Høgskulen på Vestlandet

Fakultet for ingeniør- og naturvitskap

Institutt for datateknologi, elektroteknologi og realfag

Applikasjonsutvikling i Android for Smartere Transportløsninger

Høgskulen på Vestlandet

Fakultet for ingeniør- og naturvitskap

Institutt for datateknologi, elektroteknologi og realfag

TITTELSIDE FOR HOVEDPROSJEKT

Rapportens tittel: Applikasjonsutvikling i Android for smartere transportløsninger	*Dato:* *02-06-2020*
Forfatter(e): Anh Tuyet Celina Vu, Eivydas Stankauskas, Ove Henrik Draugsvoll	*Antall sider u/vedlegg: 42* *Antall sider vedlegg: 46*
Studieretning: Informasjonsteknologi /Dataingeniør	*Antall disketter/CD-er:*
Kontaktperson ved studieretning: Rogardt Heldal, Richard Kjepso	*Gradering:* (Velg: Ingen eller Begrenset til ddmmåå)
Merknader:	

Oppdragsgiver: Bergen Kommune	*Oppdragsgivers referanse:*
Oppdragsgivers kontaktperson: Tom Osnes Svellingen	*Telefon:* 97737588

Sammendrag:

Utvikling av en posisjonsbasert applikasjon som skal informere om og gjøre det enklere å velge kollektivtransport fremfor personbil. Laget i Android Studio med Google Maps SDK, Fused Location Provider API og ein SQLite database. Et nøkkelkomponent skal være kommunikasjon mellom brukere og utgiver.

Stikkord:

Android	Java	sql

Høgskulen på Vestlandet, Fakultet for ingeniør- og naturvitskap
Postadresse: Postboks 7030, 5020 BERGEN Besøksadresse: Inndalsveien 28, Bergen
Tlf. 55 58 75 00 Fax 55 58 77 90 E-post: post@hvl.no Hjemmeside: http://www.hvl.no

FORORD

Denne rapporten er skrevet av Anh Tuyet Celina Vu, Ove Henrik Draugsvoll og Eivydas Stankauskas, som er informasjonsteknologi og dataingeniør studenter ved Høgskulen på Vestlandet, Campus Bergen.

Oppgaven går ut på å lage en mobilapplikasjon for MUST, som skal være til hjelp for deres *living lab* med utføring av ulike pilotprosjekter.

Først og fremst vil vi takke Bergen Kommune for denne flotte muligheten til å jobbe med en spennende og meningsfylt oppgave. Videre vil vi spesielt takke Tom Osnes Svellingen og resten av MUST-teamet for deres bidrag, hjelp og innspill gjennom hele prosjektet.

Til slutt vil vi takke Rogardt Heldal og Richard Kjepso for deres veiledning, hjelp og forslag til prosjektet og oppgaveskrivingen.

INNHOLDSFORTEGNELSE

1 INNLEDNING

1.1 Motivasjon og mål

Per dags dato skjer ca 76% av all reising i Norge i personbil . Dette tallet har ikke endret seg noe særlig siden 1975, men personkilometer* kjørt i personbil har økt med 240% og antallet av registrerte personbiler i Norge har nesten tredoblet seg i samme perioden. Vi reiser mer enn noen gang før, mens våre reisevaner har ikke forandret seg de siste 45 år. (Statistisk sentralbyrå, 2020) Folk reiser generelt mye oftere i personbil enn i kollektive løsninger. Derfor har forbedring av mobilitets- og transportløsninger et stort potensiale for vårt miljø og samfunn, og det er akkurat dette som er hovedmålet til MUST. De jobber med å finne gode løsninger på dagens og morgendagens transport-utfordringer.

I et samarbeidsprosjekt mellom Hordaland fylkeskommune, Skyss, Bergen kommune og Statens Vegvesen etablertes MUST - Mobilitetslaboratorium for Utvikling av Smarte Transportløsninger i Bergen. MUST er i dag et pågående og aktivt prosjekt som har to laboratorium, *data-lab* og *innovation lab***.** De skal i nær fremtid også ha en *living lab,* og vår motivasjon med dette bachelorprosjektet er å støtte MUST, ved å hjelpe dem med deres living lab der de skal holde noen pilotprosjekter i forskjellige deler av Bergen by. Planen er å hjelpe MUST med å utvikle en applikasjon med et brukergrensesnitt som hjelper med å holde brukere oppdatert om aktiviteter og nyheter, samt at vi kan få direkte tilbakemeldinger fra dem om hva som kan forbedres eller gjøres annerledes. Vi vil også la dem dele informasjon knyttet til transport og trafikk med hverandre.

Målet til dette bachelorprosjektet er å lage en app-basert løsning som gir brukerne denne praktiske funksjonaliteten og informasjonen angående aktiviteter som pågår under MUST pilotprosjekter i nærområdet, og gi dem mulighet til å være med å prøve ut og forbedre både nye og nåværende løsninger.
Det er også viktig for oss at sluttbruker får en applikasjon med en god brukeropplevelse som tilbyr interessant data og funksjoner som gjør det enkelt og trivelig å velge kollektivtransport. Om prosjektet møter opp til forventningene så kan også forurensing minskes ved mindre bruk av personbil og områder med høyt trafikknivå kan bli forbedret når flere benytter seg av kollektive løsninger.

**1 personkilometer = 1 person transportert 1 kilometer. (Statistisk sentralbyrå, 2020, ordforklaring)*

1.2 Kontekst

MUST skal så snart som mulig ha en *living lab* der de har et stort behov for et praktisk grensesnitt for toveis kommunikasjon mellom MUST og brukere under pilotprosjekter. Dette skal gjøre det lettere for MUST å informere brukere og holde dem oppdatert om forskjellige aktiviteter og tilbud som finnes rundt dem, og for brukerne til å raskt nå nyttig informasjon om tilbud som finnes rundt dem og å gi tilbakemeldinger. Dermed valgte Bergen kommune å gi oss en bacheloroppgave med fokus på disse funksjonalitetene.

Prosjektperioden er antatt som en første omgang av utviklingen, derfor vil bare de viktigste funksjonene bli implementert i applikasjonen. Med hensyn på ressurser skal teamet levere inn en ferdig alfaversjon med disse funksjonalitetene og applikasjonen vil bli vurdert ut ifra hvor nyttig den vil bli for videreutvikling og senere bruk for oppdragsgiver.

1.3 Avgrensninger

Det tok en del tid før vi virkelig kunne gå inn for å arbeide med selve oppgaven, det var altså en god del forarbeid som måtte til. Dette var fordi oppgaven som vi fikk hadde ikke blitt særlig spesifisert og den var ganske åpen på de fleste områdene. Mye av tiden vår gikk derfor til spesifisering av oppgaven og resulterte da til at vi fikk mindre tid på selve prosjektet.

I applikasjonen vår vil det også være mulighet for å finne ut om hvor full/tom en transportenhet er i et bestemt tidspunkt, men grunnet mangel på Live-data så vil ikke denne funksjonaliteten være 100% nøyaktig. Dette gir derfor muligheter for videreutvikling av applikasjonen og forbedringer i ettertid av bacheloroppgaven.

1.4 Ressurser

Ressursene vi har er våre mobiltelefoner, PC/laptop med nedlastet Android Studio og data fra MUST (inkludert API-er). Vi må bruke flere API'er for å implementere diverse funksjonalitet som for eksempel å vise ledige bysykler i nærområde osv. I tillegg må vi jobbe sammen med

oppdragsgiver for å ta i bruk data-sjøen deres for å få ulike data som trengs for å utvikle applikasjonen.

Det er også mulighet for å bruke Høgskulen ved å spørre om hjelp fra førsteamanuenser i forbindelse med faget om utvikling av mobilapplikasjoner. Til slutt har vi internett og lærebøker, hvor vi burde finne alt vi kommer til å trenge.

1.5 Oppbygging av rapporten

Rapporten består av ni kapitler som er organisert i følgende rekkefølge:

Kapittel 1: En introduksjon til prosjektet ved fokus på presentasjon av mål og motivasjon til oppgaven, kontekst, avgrensninger og ressurser.

Kapittel 2: En detaljert beskrivelse av prosjektet, der vi går dypere inn om praktisk bakgrunn og problemstillingen.

Kapittel 3: Utformingen av prosjektet med vektlegging av forskjellige måter å løse problemstillingen og hvordan planleggingen av prosjektet vil bli utført.

Kapittel 4: Arkitekturen til de forskjellige stegene og applikasjonen som en helhet.

Kapittel 5: Evaluering med bruk av forskjellige metoder og gjennom forskjellige tester.

Kapittel 6: Diskusjon angående hvordan resultatet ble og framgangsmåten som ble brukt.

Kapittel 7: Konklusjon og videreutvikling.

Kapittel 8: Referanseliste.

Kapittel 9: Vedlegg som risikoliste og GANTT-diagram.

