tabell 3.1: Gjenskapning av tabell fra Transmission Reduction between Mobile Phone Applications and RESTful APIs[2].

Når APIet returnerer data ønsker vi å bruke en standard slik at alle forskjellige enheter, enten web, Android, iOS osv, skal kunne behandle det, og det skal være uavhengig av enhet. Derfor bruker vi JSON i alle resultatene våre. Lei Gao, Chunhong Zhang og Li Sun skriver i sin artikkel [2] at XML også er vanlig brukt, spesielt tradisjonelt, men vi bruker JSON fordi JSON har mindre overhead enn XML og JSON er enkelt å behandle både i Javascript og Java/Android, som også Lei Gao, Chunhong Zhang og Li Sun nevner i sin artikkel [2].

3.3 Database-engine

One.com tilbyr mange database-motorer som kan brukes gjennom phpmyadmin. Vi har valgt å bruke InnoDB Storage Engine fremfor One.com's standard MyISAM fordi MyISAM ikke støtter fremmednøkler. I følge rackspace[4] vil man i de fleste tilfeller ønske å velge InnoDB fremfor MyISAM. MyISAM er å foretrekke dersom man har en database som nesten utelukkende leser data, og ikke skriver data til databasen, fordi den leser raskere enn InnoDB. Et annet tilfelle er hvis ACID(Atomicity, Consistency, Isolation, Durability) skulle

være irrelevant siden MyISAM ikke bruker ACID vil det gå raskere. InnoDB er nødt til å gjøre sjekker med ACID og det vil da gå tregere.
Av andre database-motorer kunne vi valgt Blackhole, Aria, Memory, Archive osv. men InnoDB er sammen med MyISAM den mest brukte database-motoren og det er da enklere å finne kompetanse til å feilsøke hvis noe skulle gå galt.

3.4 Material Design

Material design har vært under utvikling i flere år, men gjorde først og fremst sitt inntog ved lanseringen av Android Lollipop i november 2014. Vi har forholdt oss til spesifikasjonene for material design ved utviklingen av GUIet til applikasjonen.

3.4.1 Farge

Fargene i material design er sterke, knalle og vibrerende. Design-spesifikasjonene definerer en fargepalett med farger som skal passe godt sammen og utfylle hverandre. Fargepaletten består av primærfarger og aksentfarger. Det skal velges én primærfarge, og én aksentfarge fra denne fargepaletten, og disse utgjør hele fargevalget i applikasjonen [5].

3.4.2 Tekst

Material design spesifiserer også hvordan teksten skal se ut, hvilken skrifttype som skal brukes og hvilken fargenyanse det skal være på teksten, ut i fra hvordan teksten er brukt. Skrifttypen som skal brukes er Roboto. For mørk tekst på hvit bakgrunn skal hovedteksten være 87% gjennomsiktig, sekundærtekst skal være 54% gjennomsiktig og hint- eller inaktiv tekst skal være 26% gjennomsiktig. Skillelinjer skal være 12% gjennomsiktige [6].

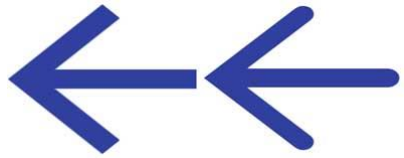
3.4.3 Ikoner

Ikoner skal skal være enkle og intuitive med enkle bakgrunner. Best practice er at alle kanter skal være rette, ikke avrundede som vi ofte har sett tidligere [7]. Se figur 3.2. Siden ikonene vil bli brukt på skjermer med forskjellig oppløsning, bør alle ikoner finnes i forskjellige størrelser - mdpi, hdpi, xhdpi, xxhdpi og xxxhdpi.

Standard gjennomsiktighet for aktive ikoner på lys bakgrunn er 54% av #000000. Inaktive ikoner skal ha en gjennomsiktighet på 26%.

Ikoner i toolbaren skal være 32x32 dp, hvor selve ikonet er 24x24 dp, og resten er padding. Oppstartsikonet skal være 48x48 dp til mobiltelefoner, og 512x512 dp til Google Play. For å sikre at ikonene kan skaleres uten å bli uskarpe eller udetaljerte, er det anbefalt å bruke vektorer.

Ikoner skal følge navnekonvensjonen for ikoner. Det vil si at ikonene navngis slik at ikoner som hører sammen, også vil grupperes sammen når de sorteres alfabetisk . Ikoner skal ha prefikset ic_ etterfulgt at hvor ikonet hører til, og deretter ikonet sitt navn [8].



Figur 3.2 Egen illustrasjon. Pilen til venstre har ikke avrundede kanter, og dette er best practice. Pilen til høyre har avrundede kanter, dette er ikke ok i material design [7].

3.4.4 Dialoger

Dialogbokser skal ha 24 dp padding. Mellomrom mellom tittel og tekst skal være 20 dp. "AVBRYT" - negativ handling - skal alltid ligge til venstre, mens "OK" - positiv handling alltid ligger til høyre. Begge deler skal være sentrert til høyre, og plassert nederst i dialogboksen.

Teksten skal alltid være i store bokstaver, og fargen skal være aksentfargen. I fulllskjermsdialoger ligger avbryt- og bekreft-handlingene i toolbaren. Avbryt-handlingen vises som en X til venstre. Denne har samme funksjonalitet som tilbake-pilen, altså indikerer den at ingenting vil lagres, og brukeren kommer tilbake til forrige skjermbilde. Bekreft-handlingen vises med et ord. Her er det viktig å bruke tydelige ord for å beskrive handlingen, ikke ord som kan være tvetydige, som "ok" eller "lukk". Brukeren bør være klar over hva handlingen medfører. "LAGRE", "LEGG TIL", "OPPDATER" er fine utvetydige ord [9].

Knapper i dialogbokser skal ikke ha noen bakgrunn, for å unngå at det blir for mange lag.

3.5 Navigation Drawer Pattern

Navigation drawer er en sidemeny som sklir inn fra venstre.

Navigation drawer pattern beskriver hvordan navigation drawer skal se ut, og har samme retningslinjer som lister [11].

- Skrifttype skal være Roboto Medium, med skriftstørrelse 14sp.
- Fargen på teksten skal være uten alpha-verdier, altså ikke gjennomsiktig.
- Eventuelle undermenyer skal være 87% gjennomsiktig.
- Ikoner skal være 16 dp fra venstre og høyre kant.
- Fargen til ikoner skal være 54% gjennomsiktig.
- Selve innholdet som hører sammen med ikonet på linjen skal være 72 dp fra venstre kant.
- Det skal være 25 dp mellomrom mellom navigation drawer og statusbaren.
- 8 dp padding mellom innholds-områdene, det vil si at det skal være 8 dp padding i toppen og bunnen av hver listegruppering, med unntak av eventuelle underoverskrifter, da de har egen padding.

Navigation drawer skal ligge under statusbaren, og er like høy som høyden på skjermen. Siden draweren ligger over annet innhold, er dette innholdet fortsatt synlig, men det blir mørkere. Innholdet i navigation drawer vises som en liste. Når et element i denne listen trykkes på, får elementet fargen til applikasjonens primærfarge, sammen med en ripple-effekt for å indikere at dette elementet er valgt.

Skillelinjer strekker seg over hele draweren, uten padding på høyre og venstre side, og med 8 dp padding på over- og undersiden av hver skillelinje. Dette er det samme som gjelder for lister [9].

3.6 Singleton-patternet

For å holde på stedene i applikasjonen valgte vi å implementere et Singleton-pattern. Dette patternet brukes i situasjoner hvor det er hensiktsmessig å ha kun én instans av en klasse. En klasse som følger patternet Singleton initierer seg selv slik at det kun finnes én av den. I

tillegg har den også en getInstance() funksjon som returner instansen. Dette er da den globale aksessmetoden som brukes hver gang funksjoner i klassen skal benyttes [12]. I vår kode er det klassen Places som følger dette patternet. Denne klassen holder på alle stedene som er lastet ned og lagt til i den lokale databasen, og sikrer at alle klasser i applikasjonen jobber mot de samme stedene. Klassen kan sees i helhet i vedlegg K - Places.class

4. Implementering av brukergrensesnitt

4.1 Farger

Vi valgte å bruke indigo som primærfarge, og lilla som sekundærfarge. I henhold til design-spesifikasjonene til material design er hovedfargen 500-verdien av indigo, og statusbaren er 700-verdien. I colors.xml har vi kalt disse colorPrimary og colorPrimaryDark. Aksentfargen er brukt i knapper og til kategori-ikonene. Se figur 4.1.

Figur 4.1: Fargesammensetningen i Stuttreist

4.2 Tekst

For å gjøre det enkelt, har vi definert alle tekstfargene i colors.xml, med navnene colorPrimaryText, colorSecondaryText og colorHintText. I knapper med lilla bakgrunn har vi brukt colorButtonText (#FFFFFF), mens på knapper uten bakgrunn har vi brukt colorAccent – aksentfargen.

4.3 Ikoner

Vi har brukt mange av standard-ikonene som hører til material design, hentet fra developer.android.com [13], men vi har også vært nødt til å lage en god del egne applikasjonsspesifikke ikoner. Her har vi forsøkt å forholde oss til retningslinjene som er gitt for ikoner, i forhold til bredde på linjene, ikke bruk av avrundede kanter og at det finnes ikoner i størrelsene mdpi, hdpi, xhdpi, xxhdpi og xxxhdpi.

Alle ikoner vi har laget er like høye som de er brede, for å unngå at de blir skjeve ved skalering.

Kategori-ikoner og kartmarkører er også laget av oss. Kategori-ikonene er enkle og intuitive hvite illustrasjoner på lilla bakgrunn (aksentfargen). Se figur 4.2. Dette er de samme fargene som er brukt i knappene, for å gi en helheti fargesammensetningen i applikasjonen. Kartmarkørene har samme illustrasjoner som kategori-ikonene, men de har i tillegg fått hver sin farge. Se figur 4.3. Markøren blir mørk grønn når brukeren trykker på den, for å indikere at den er valgt.

Figur 4.2: Våre egne ikoner.

Figur 4.3: Våre kartmarkører. Alle markørene i sin farge, historie-markøren til høyre er mørkegrønn og indikerer at markøren er valgt.

Alle ikoner er navngitt i henhold til navnekonvesjonen for ikoner. Det vil si at filene navngis slik at ikoner som hører sammen også vil grupperes sammen når de sorteres alfabetisk [6]. Derfor har vi navngitt alle ikoner med prefikset ic_, deretter følger angivelse av hvor ikonet hører til, og så navnet.

Navigation drawer: ic_drawer_navn

Meny: ic_menu_navn

Kategorier: ic_cat_navn

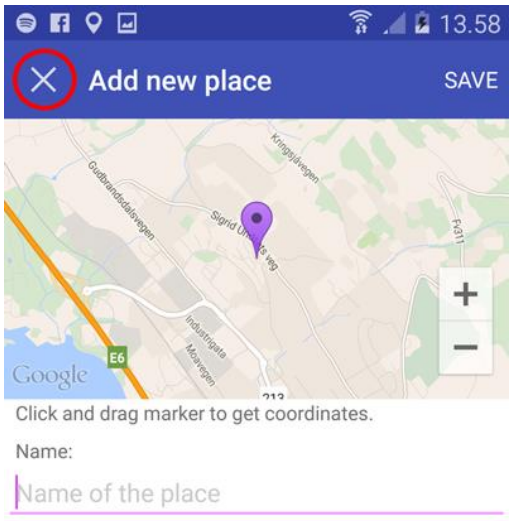
Ikoner i knapper: ic_btn_navn

Dialoger: ic_dialog_navn

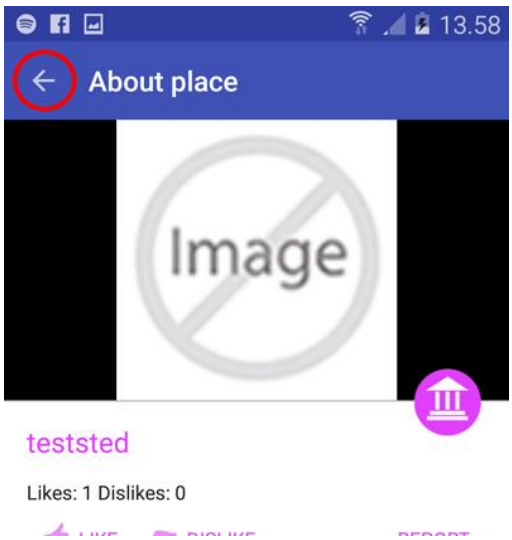
4.4 Dialoger

Vi har brukt to typer dialoger – fullskjerms dialoger og dialoger som bare bare tar opp en del av skjermen, mens bakgrunnen blir mørkere.

En viktig forskjell mellom dialoger og aktiviteter i applikasjonen, ligger i toolbaren. I fullskjermsdialoger vises et kryss i stedet for tilbake-pilen. Se figur 4.4 og figur 4.5.



Figur 4.4: Krysset indikerer at brukeren går tilbake uten å lagre, mens lagre-knappen vises til høyre.



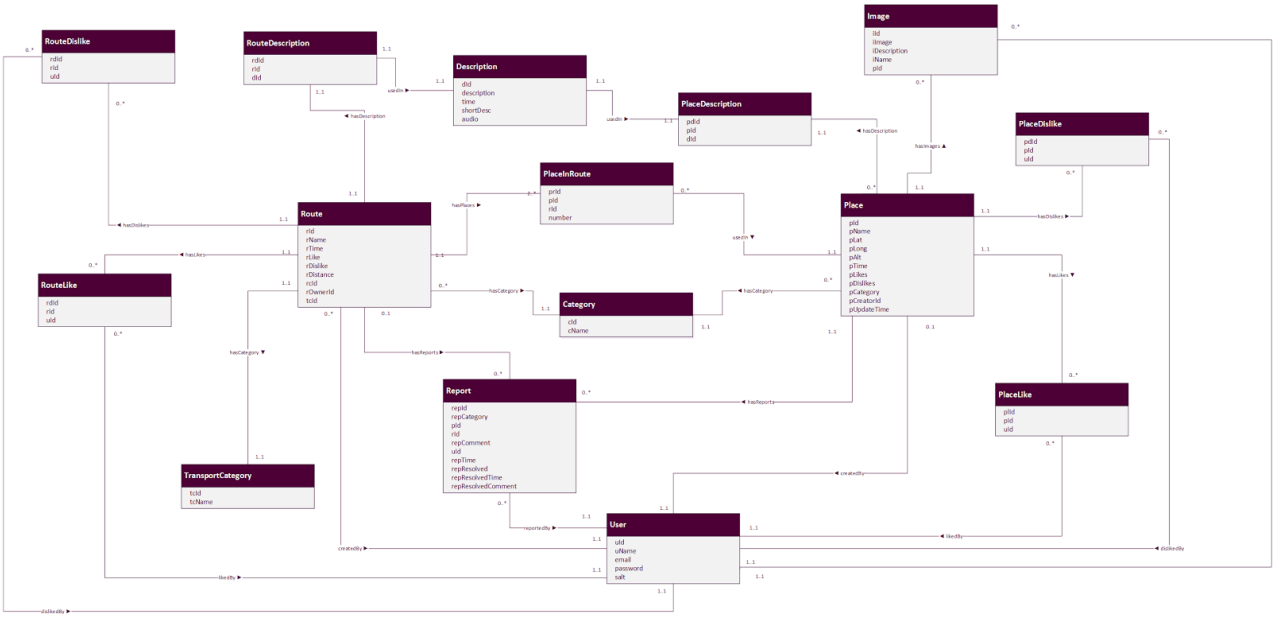
Figur 4.5: Tilbake-pilen indikerer at brukeren går tilbake til den forrige aktiviteten.

4.5 Navigation Drawer

I vår applikasjon vil navigation drawer, som er hovedmenyen, være åpen første gang applikasjonen kjøres, for at brukeren skal vite at det er her menyen er. Etter dette vil navigation drawer være lukket ved oppstart av applikasjonen. Den kan åpnes ved å trykke på ikonet til venstre i menyen, eller ved å dra fingeren fra venstre mot høyre over skjermen. På samme måte kan man også lukke navigation drawer.

5. Backend arkitektur

5.1 Server database



Figur 5.1: Database UML.

Kjernene i databasen på serversiden er tabellene Place og Route, som fungerer som samlingspunkter for alle de andre database-tabellene. For fullstendig UML-diagram, se vedlegg A.

5.1.1 Place-tabellen

Place-tabellen, se figur 5.2, inneholder informasjon om hvert sted - stedets id, navn, lengde- og breddegrad, høyde over havet, tidspunkt for når stedet ble lagt til, antall likes, antall dislikes, hvilken kategori stedet tilhører, hvem som har opprettet stedet og når stedet sist ble oppdatert. Fremmednøkkelen pCreatorId knytter Place-tabellen til User-tabellen, slik at vi vet hvilke brukere som har opprettet hvilke steder. Ett sted kan bare være opprettet av én bruker, mens én bruker kan opprette flere steder. Siden vi har et én-til-mange forhold, legger vi user-ID rett i Place-tabellen. På denne måten kan vi sikre at et sted bare kan opprettes av en eksisterende bruker, og at en bruker ikke kan slettes helt uten videre, dersom denne brukeren har opprettet et sted. Dette vil uansett fanges opp av scriptene, men kan ses på som en ekstra sikkerhet på database-siden.

Place-tabellen lagrer to tidspunkt: pTime og pUpdateTime. pTime lagrer tidspunktet for når stedet ble opprettet, mens pUpdateTime lagrer tidsspunktet for når stedets data sist ble oppdatert. Dette gjelder også oppdateringer som skjer i andre tabeller. Vi har et script som tar seg av dette. Vi kunne også ha laget en "trigger" i databasen som oppdaterer pUpdateTime i Place-tabellen hver gang en beskrivelse blir oppdatert, men dette er ingen kritisk del av løsningen, og har ikke blitt prioritert.

Figur 5.2: Place-tabellen.

Figur 5.3: Route-tabellen.

5.1.2 Route-tabellen

Route-tabellen, se figur 5.3, holder hver rutes unike id, navn, tidspunkt ruten ble opprettet, antall likes og dislikes, hvor lang ruten er, id for kategori, hvem som opprettet ruten og transport id. rcid, rOwnerId og tcId er fremmenøkler til tabellene Category, User og TransportCategory.

5.1.3 Category og TransportCategory

Category er koblet til Route og Place. Mellom disse er det fremmednøkkel som gjør at Place og Route bare kan ha kategorier som er definert fra før. Vi ønsker at Place og Route bare skal ha de forhåndsdefinerte kategoriene. Det samme gjelder for TransportCategory, men denne er bare koblet til Route, da transporttyper ikke er av noen betydning for et sted. Hvert sted eller rute kan bare ha én kategori, mens én kategori kan tilhøre flere steder. Hver rute kan bare ha én transportkategori, mens hver transportkategori kan tilhøre flere ruter.

5.1.4 Likes og Dislikes

Hver rute og hvert sted kan likes eller dislikes. Likes og dislikes lagres i hver sine tabeller, hvor PlaceLike og PlaceDislike har de samme kolonnene: pdId, pId, uId - En unik id for hvert felt, place og user.
Hver rad representerer én like eller dislike, og én user-ID kan bare forkomme sammen med én place-ID én gang. Altså kan en bruker kun like eller dislike ett sted én gang.
Det samme gjelder RouteLike og RouteDislike, som også har samme kolonner: rdId, rId, uId - unike id-er for hvert felt, route og user, slik at en bruker kun kan like eller dislike en rute én gang.

Fremmednøklene i RouteLike - rId og uId, gjør at det ikke kan opprettes en rad i RouteLike uten at det eksisterer en user-Id eller place-ID.

Vi var innom muligheten for å slå sammen PlaceLike og RouteLike. Dermed ville vi fått en tabell som i Tabell 5.1. under.

table-ID	place-ID	route-ID	user-ID
1	5	NULL	27
2	NULL	11	27

Tabell 5.1: RouteLike og PlaceLike slått sammen.

Her ville det blitt for mange tomme felter i tabellen. Vi valgte derfor å splitte opp i flere tabeller. Vi brukte den samme tankegangen når det gjelder PlaceDislike, RouteDislike, PlaceDescription og RouteDescription.

5.1.5 Image

Bilder er kun knyttet til steder. Hvert bilde kan bare tilhøre ett sted, mens hvert sted kan inneholde mange bilder. Dermed får vi et én-til-mange forhold mellom Image- og Place-tabellen. Vi må ha en place-ID for å kunne legge inn en rad i Image-tabellen. Image-tabellen har en kolonne av typen Blob, her ligger selve bildet binært. I tillegg har Image en fremmednøkkel forbundet med User, slik at vi kan vite hvilke brukere som har lastet opp

hvilke bilder. Denne fremmednøkkelen kan ikke være NULL. Vi har lagt til rette for navn og beskrivelse av bilder i databasen, men dette er ikke implementert i applikasjonen.

5.1.6 Description

Description-tabellen inneholder beskrivelse, kort beskrivelse og eventuell lyd til steder og ruter. Vi kunne i løsningen ha lagt beskrivelse rett i Route og Place, vi har valgt å legge det i en egen tabell, fordi vi ønsker å ta vare på tidligere beskrivelser som er gjort av ruter og steder, i tilfelle noe skulle overskrives ved en feiltakelse, eller at en bruker legger inn en beskrivelse som inneholder feil.

5.1.7 User

I User-tabellen lagres brukere og nødvendige brukerdata. Oppdragsgiver hadde kun krav om at vi ikke lagrer passord i klartekst, så vi har lagd en enkel hashing med salt som lagres i tabellen. User sin user-ID(uId) blir brukt som fremmednøkkel flere steder i databasen for å ha oversikt over hvilke brukere som har eierskap til steder, ruter, likes osv. Denne tabellen brukes også når brukere skal logge inn i applikasjonen.

5.1.8 Report

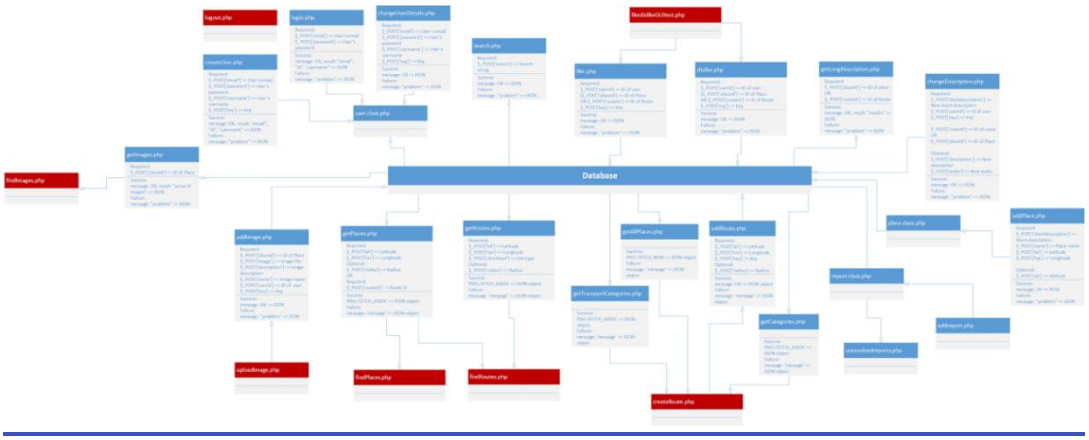
I Report-tabellen lagres alle rapporteringer som er blitt gjort på steder i applikasjonen. I tillegg til å holde på data om en rapportering har den også felter for å lagre informasjon om når og hvordan en rapportering er blitt løst.

5.1.9 PlaceInRoute

Siden Route-tabellen og Place-tabellen har et mange-til-mange-forhold bruker vi PlaceInRoute som hjelpetabell for å vite hvilke hvilke steder som hører til hvilke ruter og motsatt. Grunnen til at Route-tabellen og Place-tabellen har et mange-til-mange forhold er at det kan være mange steder i en rute og et sted kan høre til i flere forskjellige ruter. PlaceInRoute holder også styr på hvilket nummer i rekken hvert sted har i en rute i kolonnen “number”. F.eks. hvis et sted er det første stedet i en rute får den “number” 1.

5.2 API oppbygging

Figur 5.4 viser en oversikt over hvordan APIet er bygget opp. Blå representerer deler av APIet, mens Rød er testfiler vi har laget for å teste at APIet fungerer. Vi hadde en plan om å bygge ut de røde testfilene til en fullverdig webløsning hvor en administrator kunne ha muligheten til å enkelt administrere databaseinnholdet, som ruter, punkter osv.



Figur 5.4: Backend filoversikt. Blå = del av API, Rød = test-filer. Se stor versjon i vedlegg B.

Da vi begynte utviklingen av APIet fantes ingen Android-funksjonalitet til å sjekke at APIet fungerte, derfor lagde vi enkle GUI-filer for dette. Da vi senere var kommet lengre i utviklingen av appen, begynte vi å teste den nyeste funksjonaliteten direkte i applikasjonen, uten å lage noen ekstra GUI-filer for webløsningen. Dersom oppdragsgiver er interessert i å videreutvikle applikasjonen og webløsningen, ligger disse filene klare til bruk, alt som trengs er et penere brukergrensesnitt.

På noen områder, slik som bruker, rapport og sted, har vi valgt å bruke klasser som opererer direkte på databasen, og all ny funksjonalitet bruker disse klassene. Mens på annen, enkel, funksjonalitet opererer scriptene direkte på databasen. Dette kan vi se i figur 5.4.

Alle script som gjør innsettinger i databasen, tar imot data. Det varierer hva slags data de mottar, men alle tar en "nøkkel". Dette er for at ikke innsettingsfunksjonaliteten skal bli misbrukt og databasen blir fylt opp av søppeldata. Vi tillater alle å hente ut data uten å ha noen nøkkel, siden dette er data man finner i applikasjonen. Hvis AndMark senere skulle videreutvikle løsningen, eventuelt lage en ny applikasjon basert på back-end løsningen vi har

utviklet, vil det være aktuelt med en nøkkel for hver ny løsning som tillater innsetting av data.

Flere steder er bruker-ID med ved innsetting av data, slik at vi har oversikt over hvilke brukere som har opprettet hvilke steder, lastet opp hvilke bilder, og laget hvilke ruter. Noen filer, som f.eks. getRoutes.php og changeDescription.php, i figur 5.4, tar imot valgfrie data. I getRoutes.php kan man velge å sende med den radius man ønsker å finne ruter innenfor. Hvis denne ikke blir satt, setter scriptet en standard verdi for å finne ruter. I changeDescription.php kan man velge å skrive inn "description," hvis ikke blir denne satt til NULL, og NULL-verdien blir lagt til i databasen. Dette er akseptert, siden vi har bestemt at steder og ruter ikke er påtvunget å ha noen beskrivelse, bare en kort beskrivelse.

Enkelte "get"-filer, slik som getCategories.php, getAllPlaces.php, tar ikke noen data. Disse har standard oppsett og returnerer akkurat de dataene de er satt til å hente - henholdsvis kategorier og alle steder.

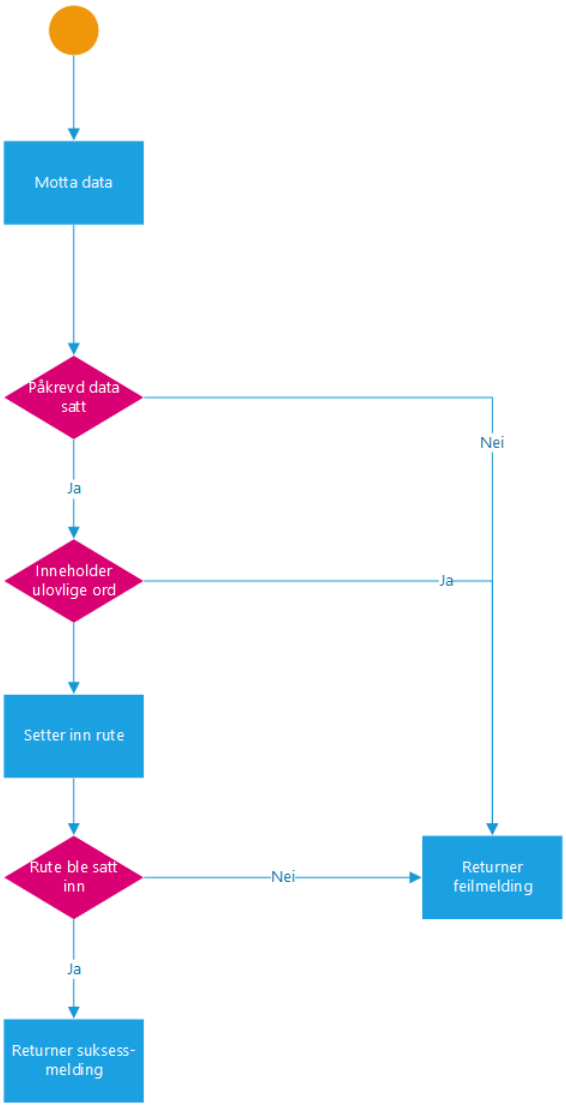
I databasen er bilde- og lydfiler av typen Blob. Dette er en binær datatype. Siden APIet vi har utviklet er JSON-basert, er det vanskelig å sende binære data. Det samme gjelder å sende binære data fra Android-enheter til APIet. Derfor har vi valgt å base64-enkode disse. Slik som addImage.php og changeDescription.php, i figur 5.4, tar imot en base64-enkodet string og dekoder denne, slik at den igjen blir binær før den bli satt inn i databasen. Når disse hentes ut, slik som i getImages.php blir de base64-kodet før de legges inn i JSON-resultatet.

5.3 Flytdiagram: Opprett rute i API

I figur 5.5 ser vi flytdiagram for innsetting av rute i APIet. Dette er en typisk situasjon for innsetting av data. Diagrammet starter med at scriptet mottar et sett med data, deretter sjekkes det om alle påkrevde data er satt. Dersom det er manglende data, returneres passende feilmelding i forhold til hvilke data som ikke er satt. Hvis det ikke er manglende data sjekkes det om beskrivelse, kort-beskrivelse eller rute-navn inneholder en av ordene i en liste med ikke tillatte ord. Hvis beskrivelse, kort-beskrivelse eller rute-navn inneholder et ikke-tillatt ord returneres en feilmelding om dette, se kode under. Hvis teksten som settes inn ikke inneholder noen ulovlige ord, vil scriptet prøve å sette inn ruten gjennom en rekke

SQL-Inserts. Hvis en av disse feiler vil det returneres en feilmelding om dette. Hvis ruten ble satt inn returneres en suksess-melding.

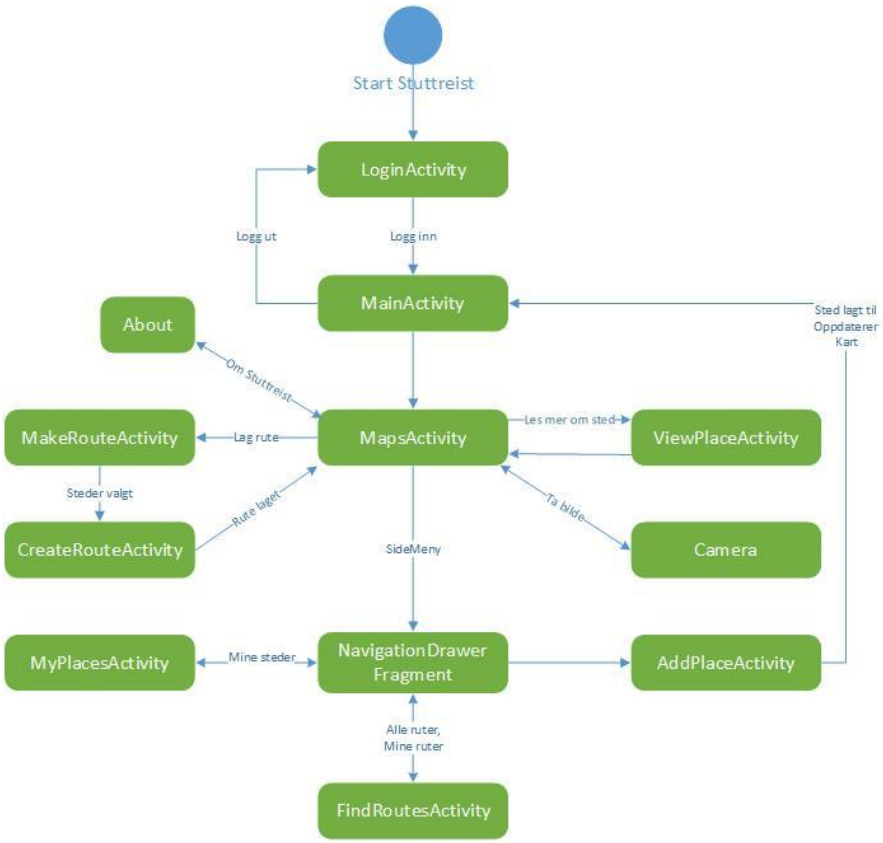
```
/*
* Checks if string contains any "illegal" words.
*
* Code from:
* http://stackoverflow.com/questions/13795789/check-if-string-contains-word-in-array
01.05.15
*/

function checkString($string) {
   $stringArray = array("faen", "fuck", "sex");
   foreach($stringArray as $item) {
      if(stripos($string, $item) !== false)
         return true;
   }
   return false;
}
```



Figur 5.5: Flytdiagram for innsetting av rute i APIet.

6. Implementering av applikasjonen

Da vi startet med implementeringen av applikasjonen begynte vi først med MapsActivity, som er selve hovedaktivitenen som knytter resten av aktivitetene sammen. Siden vi har jobbet sekvensielt, har vi lagt inn én og én aktivitet, helt til vi til slutt endte opp med aktivitene vist i figur 6.1. Figuren viser hvordan de forskjellige aktivitetene i applikasjonen henger sammen. Videre i kapittelet går vi nærmere inn på hvordan dette er implementert. Klassediagram for applikasjonen kan sees i vedlegg C.



Figur 6.1: Kart over aktivitetene brukeren kan benytte seg av og hvordan de henger sammen.

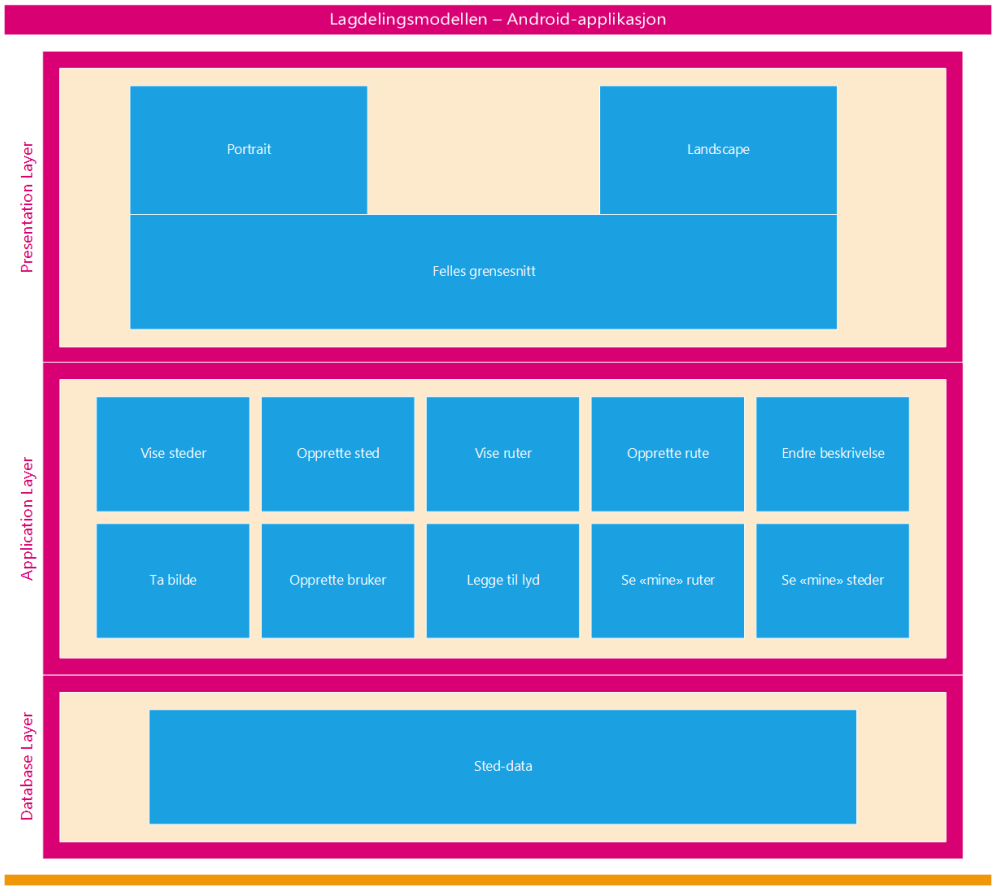
6.1 Lokal database - UML

I applikasjonen har vi bare én tabell som lagres lokalt, se figur 6.2. Tabellen holder alle steder vi henter fra APIet slik at de raskt hentes inn hver gang brukeren åpner applikasjonen. Tabellen er en samling av data fra flere tabeller fra databasen som ligger på server. Vi har gjort det på denne måten fordi applikasjonen ikke trenger å ta vare på tidligere beskrivelser og kort-beskrivelser. Her ligger også kategoriene i klartekst og ikke bare kategoriens ID. Den lokale Place-tabellen blir oppdatert når nye steder hentes inn fra APIet.

Figur 6.2: Oversikt over database-tabell i applikasjon.

6.2 Logical view

6.2.1 Lagdelingsmodellen



Figur 6.3: Lagdelingsmodell.

6.2.2 Presentation Layer

Som vi ser i Presentation Layer, i figur 6.3, fungerer applikasjonen både i portrett- og landskapsmodus. Begge disse er basert på samme felles grensesnitt. Den beste brukeropplevelsen gis ved bruk i portrettmodus, men landskapsmodus er også mulig. Når bruker velger å bruke landskap skaleres elementene, slik at de passer den nye visningen.

6.2.3 Application Layer

I Application Layer i figur 6.3 ser vi noen av operasjonene applikasjonen skal håndtere. Her er det snakk om alt applikasjonen skal gjøre av visning av kart, henting av data, sending av data, håndtering av bruker-input og behandling av binære filer og sensorer. Elementene i Application Layer er fordelt på samme måte som applikasjonen logisk sett er delt.

6.2.4 Database Layer

Database Layer, figur 6.3, viser databasetabellene vi har i applikasjonen. Her bruker vi bare én tabell som holder steder og den informasjonen om stedene som vises i applikasjonen. Application Layer henter stedsinformasjon fra denne tabellen, og viser navn, beskrivelse, posisjon osv, og viser de på kartet eller i lister.

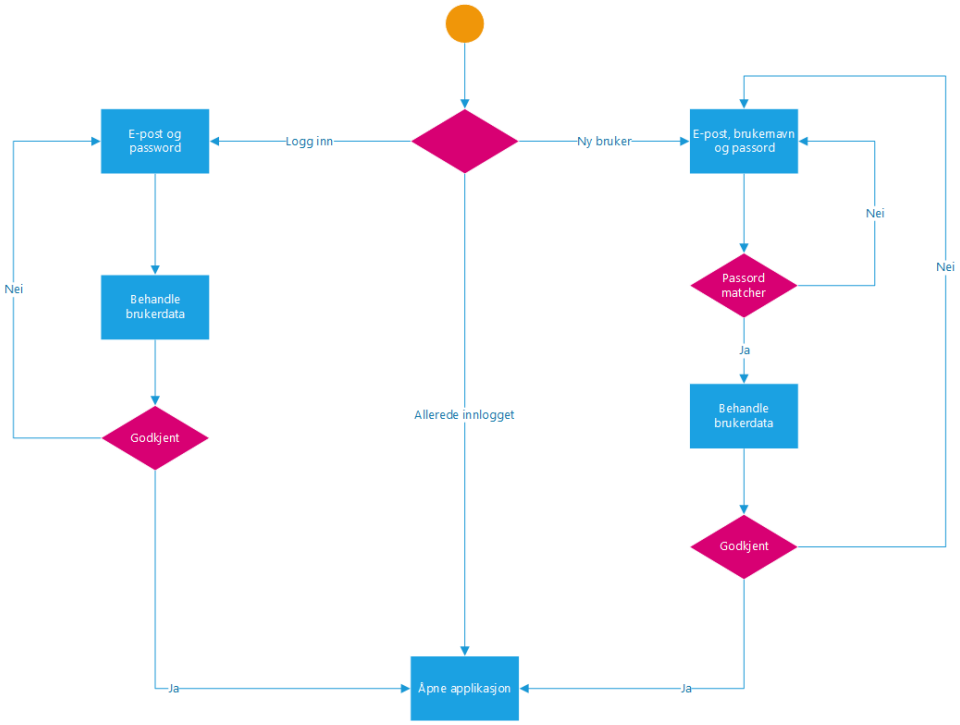
6.3 Process view

6.3.1 Flytdiagram for innlogging

I figur 6.4 ser vi flytdiagram for innlogging/oppretting av ny bruker i applikasjonen. I det brukeren starter applikasjonen sjekkes det om brukeren er logget inn. Hvis brukeren allerede er logget inn, får brukeren tilgang til applikasjonen. Hvis brukeren ikke er logget inn, kan han/hun velge mellom å logge inn med en eksisterende bruker, eller opprette ny bruker. Hvis brukeren velger å logge inn med eksisterende bruker blir han/hun bedt om å fylle inn e-post-adresse og passord. E-post-adressen og passordet blir sendt til APIet for sjekk. Svaret returneres til applikasjonen. Hvis innloggingen ble godkjent kan brukeren bruke applikasjonen, hvis den ikke blir godkjent blir brukeren bedt om å fylle inn e-post-adresse og passord på nytt.

Hvis brukeren ønsker å opprette ny bruker blir han/hun bedt om å fylle inn brukernavn, e-post-adresse, passord og bekrefte passord. Hvis passordene ikke er like vil brukeren bli bedt om å fylle inn på nytt. Hvis passordene er like vil brukerdataene bli sendt til APIet for oppretting av bruker.

Hvis opprettelsen blir godkjent vil brukeren bli sendt videre og kan bruke applikasjonen, hvis opprettelsen ikke ble godkjent, ved at for eksempel e-postadresse allerede er registrert, vil brukeren bli bedt om å fylle inn brukerdata på nytt.



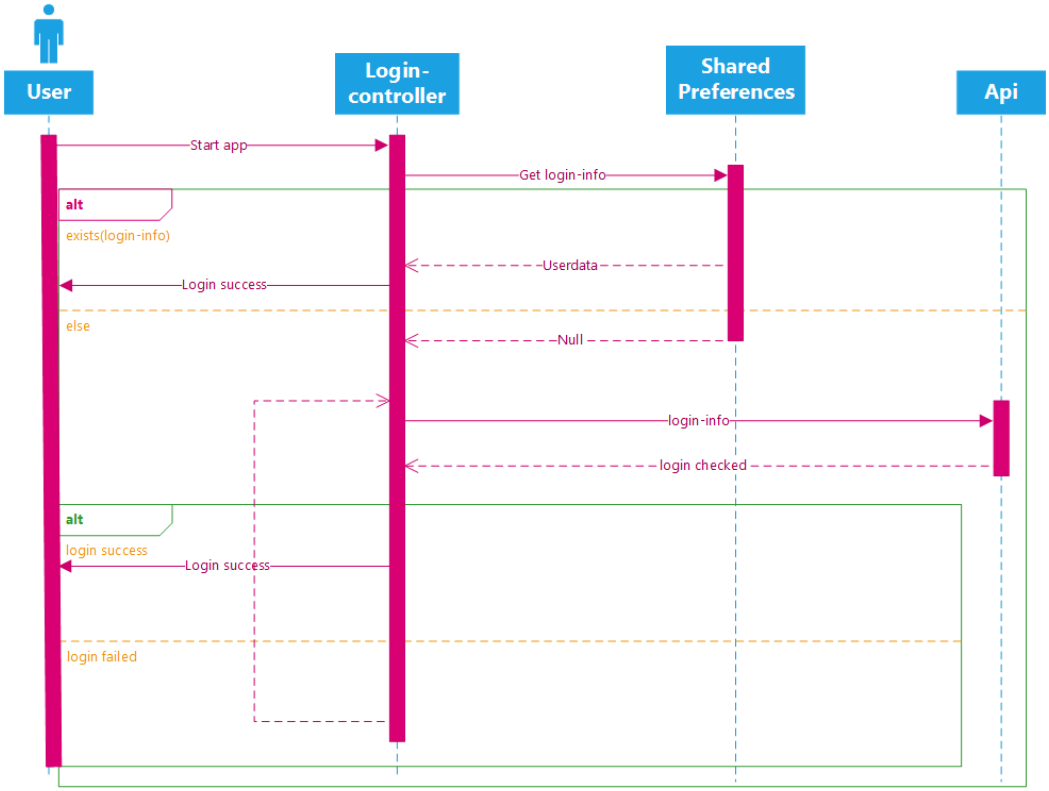
Figur 6.4: Flytdiagram for innlogging.

6.4 Sekvensdiagram for innlogging

Som vi kan se i figur 6.5, starter brukeren med å åpne applikasjonen. Deretter prøver applikasjonen å hente brukerdata fra Shared Preferences. Hvis applikasjonen får hentet ut brukerdata, vil applikasjonen fortsette videre. Hvis applikasjonen ikke mottar brukerdataene fra Shared Preferences vil brukeren bli bedt om å taste inn e-post/passord. Disse dataene vil bli sjekket med server. Hvis dataene stemmer, får vi positivt svar fra server og applikasjonen vil la brukeren benytte applikasjonen. Hvis dataene ikke stemmer vil brukeren bli sendt tilbake til innloggingsvinduet hvor brukerdata må tastes inn på nytt.

Kort sagt lagrer vi brukerinformasjon som e-post og bruker-ID i Shared Preferences, som brukes når vi sender forespørsler til server. Når brukeren starter applikasjonen, vil det først sjekkes om det ligger noe data i login-dataene i Shared Preferences, og hvis det gjør det starter applikasjonen, ellers må brukeren logge inn. Når brukeren klikker "logg ut" slettes innloggingsinformasjonen i Shared Preferences og applikasjonen avsluttes. Neste gang

applikasjonen starter vil det ikke ligge noe data i Shared Preferences, og brukeren vil få opp innloggingsvinduet.



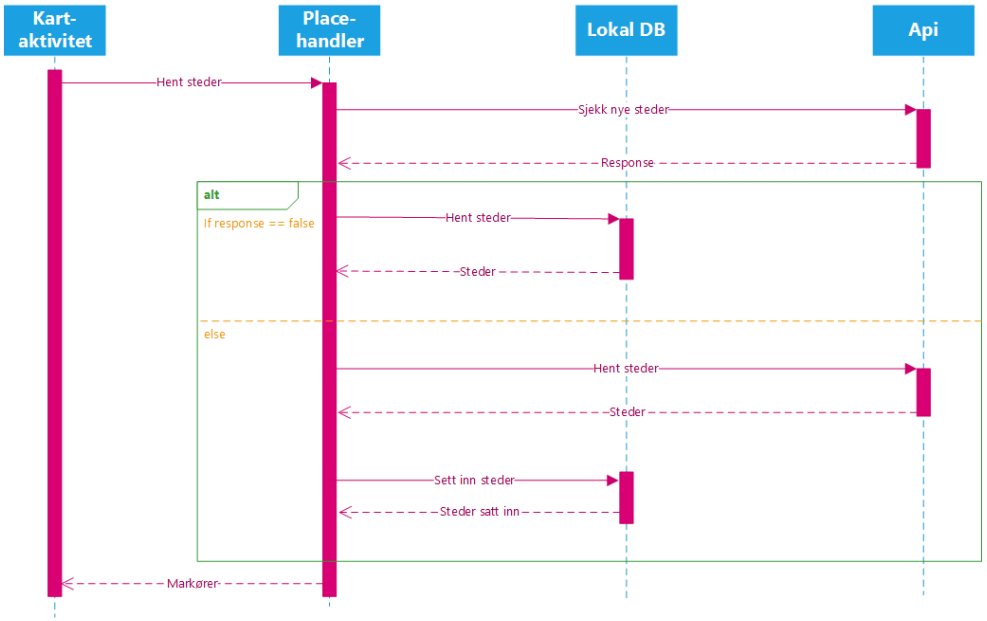
Figur 6.5: Sekvensdiagram for innlogging.