2 PROSJEKTBESKRIVELSE

2.1 Praktisk bakgrunn

Dette er første gangen MUST har samarbeidet med HVL for å skape en bacheloroppgave til studentene som tar faget DAT190 (*Bacheloroppgave-Data/informasjonsteknologi*). Oppgaven som ble introdusert bestod av mange muligheter og mye var opp til hva vi selv ønsket å arbeide med. Et av forslagene som ble nevnt var utvikling av en mobilapplikasjon og det var dette vi landet på. Siden vi har nylige kunnskaper innenfor Android fra faget DAT153 (*Mobile og distribuerte applikasjoner*), valgte vi å lage en Android mobilapplikasjon til MUST.

2.1.1 Prosjekteier

Prosjekteieren for vår bacheloroppgave er Bergen Kommune som har ansvar for MUST. Innenfor Bergen kommune så er det "Digitalisering og Innovasjons" konsernet som er den seksjonen som vi arbeider sammen med. Det er også dette konsernet som har det overordnede ansvaret for informasjons og kommunikasjonsteknologi i Bergen kommune. Digitalisering og innovasjons konsernet har også lokal i nærheten av bryggen i Bergen og deres adresse er Rosenkrantzgaten 3.

2.1.2 Tidligere arbeid

I dag kan man finne informasjon om MUST på Bergen Kommune sin hjemmeside eller på LinkedIn, men de har opptil nå ingen egen plattform hvor de f.eks informerer og legger ut sine planlagte pilotprosjekter.

Til tross for at MUST ikke har noen tidligere lignende applikasjon, så finnes det en mye brukt applikasjon som vil ha noen like trekk som oss. Dette er altså skyss applikasjonen. I dag blir skyss applikasjonen brukt mye for å vise oppdatert informasjon om når buss, bybane og båt går i Vestlandet, pluss muligheten for å planlegge reiser. Til tross for at begge applikasjonene vil ha en del ting til felles, så vil det likevel være funksjoner som skiller applikasjonene fra hverandre.

2.1.3 Initielle krav

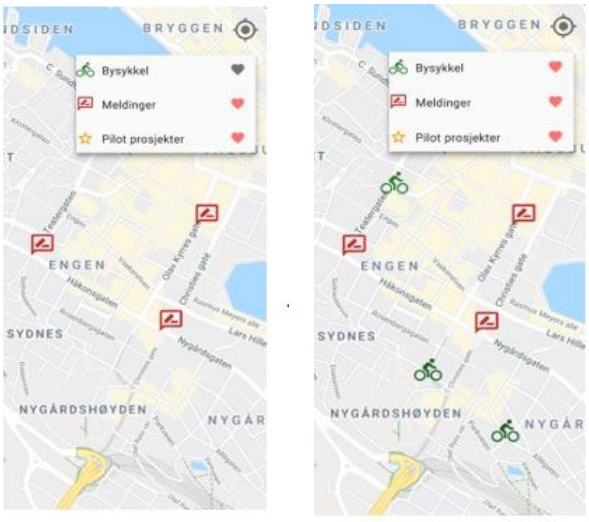
Oppdragsgiver hadde ingen konkrete krav til å starte med, men mange åpne ideer og forslag til oss. De var altså ganske fleksible når det gjaldt oppgaven. De ønsket stort sett at vi skulle fokusere

på det med å hente data eller å vise data i forbindelse med mobilitet og transport. Underveis utviklet dette seg og de ønsket å ha fokus på pilotprosjektene, der det skulle være mulighet for å få vist informasjon om de ulike pilotene i nærheten og samtidig få tilbakemelding fra brukerne angående disse pilotene. I tillegg var det ønskelig å ha en nettside fra MUST på selve hovedskjermen på applikasjonen som de selv skulle opprette og oppdatere.

2.1.4 Initiell løsnings-idé

Etter diskusjoner, tanker og ideer ble vi endelig enige om en potensiell løsning med oppdragsgiver. Vi skulle altså utvikle en Android applikasjon som skulle bestå av flere ulike funksjonaliteter. Først og fremst skulle vi ha en hovedside som består av informasjon angående MUST som skulle bli oppdatert gjennom web av oppdragsgiver. Deretter skulle vi ha en statistikk funksjon som fortalte brukerne om hvor full/tom en transportenhet er på et bestemt tidspunkt. Videre skulle vi også ha med en tilbakemeldingsfunksjon der brukerne kunne velge å sende tilbakemeldinger angående pilotprosjektene eller feilmeldinger relatert til veier, holdeplasser, transportenheter osv. Vi skal også ha en funksjon som vil bestå av trafikkvarsler, hvor brukere kan sende inn trafikkvarsler for å spre informasjon om trafikkflyten direkte til andre brukere og samtidig se trafikk varslene som andre har sendt inn. Til slutt skulle vi ha med en kartfunksjon som vil vise brukerne pilotprosjektene, bysyklene og innsendte varsler i nærheten. I tillegg har det vært prat om å inkludere nyhetsvarsler i applikasjonen, hvor varsler kan komme opp på hovedsiden av applikasjonen.

Tidlige digitale skisser:



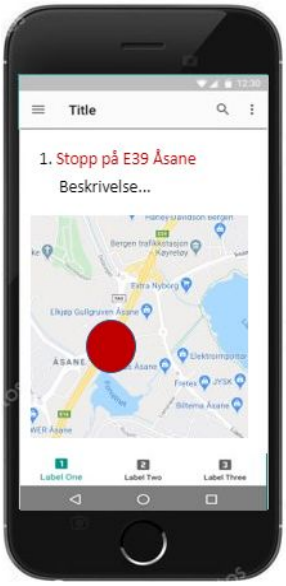
Skisse 1 & 2: Viser en avkrysningsboks der man kan velge mellom fremvisning av pilotprosjekter, bysykkel og varsler på kartet.

Skisse 3: Viser hva som skjer når brukerne trykker på en av de meldings-ikonene eller elementene på kartet. Det vil da dukke opp en infoboks som forklarer nærmere, som fremstilt i skissen.

Skisse 4: Viser hvordan feilmeldings-siden ser ut. Dette er altså det brukerne ser når de ønsker å rapportere feil/skade. Brukerne blir da omdirigert videre til en skjema-side ved å trykke på en av knappene.

Skisse 5: Viser hvordan tilbakemelding-siden vil se ut. For å sende inn tilbakemelding inn til MUST må brukeren fylle inn tittel, beskrivelse, velge område og legge til bilde om ønskelig

Skisse 6 : Viser hvordan hovedskjermen skal se ut. Den skal altså vise en nettside om MUST og deres nyheter og hvordan den ville sett ut med en varsling øverst på toppen av skjermbildet.



Skisse 7: Viser trafikkvarsel. som informerer hva som er årsaken, beskriver hendelsen og hvor på kartet hendelsen foregikk.

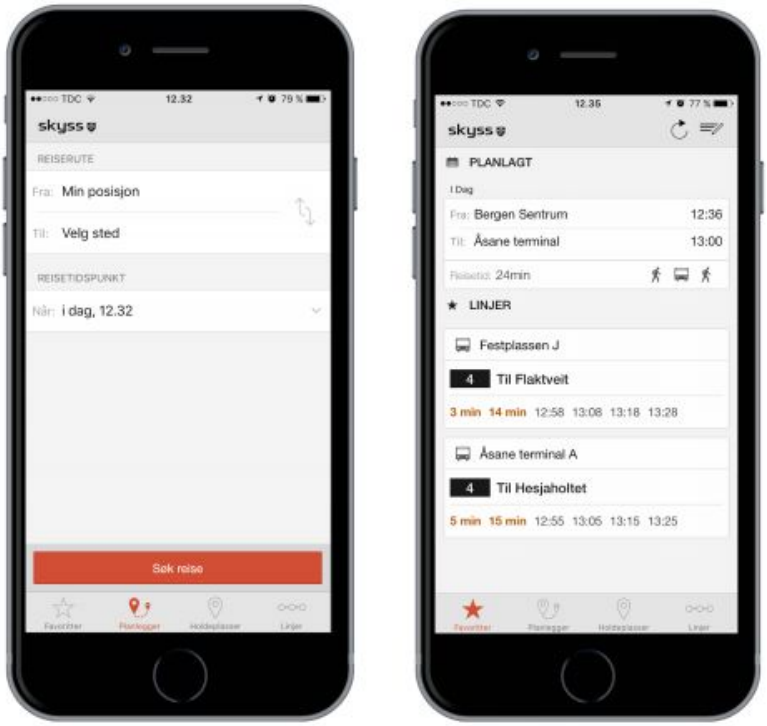
2.2 Litteratur om problemstilling

Som nevnt tidligere har vår applikasjon en del til felles med den allerede eksisterende applikasjonen “skyss reise”, hvor brukere kan planlegge/finne en reise og få sanntidsinformasjon over de forskjellige avgangene.