6.5 Sekvensdiagram for henting av steder

Ut fra figur 6.6 kan vi lese at brukeren starter med å prøve og hente steder. Deretter sender Place en forespørsel til APIet og spør om det er noen nye steder. Hvis det ikke er noen nye steder, hentes kun stedene fra den lokale databasen og returnerer de som kart-markører til brukeren. Hvis det er opprettet nye steder, vil Place hente stedene fra server og erstatte stedene i databasen med disse, deretter returneres disse nye stedene som markører til brukeren.

I praksis gjøres dette ved å hente siste oppdateringstidspunkt fra den lokale databasen. Oppdateringstidpunktet er en egen kolonne i Placetabellen(se DB-UML for applikasjonen i vedlegg A). Tidspunktet sendes til server, som sjekker med databasen om dette tidspunktet er nyere enn noen av oppdateringstidpunktene til stedene som ligger der, og returnerer

enten “true” eller “false”. Hvis applikasjonen får returnert true, ber den om å hente alle steder. Disse stedene erstatter stedene som allerede ligger i den lokale databasen, og returnerer deretter stedene som markører til applikasjonen.



Figur 6.6: Sekvensdiagram for henting av steder

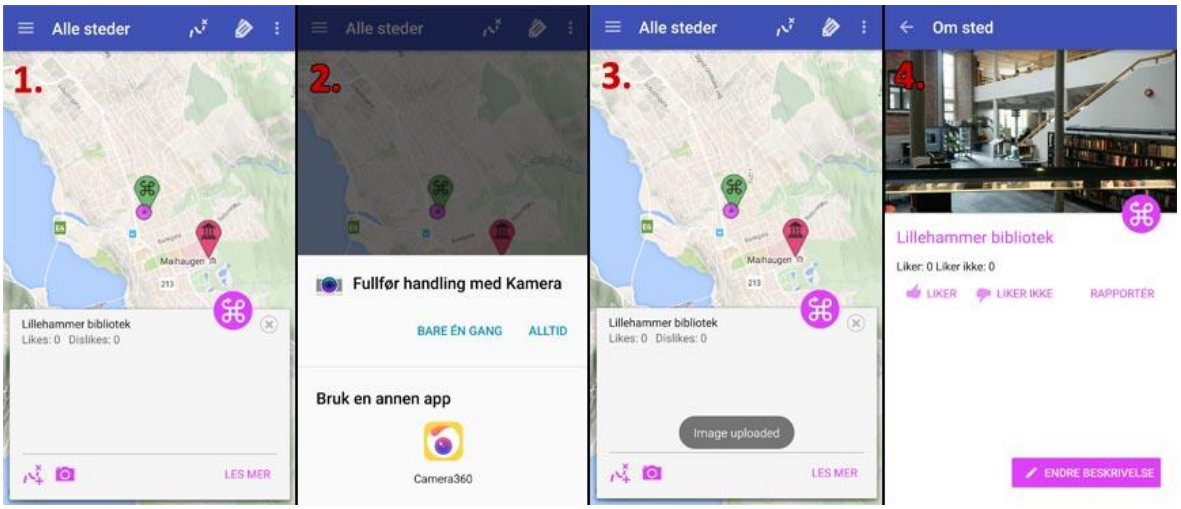
Under planleggingen av arkitekturen fant vi ut at det vil bli mange spørringer om å hente steder. Blant annet fordi kartet med markørene/stedene er en veldig sentral del av applikasjonen, og fordi vi ønsker å holde brukeren oppdatert med de nyeste stedene til enhver tid. I tillegg har mesteparten av de andre fuksjonalitetene sammenheng med kartet med markørene. Derfor lagrer vi markørene i en lokal database som brukes dersom det ikke er opprettet nye steder. Fordelen med dette er at det er raskere å hente fra lokal database, spesielt når vi tenker på at applikasjonen vil bli brukt mye utenfor de sentrale områdene hvor det er dårlig mobildekning. Vi begrenser også datamengden, som er beskrevet mer detaljert under punkt 6.13.

Vi kunne ha hentet stedene med en gang hvis det hadde vært nye steder, altså i første “response” i figuren. Slik som det er nå, gjør vi to requests for å hente stedene, mens med en slik arkitektur hadde vi bare trengt én. Men da hadde det ikke vært mulig å bare sjekke om det er nye steder uten å måtte laste ned alle. Det blir også et mer anvendelig API når

man kan spørre om det finnes nye steder og f.eks. vente med nedlastingen til enheten er koblet til trådløst nett.

6.6 Legge til bilder

For å gjøre applikasjonen mer interessant la vi opp til muligheten for å legge inn bilder fra steder. Prosessen for dette kan sees i figur 6.7. I bilde 1 i figur 6.7 har brukeren klikket på et sted på kartet og fått opp infoboks om stedet. Herfra kan brukeren klikke på fotoikonet nederst i infoboksen for å legge bilde til et sted.



Figur 6.7 : Legge til bilde på et sted.

I det brukeren klikker på ikonet starter vi med å opprette en URI for bildet for hvor det skal lagres på enheten. Ved å lagre bildet på brukerens enhet kan han/hun senere se bildene som er tatt gjennom Stuttreist. Siden vi ønsker å lagre bildet på brukerens enhet må vi først be om tillatelse til dette i AndroidManifest.xml. Dette gjøres med denne koden:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Brukeren blir varslet om at applikasjonen ønsker tilgang til enhetens filsystem når brukeren installerer applikasjonen.

Videre ønsker vi en egen katalog for bildene tatt i Stuttreist. Dette gjør vi ved ved å først opprette et File-objekt, for så å kalle metoden mkdir() som oppretter en katalog av filen, som vist under.

```
File folder = new File(Environment.getExternalStorageDirectory().getAbsolutePath() +
"/STUTTREIST/");

folder.mkdir();
```

Hvis katalogen allerede eksisterer skjer ingenting.

Deretter ønsker vi et unikt navn for alle bilder i filsystemet, vi bruker derfor tidspunktet bildet ble tatt i tillegg til ordet "STUTTREIST" for å vise hvilken applikasjon bildet tilhører. Dette gjøres i følgende kode:

```
String dateTime = new SimpleDateFormat("yyyyMMdd_HHmmss", Locale.GERMAN).format(new
Date());

imageName = "STUTTREIST_" + dateTime + ".jpg";

File imageFile = new File(Environment.getExternalStorageDirectory().getAbsolutePath()
                  + "/STUTTREIST/" + imageName);
```

Vi har valgt å ikke bygge et eget kamera i applikasjonen, så vi kaller et kamera-intent i som setter applikasjonen i pause og kjører kamera over applikasjonen. Brukeren kan selv velge en kamera-applikasjon, ut fra hvilke kameramuligheter brukeren har på sin enhet. Se bilde 2 figur 6.7. Det eneste Stuttreist-applikasjonen trenger å gjøre her er å kalle kamera, og si hvor bildet skal lagres. For at vi skal huske hvor bildet er lagret og hvilket sted bildet tilhører lagrer vi dette først i Shared Preferences før vi starter intenten. Dette ser vi i koden under:

```
fileUri = Uri.fromFile(imageFile);
SharedPreferences UriSP = getSharedPreferences(URI_PATH, 0);
UriSP.edit().putString("URI", fileUri.getPath()).apply();

SharedPreferences setId = getSharedPreferences(LAST_CLICKED_ID, 0);
setId.edit().putString("lastClickedId", markerToPlaceId.get(lastClickedMarker)+"").apply();

Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri);
startActivityForResult(intent, IMAGE_CAPTURE);
```

Hele funksjonen ligger i vedlegg I.

I det brukeren har tatt bilde og klikket lagre i valgt kamera-applikasjon returneres brukeren til Stuttreist-applikasjonen og MapsActivity. Nå vil funksjonen onActivityResult() motta svaret fra kameraet. Hvis kameraet responderer med at bildet ble tatt og lagret vil applikasjonen starte opplasting av bildet, som vist i koden under:

```
@Override
   protected void onActivityResult(int requestCode, int resultCode, Intent data) {
       super.onActivityResult(requestCode, resultCode, data);
       Log.i("Test onactivityresult", "Running");
       if (requestCode == IMAGE_CAPTURE) {
           if (resultCode == RESULT_OK) {
               Log.i("Test image-result", "Running");
               SharedPreferences getURI = getSharedPreferences(URI_PATH, 0);
               String spURI = getURI.getString("URI", null);
               Log.i("URI", spURI);
               new UploadImage().execute();
           }
       }
   }
```

Koden overfor avslutter med å opprette en ny instans av klassen UploadImage. Denne starter med å komprimere bildet, ettersom kameraet tar høyoppløselige bilder, mens vi ønsker både å holde en lav databruk i tillegg til at vi ønsker at applikasjonen skal fungere så raskt som mulig. Dette gjør vi med koden under:

```
Bitmap bmp = BitmapFactory.decodeFile(f.getPath());
ByteArrayOutputStream baos = new ByteArrayOutputStream();
bmp.compress(Bitmap.CompressFormat.JPEG, 35, baos);
```

Videre gjør vi om bildet til base64 for at vi skal kunne sende det som string:

```
String base24Image = Base64.encodeToString(bytearray, Base64.DEFAULT);
```

Hele UploadImage-klassen ligger i vedlegg J.

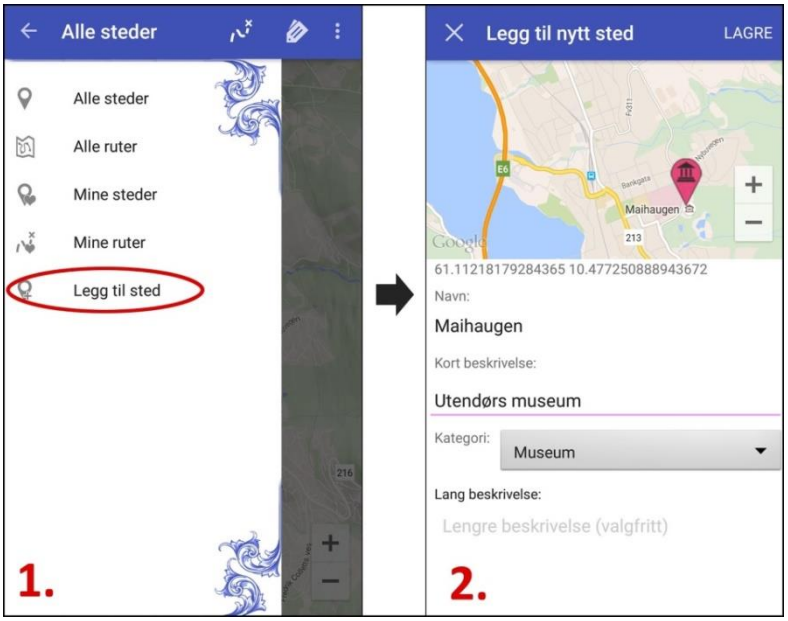
Til slutt sender vi alle påkrevde data til APIet, sammen med bildet. Hvis APIet returnerer "OK" vises en tilbakemeding på skjermen om at bildet ble lastet opp, eventuelt en feilmelding hvis det oppstod en feil. Dette skjer i bilde 3 i figur 6.7. Brukeren kan nå trykke "Les mer" på stedet i applikasjonen og maks 10 tilfeldige bilder blir hentet ut og bildet kan sees her. Vist i bilde 4 i figur 6.7.

Vi har valgt at bilder kun legges til ved at en bruker tar et bilde som deretter lastes opp, i motsetning til å laste opp et bilde som er lagret fra før. Dette er for å unngå copyright-problematikk som kan oppstå dersom brukere hadde fått muligheten til å laste opp andre bilder de har lagret på enheten fra før.

6.7 Legg til sted

Vi innså tidlig at det ville bli en stor utfordring for oss alene å skape et interessant innhold i applikasjonen, i forhold til antall steder. For å få et kvantitativt innhold er det essensielt at brukeren selv skal kunne legge inn steder og ruter.

Fra hovedmenyen (Figur 6.8, skjermbilde 1.) trykker brukeren på "Legg til sted" som fører brukeren videre til aktiviteten for å legge til sted. I denne aktiviteten (Figur 6.8, skjermbilde 2.) kan brukeren legge inn informasjon om stedet. I første omgang ble koordinatene satt til å være brukerens nåværende posisjon, men siden vi ønsket at brukeren skulle ha mulighet til å legge til steder uten å være på stedet fysisk, la vi inn et kart hvor man kan finne koordinatene til stedet man vil legge til.



Figur 6.8 : Legge til sted i applikasjonen.

Når brukeren klikker lagre kjøres funksjonen AddPlace() i AddPlaceActivity-klassen. Her foretas det først en sjekk på om de nødvendige feltene er fylt ut.

```
if(coordinatesText.getText().length() > 1 ||
  placeTitle.getText().length() > 1 ||
  shortDesc.getText().length() > 1) {
```

Deretter sjekkes det om bruker har klikket på kartet for å finne koordinater ved å sjekke om det er en markør på kartet.

```
if(marker != null) {
```

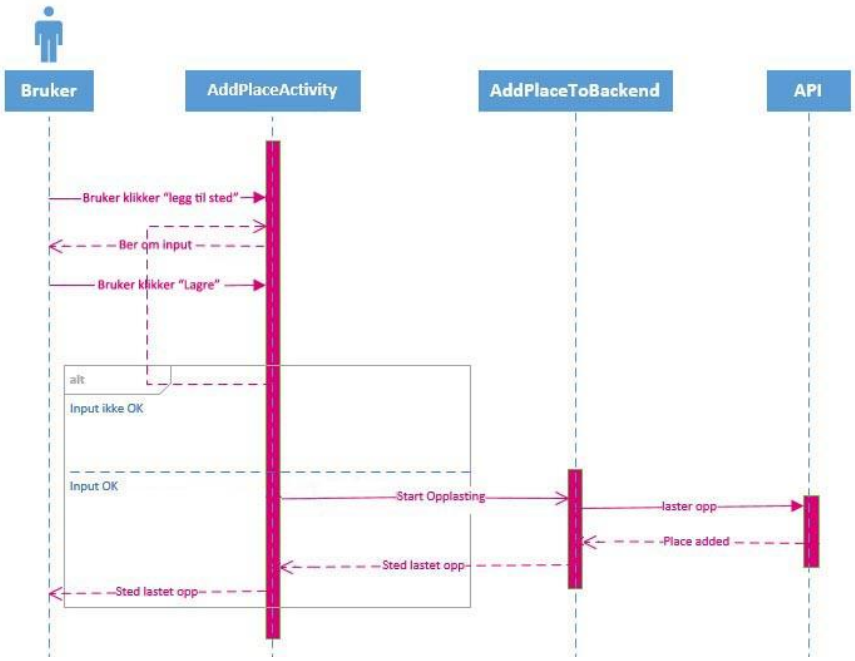
Etter at input er kontrollert legges input-dataene inn i objekter av typen `NameValuePair` som kobler verdien til variablene mot en variabel med samme navn som APIet tar imot som JSON.

```
NameValuePair nameParam = new BasicNameValuePair("name",
        placeTitle.getText().toString());
```

Når alle variablene er satt inn i hvert sitt NameValuePair legges disse inn i en liste som videre sendes til en asynkron funksjon.

```
List<NameValuePair> params = new ArrayList<>();
params.add(nameParam);
…
placeAdded = new AddPlaceToBackend().execute(params).get();
```

Den asynkrone funksjonen AddPlaceToBackend() tar imot parameterne og sender de videre til klassen JSONserviceHandler som formaterer dataene og laster opp det nye stedet. Når stedet er lastet opp oppdateres den lokale databasen med det nye stedet. Alle andre brukere som åpner applikasjonen etter at et nytt sted er blitt lagt til vil nå få sin lokale database oppdatert, og de nyeste stedene vil vises på kartet. Hele prosessen kan sees i figur 6.9.



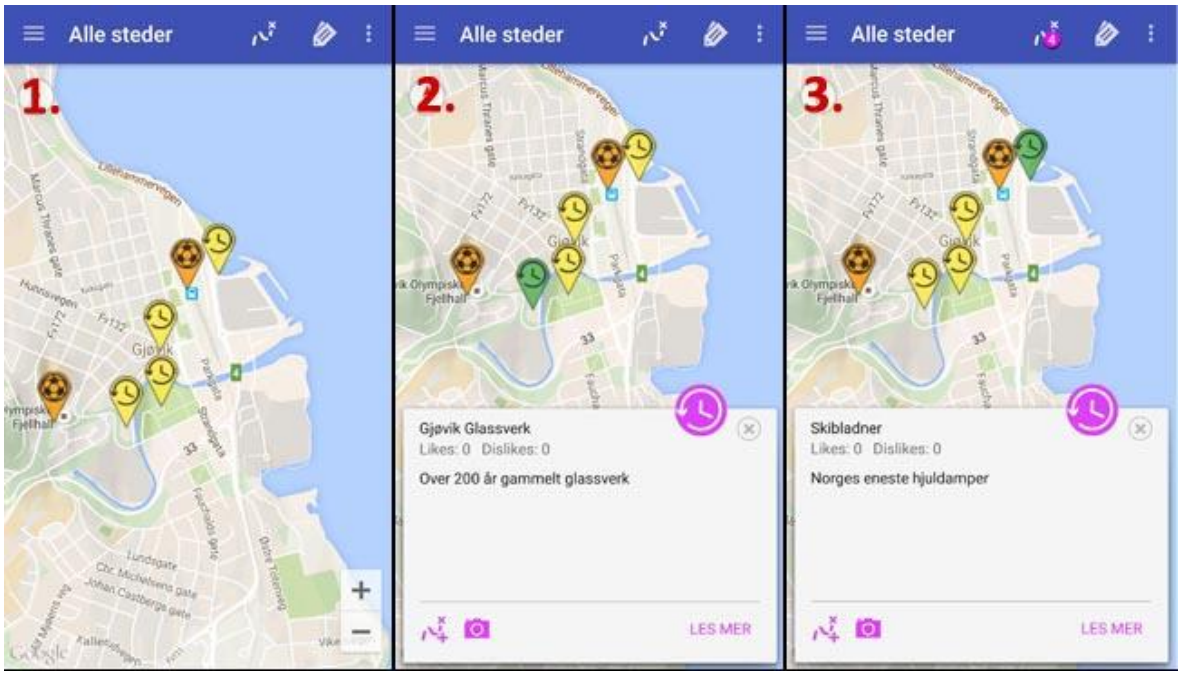
Figur 6.9 : Sekvensdiagram for å legge til sted.

6.8 Legg til rute

Når en bruker vil lage en ny rute, må hovedkartet vises, og alle steder må være lastet inn (figur 6.10, skjermbilde 1). Bruker må så opprette en liste over alle steder som skal være med i ruten. Dette gjøres ved at bruker trykker på et sted. Markøren bytter farge til grønn, og et lite infovindu vises nederst på skjermen (figur 6.10, skjermbilde 2).
Funksjonen for å vise infovinduet tar den siste valgte markør som parameter og returnerer informasjon tilhørende denne, som deretter vises i infovinduet.

```
public void showInfoWindow(Marker m){
  for(Place p : Places.getInstance().getList()){
      if((m.getPosition().latitude == p.getLat()) &&
              (m.getPosition().longitude == p.getLon())){
          mName.setText(p.getName());
          mDesc.setText(p.getShortDesc());
          mLikes.setText("Likes: " + Integer.toString(p.getLikes()));
          mDislikes.setText("Dislikes: " + Integer.toString(p.getDislikes()));
          setCategoryIcon(p);
      }
  }
  infoMenu.setVisibility(View.VISIBLE);
  mCategory.setVisibility(View.VISIBLE);
}
```

Når brukeren klikker på "legg til i rute"-knappen nederst til venstre i infovinduet, blir markøren lagt til i en markør-liste i MapsActivity. Rute-ikonet i toolbaren oppdateres med antall markører i denne listen (figur 6.10, skjermbilde 3).

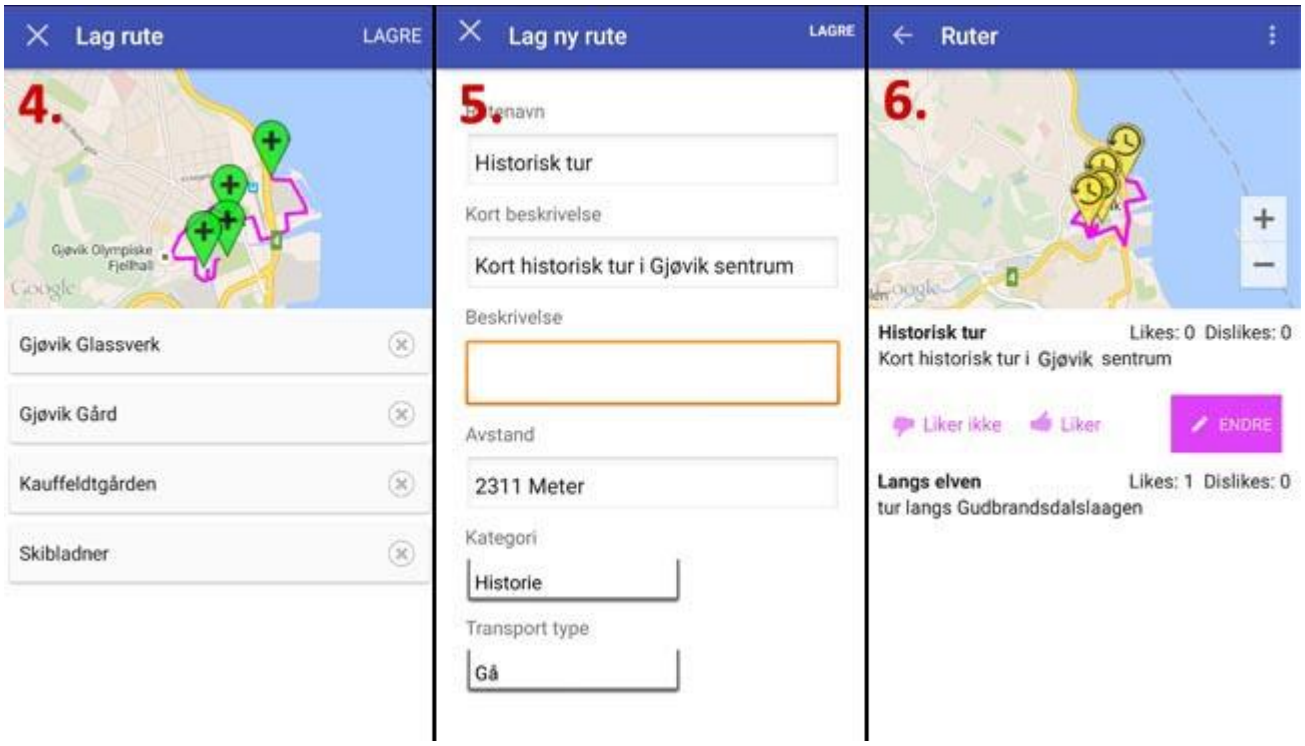


Figur 6.10 : Hvordan lage rute. Skjermbilde 1-3.

Når brukeren klikker på rute-ikonet i toolbaren, startes MakeRouteActivity (figur 6.11, skjermbilde 4). Kartet øverst på skjermen zoomer inn på brukerens nåværende posisjon, og viser markørene som ligger i markørlisten fra MapsActivity. En arrayliste med LatLng-elementer genereres ut fra markør-listen i MapsActivity. MakeRouteAdapter, se vedlegg E, bruker denne arraylisten og klassen MakeRouteContent, se vedlegg F, for å lage en liste som vises under kartet. Hver rad i listen inneholder navn, latlng og et ikon for å fjerne elementet fra listen. Hele MakeRouteActivity kan sees i vedlegg G.

Når bruker trykker på slette-ikonet på et element, vil elementet fjernes fra latlng-arraylisten OG fra markør-listen. Vi må forholde oss til to lister, siden vi trenger en markørliste for å få vist markørene på kartet, og en latlng liste for å få vist ruten mellom markørene. Listene oppdaters, kartet oppdateres, og ruten tegnes opp på nytt.

Når brukeren trykker på lagre-knappen i toolbaren, blir han/hun sendt videre til en fullskjermsdialogboks hvor mer informasjon om ruten skal fylles inn (figur 6.11, skjermbilde 5). Bruker fyller inn informasjon, og trykker på lagre-knappen i toolbaren igjen. Nå kan brukeren gå til "Mine ruter" og se på ruten som er blitt laget (figur 6.11, skjermbilde 6).



Figur 6.11 : Hvordan lage rute. Skjermbilde 4-6.

I aktiviteten MakeRoute hadde vi i utgangspunktet satt opp en listView med drag'n'drop-funksjonalitet, slik at brukeren kunne flytte elementer på listen opp eller ned, eller slette elementer. Dette fungerte bra, men så kom oppdagsgiver med krav om at vi skulle bruke material design og funksjonalitet tilknyttet disse spesifikasjonene. RecyclerView er en sentral del av materal design, og erstatter listView. I recylerView er flere operasjoner animert automatisk, for eksempel når man legger til eller fjerner elementer fra en liste. På grunn av tidsnød ble ikke drag'n'drop implementert i den nye recyclerViewen. Vi diskuterte litt frem og tilbake om vi skulle beholde den gamle listViewen med full funksjonalitet, eller om vi skulle nøye oss med recyclerView med kun mulighet for å fjerne elementer fra listen. Vi endte opp med å beholde recyclerView. Animeringen ved fjerning av elementer er penere, og mer i tråd med resten av brukergrensesnittet, og selve koden er mer ryddig og oversiktlig enn den første løsningen med listView. Vi ble enige om å komme tilbake til drag'n'drop-funksjonaliteten senere, dersom vi skulle få tid. Valget har ingen praktisk betydning for applikasjonens funksjonalitet, annet enn at det er litt mer stilig med drag'n'drop.

6.9 Rapportér sted

Et generelt problem med brukergenerert data er at man ikke kan garantere hvilken kvalitet det er på innholdet som blir lagt til i systemet. I vårt tilfelle kan dette f.eks. være at bruker legger inn sted på feil koordinater, feil i stedsinformasjonen, eller rett og slett legger inn tullesteder. For å redusere dette problemet har vi lagt inn muligheten for å rapportere steder fra applikasjonen. Med andre ord vil kvaliteten på innholdet bestemmes av brukerne - brukerne bestemmer hva som legges inn og hva som ikke holder god nok standard. Alle rapporteringer vil kunne hentes ut av oppdragsgiver i et enkelt web-grensesnitt, hvor det vil vises informasjon om hvilket sted det gjelder, tidspunkt for rapporteringen og hva rapporteringen gjelder. Dette fungerer som et enkelt ticket-system hvor oppdragsgiver kan gå gjennom rapporteringene og løse de. For å løse rapporteringer må oppdragsgiver gå direkte inn i databasen for å redigere eller fjerne problemet som er rapportert. Dette er noe vi ikke har regnet som et problem da oppdragsgiver har nødvendige kunnskaper for å gjøre dette.

6.10 Oppdatert data

For å sørge for at brukeren blir servert en oppdatert database hver gang applikasjonen kjøres, gjennomføres det en sjekk i oppstarten av applikasjonen. Det nyeste tidsstempelet som er lagret i den lokale databasen sendes til APIets getAllPlaces. Hvis det finnes nyere tidsstempel i databasen i backend vil applikasjonen fjerne den lokale databasen og laste ned de nyeste dataene. Tidsstempelet som blir sendt til APIet sammenliknes med Place-tabellens kolonne pUpdateTime i backend, som oppdateres hver gang endringer skjer på et sted. Hvis brukeren går inn for å lese utvidet informasjon om et sted hentes dataene som ikke blir hentet i fra getAllPlaces direkte fra backend. Dette inkluderer da den utvidede informasjonen om et sted, eventuelle bilder og lyd.

6.11 Material Design

Effekter som fungerer godt og ser bra ut på Lollipop-enheter, vil ikke nødvendigvis se like bra ut på pre-Lollipop-enheter. For å løse dette har vi satt opp stilark for forskjellige versjoner. For eksempel har ikke pre-Lollipop-enheter den animerte ripple-effekten som vises når

brukeren trykker på et element. Istedet får elementet samme farge som ripple, men ikke animert.

Et annet eksempel er krysset i toolbaren som vises i fullskjermsdialoger. På Lollipop-enhenter trenger vi ikke legge til noe ekstra i stilarket, men for pre-lollipop-enheter har vi lagt til dette, for at kryss-ikonet skal vises også på eldre android-versjoner.

```
<style name="CustomToolBarTheme" parent="Theme.AppCompat.NoActionBar">
<item name="homeAsUpIndicator">@mipmap/ic_menu_cancel</item>
</style>
```

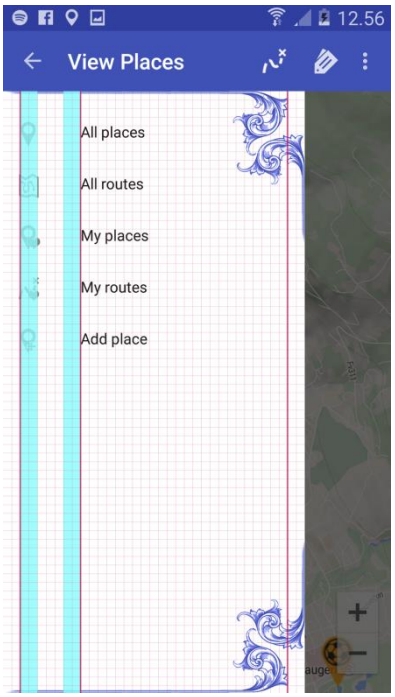
6.12 Navigation Drawer

For å kontrollere at vår UI forholder seg til spesifikasjonene satt i Navigation Drawer Pattern, har vi brukt dspec fra lucasr på GitHub [14]. Dette rutenettet er lagt over UIer vi vil validere, og vi har spesifisert rutenettet i forhold til innholdet. Filene ligger i res/raw.
Dette er spesifikasjonen brukt for å validere navigation drawer:

```
{
   "baselineGridVisible": true,
   "baselineGridCellSize": 8,
   "keylines": [
      { "offset": 16,
         "from": "LEFT" },
      { "offset": 72,
         "from": "LEFT" },
      { "offset": 16,
         "from": "RIGHT" }
   ],
   "spacings": [
      { "offset": 16,
         "size": 16,
         "from": "LEFT" },
      { "offset": 56,
         "size": 16,
         "from": "LEFT" },
      { "offset": 0,
         "size": 16,
         "from": "RIGHT" }
   ]
}
```

Cellene er 8 dp i størrelse. Vertikale røde streker viser hvor innholdet skal starte. Se figur 6.12.

I henhold til Navigation Drawer Pattern, begynner ikoner to ruter (= 16 dp) fra venstre kant, og teksten begynner ni ruter (= 72 dp) fra venstre kant.



Figur 6.12: Skjermdump av Stuttreist – navigation drawer med UI-validering.

6.13 Databruk

For å gjøre applikasjonen brukervennlig ute i skog og mark har vi på en del områder gjort tiltak for å holde datamengden som overføres mellom applikasjonen og APIet på et minimum. Dette er fordi det kan være dårlige og trege nettverk utenfor allfarvei, og brukeren vil dermed få en dårlig opplevelse når han/hun bruker funksjonalitet i applikasjonen som krever større dataoverføringer. I tillegg ønsker vi ikke å påføre brukerne store kostnader i databruk.

Vi har også sett på fordelingen av funksjonalitet, og hvordan de blir brukt i applikasjonen. Vi gjorde dette ved å se på user-storyene for applikasjonen og se hvordan de blir brukt, og i hvilke tilfeller brukerne vil utføre de forskjellige user-storyene. Ut fra dette kunne vi lage en oversikt over hvor vi bør prioritere å begrense databruken.

Vi kom frem til at brukeren vil ønske å finne steder ofte. Noe av det vi gjorde for å begrense datamengden her var å opprette en lokal database i applikasjonen som holder på stedene etter at de blir hentet fra APIet. Når brukerne dermed navigerer rundt i applikasjonen vil de ikke trenge å hente inn alle steder hver gang de returnerer til kartet med stedene. I tillegg valgte vi å begrense datamengden som er tilknyttet steder til det minimale. Bare det som

trengs for å få plottet alle steder på kartet blir sendt fra APIet. Videre, når brukeren klikker for å lese mer om et sted hentes den utfyllende informasjonen om stedet, slik som full beskrivelse, bilder og lyd som er større datamengder.

Vi konkluderte også med at user-storyen Finne ruter er en form for planleggingsaktivitet som brukerne benytter før de går ut på tur og finner steder. Vi tenker at de da befinner seg på mer sentrale steder med raskere og bedre nettverk. Derfor har vi ikke prioritert å kutte datamengden her.

Vi fant også ut at oppretting av sted er noe som brukerne gjør ute i skog og mark. Derfor har vi satt det valgfritt å legge til beskrivelse når man oppretter sted, siden denne kan være tung i forhold til kort beskrivelse og navn. Lyd er det også valgfritt å legge til på steder. Lydfiler er tunge filer, og vi ser for oss at brukere kan ønske å foreta operasjoner som krever mer databruk når de er tilkoblet trådløst nett, eller er på et sted med kjent god dekning. Vi forestiller oss at brukerne kan fylle inn utfyllende informasjon som lang beskrivelse og lyd etter at de har vært ute og opplevd stedet.

Det siste vi valgte å prioritere i forhold til databruk var funksjonalitet for å legge til bilde. Opprinnelig var planen at brukerne skulle kunne legge til navn og beskrivelse til bilde, men vi valgte å ikke implementere dette. Dermed vil dataene som overføres, både ved opplasting og nedlasting av bilder, begrenses noe. I forhold til størrelsen på et bilde blir ikke dette mye, men vi tenker oss at opplasting og deling av bilder vil være en mye brukt funksjon og datamengden som minskes vil da bli stor sammenlagt når en bruker laster opp mange bilder og går inn for å lese om mange steder.

Et tiltak vi også implementerte i sammenheng med bilder for å begrense databruken er å komprimere bildet som blir returnert fra kamera-applikasjonen. Etter å ha testet bilde-opplasting oppdaget vi at bildestørrelsen var på noen MB uten komprimering, men etter en komprimering på 20% av orginalen fikk vi bilder i størrelser på ca 300-500KB med akseptabel bildekvalitet.

7. Implementering av backend API

7.1 Tabell over API funksjonalitet

I tabell 7.1 ser vi en oversikt over script og funksjonalitet i APIet. Dette er en annen reprensentasjon av vedlegg B - API UML. Her ser vi hvilken URL som kalles for å nå scriptene, hva filnavnene til scriptene er, hva slags funksjon de forskjellige scriptene har, hvilke parametre de tar, og hva de returnerer.

URL storsveen.rocks/ guide/api/..	**Beskrivelse**	**Mottar**	**Returnerer**
/getAllPlaces	Henter alle steder, eller sjekker om det er opprettet nye steder etter et gitt tidspunkt.	Påkrevet: - Ingen Valgfri: - Dato	JSON-array med steder ELLER true/false
/addImage	Laster bilde + data til bilde inn i databasen.	Påkrevet: - Sted-ID - Nøkkel - Navn - Beskrivelse - Bilde	Message: Ok ELLER feilmelding
/addRoute	Legger “rute” inn i databasen.	Påkrevet: - Array med steder - Kort beskrivelse - Navn - Kategori - Nøkkel - Transport-kategori - Beskrivelse - Avstand	Message: Ok ELLER feilmelding
/addUser	Oppretter ny bruker.	Påkrevet: -Brukernavn -Epost -Passord -Nøkkel	Message: Ok ELLER feilmelding
/changeDescription	Endre en eksisterende beskrivelse.	Påkrevet: -Ny	Message: Ok ELLER

		beskrivelse -Ny kort-beskrivelse -Nøkkel -Bruker-ID -Sted-/rute-ID	PDO feilmelding
/changeUserDetails	Endre info om en bruker: passord/brukernavn.	Påkrevet: -Nytt brukernavn -Nytt passord -Epost -Nøkkel	Message: Ok ELLER feilmelding
/dislike	Legger til en "dislike" på et sted eller en rute.	Påkrevet: -Rute-/sted-ID -Bruker-ID -Nøkkel	Message: Ok ELLER feilmelding
/getCategories	Henter og returnerer eksisterende kategorier.	Ingenting	JSON-array med kategorier
/getImages	Henter ti tilfeldige bilder knyttet til et sted.	Påkrevet: -Sted-ID	JSON-array med bilder
/getLongDescription	Henter (lang) beskrivelse til en rute eller et sted.	Påkrevet: -Rute-/sted-ID	JSON-objekt med beskrivelse
/getPlaces	Henter steder innenfor et område ELLER Alle stedene til en rute.	Påkrevet: ENTEN -Lengdegrad -Breddegrad Valgfri: -Radius ELLER -Rute-ID	JSON-array med steder
/getRoutes	Henter ruter innenfor et område ELLER Alle ruter oppretter av en gitt bruker.	Påkrevet: ENTEN -Lengegrad -Breddegrad -Type Valgfri: -Radius	JSON-array med ruter

		ELLER -Bruker-ID	
/getTransportCategories	Henter alle transport-kategorier fra databasen.	Ingenting	JSON-array med kategorier
/login	Sjekker om et gitt par med epost og passord stemmer overens.	Påkrevet: -Epost -Passord	JSON-array med epost, navn og ID
/logout	Logger ut bruker(Bare for web).	Ingenting	Ingen
/search	Søker etter ruter ved om deres navn eller kort-beskrivelse inneholder en gitt string.	Påkrevet: Søke-streng	JSON-array med ruter

Tabell 7.1 : Tabell over API-funksjonalitet.

Under utviklingen brukte vi denne tabellen for å holde oversikt over hvilken funksjonalitet som var ferdig utviklet i backend og ikke. I tillegg var det enkelt å holde oversikt over hvilke data hvilke script skulle ta i mot, og hva de returnerte, uten å lese gjennom alle filer for å finne ut hva de gjør.

7.2 API i detalj

7.2.1 .htaccess

```
DirectoryIndex requestHandler.php

RewriteEngine on
RewriteBase /guide/
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ requestHandler.php?/$1 [L,QSA]

RewriteCond %{REQUEST_FILENAME} \.php$
RewriteRule ^(.*)$ requestHandler.php?/$1 [L,QSA]
```

Ovenfor er et utsnitt fra .htaccess-filen vi bruker i backend. .htaccess filer brukes på web-servere for å konfigurere web-serveren. .htaccess-filene brukes ofte til å redirigere forespørsler til apache-serveren, som er slik vi også bruker den i vår løsning. Vi har plassert den under .../guide/ i vårt fil-tre for å styre strømmen av forespørsler til .../guide/. Altså hvis det kommer en forespørsel til .../guide/api/getPlaces, styres forespørselen dit den skal. Hvis

det kommer en forespørsel til .../example/example.php, blir ikke denne redirigert. .htaccess redirigerer bare bestemte forespørsler til et bestemt script. I vårt tilfelle vil .htaccess redirigere forespørsler til requestHandler.php (beskrevet nærmere i punkt 7.2.2), som igjen henter ønsket fil til brukeren. Altså er det ikke .htaccess som inneholder logikken til hvilken fil som skal hentes, dette tar requestHandler seg av.

I vår løsning videresender vi forespørsler som ender med .php eller er uten filending.
For eksempel hvis brukeren sender en forespørsel om:
www.storsveen.rocks/guide/**api**/getAllPlaces, vil .htaccess sende denne forespørselen til requestHandler.php, som laster getAllPlaces.php til brukeren. Det finnes ingen "api"-katalog, men alle "api-filene" ligger under .../guide/api/. Hvis brukeren ber om .../guide/getAllPlaces.php, som er den direkte stien til getAllPlaces vil han/hun få en feilmelding, ettersom vi ønsker at brukeren skal bruke formatet vi har satt opp.
Men, hvis brukeren ber om .../guide/example.**html** vil brukeren komme dit, ettersom vi bare styrer forespørsler som ender med .php eller ingen ending. Med to ekstra kodelinjer i .htaccess kunne vi også har styrt .html-forespørselene. Siden vi bare bruker .php, trengs ikke dette.

Hvis vi ser på kode-utsnittet overfor er det `RewriteEngine on` som lar oss redirigere forespørselene.
`RewriteBase /guide/` forteller hvilke forespørsler vi skal håndtere fra.
`RewriteCond` sier hva slags "condition" eller betingelse som skal til for at vi skal redirigere disse. F.eks. hvis de slutter på .php. `RewriteRule ^(.*)$` sier hvor forespørselene skal sendes. I vår løsning sender vi alle til requestHandler, men man kunne f.eks hatt én requestHandler for .php-forespørsler og en annen for .html-forespørsler.

Vi trengte litt hjelp til .htaccess-filen, så vi ba om hjelp på stackoverflow[17]. Se gjerne i linken om hvordan vi gikk frem for å utforme den.

7.2.2 RequestHandler

```
class RequestHandler {
   public $path;

   function __construct() {
       // Getting request:
       reset($_GET);
       next($_GET);
       $this->path = strtolower(key($_GET));
```

RequestHandler håndterer forespørselene fra en bruker og laster filer basert på forespørselene. Ovenfor ser vi et utsnitt av constructor til klassen RequestHandler i requestHandler.php. Resten av klassen ligger i vedlegg H. Brukerens forespørsel legger seg i GET-variablen i skuff nummer 1 (andre skuff i arrayen). $_GET kan f.eks. se slik ut:

```
Array (
   [/requestHandler_php] =>
   [/api/getAllPlaces] =>
)
```

Eventuelle andre GET-variabler legger seg nedenfor slik: [key1] => value1… osv.
Vi starter med å resette arrayen, hopper til andre skuff, og henter nøkkelen. Vi gjør også alt om til små bokstaver slik at det skal være mer brukervennlig.

```
switch($this->path) {
    case "/api/getallplaces/":
    case "/api/getallplaces":
        include("getAllPlaces.php");
        break;
    case "/api/getimages":
    case"/api/getimages/":
        include("getImages.php");
        break;
```

Ovenfor er et utsnitt fra RequestHandlers constructor. Vi lager en switch på path, som vi tidligere har satt til å være URLen fra brukeren. Hvis f.eks. brukeren ber om .../api/getAllPlaces vil dette matche første case i switchen og vi laster getAllPlaces.php. Merk at navngivningen på URL og fil ikke trenger å være lik. Vi kan f.eks. endre getAllPlaces til fetchAllPlaces.php og legge denne som include i RequestHandler og vi trenger dermed ikke gjøre noen endringer i Android-applikasjonen.

Som default case i switchen skriver vi ut Error 404: Page not found. Merk at vi i praksis laster filene vi trenger inn i RequestHandler og trenger dermed ikke sende eventuelle POST-variable til filene. De vil filene ha tilgang til gjennom RequestHandler.

7.2.3 Hvordan APIet fungerer

APIer bruker, som nevnt tidligere, HTTP POST. Hver enkelt metode i APIet tar et sett med data som blir liggende i $_POST-arrayen. Men det varierer hvilke data som hver enkelt metode tar avhengig av funksjonen til metoden. Oversikt over hvilken funksjon de forskjellige metodene har, hvilke data de mottar og returnerer, ligger dokumentert i hver enkelt fil, i tabellen 7.1 og i UML med filoversikt for backend, vedlegg B.

7.2.4 Innsetting av data

Innsetting av data gjør vi ved at en gitt del av APIet mottar post-variable. Så sjekker vi om de påkrevde data er satt. Slik som nedenfor hvor vi ser et eksempel fra changeDescription.php. Valgfrie parametre blir satt dersom de ikke er medsendt. Parametre blir sjekket ulikt avhengig av type. F.eks. blir heltall sjekket at de er heltall, e-post oppbygd som e-post osv. Det blir også sjekket om beskrivelser og korte beskrivelser inneholder ikke tillatte ord som er satt i en forhåndsdefinert array. Dette er for å prøve å holde innholdet i løsningen rent og relevant.

```
if(!(isset($_POST['description']))||strlen($_POST['description'])<1) {
   die (json_encode(array('message' => 'Description not set')));
} else if(!(isset($_POST['shortdescription']))||strlen($_POST['shortdescription'])<1) {
   die (json_encode (array ('message'=>'Short-description not set')));
```

Scriptet dør og sender tilbake feilmelding hvis ikke påkrevet data er satt.

```
'INSERT INTO '.$DBPlaceDescription->tableName.' ('.$DBPlaceDescription->descriptionId.',
'.$DBPlaceDescription->placeId.') VALUES (?, ?);'
```

Videre prøver vi å sette inn dataene, med en SQL-setning, som den over som blir brukt i changeDescription.php. Vi har lagret tabellnavn og kolonnenavn på alle tabellene i databasen i en egen fil, som vi henter når vi lager SQL-setninger. På denne måten kan vi fritt endre navn på kolonner i databasen for så å bare endre det i denne ene filen, dermed vil det bli endret alle steder hvor det blir brukt. Istedet for å måtte gå gjennom alle script for å se hvor denne kolonnen eller denne tabellen blir brukt for å endre det. Dette er også sikrere

siden man ofte glemmer å endre til nytt kolonnenavn overalt, og da vil løsningen slutte og fungere.

SQLen ovenfor blir til denne strengen slik som kolonnenavnene er nå:

```
'INSERT INTO PlaceDescription (dId, pId) VALUES (?, ?);'
```

addImage.php og changeDescription.php skal sette inn bilde og lyd. Disse er opprinnelig i binært format. Siden det er vanskelig å få til sending av binære data i Android har vi valgt å base64-kode filene i applikasjonen før vi sender dem. APIet tar dermed imot stenger istedet for filer. I scriptet gjør vi dermed:

```
$content = base64_decode($_POST['content']);
```

som er en innebygd php-funksjon som base64-dekoder strenger. Dermed setter vi nå inn `$content`, som er den binære filen.

7.2.5 Returnering av data

Returnering av data avhenger av om vi skal sette inn data eller hente ut data. Vi ønsker gjerne å få en respons på at dataene ble satt inn etter en innsetting. Derfor returnerer vi JSON-objektet "message => OK" hvis dataene ble satt inn. Mens hvis det oppstår et problem under innsettingen, dette kan være f.eks. at en bruker har "liket" et sted eller en rute fra før, så returnerer vi JSON-objektet: message => feilmelding.
Vi bruker alltid "message" i responsen på innsettinger siden det da er enkelt å hente ut meldingen. Det hender også at vi ønsker å få returnert en ID eller lignende. Denne blir dermed liggende under "response". Sammenlagt får vi dermed JSON: "message": "OK", "response": "id1", "id2"...

Når vi skal hente ut data returnerer vi dataene i JSON-arrayer hvor hvert objekt er f.eks. en rute, sted eller et bilde. Når vi henter ut data fra databasen med fetchAll får vi en array med data. Hvis vi hadde direkte JSON-kodet dette ville vi fått et JSON-objekt med arrayer, vi ønsker det motsatt, altså en array med objekter. Derfor bruker vi metoden under for å gjøre om dataene til JSON-array.

```
$result = $sth->fetchAll(PDO::FETCH_ASSOC);

$objArray = [];
foreach($result as $place) {
```

```
    $objArray[] = ((object)$place);
}
die (json_encode($objArray));
```

Først henter vi dataene, deretter oppretter vi en array, og så henter vi hver "rad" fra resultatet, gjør dette om til et objekt, og legger dette til i arrayen. Til slutt JSON-enkoder vi objekt-arrayen og returnerer den.

7.2.6 Hente steder/punkter fra databasen

For å finne alle punktene som er i nærheten av deg for øyeblikket, bruker vi scriptet getPlaces.php. Dette scriptet tar en breddegrad(lat) og en lengdegrad(lon), og mulighet for å ta en radius. Hvis ikke radius er satt blir den satt standard til 10km (10000 meter). Scriptet er bygget videre på en metode for å hente punkt innenfor område[15].