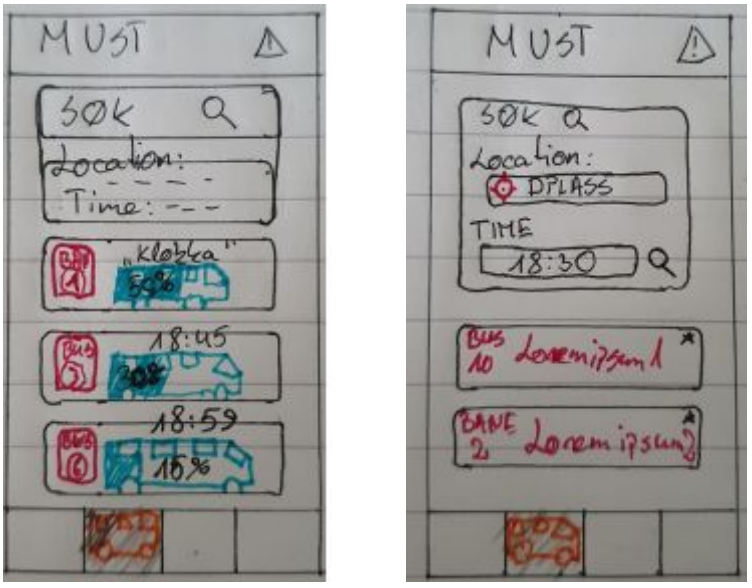
Skyss reise er i dag tilgjengelig både for iOS-brukere og Android brukere, men vår applikasjon vil bare være på Android-plattformen i første omgang.

Skyss Reise ble ikke lansert før i 2014. Det var en etterspørsel og et ønske om å gjøre terskelen for kollektivtransport lavere (Colliander, 2014). Det initielle formålet med applikasjonen var å gi sanntidsinformasjon til brukerne over de ulike transport-enhetene på forhånd, uten å måtte være tilstede i holdeplassene. Før denne applikasjonen ble lansert hadde skyss også en annen applikasjon ved navnet "Skyss Reiseplanlegger"; som omhandlet planlegging av reiser. Skyss Reiseplanlegger ble senere i desember 2016 mindre etterspurt, og anbefalt fjernet, da dens funksjoner var blitt inkludert i Skyss Reise applikasjonen. Man hadde derfor ikke bruk for en egen reiseplanlegger applikasjonen lenger. Dermed er det slik at dagens "Skyss Reise" er en videreutvikling av den initielle applikasjonen som i første omgang bare handlet om visning av sanntidsdata, men ble senere utviklet og implementert med planleggings-funksjonen (Skyss, 2016).

Likhetene mellom vår og Skyss sin applikasjon er at de begge inneholder reiseplanlegger funksjonen som gjør det mulig å finne en bestemt avgang for reise. I tillegg til at begge appene kan vise frem hvor det er bysykler på kartet, men på Skyss sin versjon var det vanskeligere å finne frem til dette og vår applikasjonen vil derfor vise dette på en mer selvforklarende måte. Forskjellen er da at vår applikasjon vil ha pilotprosjektene som hovedfokus, med et grensesnitt som skal fungere som en toveis kommunikasjon mellom brukerne og MUST. Det vil altså si at applikasjonen vår vil legge mer vekt på andre funksjonaliteter som vil være nyttig for MUST.



Bilde 2.1: Skjermbilde av skyss reise sin applikasjon og dens funksjonalitet



Bilde 2.2: Skjermbilde av håndtegnede skisser av reiseplanlegger funksjonaliteten med statistikk

3 DESIGN AV PROSJEKTET

3.1 Forslag til løsning

3.1.1 Web Applikasjon

Webapplikasjon er et alternativ som hadde hjulpet oss å lage en løsning som er tilgjengelig på flest antall enheter. Dette hadde inkludert bruk av HTML5, JavaScript, Angular/React som ville ha gjort applikasjonen tilgjengelig for alle enheter som kan ta i bruk nettlesere (PC, laptop, nettbrett etc.)

3.1.2 Cross Platform

Cross platform utvikling er et alternativ som lar utviklere skrive en kode som er kjørbar på flere operativsystemer. I vårt tilfelle hadde det hjulpet med å utvikle applikasjonen for både iOS og Android samtidig. Det finnes en del forskjellige løsninger og rammeverk til dette b.a. Apache Cordova og React Native. Dette hadde inkludert Apache Cordova/Phonegap/React Native, HTML5 og JavaScript.

3.1.3 Android Native

Android Native er en applikasjon som kan kun kjøres på Android operativsystem. Android er det mest populære operativsystem for mobiltelefoner og nettbretter. Dette er et vel beskrevet og dokumentert system som har mange libraries som vi kan ta i bruk. Dette hadde inkludert Java eller Kotlin og Android Studio Libraries.

3.1.4 iOS

iOS er *Apple* sitt operativsystem som brukes på *iPhones*, og tidligere *iPads*. Dette var en mindre attraktiv løsning siden vi ikke er kjent med Swift programmeringsspråk og iOS utvikling, og det hadde krevd en del ekstra tid. Swift utviklingsmiljø er laget for MacOS som kun kan kjøres på *Apple* sine datamaskiner, noe vi ikke hadde tilgang til. Dette hadde inkludert Swift og dets biblioteker.

3.1.5 Diskusjon av alternativene

Før vi faktisk bestemte oss for en Android applikasjon så ble alle alternativene nevnt ovenfor tenkt godt igjennom.
Vi vurderte webapplikasjon siden dette hadde gjort tjenesten tilgjengelig på flere enheter enn bare mobiltelefoner og nettbrett. Utviklingen ville sannsynligvis ha gått raskere og, men oppdragsgiver ville ha en løsning som hadde mulighet for innhenting av brukernes data som posisjon og reisemønster, noe som var begrenset i en webapplikasjon.

iOS var noe vi kunne tenke oss siden det er mange iPhone brukere, spesielt her i Norge, men dette skapte ekstra problemer og risikoer siden ingen av oss hadde tilgang til MacBook, Xcode eller SwiftUI. Vi var heller ikke kjent med Swift programmeringsspråk og dette hadde ført til ekstra tidsforbruk, derfor valgte vi å ikke gå for dette alternativet. Den mest ideelle løsningen hadde gjerne vært cross-platform utvikling som ville ha inkludert både Android og iOS enheter, men det ga det samme problemet hvor vi hadde mangel på kunnskap og erfaring på dette området. Det ble regnet som en unødvendig risiko å utvikle produktet på en ukjent platform.

Android, som er alternativet vi gikk for, er det operativsystemet som har flest brukere. Dette er også systemet som vi er best kjent med, som lar oss hoppe rett inn og fokusere på produktet uten og bruke alt for mye tid og ressurser på å lese dokumentasjon og lære oss et helt nytt språk. Dette gir oss en raskere utviklingsfase og implementasjon av flere funksjoner.

3.2 Valgt løsning

Grunnen til at vi valgte Android Studio og Java er som nevnt kjennskap til programvare, språket og effektiviteten som kom med det. Muligheten til å sette i gang utviklingen fortest mulig uten å bruke for mye tid på å lese oss opp på dokumentasjon og om forskjellige verktøy på en ukjent plattform.

Siden alle på gruppen er kjent med både Android Studio, Java og forskjellige Android Libraries, så fører dette til best mulig tidsforbruk og det øker effektiviteten. Siden første utgave av produktet er kun ment til et begrenset antall brukere som skal vere test gruppen vår, så lar dette oss å fokusere på implementasjon av nødvendige funksjoner og design etter det.

3.3 Valg av verktøy

Android Studio - Dette er det beste *Integrated Development Environment* (IDE) til Android applikasjon utvikling. IDE er en type programvare som brukes til å lage en annen programvare, og inkluderer en Debugger, automatisert Build verktøy og kildekode editor.

Android Libraries og APIer - Er forskjellige biblioteker til implementering av nødvendige funksjonaliteter.

- **WebView** - Visning av nettsider direkte i applikasjonen
- **Google Maps SDK for Android** - Bruk og visning av Google Maps direkte i applikasjonen med mulighet til å endre på utseende og funksjonalitet rett i Android Studio
- **Fused Location Provider API** - Bruk av Android sine posisjonsbaserte tjenester
- **Room Persistence Library -** Gir oss et abstraksjonsnivå over SQLite som gjør det lettere å jobbe med databaser.
- **Espresso -** Gjør det mulig å teste UI (User Interface) ved å simulere en person som bruker applikasjonen.

GitHub - Verdens største plattform for deling av prosjekter og kildekoden. Her kan vi synkronisere prosjektet fra flere brukere samtidig.

Java - Et programmeringsspråk vi er mest kjent med som kan bli brukt til Android applikasjonsutvikling.

JSON/XML/TypeScript/JavaScript - Forskjellige typer scripter, meta-språk og datalagringsløsning til innhenting og lagring av data som er relatert til offentleg transport og er tilgjengelig til oss for å bruke.