For å finne punktene innenfor et område regner vi ut høyeste og laveste mulig breddegrad og lengdegrad. For å finne høyeste mulig breddegrad starter vi med å dele radius for jordens radius(6371000m/6371km):
Eks. **10000/6371000**
Dette er i radianer, så vi gjør det om til grader:
10000/6371000 *** 180/pi**
Så legger vi til den medsendte breddegraden:
60.789927 + *(10000/6371000 * 180/pi)*

For minste mulige breddegrad blir det samme formel, bare at vi trekker fra i stedet:
60.789927 **-** *(10000/6371000 * 180/pi)*

For å finne lengdegrad tar vi utgangspunkt i samme formel, men deler på cosinus av den medsendte breddegraden omgjort til radianer:
10.696661 ± *((10000/6371000 * 180/pi)* **/ cos(60.789927*pi/180)**

For å hente alle punktene innenfor området vårt kjører vi en spørring til databasen om å hente ut alle punkter som har breddegrad mellom minste breddegrad og høyeste breddegrad OG lengegrad mellom minste lengdegrad og høyeste lengdegrad:

```
SELECT pId, pName, pLat, pLong, pAlt, pLikes, pDislikes
FROM Place
WHERE pLat > laveste_breddegrad AND pLat < høyeste_breddegrad
AND pLong > laveste_lengdegrad AND pLong < høyeste_lengdegrad
```

Videre må vi legge dette uttykket inn i et annet uttrykk som tar punktene i resultatet fra dette uttrykket og regner ut avstanden fra den medsendte posisjonen (vår breddegrad og lengdegrad) til alle punktene i resultatet og sjekker at avstanden er mindre enn den medsendte radiusen.

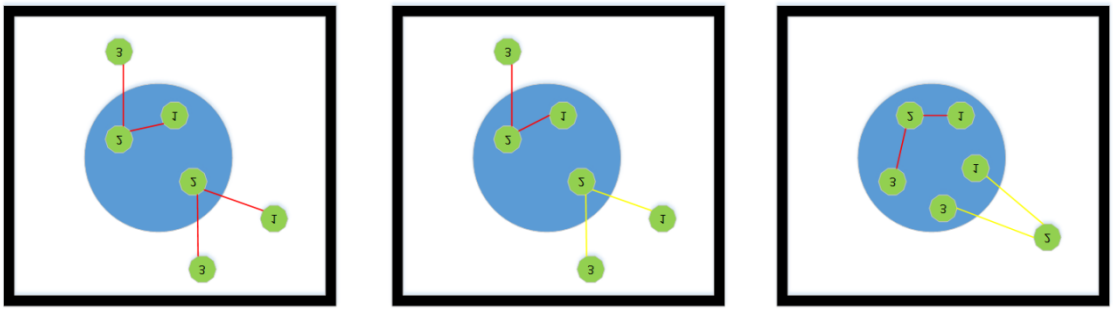
Den fullstendige spørringen blir da:

```
SELECT pId, pName, pLat, pLong, pAlt, pLikes, pDislikes,
                    :EARTH_RADIUS * acos(sin(:latitude*PI()/180)*sin(pLat*PI()/180) +
cos(:latitude*PI()/180)*cos(pLat*PI()/180)*cos(pLong*PI()/180-:longitude*PI()/180))
AS                        Distance
                FROM (
                    SELECT pId, pName, pLat, pLong, pAlt, pLikes, pDislikes
                    FROM Place
                  WHERE pLat > :latitude-(:radius/:EARTH_RADIUS*180/PI()) AND pLat <
                  :latitude+(:radius/:EARTH_RADIUS*180/PI())
                  AND pLong > :longitude-
                  ((:radius/:EARTH_RADIUS*180/PI())/cos(:latitude*PI()/180)) AND pLong <
                  :longitude+((:radius/:EARTH_RADIUS*180/PI())/cos(:latitude
                        *PI()/180))
                ) AS Points
                HAVING Distance < :radius
                ORDER BY Distance
```

7.2.7 Finne ruter

Når vi skal finne ruter tenker vi at brukeren ønsker å se de rutene som er i nærheten. Det er forskjellige måter å finne ruter i nærheten på, og vi lar brukeren velge mellom disse tre alternativene:

1. Hente ruter hvor minst ett punkt er innenfor gitt radius, men hvilket som helst punkt i ruten.
2. Hente ruter hvor hvertfall første punkt er innenfor gitt radius.
3. Hente ruter hvor alle punktene i ruten er innenfor gitt radius.



Figur 7.2: Metode 1,2,3. Rød linje er rutene som blir hentet.

Vi tar utgangspunkt i samme metode som når vi finner steder, og gjør det samme med ruter. For metode 2 henter vi ruter med "number" 1 innenfor radiusen. Mens for metode 3 må alle rutens punkter være innenfor radiusen, dette er litt komplisert ettersom vi først har en SQL som finner ruter med ingen punkter innenfor området, så left-joiner vi denne med alle ruter og henter de som har NULL og får dermed rutene med alle punktene innenfor.

I applikasjonen kan brukeren også sette radiusen/avstanden fra sitt punkt, som han/hun ønsker å hente ruter fra. Den er satt standard slik at det skal være enkelt for brukeren å finne ruter, men hvis det er få eller ingen ruter i området kan brukeren øke radiusen selv til ønsket nivå og finne ruter.

8. Avslutning

I forhold til resultatmålene som ble satt i starten av prosjektet, føler vi at vi har nådd alle. Vi har utviklet en applikasjon hvor brukerne selv genererer innholdet. Brukerne kan både finne severdigheter, lage egne severdigheter, finne ruter og lage egne ruter. Applikasjonen kan brukes over hele landet, og fungerer fra Android 4.0, inkludert ny funksjonalitet i Android 5.0. Ut over dette hadde vi en liste med ekstra funksjonalitet som vi også gjerne ville ha med. Både det å la brukerne legge til egne bilder og egen lyd er med i applikasjonen. Det at brukerne har mulighet for å høre lydspor - at andre forteller om en severdighet eller et sted, er det som gjør at vår applikasjon skiller seg ut, og gjør at Stuttreist virkelig er din lokale reiseguide.

I forhold til læringsmål har vi fått veldig mye mer kompetanse innenfor mobilutvikling og god kjennskap til material design. Gjennomføringen av prosjektet har gått greit, men det har vært vanskelig å tidsestimere de forskjellige oppgavene. Det har gått med mye mer tid til å rette bugs og "småting" i applikasjonen enn det vi hadde forventet. I tillegg måtte vi bruke en god del ekstra tid på å legge om GUIen til material design da oppdragsgiver kom med krav om dette.

Arbeidsfordelingen har vært veldig grei. I starten ble det en slags naturlig fordeling av arbeidsområder, hvor alle hadde "sine" ting å jobbe med. Mot slutten har vi jobbet mer sammen, da områdene begynte og overlappe en del.

I forhold til oppdragsgiver har vi fått flere erfaringer. For det første opplevde vi at den opprinnelige oppdragsgiveren trakk seg tidlig i januar, da vi tok kontakt for å avtale møte om signering av kontrakt. Vi hadde da allerede hatt et møte før jul, hvor vi diskuterte litt hva som skulle gjøres og hvordan. Vi fikk samlet oss raskt og plukket ut en ny oppgave og dermed ny oppdragsgiver. Siden AndMark også er studenter på HiG, har møtene vi har hatt med de vært uformelle og relativt korte. Vi har fått stor frihet til å utvikle applikasjonen etter eget ønske. De har hatt noen få krav, som REST og material design. Disse kravene har vi etterfulgt. Det at kravene var i hovedsak teknologibaserte fremfor funksjonsbaserte er noe vi tenker kan være uvanlig forhold til andre oppdragsgivere. Anders og Markus i AndMark er

veldig kunnskapsrike på teknologiområdene vi har vært innom, så de har på en måte også fungert litt som veiledere.

Vi har ikke laget noen fullverdig nettløsning for applikasjonen. Oppdragsgiver ville ha en kort informasjonsside på internett, med eventuelt utvidet funksjonalitet. Vi har laget en løsning som oppdragsgiver kan benytte seg av, men den er ikke tilpasset vanlige brukere, og ligger ikke på et eget domene (som f.eks. stuttreist.no, som forøvrig er ledig).

I forhold til oppgaven burde vi ha avsluttet poleringen av appen mye tidligere, og brukt tiden på rapporten, men det var vanskelig å legge appen fra seg før vi kunne si oss fornøyde med funksjonaliteten, og hadde rettet de verste bugsene.

Applikasjonen kan videreutvikles på flere områder. For det første ligger mye funksjonalitet klart i forhold til å opprette en fullverdig nettløsning tilhørende applikasjonen. Script for å finne rute, opprette rute, finne sted, legge til sted, laste opp bilder og løse rapporteringer er klare og kan benyttes som administratorløsninger. Med litt videreutikling kan disse også benyttes av applikasjonens brukere via en webløsning. Det kan være hensiktsmessig å legge til funksjonalitet for at bruker får en notification på telefonen når en severdighet er i nærheten, det kan også være aktuelt å legge til flere lydspor for et sted, eller også videosnutter. For deling i sosiale nettverk kan det være aktuelt å legge til en funksjon for å "sjekke inn" på et sted, sammen med et bilde fra stedet. Navn på bilde og bildetekst er tatt henyn til og lagt inn i databasen, men er ikke implementert i applikasjonen. Det kan også være aktuelt å lage å lage applikasjonen på andre plattformer som iOS og Windows Phone ettersom dagens løsning ikke dekker alle brukere. I tillegg kunne man laget en administratorløsning hvor man f.eks. stenger brukere ute fra deler av løsningen pga. dårlig oppførsel, mulighet for å ta backup av data, og generell vedlikehold av løsningen.

Når det kommer til publisering av applikasjonen er dette noe oppdragsgiver tar seg av om ønskelig. Vi har laget en applikasjon som er klar til publisering i release-versjon. Hva som skjer videre med Stuttreist er opp til AndMark, enten det er publisering som en stand-alone-applikasjon i Google Play eller implementering av funksjonalitet i dagens Stolpejakten. For

spesielt interesserte kan applikasjonen lastes ned fra www.storsveen.rocks/guide/stuttreist.apk.

Gruppa har arbeidet godt sammen, og det har vært lite uenighet om hvordan vi skulle legge opp løpet. Trine ble tildelt rollen som prosjektleder, og Pål ble tildelt rollen som webansvarlig. Arbeidsfordelingen har vært jevn og uformell. Vi har (heldigvis) hver våre interesseområder, og oppgavene ble fordelt etter hva vi synes selv var mest interessant å jobbe med. Dennis og Pål begynte med backendløsningen, Trine begynte med applikasjonen - kart, location og hente inn alle steder. Dennis fortsatte med backend og databasestruktur, mens Pål begynte med "legg til sted" i applikasjonen. På dette tidspunkt kom kravet om material design, så Trine gjorde nesten ferdig "lag rute", og begynte og sette seg inn i material design. Her kom Dennis inn i applikasjons-utviklingen og tok over "lag rute", mens Trine begynte å sette opp alle xml-filene på nytt i henhold til material design-spesifikasjonene. Trine og Pål har laget ikoner og markører. Dennis har tatt seg av det meste av funksjonaliteten som hører til "lag rute" og hele delen med "legg til bilde". Gruppesamarbeidet har gått merkelig smertefritt, vi var forbredt på at det kom til å bli litt krangling og uenigheter underveis, men med unntak av litt søskenkjekling har vi unngått de store kranglene. Her har Dennis, fornuftig nok, forholdt seg nøytral. Mot slutten av prosjektperioden har vi jobbet mer sammen, med bugfiksing, Lint og selve rapporten.

Med unntak av faste tirsdagsmøter, og veiledermøter annenhver onsdag har vi hatt stor frihet i forhold til hvordan vi har valgt å jobbe med prosjektet. ScrumChatten og Facebook-gruppen har vært gode møteplasser for oppdateringer og diskusjoner. Ved at vi har oppdatert hverandre på disse arenaene har vi også hatt friheten til å jobbe med prosjektet på de tidspunktene det passer oss best (i forhold til jobb og andre emner i studiet). Det å bruke Scrum som utviklingsmetodikk har bidratt til at vi har hatt god oversikt over hvem som jobber med hva, hva som er ferdig og hva som gjenstår. Siden vi har valgt å jobbe sammen i så liten grad som vi har gjort, har vi også hatt mye fokus på at alle til enhver tid skal vite hva de andre holder på med. Dette ble gjort via ScrumChatten, og via kontinuerlige oppdateringer i Trello. På tirsdagsmøtene startet vi alltid med hva vi har gjort siden sist, og fordelte i grove trekk oppgaver. Ellers har vi plukket oppgaver fra Trello, etter "first-come-first-serve"-metoden.

For oss har prosjekt som arbeidsform fungert totalt sett bra. Vi har hatt stor frihet både til å forme oppgaven slik vi ønsket selv og kunne velge de delene av oppgaven som passet oss godt. Vi har også utfylt hverandre bra med at vi har ønsket å jobbe med forskjellige deler av løsningen og dermed, sammen, skapt et totaltsett godt resultat. I tillegg har vi på de tyngre oppgavene samarbeidet godt. Ved at vi har ønsket å jobbe med forskjellige deler av løsningen samtidig som vi har støttet opp hverandre på alle deler av løsningen har vi bygd opp hverandres kompetanse.

Vi kan konkludere med at vi er godt fornøyd med resultatet av applikasjonen. Vi burde ha kuttet noe funksjonalitet (som lyd og/eller bilde), til fordel for å perfeksjonere applikasjonen og skrive rapporten, men vi følte at det var viktige elementer å ha med for å gi applikasjonen "det lille ekstra". Vi har lært mye mer om androidutvikling, vi har hatt det gøy når vi har fått til noe stilig, og hatt det litt kjipt når vi har hatt problemer med å rette feil. Gruppa har samarbeidet godt og lagt ned en god innsats i prosjektet.

Litteraturliste

1. Android.Plattform Versions [Internett]. 2015 [besøkt 6. mai 2015]. Tilgjengelig fra: https://developer.android.com/about/dashboards/index.html

2. Gao, Lei, Chunhong Zhang, and Li Sun. "RESTful web of things API in sharing sensor data." Internet Technology and Applications (iTAP), 2011 International Conference on. IEEE, 2011.

3. Tsai, Chin-Liang, et al. "Transmission reduction between mobile phone applications and RESTful APIs." Proceedings of the 2011 ACM Symposium on Applied Computing. ACM, 2011.

4. Rackspace.com Knowledge centre, MySQL Engines - MyISAM vs Innodb | Knowledge Center **|** Rackspace Hosting [Internett] 2015 [besøkt 10. mai 2015]. Tilgjengelig fra: http://www.rackspace.com/knowledge_center/article/mysql-engines-myisam-vs-innodb

5. Google design guidelines. Color - Style - Google design guidelines [Internett]. 2015 [besøkt 16. april 2015]. Tilgjengelig fra: http://static.googleusercontent.com/design/spec/style/color.html

6. Google design guidelines. Typography - Style - Google design guidelines [Internett]. 2015 [besøkt 16. april 2015]. Tilgjengelig fra: http://www.google.com/design/spec/style/typography.html

7. Google design guidelines. Icons - Style - Google design guidelines [Internett]. 2015 [besøkt 25. april 2015]. Tilgjengelig fra: http://static.googleusercontent.com/design/spec/style/icons.html#icons-system-icons

8. Developer.android.com. Iconography | Android Developers [Internett]. 2015 [besøkt 25. april 2015]. Tilgjengelig fra: http://developer.android.com/design/style/iconography.html

9. Google design guidelines. Dialogs - Components - Google design guidelines [Internett]. 2015 [besøkt 25. april 2015]. Tilgjengelig fra: http://static.googleusercontent.com/design/spec/components/dialogs.html

10. Google design guidelines. Buttons - Components - Google design guidelines [Internett]. 2015 [besøkt 25. april 2015]. Tilgjengelig fra: http://static.googleusercontent.com/design/spec/components/buttons.html

11. Google design guidelines. Navigation drawer - Patterns - Google design guidelines [Internett]. 2015 [besøkt 30. april 2015]. Tilgjengelig fra: http://static.googleusercontent.com/design/spec/patterns/navigation-drawer.html

12. Oodesign.com. Singleton Pattern | Object Oriented Design [Internett]. 2015 [besøkt 5. mai 2015]. Tilgjengelig fra: http://www.oodesign.com/singleton-pattern.html

13. Developer.android.com. [Internett]. 2015 [besøkt 25. mars 2015]. Tilgjengelig fra: http://developer.android.com/downloads/design/Android_Design_Icons_20131106.zip

14. GitHub. lucasr/dspec [Internett]. 2015 [besøkt 28. mars 2015]. Tilgjengelig fra: https://github.com/lucasr/dspec

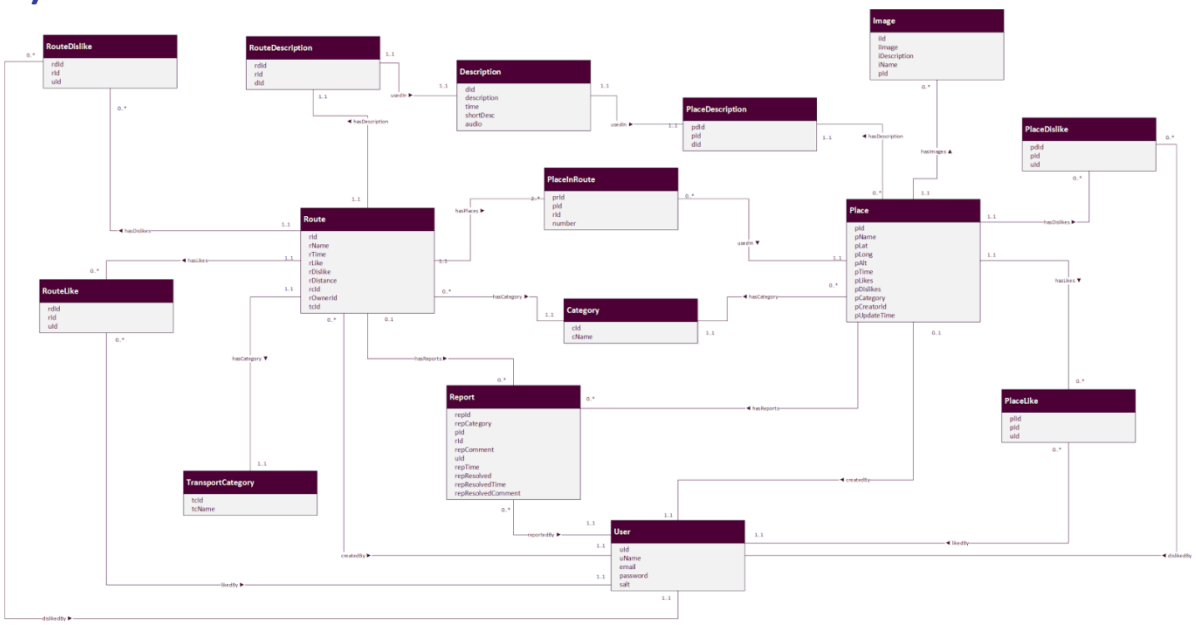
15. Chris Veness, Selecting points from a database by latitude/longitude within a bounding circle [Internett]. 2015 [besøkt 10. februar]. Tilgjengelig fra: http://www.movable-type.co.uk/scripts/latlong-db.html

16. Opensource.org. The MIT License (MIT) | Open Source Initiative [Internett]. 2015 [besøkt 11. mai 2015]. Tilgjengelig fra: http://opensource.org/licenses/mit-license.php

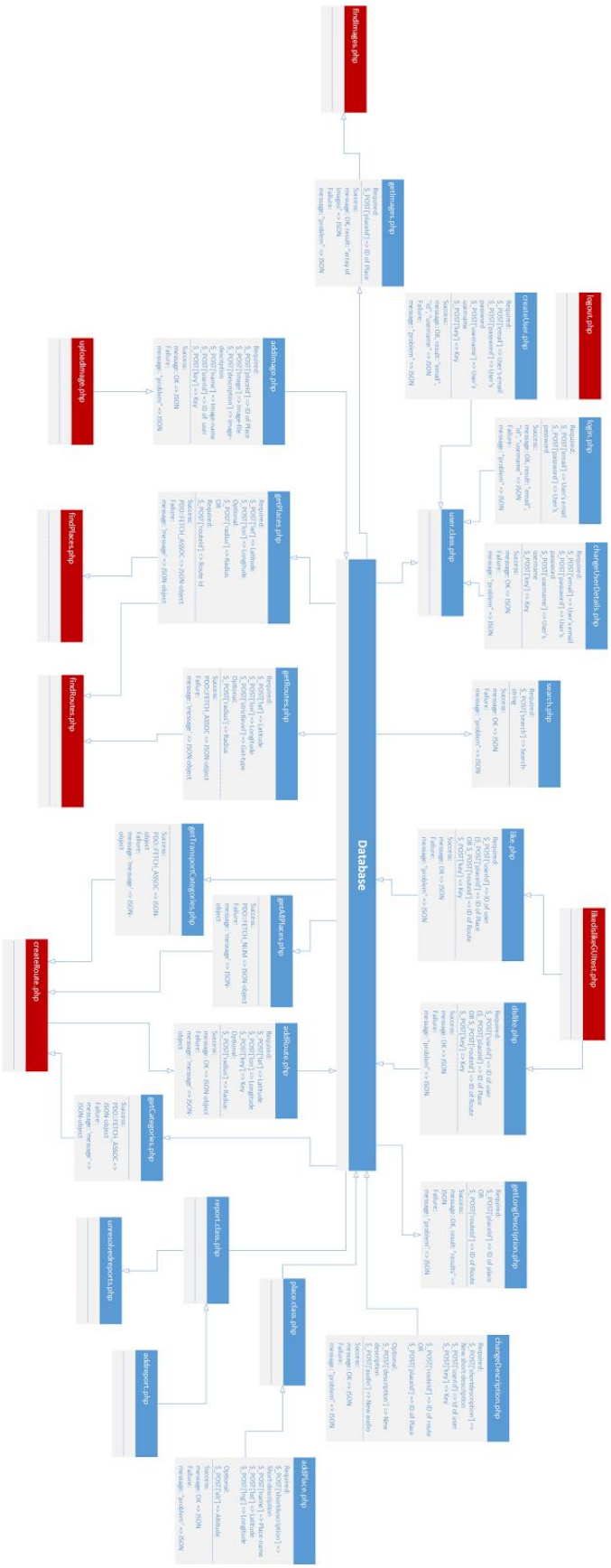
17. files . .htaccess doesn't redirect direct link to .php files [Internett]. Stackoverflow.com. 2015 [besøkt13. mai 2015]. Tilgjengelig fra: http://stackoverflow.com/questions/28879154/htaccess-doesnt-redirect-direct-link-to-php-files/28881089

VEDLEGG

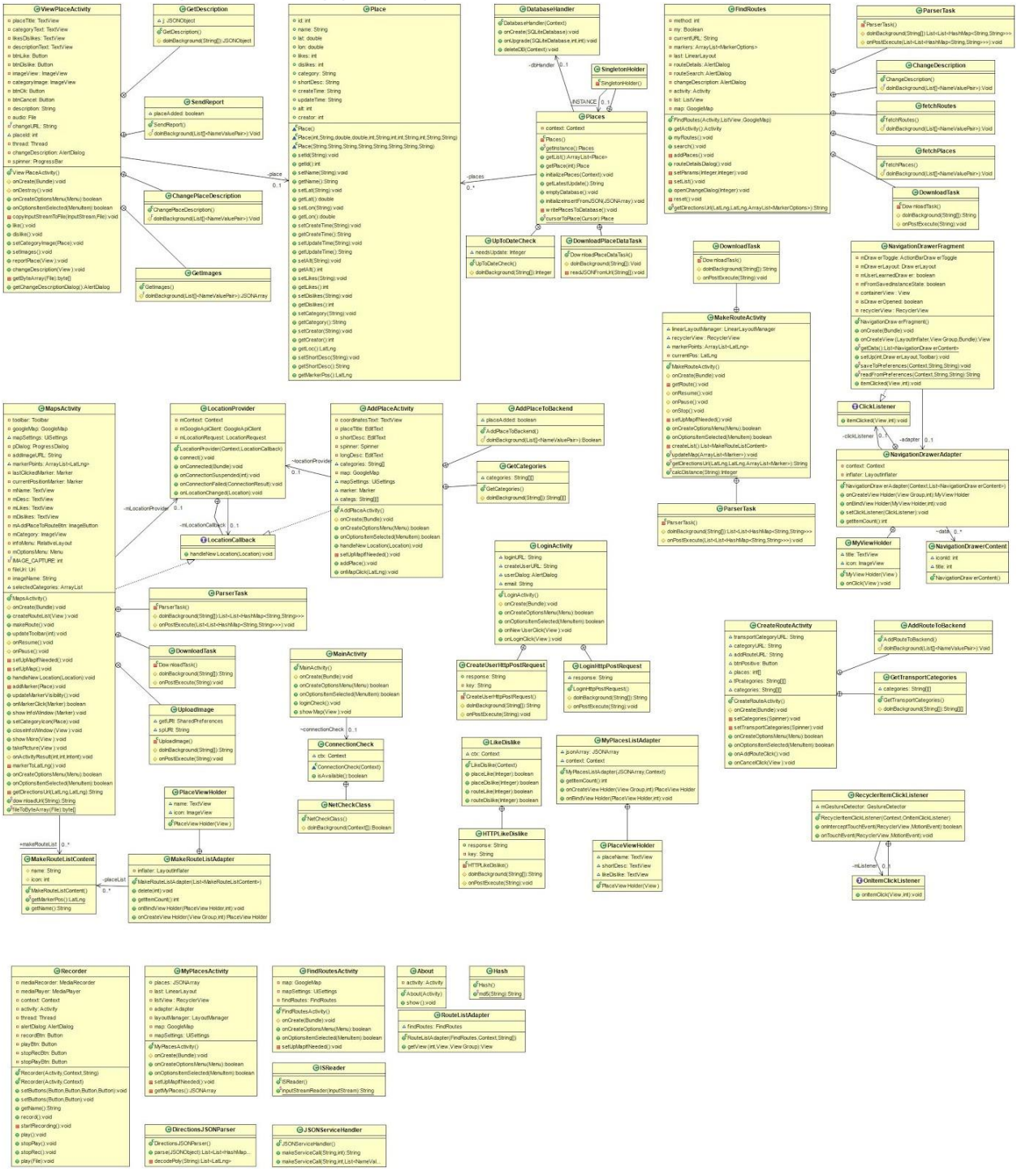
A) Full DB UML back-end:



B) API UML:


findImages.php
getImages.php
createUser.php
logout.php
login.php
user.class.php
changeUserDetails.php
uploadImage.php
addImage.php
findPlaces.php
getPlaces.php
findRoutes.php
getRoutes.php
search.php
Database
like.php
likedIsReqCollHost.php
dislike.php
getTransportCategories.php
getAllPlaces.php
createRoute.php
addRoute.php
getCategories.php
getLongDescription.php
report.class.php
unresolvedreport.php
place.class.php
changeDescription.php
addreport.php
addPlace.php

C) Klassediagram for applikasjonen



D) MIT lisens

E) Kode - MakeRouteListAdapter

```
public class MakeRouteListAdapter extends
RecyclerView.Adapter<MakeRouteListAdapter.PlaceViewHolder> {

  private List<MakeRouteListContent> placeList;

  public MakeRouteListAdapter(List<MakeRouteListContent> placeList) {
      this.placeList = placeList;
  }

  /**
   * Deletes an item from list of places.
   * Also deletes the corresponding marker.
   * Then the map is updated.
   * @param position
   */
  public void delete(int position){
      placeList.remove(position);
      MapsActivity.makeRouteMarkerList.remove(position);
      MakeRouteActivity.route.remove(position);
      notifyItemRemoved(position);
      notifyItemRangeChanged(position, MapsActivity.makeRouteMarkerList.size());
      notifyDataSetChanged();
      MakeRouteActivity.updateMap(MapsActivity.makeRouteMarkerList);
  }

  @Override
  public int getItemCount() {
      return placeList.size();
  }

  /**
   * Binds the content of the placeList to the corresponding recyclerview-item
   * @param placeViewHolder
   * @param position
   */

  @Override
  public void onBindViewHolder(PlaceViewHolder placeViewHolder, int position) {
      MakeRouteListContent current = placeList.get(position);
      placeViewHolder.name.setText(current.name);
      placeViewHolder.icon.setImageResource(current.icon);
  }

  /**
   * inflates the layout for each row in the list
   * @param parent
   * @param viewType
   * @return
   */
  @Override
  public PlaceViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
      View itemView = LayoutInflater.
              from(parent.getContext()).
              inflate(R.layout.custom_row_make_route, parent, false);
```

```
        return new PlaceViewHolder(itemView);
    }

    /**
     * PlaceViewHolder class that connects the data to the view
     */
    public class PlaceViewHolder extends RecyclerView.ViewHolder {
        TextView name;
        ImageView icon;

        public PlaceViewHolder(View itemView) {
            super(itemView);
            name =  (TextView) itemView.findViewById(R.id.name);
            icon = (ImageView) itemView.findViewById(R.id.icon);

            icon.setOnClickListener( new View.OnClickListener() {
                @Override
                public void onClick(View v){
                    delete(getPosition());
                }
            });
        }
    }
}
```

F) Kode - MakeRouteListContent

```
/**
* Class to hold the content of the routelist in MakeRouteActivity
*/
public class MakeRouteListContent {
  protected String name;
  protected int icon;
  protected static LatLng markerPos;

  public static LatLng getMarkerPos(){
      return markerPos;
  }

  public String getName(){
      return name;
  }

}
```

G) Kode - MakeRouteActivity

```
/**
*
* Creates a route based on the markers the user has decided to visit, with
* polylines between each marker. Also updates when item is removed from the list.
*/
public class MakeRouteActivity extends ActionBarActivity {
  LinearLayoutManager linearLayoutManager;
  RecyclerView recyclerView;
  //    public static ArrayList<Marker> routeMarkers = new ArrayList<>();
  public static ArrayList<LatLng> route = new ArrayList<LatLng>();

  static Integer routeDist;

  ArrayList<LatLng> markerPoints;

  private static GoogleMap googleMap;
  private LatLng currentPos;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_make_route_appbar);

      Toolbar toolbar = (Toolbar) findViewById(R.id.app_bar);
      setSupportActionBar(toolbar);

      markerPoints = new ArrayList<>();

      getSupportActionBar().setHomeButtonEnabled(true);
      getSupportActionBar().setDisplayHomeAsUpEnabled(true);
      getSupportActionBar().setHomeAsUpIndicator(R.mipmap.ic_menu_cancel);

      recyclerView = (RecyclerView) findViewById(R.id.placeList);
      recyclerView.setHasFixedSize(true);
      linearLayoutManager = new LinearLayoutManager(this);
      linearLayoutManager.setOrientation(LinearLayoutManager.VERTICAL);
      recyclerView.setLayoutManager(linearLayoutManager);

      MakeRouteListAdapter adapter = new MakeRouteListAdapter(createList());
      recyclerView.setAdapter(adapter);

      currentPos = MapsActivity.currentPosition;

      getRoute();
      setUpMapIfNeeded();

      CameraPosition cameraposition = new CameraPosition.Builder()
              .target(currentPos)
              .zoom(11)
              .build();
      googleMap.moveCamera(CameraUpdateFactory.newCameraPosition(cameraposition));

      updateMap(MapsActivity.makeRouteMarkerList);
  }
```

```
private void getRoute() {
    for (Marker m : MapsActivity.makeRouteMarkerList) {
        LatLng pos = m.getPosition();
        route.add(pos);
    }
}

@Override
protected void onResume(){
    super.onResume();
    setUpMapIfNeeded();
}

@Override
protected void onPause(){
    super.onPause();
    if(googleMap != null) {
        googleMap.clear();
        googleMap = null;
    }
    if(MapsActivity.makeRouteList.size() > 0) {
        MapsActivity.makeRouteList.clear();
    }
    if(MapsActivity.makeRouteMarkerList.size() > 0) {
        MapsActivity.makeRouteMarkerList.clear();
    }
}

@Override
protected void onStop(){
    super.onStop();
    if(googleMap != null) {
        googleMap.clear();
        googleMap = null;
    }
    if(MapsActivity.makeRouteList.size() > 0) {
        MapsActivity.makeRouteList.clear();
    }
    if(MapsActivity.makeRouteMarkerList.size() > 0) {
        MapsActivity.makeRouteMarkerList.clear();
    }
}

private void setUpMapIfNeeded(){
    if(googleMap == null){
        googleMap = ((SupportMapFragment) getSupportFragmentManager()
                .findFragmentById(R.id.makeRouteMap)).getMap();
        if(googleMap != null){
            updateMap(MapsActivity.makeRouteMarkerList);
        }
    }
}


@Override
```

```
  public boolean onCreateOptionsMenu(Menu menu) {
      // Inflate the menu; this adds items to the action bar if it is present.
      getMenuInflater().inflate(R.menu.menu_save, menu);
      return true;
  }

  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
      int id = item.getItemId();

      if (id == R.id.btnSave){
          int[] places = new int[MapsActivity.makeRouteMarkerList.size()];
          for(int i = 0; i < MapsActivity.makeRouteMarkerList.size(); i++) {
              places[i] =
MapsActivity.markerToPlaceId.get(MapsActivity.makeRouteMarkerList.get(i));
          }
          Log.i("Places.length: ", places.length+"");

          Intent intent = new Intent(this, CreateRouteActivity.class);
          intent.putExtra("places", places);
          intent.putExtra("distance", routeDist);
          startActivity(intent);
      }
      return super.onOptionsItemSelected(item);
  }

  private List<MakeRouteListContent> createList() {
      return MapsActivity.makeRouteList;
  }

  /**
   * Function for updating the map when an item is removed from the list
   * @param routeList
   */
  public static void updateMap(ArrayList<Marker> routeList) {
      //first clear the map of all markers
      googleMap.clear();

      // Then add the markers again
      for(Integer i = 0; i<MapsActivity.makeRouteList.size(); i++) {
          MarkerOptions placeMarker = new MarkerOptions()
                  .position(MapsActivity.makeRouteMarkerList.get(i).getPosition())
                  .title(""+(i+1)+". "+MapsActivity.makeRouteMarkerList.get(i).getTitle())
                  .alpha(0.8f)
                  .snippet(MapsActivity.makeRouteMarkerList.get(i).getSnippet())
                  .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_A
ZURE));

          googleMap.addMarker(placeMarker);
      }

      // Checks, whether start and end locations are captured
      if(routeList.size() >= 2){
          LatLng origin = routeList.get(0).getPosition();
          LatLng dest = routeList.get(routeList.size()-1).getPosition();
```

```
            // Getting URL to the Google Directions API
            String url = getDirectionsUrl(origin, dest, routeList);

            DownloadTask downloadTask = new DownloadTask();

            // Start downloading json data from Google Directions API
            downloadTask.execute(url);
        }
    }


    /**
     * Drawing the polylines on the map
     * @param origin
     * @param dest
     * @param routeMarkerList
     * @return
     */
    public static String getDirectionsUrl(LatLng origin,LatLng dest, ArrayList<Marker>
routeMarkerList){
        // Origin of route
        String str_origin = "origin="+origin.latitude+","+origin.longitude;

        // Destination of route
        String str_dest = "destination="+dest.latitude+","+dest.longitude;

        // Sensor enabled
        String sensor = "sensor=false";

        // Waypoints
        String waypoints = "";
        if(routeMarkerList.size()>2) {
            waypoints = "waypoints=";
            for (int i = 1; i < routeMarkerList.size() - 1; i++) {
                LatLng point = (LatLng) routeMarkerList.get(i).getPosition();
                waypoints += point.latitude + "," + point.longitude + "|";
            }
        }

        // Building the parameters to the web service
        String parameters = str_origin+"&"+str_dest+"&"+sensor+"&"+waypoints;

        // Output format
        String output = "json";

        // Building the url to the web service
        String url =
"https://maps.googleapis.com/maps/api/directions/"+output+"?"+parameters;

        return url;
    }

    /**
     *  Fetches data from url passed
     */
```

```
private static class DownloadTask extends AsyncTask<String, Void, String>{

    // Downloading data in non-ui thread
    @Override
    protected String doInBackground(String... url) {

        // For storing data from web service
        String data = "";

        try{
            // Fetching the data from web service
            data = MapsActivity.downloadUrl(url[0]);
        }catch(Exception e){
            Log.e("Background Task",e.toString());
        }

        return data;
    }

    /**
     * Executes in UI thread, after the execution of
     * doInBackground()
     * @param result
     */
    @Override
    protected void onPostExecute(String result) {
        super.onPostExecute(result);

        ParserTask parserTask = new ParserTask();

        // Invokes the thread for parsing the JSON data
        if(result != null) {
            parserTask.execute(result);
            routeDist = calcDistance(result);
        } else {
            Log.e("DownloadTask", "result == null");
        }
    }
}

/**
 * Calculates the distance of the route
 * @param response
 * @return
 */
public static Integer calcDistance(String response) {
    Integer res = 0;
    try {
        JSONObject obj1 = new JSONObject(response);
        JSONArray arr1 = new JSONArray(obj1.get("routes").toString());
        JSONObject obj2 = new JSONObject(arr1.getJSONObject(0).toString());
        JSONArray arr2 = new JSONArray(obj2.getJSONArray("legs").toString());
        Log.i("Test calcDist arr2.len", ""+arr2.length());
        for(int i = 0; i<arr2.length(); i++) {
            JSONObject obj3 = new JSONObject(arr2.getJSONObject(i).toString());
            JSONObject obj4 = new JSONObject(obj3.getJSONObject("distance").toString());
```

```
                res += obj4.getInt("value");
            }
            Log.i("Test calcDistance res", ""+res);
        } catch (JSONException e) {
            e.printStackTrace();
        }
        return res;
    }

    /**
     * A class to parse the Google Places in JSON format
     */
    private static class ParserTask extends AsyncTask<String, Integer,
List<List<HashMap<String,String>>> >{

        // Parsing the data in non-ui thread
        @Override
        protected List<List<HashMap<String, String>>> doInBackground(String... jsonData) {

            JSONObject jObject;
            List<List<HashMap<String, String>>> routes = null;

            try{
                jObject = new JSONObject(jsonData[0]);
                DirectionsJSONParser parser = new DirectionsJSONParser();

                // Starts parsing data
                routes = parser.parse(jObject);
            }catch(Exception e){
                e.printStackTrace();
            }
            return routes;
        }

        // Executes in UI thread, after the parsing process
        @Override
        protected void onPostExecute(List<List<HashMap<String, String>>> result) {

            ArrayList<LatLng> points = null;
            PolylineOptions lineOptions = null;

            if (result != null) {
                for (int i = 0; i < result.size(); i++) {
                    points = new ArrayList<LatLng>();
                    lineOptions = new PolylineOptions();

                    // Fetching i-th route
                    List<HashMap<String, String>> path = result.get(i);

                    // Fetching all the points in i-th route
                    for (int j = 0; j < path.size(); j++) {
                        HashMap<String, String> point = path.get(j);

                        double lat = Double.parseDouble(point.get("lat"));
                        double lng = Double.parseDouble(point.get("lng"));
                        LatLng position = new LatLng(lat, lng);
```

```
                    points.add(position);
                }

                // Adding all the points in the route to LineOptions
                lineOptions.addAll(points);
                lineOptions.width(10);
                lineOptions.color(Color.MAGENTA);
            }

            // Drawing polyline in the Google Map for the i-th route
            if (googleMap != null) {
                googleMap.addPolyline(lineOptions);
            }
        } else {
            Log.e("Error", "Draw route result == null");
        }
    }
  }
}
```

H) Kode - Requesthandler.php

```php
<?php

/**
* Handles URLs from www.storsveen.rocks/guide/...
* Sends the request to the correct script.
*/

// Class to handle requests
class RequestHandler {
   public $path;

   function __construct() {
       // Getting request:
       reset($_GET);
       next($_GET);
       $this->path = strtolower(key($_GET));

       switch($this->path) {                    // Switch through possible urls. If none
match, user will be sent to front-page.
           case "/api/getallplaces/":
           case "/api/getallplaces":
               include("getAllPlaces.php");
               break;
           case "/createroute/":
           case "/createroute":
               include("createRoute.php");
               break;
         case "/createplace/":
           case "/createplace":
               include("createPlace.php");
               break;
           case "/api/getplaces/":
           case "/api/getplaces":
               include("getPlaces.php");
               break;
           case "/api/getcategories/":
           case "/api/getcategories":
               include("getCategories.php");
               break;
           case "/api/gettransportcategories/":
           case "/api/gettransportcategories":
               include("getTransportCategories.php");
               break;
           case "/api/getlongdescription/":
           case "/api/getlongdescription":
               include("getLongDescription.php");
               break;
           case "/findplaces/":
           case "/findplaces":
               include("findPlaces.php");
               break;
           case "/api/addroute/":
           case "/api/addroute":
               include("addRoute.php");
               break;
```

```
    case "/findroutes/":
    case "/findroutes":
        include("findRoutes.php");
        break;
    case "/api/getroutes/":
    case "/api/getroutes":
        include("getRoutes.php");
        break;
    case "/likedislikeguitest/":
    case "/likedislikeguitest":
        include("likedislikeGUItest.php");
        break;
    case "/api/like/":
    case "/api/like":
        include("like.php");
        break;
    case "/api/dislike/":
    case "/api/dislike":
        include("dislike.php");
        break;
case "/api/addplace/":
    case "/api/addplace":
        include("addPlace.php");
        break;
    case "/addreport/":
    case "/addreport":
        include("addreport.php");
        break;
    case "/unresolvedreports/":
    case "/unresolvedreports":
        include("unresolvedreports.php");
        break;
    case "/api/adduser/":
    case "/api/adduser":
        include("addUser.php");
        break;
    case "/createuser/":
    case "/createuser":
        include("createUser.php");
        break;
    case "/api/login/":
    case "/api/login":
        include("login.php");
        break;
    case "/api/logout/":
    case "/api/logout":
        include("logout.php");
        break;
    case "/api/changeuserdetails/":
    case "/api/changeuserdetails":
        include("changeUserDetails.php");
        break;
    case "/api/changedescription/":
    case "/api/changedescription":
        include("changeDescription.php");
        break;
    case "/uploadimage":
```

```
            case"/uploadimage/":
                include("uploadImage.php");
                break;
            case "/api/addimage":
            case"/api/addimage/":
                include("addImage.php");
                break;
            case "/api/getimages":
            case"/api/getimages/":
                include("getImages.php");
                break;
            case "/findimages":
            case"/findimages/":
                include("findImages.php");
                break;
            case "/api/search":
            case"/api/search/":
                include("search.php");
                break;
            case "/api/removeplace":
            case"/api/removeplace/":
                include("removePlace.php");
                break;
            case "":
            case "/":
                include("index.php");
                break;
            default:
                echo 'ERROR 404 '.$this->path.' not found. <br> Navigate <a
href="http://www.storsveen.rocks/guide">here</a> for frontpage';
                break;
        }
    }
}

$handler = new RequestHandler();        // Creates new instance of request-handler.
```

I) Kode - public void takePicture - Mapsactivity.java

```
public void takePicture(View view) {
       Log.i("Test TakePicture", "Running");
       //Integer placeId = markerToPlaceId.get(lastClickedMarker);

       if (getPackageManager().hasSystemFeature(PackageManager.FEATURE_CAMERA)){
           String dateTime = new SimpleDateFormat("yyyyMMdd_HHmmss",
Locale.GERMAN).format(new Date());
           imageName = "STUTTREIST_" + dateTime + ".jpg";

           folder.mkdir();

           File imageFile = new
File(Environment.getExternalStorageDirectory().getAbsolutePath()
                   + "/STUTTREIST/" + imageName);

           Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
           fileUri = Uri.fromFile(imageFile);
           Log.i("Test takePicture", "fileURI = "+fileUri.getPath());
           Log.i("Test takePicture", "fileURI.tostring = "+fileUri.getPath().toString());
           SharedPreferences UriSP = getSharedPreferences(URI_PATH, 0);
           UriSP.edit().putString("URI", fileUri.getPath()).apply();

           Log.i("Test takePicture", "lastClickedMarker-ID =
"+markerToPlaceId.get(lastClickedMarker));
           SharedPreferences setId = getSharedPreferences(LAST_CLICKED_ID, 0);
           setId.edit().putString("lastClickedId",
markerToPlaceId.get(lastClickedMarker)+"").apply();

           intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri);
           startActivityForResult(intent, IMAGE_CAPTURE);
       }
   }
```

J) Kode - private class UploadImage

```
private class UploadImage extends AsyncTask<String, Void, String>{
       SharedPreferences getURI = getSharedPreferences(URI_PATH, 0);
       String spURI = getURI.getString("URI", null);
       // Downloading data in non-ui thread
       @Override
       protected String doInBackground(String... uri) {
           Log.i("Test uploadimage", "fileuri: "+spURI);
           File file = new File(spURI);
           Log.i("Test uploadimage", "filesize: "+file.length());
           byte[] bytearray = fileToByteArray(file);

           String base24Image = Base64.encodeToString(bytearray, Base64.DEFAULT);

           SharedPreferences checkLogin =
getApplicationContext().getSharedPreferences(CreateRouteActivity.LOGIN_PREFS_NAME, 0);
           String userId = checkLogin.getString("userID", null);

           SharedPreferences getLastClicked = getSharedPreferences(LAST_CLICKED_ID, 0);
           String lastClickedId = getLastClicked.getString("lastClickedId", null);

           List<NameValuePair> params = new ArrayList<NameValuePair>();
           NameValuePair nameParam = new BasicNameValuePair("name", "newname");
           NameValuePair contentParam = new BasicNameValuePair("content", base24Image);
           NameValuePair idParam = new BasicNameValuePair("placeId", lastClickedId);
           NameValuePair userParam = new BasicNameValuePair("userId", userId);
           Log.i("Test placeid", markerToPlaceId.get(lastClickedMarker)+"");
           Log.i("Test lastCLickedID", lastClickedId);
           NameValuePair keyParam = new BasicNameValuePair("insert_key",
Hash.md5("my_key"));
           params.add(keyParam);
           params.add(userParam);
           params.add(nameParam);
           params.add(contentParam);
           params.add(idParam);
           Log.i("Test uploadimage", "params added: "+params.get(0)+" "+params.get(1)+"
"+params.get(3));

           try {
               Log.i("Test uploadimage", "Making service-handler");
               JSONServiceHandler sh = new JSONServiceHandler();
               Log.i("Test uploadimage", "Making servicecall");
               String jsonStr = sh.makeServiceCall(addImageURL, JSONServiceHandler.POST,
params);
               Log.i("Test uploadimage", "jsonStr: "+jsonStr);
               JSONObject json = new JSONObject(jsonStr);
               Log.i("ServerResponse", jsonStr);
               //Log.i("ServerResponse message", json.getString("message"));
               if(json.getString("message").equals("OK")) {
                   Log.i("Test addimage", "Image uploaded");
                   runOnUiThread(new Runnable() {
                       @Override
                       public void run() {
                           Toast.makeText(getApplicationContext(), "Image uploaded",
Toast.LENGTH_LONG).show();
                       }
```

```
                    });
                    return jsonStr;
                }
            } catch (Exception e) {
                Log.e("URL:", "Could not retrieve JSON data from guidedTour" +
e.getMessage());
            }
            return "Problem uploading picture. Try again later";
        }

        @Override
        protected void onPostExecute(String result) {
            super.onPostExecute(result);

            Log.i("Test uploadi response", result);

        }
    }


    // https://androidsnippets.wordpress.com/2012/08/07/how-to-convert-a-file-to-byte-array/
    public static byte[] fileToByteArray(File f) {
        Log.i("Test filetobyarray", "Running");
        byte[] bytes = null;
        try
        {
            Bitmap bmp = BitmapFactory.decodeFile(f.getPath());
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            bmp.compress(Bitmap.CompressFormat.JPEG, 35, baos);

            InputStream inputStream = new ByteArrayInputStream(baos.toByteArray());
            ByteArrayOutputStream bos = new ByteArrayOutputStream();
            byte[] b = new byte[1024*8];
            int bytesRead =0;

            while ((bytesRead = inputStream.read(b)) != -1) {
                bos.write(b, 0, bytesRead);
            }

            bytes = bos.toByteArray();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        Log.i("Test filetobyarray", "Returning bytearray");
        return bytes;
    }
```