3.4 Prosjektmetodikk

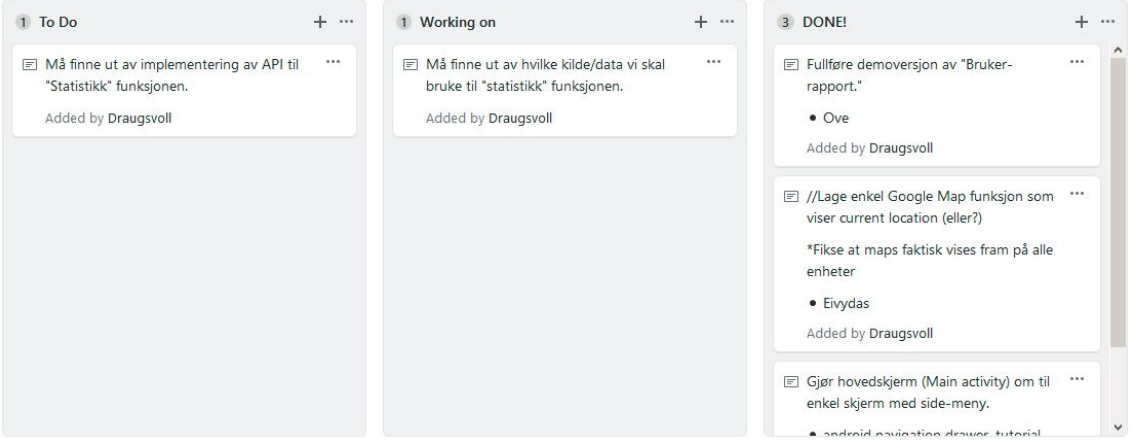
3.4.1 Utviklingsmetodikk

Vi bestemte oss for å bruke en agil utviklingsmetode som heter Scrum. Dette er en utviklingsmetode som er designet for et team mellom 3-9 personer, som passer veldig godt for oss. Scrum går ut på å dele prosjektet i flere tidsbegrenset iterasjoner, og at en iterasjon blir fullført før man går til den neste. En iterasjon er som regel en 2-4 ukers periode som kalles for en “sprint”.

I begynnelsen av hver sprint skal teamet lage en plan for sprinten med å plukke oppgaver fra en *backlog* som skal bli implementert gjennom sprinten. Til å holde styr over alle oppgaver og holde oss oppdatert skal teamet bruke github sin *projects* fane som vi kan bruke som et *scrum board* (Bilde 3.1) og "To do" kort. Dette skal gjøre det lettere for teamet å fordele spesifikke arbeidsoppgaver. Gjennom hele sprinten skal teamet ha daglige møter der vi går gjennom hva som ble gjort dagen før, fordele nye oppgaver for dagen og diskutere hva vi står igjen med og eventuelle problemer som oppstår under utviklingen.
Med å bruke scrum board og "To do" kort, skal vi sørge for at hvert medlem av teamet er oppdatert på hva som må gjøres, hva andre medlemmer jobber med og hvilke oppgaver vi er ferdig med. Dette skal også hjelpe medlemmer til å ha en mer organisert arbeidsflyt der hvert medlem har en klar oversikt over status av prosjektet, og kan lett plukke opp nye oppgaver og jobbe med de.

Når en *sprint* er fullført, skal teamet holde et lenger møte der vi går grundig gjennom alt som ble gjort, problemer som ikke ble løst i gjennom sprinten, eventuelle løsningsforslag og forbedringer vi kan gjøre til neste sprint. Dette kan bety at noen oppgaver krever andre typer løsninger enn vi først hadde tenkt oss, eller ikke kan bli implementert i nåværende nivå og må derfor flyttes tilbake til backloggen og vil bli inkludert i en senere sprint.

Hovedideen med Scrum er at det er et iterativ og inkrementell rammeverk som er fleksibel. Målet med den er at utviklingsteam skal ha myndighet til å løse oppgavene slik de finner mest hensiktsmessig og at teamet ikke har en utnevnt leder, men at teamet selv planlegger iterasjoner og fordeler arbeidsoppgaver innenfor inkrementasjoner. Dette innebærer også at hver utvikler hjelper hverandre slik at vi ikke ender opp med enkeltpersoner som blir flaskehalser eller hindrer måloppnåelse. Dette gjør prosjektet mer effektivt og gir mulighet for forbedret kommunikasjon og samvirkning mellom utviklerer.



Bilde 3.1: Skjermbilde av Scrum board som blei brukt til første sprinten

4 DETALJERT DESIGN

Som nevnt tidligere, så var oppgaven såpass åpen i begynnelsen at planleggingen av designet kom noe tregt i gang. Dette førte også til noen forandringer underveis som gjorde at vi måtte gå tilbake til skissene og ta ting litt i fra begynnelsen igjen.

4.1 UI design

Siden starten så har målet til prosjektet vårt, vært å lage en alfa versjon av applikasjonen som skal testes i en pilotgruppe og utvikles videre basert på behov og tilbakemeldinger. Siden dette tok tid, så måtte vi prioritere å implementere nødvendige funksjonaliteter fremfor et komplett design for farger og elementer i appen.

Selv om vi ikke bestemte oss for et spesifikt UI design, og prosjektet heller ikke hadde en spesifikk målgruppe vi kunne anvende applikasjonen til, så har vi fått til et UI design som regnes som ren, klar og intuitiv, med å ha kun nødvendige ting på hver side/activity. Tanken bak brukergrensesnittet gjennom hele utviklingen var at alle funksjonene må være lett tilgjengelige og selvforklarende, noe som førte til en delvis bruk av *three-click rule*, som er en uformell regel brukt i webdesign og utforming. Regelen antyder at en bruker av et nettsted skal kunne finne all informasjon uten mer enn tre klikk. Denne regelen fungerte som en retningslinje for oss til å tenke på hva som er det viktigste å vise brukeren i hver skjerm og ikke skjule viktig informasjon og funksjoner under mange menyvalg. Vi håper at dette gjør applikasjonen intuitiv og brukervennlig for de fleste målgrupper.

Alle funksjoner ligger i en sidemeny rett på startskjermen. I starten av oppgaven vurderte gruppen å ha en meny nederst på skjermen (bottom navigation), noe som er synlig på tidligste håndskissene (Bilde 2.2). Dette ble endret på med tanke på at applikasjon vil etterhvert få flere funksjoner, og at en sidemeny gir applikasjonen et mer minimalt utseende samtidig som at alle funksjoner er fortsatt få klikk unna.

Siden denne applikasjonen er en fungerende prototype, så må UI testes med ekte brukere i forskjellige målgrupper og justeres etter tilbakemeldinger. Disse UI forandringene burde ta minimalt med tid i Android Studio på grunn av at i gjennom hele utviklingen ble det kun brukt *material design* komponenter som er innebygd i Android Studio. Komponenter kan ved behov lett forandres på i applikasjonen og det er mulig å legge til et felles globalt 'theme' til hele brukergrensesnitt.

4.2 Funksjonalitet

Hovedskjerm:
Nåværende hovedside er en nettside fra Bergen Kommune som informerer om MUST. Det skal etterhvert vises en egen nettside fra MUST som de selv oppdaterer og legger ut nyheter, pilotprosjekter osv.

Hovedmeny:
En meny som viser brukeren alt som er tilgjengelig i appen. Brukes til å navigere seg rundt på applikasjonen.

Trafikk-Varsler:
Bruker får opp trafikk-varsler. Her kan man enkelt gå gjennom flere varsler og legge til et nytt varsel selv. Ved å trykke på 'Vis I Kart' knappen så får man opp posisjonen til et varsel i kart.

Legg til varsel:
Enkel skjerm som bruker må fylle inn et nytt varsel med både posisjon og bilde.

Finn reise: Dette ble etterhvert bestemt som en enkel reiseplanlegger. Her skal vi også ha 'fyllingsgrad' funksjonen som forteller hvor full bussen er på søkt tidspunkt.

Kart:
Dette er et interaktivt kart som gir brukeren oversikt over forskjellige informasjon rundt dem. I første versjonen så viser kartet bysykler, pilotprosjekter fra MUST og trafikk varsler fra andre.