K) Kode - Places.class

```
/**
* Container class for <Places>
* Singleton
*
* Contains all the places in a list which
* is retrieved from local database or
* through a call to the API
* if the local database is empty
*/
public class Places {
   private ArrayList<Place> places;
   private DatabaseHandler dbHandler;
   private Context context;

   private Places() {
       context = null;
       places = null;
       dbHandler = null;
   }

   private static class SingletonHolder {
       private static final Places INSTANCE = new Places();
   }

   /**
    * Gets the Places instance
    * @return INSTANCE
    */
   public static Places getInstance() {
       return SingletonHolder.INSTANCE;
   }

   /**
    * @return the list of places
    */
   public ArrayList<Place> getList() {
       return places;
   }

   /**
    * Returns a Place object of a certain id
    * @param id
    * @return Place-object
    */
   public Place getPlace(int id) {
       Place place = null;
       for(Place p : places){
           if(p.getId() == id)
               place = p;
       }
       return place;
   }

   /**
    * Initializes the places list by accessing the local database
```

```
     * or the web if the local database is empty or outdated.
     * @param context for the DatabaseHandler
     */
    public void initializePlaces(Context context) {
        this.context = context;
        this.places = new ArrayList<Place>();
        this.dbHandler = new DatabaseHandler(context);

        //getting latest update
        String lastUpdate = getLatestUpdate();
        Integer needsUpdate;
        if(lastUpdate != null) { //if database exists
            try {
                //checks with backend if DB is up to date
                needsUpdate = new
                           UpToDateCheck().execute(lastUpdate).get();
                if (needsUpdate == 1) {
                    emptyDatabase();
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            } catch (ExecutionException e) {
                e.printStackTrace();
            }
            Log.i("Last update", lastUpdate.toString());

            SQLiteDatabase db = dbHandler.getWritableDatabase();

            final Cursor c = db.query(TABLE_PLACE,
                    new String[]{PLACE_ID, PLACE_NAME, PLACE_LAT,
                            PLACE_LONG, PLACE_ALT, PLACE_TIME,
                            PLACE_LIKES, PLACE_DISLIKES,
                            PLACE_CAT, PLACE_CREATOR, PLACE_DESC,
                            PLACE_UPDATE_TIME},
                            null, null, null, null, null);

            // make sure you start from the first item
            c.moveToFirst();
            while (!c.isAfterLast()) {
                final Place place = cursorToPlace(c);
                places.add(place);
                c.moveToNext();
            }
            // Make sure to Fclose the cursor
            c.close();
        }

        if(places.isEmpty()) {
            Log.i("DB", "places is empty");
            new DownloadPlaceDataTask().execute(URL_PLACES);
        }
    }


    /**
     * Gets the latest update timestamp from the DB
```

```
     * @return  String timestamp
     */
    public String getLatestUpdate(){
        SQLiteDatabase db = dbHandler.getWritableDatabase();
        String qu = "SELECT " + PLACE_UPDATE_TIME + " FROM " +
                    TABLE_PLACE + " WHERE "
                    + PLACE_UPDATE_TIME + " in (SELECT max(" +
                    PLACE_UPDATE_TIME+ ") FROM " + TABLE_PLACE + ")";
        final Cursor t = db.rawQuery(qu, null);
        t.moveToFirst();

        if(t.getCount() != 0)
            return t.getString(0);
        else
            return null;

    }

    /**
     * empties the outdated db
     */
    public void emptyDatabase(){
        /*SQLiteDatabase db = dbHandler.getWritableDatabase();
        db.rawQuery("DELETE FROM " + TABLE_PLACE, null);

        db.rawQuery("VACUUM", null);

        Cursor test = db.rawQuery("SELECT * FROM "
                                  + TABLE_PLACE, null);
        Log.i("DB DATA", test.getCount() + " After delete");

        db.close();*/
        dbHandler.deleteDB(context);
        this.dbHandler = new DatabaseHandler(context);
        Log.i("DB", "DB dropped");
    }

    /**
     * This function shall only be called if the
     * local database is empty
     *
     * Initializes datastructure from JSON and syncs with database
     *
     * @param array JSON ecoded array
     */
    public void initializeInsertFromJSON(JSONArray array) {
        try {
            for(int i = 0; i < array.length(); i++) {

                JSONObject row = array.getJSONObject(i);
                Place place = new Place(row.getInt(PLACE_ID),
                            row.getString(PLACE_NAME),
                            row.getDouble(PLACE_LAT),
                            row.getDouble(PLACE_LONG),
                            row.getInt(PLACE_ALT),
                            row.getString(PLACE_TIME),
```

```
                            row.getInt(PLACE_LIKES),
                            row.getInt(PLACE_DISLIKES),
                            row.getString(PLACE_CAT),
                            row.getInt(PLACE_CREATOR),
                            row.getString(PLACE_DESC),
                            row.getString(PLACE_UPDATE_TIME));
                    places.add(place); // adding to list
                }
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }

        private void writePlacesToDatabase() {

            SQLiteDatabase db = dbHandler.getWritableDatabase();

            Cursor dbCursor = db.query(TABLE_PLACE, null, null, null, null, null, null);
            String[] columnNames = dbCursor.getColumnNames();
            for(int i = 0; i < columnNames.length; i++)
                Log.i("Databasion", columnNames[i]);

            ContentValues values = new ContentValues();

            for(Place place : places) {

                values.put(PLACE_ID, place.getId());
                values.put(PLACE_NAME, place.getName());
                values.put(PLACE_LAT, place.getLat());
                values.put(PLACE_LONG, place.getLon());
                values.put(PLACE_ALT, place.getAlt());
                values.put(PLACE_TIME, place.getCreateTime());
                values.put(PLACE_LIKES, place.getLikes());
                values.put(PLACE_DISLIKES, place.getDislikes());
                values.put(PLACE_CAT, place.getCategory());
                values.put(PLACE_CREATOR, place.getCreator());
                values.put(PLACE_DESC, place.getShortDesc());
                values.put(PLACE_UPDATE_TIME, place.getUpdateTime());
                db.insert(TABLE_PLACE, null, values);
            }
            db.close();
        }

        /**
         * @param c table cursor
         * @return a place set with data from the database
         */
        public static Place cursorToPlace(Cursor c) {
            final Place place = new Place();

            place.setId(c.getString(c.getColumnIndex(PLACE_ID)));
            place.setName(c.getString(c.getColumnIndex(PLACE_NAME)));
            place.setLat(c.getString(c.getColumnIndex(PLACE_LAT)));
            place.setLon(c.getString(c.getColumnIndex(PLACE_LONG)));
            place.setAlt(c.getString(c.getColumnIndex(PLACE_ALT)));
            place.setCreateTime(c.getString(c.getColumnIndex(PLACE_TIME)));
```

```
        place.setLikes(c.getString(c.getColumnIndex(PLACE_LIKES)));
        place.setDislikes(c.getString(c.getColumnIndex(PLACE_DISLIKES)));
        place.setCategory(c.getString(c.getColumnIndex(PLACE_CAT)));
        place.setCreator(c.getString(c.getColumnIndex(PLACE_CREATOR)));
        place.setShortDesc(c.getString(c.getColumnIndex(PLACE_DESC)));
        place.setUpdateTime(c.getString(c.getColumnIndex(PLACE_UPDATE_TIME)));

        Log.i("PLACE", place.getName() + place.getAlt());
        return place;
    }


    static final String URL_PLACES = "http://www.storsveen.rocks/guide/api/getAllPlaces";
    static final String TABLE_PLACE = "place";

    static final String PLACE_ID = "pId";
    static final String PLACE_NAME = "pName";
    static final String PLACE_LAT = "pLat";
    static final String PLACE_LONG = "pLong";
    static final String PLACE_ALT = "pAlt";
    static final String PLACE_TIME = "pTime";
    static final String PLACE_LIKES = "pLikes";
    static final String PLACE_DISLIKES = "pDislikes";
    static final String PLACE_CAT = "cName";
    static final String PLACE_CREATOR = "pCreatorId";
    static final String PLACE_DESC = "shortDesc";
    static final String PLACE_UPDATE_TIME = "pUpdateTime";


    public static final String PLACE_CREATE_TABLE = "CREATE TABLE " + TABLE_PLACE
            + " (" + PLACE_ID + " INTEGER PRIMARY KEY, "
            + PLACE_NAME + " TEXT,"
            + PLACE_LAT + " DOUBLE,"
            + PLACE_LONG + " DOUBLE,"
            + PLACE_ALT + " INTEGER,"
            + PLACE_TIME + " TEXT,"
            + PLACE_LIKES + " INTEGER,"
            + PLACE_DISLIKES + " INTEGER,"
            + PLACE_CAT + " TEXT,"
            + PLACE_CREATOR + " TEXT,"
            + PLACE_DESC + " TEXT,"
            + PLACE_UPDATE_TIME+ " TEXT" +
            ");";


    public class DownloadPlaceDataTask extends AsyncTask<String, Void, Void> {

        @Override
        protected Void doInBackground(String... urls) {
            try {
                readJSONFromUrl(urls);
                Log.i("baalls", "Places");
            } catch (Exception e) {
                Log.e("URL:", "Could not retrieve JSON data from guidedTour"
                        + e.getMessage() + " " + urls);
            }
```

```
            return null;
        }

        private void readJSONFromUrl(String... url) throws IOException, JSONException {
            JSONServiceHandler sh = new JSONServiceHandler();

            String jsonStr = sh.makeServiceCall(url[0], JSONServiceHandler.GET);

            if(jsonStr != null) {
                try {
                    JSONArray placesJSONArray = new JSONArray(jsonStr);
                    Places.getInstance().initializeInsertFromJSON(placesJSONArray);
                } catch (JSONException e) {
                    e.printStackTrace();
                }
            }
            writePlacesToDatabase();
        }
    }


    public class UpToDateCheck extends AsyncTask<String, Void, Integer> {
        Integer needsUpdate = 0;
        @Override
        protected Integer doInBackground(String... latestUpdate) {
            //formating timestamp, replaceing space and colon
            String formatedDate = latestUpdate[0].replaceAll(" ", "%20");
            formatedDate = formatedDate.replaceAll(":", "%3A");
            String url = URL_PLACES + "?updatetime=" + formatedDate;
            try {
                JSONServiceHandler sh = new JSONServiceHandler();
                String jsonStr = sh.makeServiceCall(url, JSONServiceHandler.GET);
                if(jsonStr != null) {
                    try {
                        JSONObject job = new JSONObject(jsonStr);
                        String result = job.getString("message");
                        Log.i("DB needs update", result);
                        if (result.equals("false")) {
                            Log.i("DB", "Do not need update: " + url);
                        } else {
                            Log.i("DB", "needs update: " + url);
                            needsUpdate = 1;
                        }
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }
            } catch (Exception e) {
                Log.e("URL:", "Could not retrieve JSON data from guidedTour"
                        + e.getMessage() + " " + url);
            }

            return needsUpdate;
        }
    }
```

L) Møtereferat

AndMark Møtereferat 15.1.15

Møtedeltakelse: Pål, Dennis, Anders, Markus

Starttidspunkt: 12.00

Slutttidspunkt: 12.35

- Mer påfyll til oppgaven/ mer utfordrende.
    - Lage spill ut av oppgaven: poeng.
    - Ruter i mange kategorier: Fjellhallen, shopping, kultur, idrett.
    - Wear/smartklokke. Høre med Game-laben om utstyr. Mariusz/Simon.
        - Vise informasjon. Statiske kart, bilder på enheten.
- Løsningen lages så den ikke brukes bare i gjøvik. Hele landet.
- Push-meldinger når brukere nærmer seg et punkt.
- Ruter og nye attraksjoner fra brukere må godkjennes (?)
- Regler for størrelse på bilder ++. / bestemt mal på hvordan infosiden om attraksjon skal se ut.
- Hvordan lage nettside: wordpress.
- Filtrering av ruter: Gå, sykle, vansklighetsgrad, funksjonshemmede.
- Logging av brukeraktivitet: tider o.l.
- Kontakte historielag o.l. om å få punkter av interesse, informasjon ++.
- Web-grenesnitt for brukere for å legge inn ruter er noe som kan vurderes.
- Lage for alle aldre og alle typer mennesker.
- Valg av teknologi: Vi kan velge det vi vil.
    - MySQL anbefales ettersom alle webhotel støtter dette. Kan også bruke MongoDB eller andre, men de anbefaler MySQL.
    - Vi bruker vår egen brukerdatabase.
    - Vi burde samle data/statistikk om brukere som vi kan bruke.
- Det er penger som sponses til helse- og kulturprosjekter, så det det kan være lurt å vise frem/markedsføre appen. Statlig/kommunal støtte.
- Ikke bruke lyse farger på hvitt. Kontraster. Ikke bruke pastell: passer dårlig til skjerm.
- Det er nok av muligheter/funksjoner som kan legges til i appen.

- Navn på appen: "Kortreist", "Nasjonalromantisk", norsk.
- Skriv rapporten tidlig/under veis. Start på arkitekturen nå. Use case osv.
- Forprosjektplanen kan endres. Skriv i endelig rapport hva som vi endret i forhold til forprosjektplanen.
- Kommunikasjon mellom oppdragsgiver og bachelorgruppen skal først å fremst foregå gjennom Gruppeleder. Samle det vi lurer på i en mail. Ryddigere. Kontakt anmark på post@andmark.no så begge kan svare. Gjerne samle en liste med spørsmål i mailen.
- Prøve å inkludere oppdragsgiver under hele perioden. Komme med demoer tidlig.
- Sjekk ut "Material Design".(Layout). Android 5.
- Veileder skal hjelpe oss med rapporten og ikke nødvendigvis det teknologiske.

- Brukere kan lage ruter i appen, enten med å plotte i kart eller markere mens man går ruten.
- Sortere ruten på mange/flere forskjellige parametere.
    - Rutesøk/rutefilter.
- Mulighet for et webgrensesnitt for brukeren for å lage ruten. For å utvide oppgaven.
- Både A-B ruter og runder.
- Mulighet for butikkkampanjer.

Møte med veileder 21.01.15

Tilstede: Pål, Dennis, Trine, Ivar

Starttdspunkt: 09:15

Sluttidspunkt: 10:00

Algoritme for å finne en rute? (hvis du eks vil gå en rute på 2 timer, så finner den dette)
Veileders oppgave: Mener det er lurest om vi diskuterer teknologiske valg med AndMark, veileder brukes til prosess, fremdrift, rapporten.

Veileder anbefaler at vi har en eller annen form for daily scrum (eller kanskje annenhver dag).
Dele inn i sprinter (eks med farger i Trello, evt Jira).
Smartklokke = høyere risiko
Skriv avgrensning til bare Android i forprosjektrapporten.

Veileder vil gjerne være med på sprintretorspekts (i tillegg til møter i forbindelse med statusrapportene). Code review er veldig nyttig, at alltid minst en av de andre ser over koden. (legge code review inn i Trello). Sitte sammen under reviewen.
Anbefaler å teste på lokal server, fremfor live server (spesielt ifht Git).

Sette opp faste møter på onsdager, feks annenhver onsdag (når som helst før lunsj) kl 09.15. Avtalt møte førstkommende onsdag kl 09.15.

Vi har bedt om utsettelse på forprosjektrapporten til helga etter 28.01, på grunn av at forrige oppdragsgiver trakk seg i begynnelsen av januar. Trine sender mail til Ivar.

Møte med AndMark 21.01.15

Tilstede: Anders, Markus, Pål, Dennis, Trine

Starttidspunkt: 13:00

Sluttidspunkt: 13:30

Ting vi vil ta opp:

- Lyd til turen
- Forslagene våre er p.t. veldig likt TripAdvisor..
    - Gjøre appen mer "lokal"
    - Begrense til historie/kultur/sport
    - Muligheter for å lage rute
    - Lydspor som forteller deg om stedet du er på, istedet for å lese
    - Mulighet for å ha på appen, uten rute og få notification nå du er i nærheten av noe

Oppdragsgiver er utelukkende positive til alle forslag. Uttrykker ønske om å gjøre appen veldig norsk - stikkord: bunad, koselig, lokal, rosemaling

Møte med veileder 28.01.15

Tilstede: Pål, Dennis, Trine, Ivar

Starttidspunkt: 09:15

Sluttidspunkt: 10:00

Ting vi vil ta opp:

- Bør vi ha med Use Case og/eller Personas i forprosjektrapporten?
- Bør vi ha med mockups av tenkt resultat i forprosjektrapporten?

KAN ta med mockups. Trenger ikke Use Case eller Personas.

Levere prosjektavtalen i dag.

Avklare lengden på sprinter med oppdragsgiver, de skal være med på retrospektmøter.

Faste møter annenhver onsdag kl 09.15

Ta rapportskrivingen utenom sprintene. Vi bør begynne og skrive noteter til rapporten allerede nå.

Målgruppe for rapporten: "oss selv, i forhold til det vi kan nå". Dvs: alt nytt vi lærer skal være med i rapporten.

Legge prioriteringsliten inn i Trello.

Anbefaler å evt legge ID-nummer på sakene i Trello, for å gjøre det enklere å holde oversikt.

Hva med testing? jUnit? Automatisert testing?

Veileder vil gjerne være med på de føste par oppstartssprintgreiene.

Møte med veileder 25.02.15

Tilstede: Ivar, Dennis, Pål, Trine

Starttidspunkt: 09.15

Sluttidspunkt: 10:00

Vi kan evt bruke git-k for lokal branch oversikt i repoen

statusrapport - ifht til fremdriftsplan, avvik?

Møte med veileder 11.02.15

Tilstede: Ivar, Pål, Trine, Dennis

Starttidspunkt: 09.15

Sluttidspunkt: 10:00

Snakket litt om hva vi har gjort så langt. Ivar anbefaler fortsatt Jira ifht tidsestimering.

Hør med AndMark ang testing.

Samlet statusrapport med gruppe 10.03.15, med veileder og AndMark 11.03.15

Tilstede, gruppemøte 10.03.15: Dennis, Pål, Trine

Tilstede, veildermøte 11.03.15: Ivar, Dennis, Pål, Trine

Tilstede, møte med AndMark 11.03.15: Anders, Markus, Dennis, Pål, Trine

Status pr i dag er at vi er ferdige med følgende elementer i appen:

- Startside
- Vise brukers lokasjon i app
- Vise steder i app
- Lage liste over steder man vil besøke
- Vise liste over steder man vil besøke
- (Lage en rute over valgte steder)

Vi ser nå at vi ikke får tid til å lage noen annen type app (for iOS eller Windows).
Vi burde ha kommet mye lenger, da vi har brukt veldig mye lenger tid enn estimert på oppstarten i appen. Det tok mye lenger tid enn forventet å hente inn lokasjon på den måten vi vil, og å hente ut places. Når det gjelder å hente ut places har vi hoppet litt frem og tilbake i forhold til backenløsningen og samspillet med appen, med tanke på hva slags informasjon vi vil hente ut og hvordan. Her kunne gjort det klarere på forhånd hva vil vil hente ut og hvordan vi vil hente det ut. I tillegg har vi brukt mer tid enn estimert på databaseløsningen i appen, og vi måtte gjøre en omstrukturering av arkitekturen.

Etter å ha gått gjennom status for hva som mangler, bør vi egentlig klare å kunne være ferdig med appen innen 31. mars. Etter dette er det stopp for å legge til ny funksjonalitet, og vi bruker tid på å "polere" appen - fokus på GUI og koderevisjon.

- Vi bør ha en uke på å legge til "ordentlige" steder, med skikkelig description.

Dennis har fullt fokus på backendløsningen.

Status for backendløsningen er at vi er ferdige med:

- Legge til sted

- Legge til rapport
- Legge til rute
- Legge til bruker
- Lage rute GUI
- Opprette bruker GUI
- Dislike
- Finn steder
- Finn ruter
- Hent alle steder
- Hent kategorier
- Hent description
- Hent steder
- Hent ruter
- Hent transportkategorier
- Like
- Rapporter
- Like og Dislike GUI
- RESTful API

Etter planen skal vi være ferdig med backendløsningen innen 23. mars. Pt ser det ut til at klarer å overholde denne fristen.
Testing av backend skal være ferdig
Vi har satt opp websiden for prosjektet.

Uke 11:
Appen:

- Bli ferdig med “lag rute” (Trine)
- Bli ferdig med å sjekke for oppdateringer ved oppstart (Pål)
- Bli ferdig med “les mer om punkt” (Pål)
- Begynne med rute delen: hente, vise og velge (Trine)
- Bli ferdig med backenddelen, (- bilder) (Dennis)

Uke 12:

- Bli ferdig med hente, vise og velge rute. (Trine)
- Begynne med lagre egen rute (gjøre den offentlig)
- Bli ferdig med "les mer om punkt" - GUI-delen inkl likes og bilder
- Bli ferdig med bilde-delen av backendløsningen (Dennis og evt Pål)
- Dennis blir med på app-delen.

Uke 13:

- Sorteringsfunksjoner - kategori og transporttype
- Lage eget punkt, med all funksjonalitet
- GUI-delen av appen, lage styles mm, få en helhelt og sammenheng i GUIen

Kort oppsummert: Vi må bruke MYE mer tid på appen i de neste tre ukene. 110% fokus.

Møte med AndMark 15.04.15

Tilstede: Dennis, Pål, Trine, Markus, Anders

Starttidspunkt: 10:30

Sluttidspunkt: 11:00

About - en slags kort brukerveiledning på appen

Error-meldinger i dialogbokser, ikke i toast.

Tall på ikonet når det legges til flere punkter i rute

Sende kildekoden på bitbucket til AndMark.

Hvis vi legger ut på Google Play, spesifiser at det er studentprosjekt, bacheloroppgave

Label over inputfeltene.

Går tilbake til hovedmenyen etter å ha lagt til sted, det trenger den ikke gjøre - Gå tilbake til kartet.

Møte med Ivar 29.04.15

Tilstede: Dennis, Pål, Trine, Ivar

Starttidspunkt: 09:15

Sluttidspunkt: 10.00

Design og implementering av det
"summen av user stories" i innledning
hvordan designet er - fra brukerens side
hvordan implementeres det
figurer - hvor kommer den fra, hvorfor har vi lov til å bruke den
Figur1 skal henvises til i teksten. Eksempel er vist i figur1.

Usecase mye tidligere (kravspek), helt i begynnelsen først overordnet, så brutt ned. (koblet opp mot trello).
Innlogging mangler use case, sekvensdiagram er spesialisering av usecasediagram (fortsatt på funskjonalitet, så design, så implementering, så kode).
Mer konsistent. Begynn med innholdsfortegnelse, se hva vil vi ha i detalj og hva vi ikke har i detalj. (et par eksempler på de mest interessante). Ta en ting og vis hele rekken i det. Hva tar vi med og hvorfor.Sjekk lisenser på biblioteker også.
Ta også med noe om hva scrum er (gjerne bok).
Bruke Vancouver-stilen

Møte med Ivar 13.05.15

Tilstede: Dennis, Pål, Trine
Starttidspunkt: 09:15
Sluttidspunkt: 10:00

Skal kunne lese rapporten uten å måtte kikke på vedleggene.
Kan evt skille ut teknologier vi har brukt i et eget kapittel (som en konsekvens av kravspek/design - plasseres etter en eller begge av de.)
Bytte ut en mannen i api-greiene med en “disk”.

IMPLEMENTASJON:
beskrive lagene overordnet - først på en måte beskrive hvor databasen kommer inn i bildet, APIet - osv. Gjerne ha med mer kode - spesielt den koden vi er mest fornøyd med.
Sette inn databasen i dokumentet.
Mange-til-mange-relasjoner og hjelpetabeller.

Ha med klassediagram.

Skriv inn i teksten av dette er noen eksempler (use-case-beskrivelses-greiene).

Innholdsfortegnelsen bør få plass på en side

Hvilke føringer gir det at vi bruker MIT-lisens?