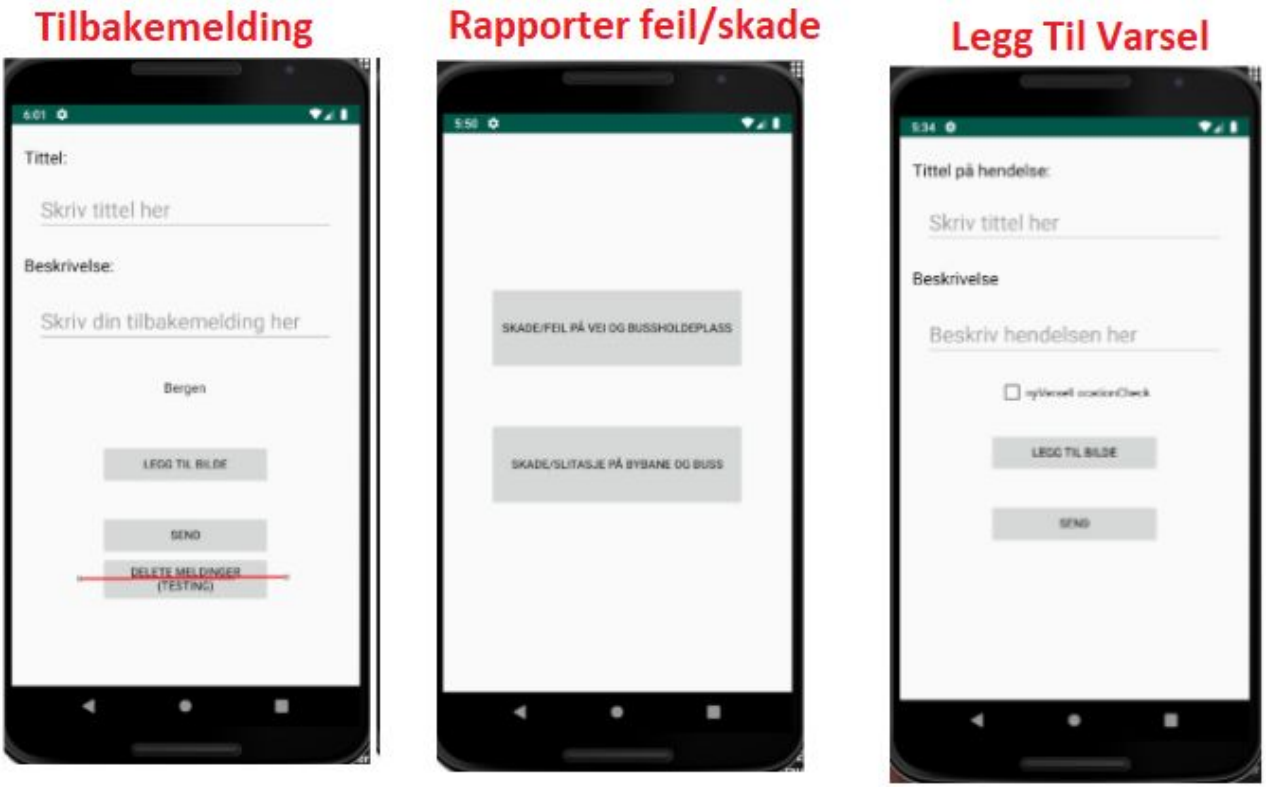
Kart funksjonen kan enkelt utvides med andre typer data som f.eks. bussholdeplass, billett terminaler og annen nyttig informasjon.

Tilbakemelding:

Her fylles inn en tilbakemelding inn til MUST. Vi valgte et veldig likt design som i 'Legg Til Varsel' for å gjøre brukeropplevelsen konsekvent.

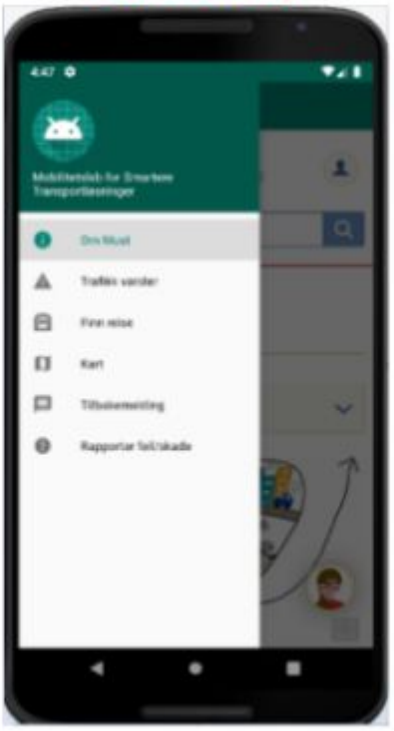
Rapporte feil/skade:

Her kan bruker velge mellom to kategorier, ettersom hva de vil rapportere. Begge knappene er en link som tar en videre til en relevant side med skjema.



Hovedskjerm

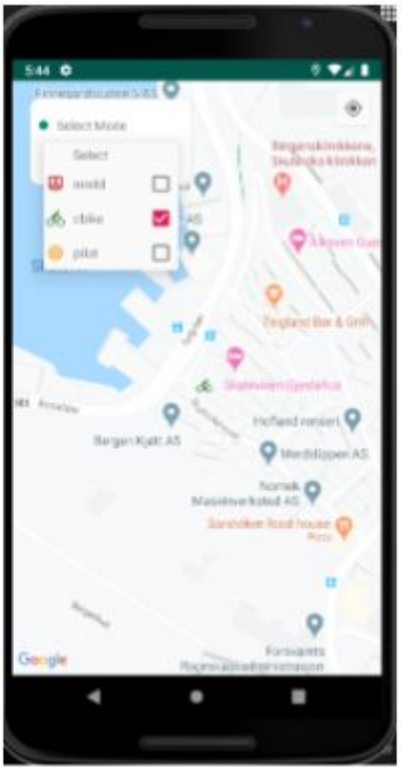
Hovedmeny



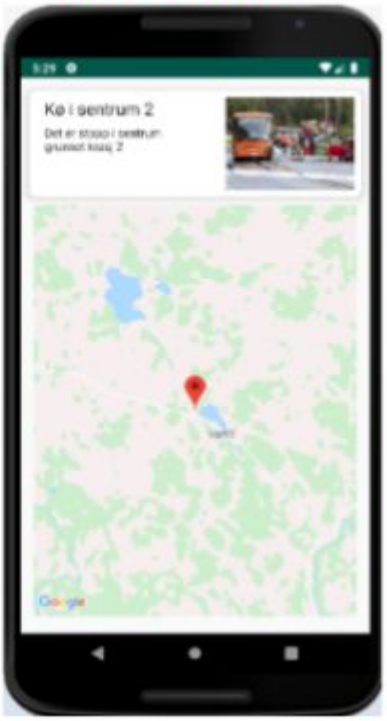
Trafikk Varsler

Finn Reise

Kart



Enkelt Varsel



4.3 Code design

Hovedskjerm:

Her bruker vi et fragment som inneholder et "WebView" for å vise frem en nettside. Dette er en funksjon i Android som gjør det lett å vise nettsider inne i appen.

```
public class Informasjon extends Fragment {

   public Informasjon() {
   }
   @SuppressLint("SetJavaScriptEnabled")
   @Nullable
   @Override
   public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container,
@Nullable Bundle savedInstanceState) {
       //Inflate layouten til fragment
       View v = inflater.inflate(R.layout.info, container, false);
       WebView webView = (WebView)v.findViewById(R.id.webView);
       webView.getSettings().setJavaScriptEnabled(true); //enable Javascript
       webView.setWebViewClient(new WebViewClient()); //viktig for å åpne url i appen
       webView.loadUrl("https://www.bergen.kommune.no/hvaskjer/tema/must"); //bytter ut vår
uri med linken
       return v;
   } // onCreate
```

Hovedmeny:

Vi valgte en Navigation Drawer meny, som er en type meny i Android Studio. Vi har brukt fragments for å vise hovedmenyen sammen med hovedskjermen. Kodesnutt viser hvordan menyen responderer med forskjellige cases, etter hva bruker klikker på.

```
/ NavigationDrawer
@Override
public boolean onNavigationItemSelected(@NonNull MenuItem item) {
   switch (item.getItemId()) {
       case R.id.info:
           getSupportFragmentManager().beginTransaction().replace(R.id.fragment_container,
new Informasjon()).commit();
           break;
       case R.id.kart:
       Intent intent = new Intent (MainActivity.this, MapsActivity.class);
       startActivity(intent);
           break;
       case R.id.tilbakemelding:
           Intent intent2 = new Intent (MainActivity.this, Tilbakemelding_Activity.class);
           startActivity(intent2);
           break;
```

```
        case R.id.feilmelding:
            Intent intent3 = new Intent (MainActivity.this, Feilmelding_Activity.class);
            startActivity(intent3);
            break;
        case R.id.trafikkVarsler:
            Intent intent4 = new Intent (MainActivity.this, TrafikkVarsel.class);
            startActivity(intent4);
            break;
        case R.id.reiseplanlegger:
            Intent intent5 = new Intent (MainActivity.this, TrafikkStatistikk.class);
            startActivity(intent5);
    }
```

Trafikk varsler:

En relativt enkel skjerm som inneholder en 'Recycler View'. Det er denne som gir oss en liste med data (trafikk-varsler fra databasen). En Recycler View henter data ved å kommunisere med et adapter som vi har laget. Dette adapteret henter data fra SQLite database. Den 'binder' data inne på listen i Recycler View. Når man trykker på 'Vis I Kart' så får man opp skjermen som viser på et kart hvor hendelsen tok plass. Kartet er et Android fragment som tar i bruk Maps SDK for Android og plasserer en markør ved gitte koordinater. Her vil man få opp hele tekst beskrivelsen, som kanskje var for lang til å passe i varsel boksen på forrige skjerm.x

Planen for SQLite Databasen:

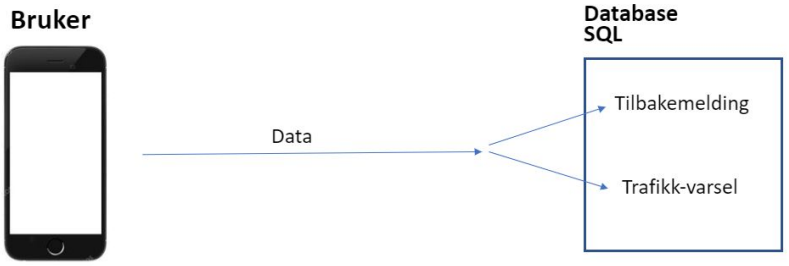
Database skisse

Vi har en database som lagrer både 'Tilbakemelding' og 'Trafikk-varsel'