M) Arbeidslogg

Uke	Hva	Timer
3.	Forprosjektsrapport	Trine: 22 timer Dennis: 18 timer Pål: 18 timer
4.	Forprosjektsrapport	Trine: 18 timer Dennis: 21 timer Pål: 19 timer
5.	Forprosjektsrapport	Trine: 19 timer Dennis: 19 timer Pål: 19 timer
6.	Planlegging	Trine: 13 timer Dennis: 18 timer Pål: 20 timer
7.	Backend API, planlegging applikasjon	Trine: 33 timer Dennis: 30 timer Pål: 27 timer
8.	Backend API, utvikling applikasjon	Trine: 18 timer Dennis: 33 timer Pål: 22 timer
9.	Backend API, utvikling applikasjon	Trine: 22 timer Dennis: 32 timer Pål: 24 timer
10.	Utvikling applikasjon	Trine: 15 timer Dennis: 13 timer Pål: 16 timer
11.	Utvikling applikasjon	Trine: 20 timer Dennis: 20 timer Pål: 37 timer
12.	Utvikling applikasjon	Trine: 18 timer Dennis: 35 timer Pål: 25 timer
13.	Utvikling applikasjon	Trine: 25 timer Dennis: 35 timer Pål: 29 timer

14.	Utvikling applikasjon	Trine: 25 timer Dennis: 32 timer Pål: 24 timer
15.	Utvikling applikasjon	Trine: 42 timer Dennis: 41 timer Pål: 22 timer
16.	Utvikling applikasjon	Trine: 38 timer Dennis: 29 timer Pål: 25 timer
17.	Rapport, Utvikling applikasjon	Trine: 18 timer Dennis: 37 timer Pål: 28 timer
18.	Rapport	Trine: 25 timer Dennis: 18 timer Pål: 24 timer
19.	Rapport	Trine: 20 timer Dennis: 20 timer Pål: 21 timer
20.	Rapport	Trine: 33 timer Dennis: 33 timer Pål: 33 timer

N) Prosjektavtale

HØGSKOLEN I GJØVIK

PROSJEKTAVTALE

mellom Høgskolen i Gjøvik (HiG) (utdanningsinstitusjon),

AndMark Software Development DA

(oppdragsgiver), og

Dennis André Østvik Gjerdingen
Pål Arild Storsveen
Trine Jeanette Storsveen

(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 10.01.15 til 15.05.15.

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der HiG 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 HiG 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 HiG. Studentene dekker utgifter for trykking og ferdigstillelse av den skriftlige besvarelsen vedrørende prosjektet.
   - 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. HiG 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 faglærer/veileder og sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode, disketter, taper mv. som inngår som del av eller vedlegg til besvarelsen, gis det en kopi av til HiG, som vederlagsfritt kan benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av HiG 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 HiG og/eller studenter har interesser.

   Besvarelser med karakter C eller bedre registreres og plasseres i skolens bibliotek. Det legges også ut en elektronisk prosjektbesvarelse uten vedlegg på bibliotekets del av skolens internett-sider. Dette avhenger av at studentene skriver under på en egen avtale hvor de gir biblioteket tillatelse til at deres hovedprosjekt blir gjort tilgjengelig i papir og nettutgave (jfr. Lov om opphavsrett). Oppdragsgiver og veileder godtar slik

offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og dekan om de i løpet av prosjektet endrer syn på slik offentliggjøring.

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 Fronter. I tillegg leveres et eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med et eksemplar til hver av partene. På vegne av HiG er det dekan/prodekan som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og HiG som nærmere regulerer 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, skjer dette uten HiG som partner.

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

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene i mellom. 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:

HiGs veileder (navn): Ivar Farup

Oppdragsgivers kontaktperson (navn): Anders Hagbakken og Markus Brovold

Student(er) (signatur): Pål S. dato 21/01-15

Trine J. Storsveen dato 21.01.15

Dennis Gjerdingen dato 21/1/15

_____ dato _____

Oppdragsgiver (signatur): [signature] dato 21/1/15

IMT Dekan/prodekan (signatur): _____ dato _____

Versjon Januar 2011hb

O) Forprosjektrapport

1. MÅL OG RAMMER

1.1. Bakgrunn

AndMark Software Development AS, i samarbeid med Høgskolen i Gjøvik, ønsker å få utviklet en android applikasjon hvor konseptet er Guided Tour. Tanken bak applikasjonen er at brukerne på en enkel måte introduseres for severdigheter i Gjøvik og omegn. Applikasjonen skal også kunne brukes over hele landet, og brukerne skal selv kunne lage ruter og opprette egne severdigheter. Målgruppen for Guided Tour er først og fremst nyinnflyttede, men den kan også være interessant for mennesker som har bodd lenge i byen, og har et ønske om å bli bedre kjent og lære mer om byen de bor i.

1.1.1 AndMark Software Development

AndMark Software Development ble grunnlagt i mai 2014 av Anders Hagebakken og Markus Brovold. Begge studerer Master i Applied Computer Science ved Høgskolen i Gjøvik. AndMark står bak Android og iOS -appen Stolpejakten, med tilhørende database og nettside, som de fikk stor suksess med etter sin Bacheloroppgave. AndMark Software Development utvikler skreddersydde, mobile applikasjoner for iOS og Android. I tillegg utvikler de nettsider både fra bunnen av eller nettsider som kan integreres med eksisterende løsninger.

1.2. Prosjektmål

1.2.1 Effektmål

- Oppdragsgiver får økt brukergruppe.
- Mulighet for økt funksjonalitet i dagens Stolpejakt.
- 500 installasjoner i løpet av 2015

1.2.2 Resultatmål

- Utvikle en Android-applikasjon under konseptet Guided Tour med brukergenerert innhold.
- Applikasjonen lar brukere finne severdigheter i sitt nærområde og sette opp ruter/turer basert på disse. Brukeren skal også kunne benytte seg av forhåndsgenererte ruter.
- Applikasjonen skal la brukerne opprette brukere og gjennom disse opprette og dele severdigheter og ruter/turer.
- Applikasjonen kan brukes aktivt over hele landet.
- Prosjektet skal ha en tilhørende nettside for informasjon og/eller utvidet funksjonalitet.
- Deler av applikasjonen kan integreres i Stolpejakten av AndMark.
- Applikasjonen skal fungere for Android 4.0 og oppover.

1.2.3 Læringsmål

- Få erfaring med å jobbe sammen med en oppdragsgiver
- Bruke allerede tilegnet kunnskap i planlegging og gjennomføring av prosjektet
- Få erfaring med å jobbe med prosjekt
- Få kjennskap til arbeidsfordeling og det å ha forskjellige ansvarsroller
- Få større faglig kompetanse innen mobilutvikling for Android
- Få større faglig kompetanse innen brukervennlig mobildesign

1.3. Rammer

1.3.1 Tidsrammer
Rammene for prosjektet er satt av Høgskolen i Gjøvik.
Levering av prosjektavtale i Fronter innen 28.01.15 kl 23:59
Levering av prosjektplanen på mail til veileder innen 02.02.15 kl 23:59
Levering av bacheloroppgave med vedlegg i Fronter innen 15.05.15 kl 12:00
Levering av individuelt refleksjonsnotat innen 22.05.15 kl 23:59
Levering av A3 plakat til utstilling innen 22.05.15 kl. 23:59
Presentasjon av oppgaven vil skje i løpet av uke 23 (1.-5. juni 2015)

1.3.2 Verktøyrammer
- Vi vil utvikle applikasjonen i IDEet Android Studio utviklet av Google. Dette har vi valgt fordi det er et verktøy alle gruppemedlemmene har gode erfaringer med.
- For versjonskontroll har vi valgt å bruke Git med repository på Bitbucket.org.
- Siden vi har valgt Scrum som utviklingsmodell så er det naturlig for oss å bruke Trello som Scrum-board for å organisere oppgavene som gjøres i prosjektet.
- For å sikre god kvalitet på koden så vil vi bruke Lint-report funksjonaliteten i Android Studio.
- Rapporter vil bli skrevet i delte Google Docs dokumenter som alle i gruppen har tilgang til.
- Prosjektets nettside vil bli opprettet ved hjelp av CMS-løsningen Wordpress.
- Grafikk som skal brukes på nettsiden eller i applikasjonen vil bli utarbeidet i Adobe Photoshop.
- Prosjektrapporter settes opp i Adobe InDesign.
- For å lage modeller til å illustrere prosjektets oppbygning eller struktur vil vi bruke Microsoft Visio.
- Eventuell nettside for applikasjonen vil bli utviklet i Adobe Dreamweaver.
- Databasen vil ligge på et felles live server som alle har tilgang til via phpMyAdmin og scripts via FTP.

2. OMFANG
2.1 Fagområde
Utvikling av applikasjonen vil være for Android, med fokus på brukervennlige løsninger. Backend-løsningen vil utvikles i PHP og database i MySQL.

2.2 Avgrensning
Prosjektet avgrenses til å kun gjelde utvikling for Android. Utvikling av en iOS applikasjon i tillegg er ikke en del av prosjektet.

2.3 Oppgavebeskrivelse/avgrensning

Det skal utvikles en applikasjon for Android hvor hovedkonseptet er Guided Tour. Navnet på applikasjonen bestemmes ved et senere tidspunkt. I samarbeid med oppdragsgiver er det blitt satt en del ønsker og krav til applikasjonen. Først og fremst skal brukerne kunne lage sin egen rute ved å plukke ut severdigheter/attraksjoner fra et kart. Brukere skal også kunne legge inn egne attraksjoner og egne severdigheter. Det skal også finnes offentlige ruter, som er tilgjengelige for alle. Disse rutene skal kunne rates, og brukere kan stemme ruter og severdigheter opp eller ned. Severdighetene skal kategoriseres og det skal være mulig å velge hvilken type severdighet man vil se. Aktuelle kategorier er i

utgangspunktet utkikkspunkt, historie, kultur, sport. Brukerne skal også kunne velge ruter som er egnet for sykling, gåing, bevegelseshemmede og eventuelt bil.

Andre elementer det kan være aktuelt å ha med i applikasjonen er skritteller, mulighet for å se bilder fra attraksjonene/severdighetene, mulighet til å se video/høre lyd. Brukere skal kunne gå en egen rute og legge inn severdigheter underveis (med bilder og tilhørende tekst), og etterpå dele denne ruten med andre brukere, sammen med informasjon om ruten. Det kan også være aktuelt å la en bruker gå fritt rundt uten å følge en gitt rute. Det vil da komme varsler på telefonen om at bruker befinner seg i nærheten av en attraksjon. Det åpnes for å legge til en funksjon for å la brukere se hva som skjer i nærheten i tiden fremover.

Grunnelementene som appen skal inneholde er satt. Det er åpent for å legge til nye elementer og videreutvikle ideen som allerede er gitt. Alle elementer er satt i prioritert rekkefølge, basert på viktighet og funksjonalitet (se punkt. 6.2).

Det skal også utvikles en nettside som tilhører applikasjonen. Denne siden trenger ikke inneholde så veldig mye mer enn informasjon om applikasjonen, men kan utvikles til å fungere som en administrasjonsløsning for utviklere og/eller brukere.

Oppdragsgiver har et ønske om at denne appen skal være veldig norsk, den skal være koselig og lokal og appelere til nasjonalromantikken hos brukerne. Stikkord fra oppdragsgiver er bunad, rosemaling og lokal historie.

3. PROSJEKTORGANISERING

3.1. Ansvarsforhold og roller

Kunde
Kunden for dette prosjektet er AndMark Software Development AS. Gruppens kontaktperson hos kunden vil være Anders Hagebakken og Markus Brovold.

Prosjektleder
Rollen som prosjektleder er blitt tildelt Trine Storsveen.

Veileder
Gruppens veileder vil være professor Ivar Farup ved HiG.

Kontaktperson
Gruppens kontaktperson mot kunden vil være Trine Storsveen, og all formell kontakt mellom gruppen og kunden skal gå via henne. Trine Storsveen er også gruppens kontaktperson i forhold til veileder Ivar Farup, og all annen ekstern kontakt utover dette.

Webmaster
Pål Storsveen er ansvarlig for at prosjektet har en fungerende nettside. Ansvarsområdet inkluderer innholdet på siden og å oppdatere nettsiden med hensiktsmessing informasjon i løpet av utviklingsløpet. De andre gruppemedlemmene skal også ha mulighet til å legge ut innhold på nettsiden.

3.3 Rutiner og regler i gruppen

Gruppen har blitt enige om og signert et sett med regler for hvordan vi skal gjennomføre dette prosjektet. Se punkt 7.2.

4. PLANLEGGING, OPPFØLGING OG RAPPORTERING

4.1 Valg av utviklingsmodell

Vi har valgt i dette prosjektet å bruke Scrum som utviklingsmodell. Vi vil benytte 2 ukers sprinter og dermed ha korte iterasjoner og raske leveranser. Grunnen til at vi har valgt å bruke 2-ukers sprinter er at prosjektet består av mange små moduler som vil passe godt å ferdiggjøre innenfor én sprint. Prosjektet består også av noen større moduler som vil gå over flere sprinter, men vi tenker da at vi deler dem inn i innlednings-fase, utviklings-fase og test-fase som vil passe til sprintene.

Grunnen til at vi har valgt Scrum som utviklingsmodell er at prosjektet mest sannsynlig vil endre seg i løpet av prosjektperioden og Scrum er tilpasset det. Som nevnt tidligere vil også sprintene passe bra til de enkelte modulene som prosjektet består av. Vi ønsker også å bruke Scrum fordi det er den mest brukte system-utviklingsmodellen i arbeidslivet og vi ønsker at denne oppgaven skal skape en realistisk setting på hvordan vi vil arbeide etter ferdig utdanning.

Gruppen har scrummøter annenhver dag kl 09.00 (mandag-onsdag-fredag), i tillegg til et fast gruppemøte på tirsdager. De dagene vi ikke sitter sammen og jobber, foregår disse møtene i en egen chattegruppe på facebook.

4.2. Planlagt møtevirksomhet

- Faste gruppemøter vil være hver tirsdag. Tidspunkt og sted avtales fortløpende.
- Veiledningsmøter vil foregå annenhver onsdag kl. 09.15, med mulighet for å bytte til 3. hver uke etter behov.
- Møte med oppdragsgiver vil foregå etter behov, og i forbindelse med sprinter.
- Daily scrum vil foregå digitalt mandag, onsdag og fredager i avtalt forum.

Vi vil også planlegge noen tidspunkter i prosjektperioden hvor vi oppdaterer prosjektets hjemmeside slik at studenter og andre interesserte kan holde seg oppdatert på utviklingen.

5. Organisering av kvalitetssikring

5.1. Kritiske suksessfaktorer

Moduler må fungere tilfredstillende innen satt tidsfrist.

5.2. Risikoanalyse

Område	Beskrivelse	Innvirkning	Sannsynlighet*	Betydning**
Team	Tap av nøkkelperson	Tap av verdifull tid	Lav	Svært Kritisk
Prosjekt	Ikke god nok planlegging	Prosjektet vil ikke kunne bli utviklet slik det var planlagt	Middels	Kritisk
Prosjekt	Tap av data	Arbeid må gjøres om igjen	Lav	Kritisk
Team	Utilstrekkelig	Tap av tid til	Middels	Kritisk

	kompetanse hos utviklere	kompetanseheving/ manglende funksjonalitet i sluttprodukt		
Prosjekt	Feil i tidsestimater	Manglende funksjonalitet i sluttproduktet	Middels	Kritisk
Krav	Endring i kravspesifikasjon	Tap av verdifull tid	Middels	Middels
Teknologi	Valg av løsninger fungerer ikke sammen med valgt SDK-versjon	Manglende funksjonalitet i sluttprodukt hos enkelte brukere	Middels	Middels
Teknologi	Utilgjengelig organiseringsverktøy	Tap av verdifull tid, økonomiske konsekvenser	Lav	Lav
Teknologi	Begrensninger i teknologivalg	Manglende funksjonalitet i sluttprodukt	Middels	Middels

**Rangering sannsynlighet: Lav, middels, høy, svært høy*
***Rangering betydning: Lav, middels, kritisk, svært kritisk.*

Som vi kan se av tabellen over så er det flere punkter som vi må ta hensyn til. Vi kan ikke av praktiske årsaker ta 100% hensyn til alle, derfor er det disse punktene som vi skal fokusere mest på, og ta ekstra hensyn til i planleggingen.

1. Tap av nøkkelperson.
    - Vi vil her legge opp planleggingen slik at én person ikke besitter informasjon som er essensielt for ferdiggjøringen av prosjektet.
    - Vi vil også planlegge slik at minst 2 personer har full oversikt over moduler som er kjernefunksjonalitet. Slik vil vi kunne fortsette utviklingen selv om et gruppemedlem skulle bli syk midlertidig, eller frafalle.
2. Feil i tidsestimater.
    - Vi vil, for å formiske sannsynligheten og betydningen, ha mye fokus på kjernefunsjonalitet og sette av ekstra tid til særdeles viktige moduler.
    - Vi vil også jobbe med de viktigste komponentene i starten av prosjektperioden, og ekstrafunksjonalitet i slutten av prosjektperioden, slik at, hvis vi tar feil i tidsestimatene kan vi strekke tidsbruken på kjernefunksjonaliteten og eventuelt droppe noe av ekstra-funksjonaliteten.
3. Endring i kravspesifikasjon.
    - For å formiske betydningen av dette vil vi, i arkitekturen og planleggingen, legge til rette for endringer og utvidelser.

De punktene som ikke er tatt anser vi som mindre viktige, men vi vil likevel ta hensyn til dem i utviklingen.

5.3. Dokumentasjon, standardbruk og kildekode

Vi forholder oss til gjeldende kodekonvensjoner for Java, PHP, MySQL og eventuelle andre utviklingsspråk vi tar i bruk. Koden dokumenteres grundig og vi gjennomfører en code review av all kode før endelig godkjenning av moduler som utvikles. Lisensvalg er i samråd med oppdragsgiver satt til å være MIT-lisens.

6. GJENNOMFØRING

6.1 Gannt-skjema

	Start	Slutt	Jan 3	4	5	6	Feb 7	8	9	10	Mar 11	12	13	14	Apr 15	16	17	18	Mai 19	20	21	22	Jun 23	24	*Ukenr*
Sprint 1	**Jan-13**	**Jan-26**																							
Prosjektplan	Jan-13	Jan-26																							
Lage Webside	Jan-13	Jan-26																							
Undersøke teknologier	Jan-13	Jan-26																							
Møte med veileder	Jan-21	Jan-21																							
Lage gruppeavtale	Jan-13	Jan-19																							
Møte med oppdraggiver	Jan-15	Jan-15																							
Sprint 2	**Jan-27**	**Feb-9**																							
Lage kravspesifikasjon	Jan-27	Feb-9																							
Designe database	Jan-27	Feb-9																							
Utvikle server/back-end/database	Jan-27	Mar-23																							
Utvikle android/mobilapplikasjon/klient	Jan-27	Mar-23																							
Møte med oppdraggiver	Jan-27	Jan-27																							
Sprint 3	**Feb-10**	**23-Feb**																							
Møte med veileder	Feb-10	10-Feb																							
Møte med oppdraggiver	Feb-10	10-Feb																							
Sprint 4	**Feb-24**	**9-Mar**																							
Ferdiggjøre server/back-end/database	Feb-24	24-Feb																							
Oppdatere nettside	Mar-2	9-Mar																							
Sprint 5	**Mar-10**	**23-Mar**																							
Møte med veileder	Mar-10	10-Mar																							
Teste backend	Mar-17	23-Mar																							
Sprint 6	**Mar-24**	**6-Apr**																							
Vurdere utvikling/utvikle for iOS/win. phone	Mar-24	6-Apr																							
Teste server/back-end/database	Mar-24	6-Apr																							
Første testfase av android-applikasjon	Mar-24	30-Mar																							
Utbedre feil i android-applikasjon	Mar-31	13-Apr																							
Sprint 7	**Apr-7**	**20-Apr**																							
Møte med veileder	Apr-7	7-Apr																							
Ferdiggjøring av android-applikasjon	Apr-14	20-Apr																							
Sprint 8	**Apr-21**	**3-May**																							
Siste testfase av at alt fungerer	Apr-21	3-May																							
Sprint 9	**May-4**	**17-May**																							
Ferdiggjøre formatet/designet på rapporten	May-4	14-May																							
Levere rapport	May-14	15-May																							
Sprint 10	**May-18**	**31-May**																							
Forberede presentasjon	May-18	2-Jun																							
Lage plakat	May-18	31-May																							
Sprint 11	**Jun-1**	**14-Jun**																							
Presentere prosjekt	Jun-3	3-Jun																							

7. GRUPPEREGLER

Grupperegler

1. Prosjektleder

Prosjektleders ansvar er å kalle inn til møter, skrive møtereferat, ha kontroll på tidsfrister og fremdrift og være gruppens kontaktperson mot oppdragsgiver. Prosjektleder har vetorett ved "små" uenigheter, og har myndighet til å signere på gruppens vegne. Dersom resterende gruppemedlemmer ikke er fornøyd med prosjektleders innsats, og tilbakemeldinger om dette ikke tas til etterretning, skal det velges en ny prosjektleder. Trine Jeanette Storsveen er gruppens prosjektleder.

2. Uenigheter

Dersom det oppstår uenighet i gruppen, forsøker vi å løse dette ved å:

a. Ta en pause før vi tar opp igjen diskusjonen, eller setter et nytt, senere tidspunkt for diskusjonen
b. Det stemmes over uenighetene
c. Ved større uenigheter involveres veileder

3. Møtehyppighet/Arbeidstider

Vi har faste gruppemøter på tirsdager, hvor alle gruppedeltakere har møteplikt. Møter med veileder/oppdragsgiver avholdes fortrinnsvis på onsdager.

4. Oppgavefordeling

Vi fordeler oppgaver på hvert tirsdagsmøte. Med alle oppgaver følger en frist, hvis ikke annet er avtalt. Oppgavene legges inn i Trello-board. Denne fristen **skal** overholdes. Hvis en gruppedeltaker ser at fristen ikke kan overholdes, skal det gis beskjed i felles gruppeforum (fellesgruppe på facebook). Alle gruppemedlemmer er selv ansvarlig for å holde seg oppdatert på sine oppgaver, og på diskusjoner som foregår på forumet.

5. Sykdom/Fravær

Ved sykdom/annet fravær skal dette så snart som mulig gis beskjed om på felles forum, sammen med antatt bedringsdato.

6. Sanksjoner

Gruppemedlemmer som ikke overholder frister eller bryter gruppereglene, kan tildeles straffepoeng etter fastsatte regler. For hvert 5. poeng utdeles det også en advarsel.
1 poeng for hver oppgave som ikke er utført innen fristen (og man ikke har sagt fra om at man ikke rekker å gjøre oppgaven ferdig). Se på Trelloboard, spør i forumet dersom noe er uklart rundt oppgaven.
1 poeng for å ikke møte opp på fastsatte møter, uten å ha sagt fra på forhånd.

To medlemmer kan bli enige om en skjønnsmessig utdeling av poeng til et tredje gruppemedlem, ut over den poengfordeligen som er nevnt her.

7. Avskjedigelse fra gruppen

Ved oppnåelse av 5 straffepoeng vil gruppemedlemmet få en muntlig advarsel (denne dokumenteres skriftlig til gruppemedlemmet og veileder)
Ved oppnåelse av 10 straffepoeng, vil gruppemedlemmet få en skriftlig advarsel (med kopi til veileder).
Ved oppnåelse av 15 straffepoeng stå gruppemedlemmet i fare for å bli kastet ut av gruppen. Det avtales et ekstraordinært møte med veileder hvor denne problemstillingen diskuteres.

8. Tirsdagsmøter

Faste møter kl 10.00 hver tirsdag, på Lillehammer bibliotek (med mindre annet er avtalt).
Fast møteplan er som følger:

1. Gjennomgang av oppgaver vi fordelte på sist møte
2. Følge møteplanen vi satte opp forrige møte
3. Fordele oppgaver og sette møteplan for neste møte

Resten av tiden brukes til å jobbe sammen på oppgaver.

9. Utgifter

Eventuelle påbeløpte utgifter fordeles på alle gruppemedlemmer, med mindre annet er avtalt.

Trine J. Storsveen 21.01.15

Trine Jeanette Storsveen *Dato*

Dennis Gjerdingen 21.1.15

Dennis André Østvik Gjerdingen *Dato*

Pål S. 21/01-15

Pål A. Storsveen *Dato*