Begge type data kommer fra en bruker. Når brukere sender data fra appen, så faller den innenfor en av disse kategoriene

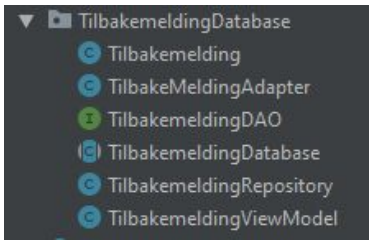
```
TABLE Tilbakemelding (
  melding_ID int NOT NULL PRIMARY KEY,
  Kategori ENUM('kategori1', 'kategori2'..),
  MeldingTittel varchar (255),
  meldingTekst varchar (255),
  Image blob
);
```

```
TABLE Trafikk-varsel (
  varsel_ID int NOT NULL PRIMARY KEY,
  VarselTittel varchar (255)
  varselTekst varchar (255),
  Latitude double,
  Longitude double,
  Image blob
);
```



Det ble noen små forandringer. Vi mangler lagring av bilder fordi det ble noen tekniske problemer når vi skulle gjøre det på riktig måte. Det vil si at vi ikke lagrer bilde direkte i databasen, fordi de

tar for stor plass. For å gjøre det riktig så lagrer vi bildene i en separat mappe, og databasen peker på hvor bildet ligger. Her skal det også tas i bruk komprimering av filen.



Figur 4.3.1:Mappe fordelingen av databasen i prosjektet

Vi brukte Room Persistent Library som hjelper med persistent lagring via SQLite, hvor vi slipper mye av selve SQL kodingen. Her lager vi et 'repository' som kommuniserer med dataene, og putter de videre inn i 'viewModel' som fungerer som en liste med LiveData. Det er altså viewModel som gjør det mulig for oss å 'observere' dataene i selve appen. Denne forsvinner når aktiviteten(skjermen) som 'observerer' listen ikke lenger er i bruk. Dette oppsettet gjør det enkelt å håndtere oppdateringer og forandring på databasen.

Nedenfor er et repository som komuniserer med databasen via en DAO(Data Access Object). Her ser vi metodene den tar i bruk, som er vanlige database operasjoner. Den bruker AsyncTask for å utføre operasjoner på databasen i bakgrunnen, og ikke på main thread (UI thread) på mobilen,siden det ville vært lite robust. AsyncTask Metodene er videre fylt ut lenger nede i klassen, men det ble litt mye kode å ta med.

```
public class VarselRepository {

    private VarselDAO varselDAO;
    private LiveData<List<Varsel>> allVarsler;

    public VarselRepository(Application application){
        VarselDatabase database = VarselDatabase.getInstance(application);
        varselDAO = database.varselDAO();
        allVarsler = varselDAO.getAllVarsler();
    }


    public void insert(Varsel varsel){
        new InsertVarselAsyncTask(varselDAO).execute(varsel);
    }
```

```
    public void update(Varsel varsel){
        new UpdateVarselAsyncTask(varselDAO).execute(varsel);
    }

    public void delete(Varsel varsel){
        new DeleteVarselAsyncTask(varselDAO).execute(varsel);
    }

    public void deleteAllVarsler(){
        new DeleteAllVarslerAsyncTask(varselDAO).execute();
    }

    public LiveData<List<Varsel>> getAllVarsler(){
        return allVarsler;
    }
```

Her er et eksempel på hvordan vi gjør java klassen (varsel objektet) om til en entitet i SQL via enkel bruk av annoteringer.

```
@Entity(tableName= "varsel_table")
public class Varsel {

    @PrimaryKey (autoGenerate = true)
    @NotNull
    int varsel_ID;
    String varselTittel = "";
    String varselTekst = "";
    double latitude;
    double longitude;


    // getterrs
    public double getLatitude(){ return latitude; }
    public double getLongitude(){
        return longitude;
    }
    public int getVarsel_ID(){
        return varsel_ID;
    }
    public String getVarselTittel() {
        return varselTittel;
    }
    public String getVarselTekst() {
        return varselTekst;
    }


    // constructor
    public Varsel(String varselTittel, String varselTekst, double latitude, double longitude){
        this.varselTittel =varselTittel;
```

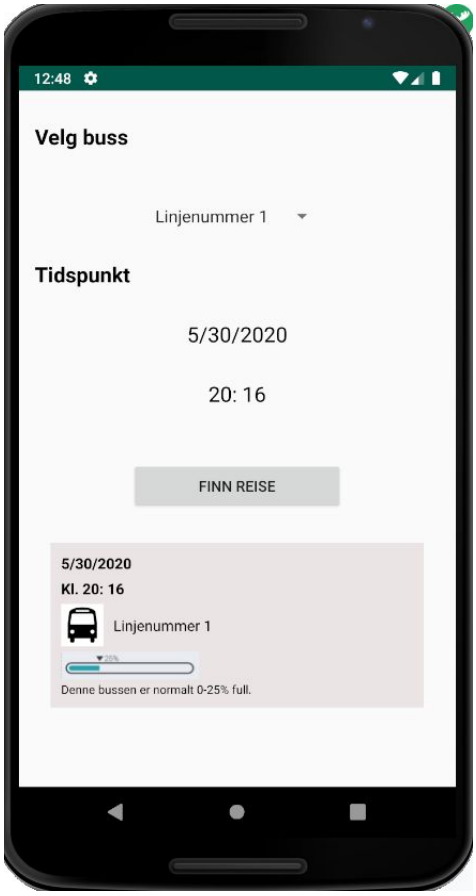
```
        this.varselTekst = varselTekst;
        this.latitude = latitude;
        this.longitude = longitude;
    }
```

Legg til varsel:

Her kan både bilde og brukers posisjon legges til varselet, men om begge må være nødvendig har vi ikke helt bestemt. Posisjon legges til ved å klikke på 'check-boxen'. Når bruker legger til bilde så kommer det også opp et lite preview av dette på skjermen. 'Send' knappen gjør at et nytt varsel objekt opprettes og legges til i 'varsel_table' i databasen.

For å legge til posisjon så kreves det 'permission' fra brukerens mobil.
Fused Location Provider API brukes til å hente brukerens posisjon og gjør det om til koordinater som kan da lagres i SQL database som to 'double' variabler.

Finn reise: Planen her var å få en API fra MUST som skulle ta imot inputs, for å vise fyllingsgrad på en valgt buss. En reiseplanlegger som viser alle tilgjengelige reiser skulle først ikke være med. Senere ble det ønskelig at en reiseplanlegger skulle kombineres med denne fyllingsgrad funksjonaliteten. På grunn av at planen ble endret underveis og det ble litt mye å kombinere disse sent i prosjektet, har vi dessverre ikke fått implementert disse. Vi har laget kode som simulerer det som var planlagt. Her velger man buss, dato og tidspunkt. Når man trykker 'Finn Reise' (Figur 4.3.2) så henter vi disse inputs'ene og viser frem en teoretisk tilgjengelig reise med fyllingsgrad.



Figur 4.3.2: Skjermbilde av foreløpige reiseplanlegger funksjonaliteten med statistikk

På grunn av Covid-19 situasjonen så ble MUST og deres data-team en del forsinket. Vårt første møte med deres data-teamet ble ikke arrangert før 19.mai. Selv om de hadde store mengde data, så viste det seg at de hadde problemer med å utvikle en API med all nødvendig funksjon på grunn av at ikke alle dataene var nøyaktige nok for dette.

Plan/skisse for API:

Input/output for statistikk API #2

API kan hente antall passasjerer på hvert enkelt stopp. Dette gir oss data med høy nøyaktighet.

Input:

- Linjenummer
- Dato
- Klokkeslett
- Påstignings-stopp
- Avstignings-stopp

Output:

- Algoritmen gir et tall f.eks mellom 1-100
- API'en gir f.eks 3tall tilbake. Et tall for de neste 3 tidsblokker.

Skisse/dokumentasjon på algoritmen: Se vedlegg i kapittel 9

Kart: Kart funksjonen implementerer Maps SDK* for Android og bruker Maps fragment til å vise kartet på skjermen og gir mulighet til å hente brukers posisjon.

For å hente informasjon om forskjellige steder som skal vises på kartet blir det brukt en JSON fil som blir tilgjengelig på brukers enhet. JSON står for *JavaScript Object Notation* og er et datautvekslingsformat som er basert på JavaScript men formateres kun som tekst. Objektene fra filen blir lest av koden og oversettes til Java Objekter med hjelp av Gson library, d.v.s. *void readDictionaryfromJson* metode leser igjennom *data.json* fil og oppretter Java Objekter ved hjelp av *PlacePojo* klasse.

Tanken bak dette var at JSON filer ikke tar opp mye plass og brukeren ikke trenger en konstant internett tilkobling og innsending av mange forskjellige *queries* til å få opp informasjon på skjermen. Filen kan da oppdateres med jevne intervaller eller ved oppstart rett fra SQL server.

```
void readDictionaryfromJson() {

   try {
       JSONArray object = new JSONArray(loadJSONFromAsset());
       Log.d("INDEX", "readDictionaryfromJson: "+object.length());
       for (int i = 0; i < object.length(); i++) {
           try {
```

```
                Gson gson = new Gson();
                PlacePojo pojo = gson.fromJson(object.get(i).toString(), PlacePojo.class);
                Log.d("JSON", "readDictionaryfromJson: Language  " + i + " " +
pojo.getVarselTitle());
                if(pojo!=null){
                    placePojos.add(pojo);
                }
```

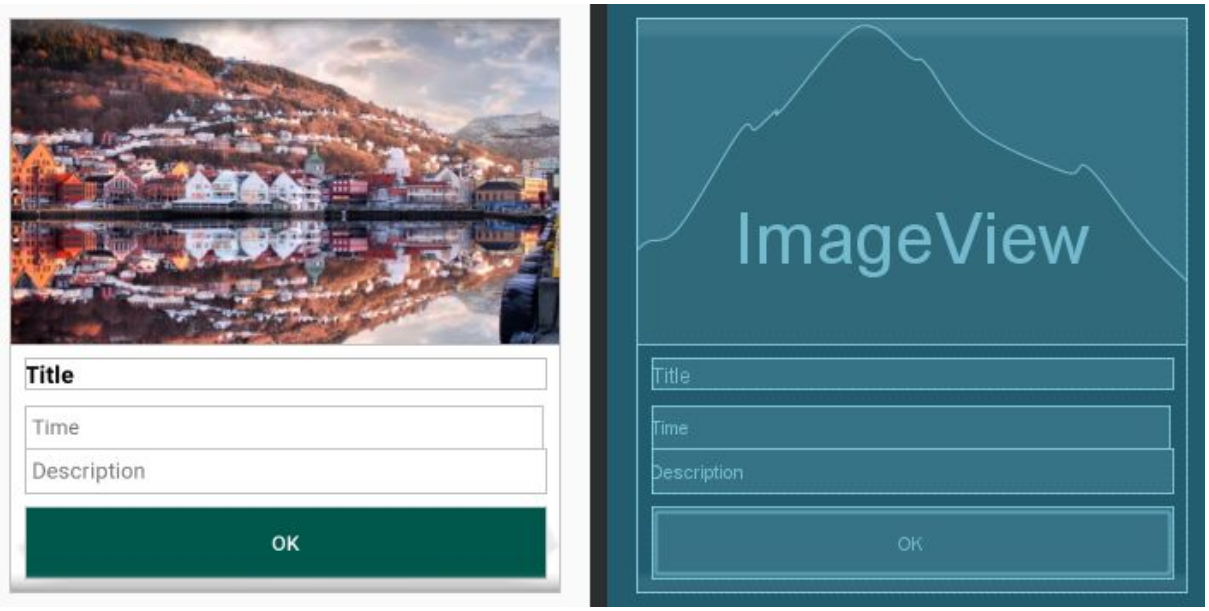
Objektene blir plassert på kartet ved å bruke *AddMarker* metode som sjekker etter varsel type og setter det på kartet med en tilsvarende ikonet ved bruk av koordinater.

```
void AddMarker(PlacePojo placePojo  , int position) {
  if(placePojo.getVarselType().equals("inmld")) {
      Drawable circleDrawable = getResources().getDrawable(R.drawable.inmld);
BitmapDescriptor        bitmapDescriptor = getMarkerIconFromDrawable(circleDrawable);

      mMap.addMarker(new MarkerOptions()
              .position(new LatLng(placePojo.getVarselLatit(), placePojo.getVarselLongt()))
              .title(placePojo.getVarselTitle())
              .snippet(placePojo.getVarselDescription())
              .icon(bitmapDescriptor).anchor(0,0)).setTag(position)
      ;
  }
```

Etter at brukeren trykker på et objekt ikon i kartet, en dialogboks kommer da opp og viser utvidet informasjon om valgte objekt, dette skjer ved hjelp av *public void showDialog(PlacePojo pojo)* metode som henter data fra valgt Java Objekten og plotter det i en *showDialog layout* (Figur 4.3.3)

**software development kit*



Figur 4.3.3: showDialog layout.

Tilbakemelding: I tillegg til meldingen har vi lagt til en 'Spinner' hvor man kan velge området som er relevant for tilbakemeldingen. Dette blir ikke brukt i første omgang men tenkes å ta i bruk hvis appen videreutvikles, siden appen egentlig kun gjelder for Bergen akkurat nå. Et bilde kan legges til om det vurderes som relevant. Når man trykker 'Send' så opprettes et tilbakemeldings-objekt som legges til i 'tilbakemelding_table' i databasen. Kode design og funksjonalitet er stort sett det samme som i Trafikk-Varsler funksjonen.

Rapporter feil/skade: Skade/slitasje på buss og bybane er relevant for Skyss, så bruker kommer til et utfyllingsskjema innpå Skyss sin nettside. Skade på vei og bussholdeplass er relevant for kommunen. Bruker sendes dermed til Bergens kommune nettside hvor en kan fylle ut et skjema. Dette trenger gjerne en alternativ løsning om appen skal videreutvikles og ta imot data fra mange andre områder. Når en av knappene klikkes på så tar vi i bruk et 'browser intent' så applikasjonen vet at vi vil vise nettsiden med en gitt URL.

5 Evaluering

Evaluering er et viktig trappetrinn innenfor prosjektet og vi bruker evaluering for å finne ut om vi er på riktig retning, og for å sikre oss at prosjektets mål blir møtt.

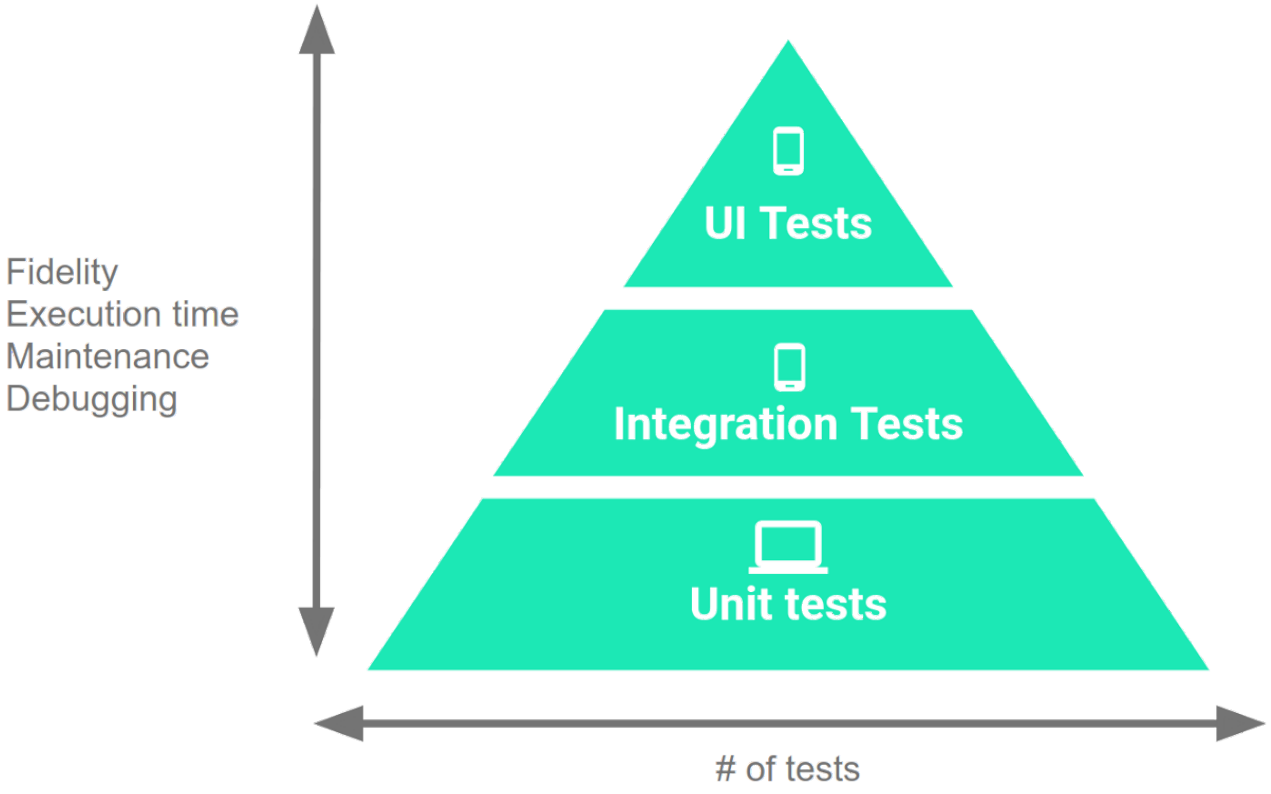
5.1 Evalueringsmetode

Målet for vårt bachelorprosjekt var å utvikle en alfaversjon av en applikasjon, som skulle være brukervennlig og nyttig både for brukerne og for oppdragsgiver.

For å sørge for at vi oppfyller oppdragsgiver sine krav, så har vi gjennomgang av de ulike funksjonalitetene og endringene som vi gjør underveis med dem. På denne måten kan vi finne ut av oppdragsgiverens tanker angående arbeidet så langt og deres innspill for endringer. Slik kan vi være sikre på at begge parter er enig og tenker likt i forbindelse med prosjektet. Det samme gjelder på slutten av hver sprint, hvor vi har et møte relatert til oppgaver utført sammen med oppdragsgiver og internt.

Når det gjelder tilbakemelding fra test-brukere av appen, så har vi hatt en utfordring. Vi tenkte at vi kunne f.eks utføre testing av brukere på høgskolen, hvor det er stort folketall og vi kunne sannsynligvis fått mange titalls test-brukere i løpet av en enkel dag. Dette gikk jo ikke på grunn av Covid-19 situasjonen. Ellers har tanken vært at vi kunne distribuere appen direkte til venner og bekjente via å dele APK-filen, altså installasjonsfilen som vil bli publisert på Google Play. Det var også en utfordring på grunn av at nøkkelfunksjonene som var essensielle for appen krevde mye forarbeid for å være ferdig. Vi har dermed fått til en testgruppe på ca 9-10 personer som har besvart på et skjema vi delte ut.

Når det gjelder testing så finnes det forskjellige typer av disse innenfor Android Studio. Testene blir delt inn i tre nivåer der den første er ***enhetstester (unit tests)*** som blir brukt for å teste en spesifikk del av koden uavhengig av resten av koden. Deretter finnes det ***integrasjonstester (integration tests)*** som tester grensesnittet mellom komponentene og sørger for at de fungerer som det skal når de er satt sammen. Til slutt har man ***brukergrensesnitt-tester (UI tests)*** som tester at de forskjellige komponentene i brukergrensesnittet fungerer som de skal.



Figur 5.1: Android testing pyramiden på engelsk

5.2 Evalueringsresultat

5.2.1 Evalueringsmøte

Vi hadde møte med oppdragsgiver på slutten av hver sprint og på det siste møtet så presenterte vi sluttresultatet. Dette møtet ble holdt gjennom Teams og applikasjonen ble presentert i form av en video av applikasjonen fordi emulatoren fungerte ikke like bra og oppdragsgiver hadde ikke en Android enhet i nærheten. Gjennom fremvisningen av applikasjonen vår med oppdragsgiver ønsket vi å få tilbake noen tilbakemeldinger og evalueringer på vårt sluttprodukt for å se om prosjektmålet ble møtt. Tilbakemeldingen som vi fikk fra oppdragsgiver var da at applikasjonen var grei å navigere seg rundt og at den fungerte bra som en app, og det var synd at vi ikke hadde fått tid til å implementere ferdig reiseplanlegger og statistikk funksjonen, selv om statistikk delen ikke var helt mulig grunnet mangel på ressurser. Videre ble det nevnt at dersom vi hadde hatt mer tid til prosjektet, hadde vi muligens fått på plass alle funksjonalitetene ved å bruke andre alternativer og ville da ha fått oppfylt alle ønskene fra MUST.

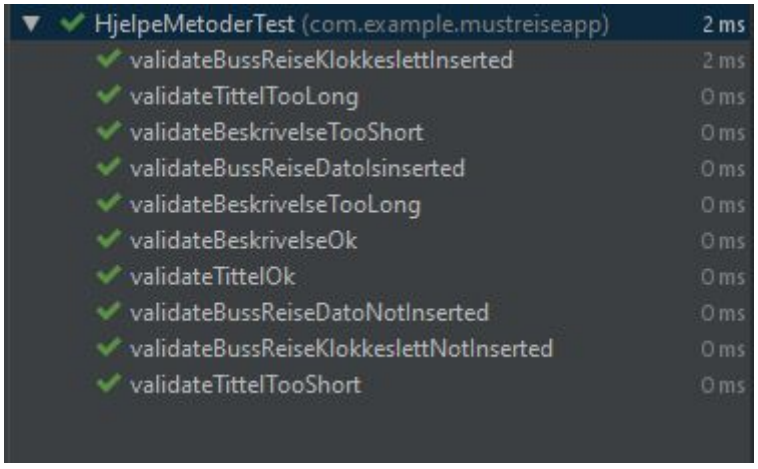
5.2.2 Undersøkelse av applikasjonen

Grunnet korona situasjonen vi er i dag ble applikasjonen testet blant en begrenset gruppe gjennom nett, med at vi delte en lenke som inneholdt APK-filen som gjør det mulig for Android-brukere å installere applikasjonen på deres mobil. Vi testet da brukeropplevelsen med at vi distribuerte apk-filen til bekjente som har Android mobiltelefon. I tillegg sendte vi med en link til en google-forms undersøkelse om brukeropplevelsen som testbrukerne kan svare på. Ut ifra resultatene fra undersøkelsen viste deg seg at brukerne synes at applikasjonen var grei og brukervennlig, men når det gjaldt om den var nyttig, var det derimot mye som kunne forbedres. Applikasjonen vil derfor videre også bli testet av en aktuell pilotgruppe (*test-gruppe)*.

Bilde 5.2.1: Screenshot av en del av undersøkelsen i Google-forms som ble gitt ut til testbrukerne

5.2.3 Unit og UI Testing

Vi brukte først og fremst Unit testing for å teste hjelpemetoder som validerer ulike inputs fra brukeren. Nedenfor vises et kodesnutt av noen av testene.



Figur:5.2.2: Skjermbilde av Unit testene som validerer inputs fra brukeren og kjører grønt..

```
public class HjelpeMetoderTest {


	private String testString="";
	private HjelpeMetoder hjelp = new HjelpeMetoder();


	@Test
```

```
        public void validateTittelTooShort() {
            testString = "asdvg";
            boolean godkjent = hjelp.validateTittel(testString);

            assertFalse(godkjent);
        }

}
```

Videre brukte vi noen UI tester, og her er blant annet et eksempel som sjekker at klikk på "Finn Reise" knappen henter og leverer inputs fra bruker på riktig måte.

Figur 5.2.3: Skjermbilde av forskjellige UI tester som godkjennes.

```
@RunWith(AndroidJUnit4.class)
public class UI_Test {


   @Rule
   public ActivityTestRule<TrafikkStatistikk> activity2 = new
ActivityTestRule<>(TrafikkStatistikk.class);


   @Test
   public void finnReiseMottarLinjenummerInput(){
       onView(withId(R.id.soekButton)).perform(click());
       onView(withId(R.id.textLinjeNr)).check(matches(withText("Linjenummer 1")));
   }

   @Test
   public void FinnReiseMottarDatoInput(){
       onView(withId(R.id.soekButton)).perform(click());
       onView(withId(R.id.textDato)).check(matches(withText("Velg Dato")));
   }

}
```

Her mangler man linjen:

```
onView(withId(R.id.etTittel)).perform(typeText("Test Tittel"));
```

som skriver inn tekst for oss. Ved kjøring av denne så fikk vi diverse errors på grunn av animasjoner og overganger i emulatoren som forvirrer og påvirker testresultatet.

5.2.4 Oppsummering av evalueringsresultatene

Evaluerings- resultatene bidrar til de overordnede målene med at vi først og fremst gjennom fremvisning av applikasjonen til oppdragsgiver, fikk tilbakemelding fra deres side og forståelse over hvilke krav vi hadde klart å møtt. Gjennom deres tilbakemelding fikk vi høre at vi vi hadde klart å oppfylle målet når det gjaldt implementering av funksjonalitetene som gjaldt rundt pilotprosjektene, men at vi også ikke hadde fått gjort applikasjonen komplett grunnet mangel av reiseplanlegger/statistikk funksjonalitet. Videre brukte vi undesøkelser på testbrukerne der resultatene var med på å bidra i å fortelle oss om vi hadde klart å oppfylle målene når det gjaldt brukeropplevelsen. Ut ifra undersøkelsen kom det tydelig frem at applikasjonen var enkel og grei å bruke, som vil si at vi har klart å nå målet angående brukervennlighet. Til slutt hadde vi med noen tester i koden som var med på å bidra til å sikre oss for at applikasjonen vår fungerte som den skulle. Testene fortalte oss da at koden og applikasjonen fungerte som den skulle og bidro til at vi fikk en brukbar alfa versjon av applikasjonen.

6 DISKUSJON OM RESULTAT OG FREMGANGSMÅTE

Fra første møte med oppdragsgiver, så var oppgaven og konseptet ganske bredt. Vi har lært at det er derfor viktig å ha både en felles visjon for oppgaven, samt en konkret oppskrift. Dette er noe som er trengs helt ifra begynnelsen. Selv med en ferdig definert oppgave, så finnes det mange måter å utføre den på. Siden visjonen for sluttproduktet var litt tvetydig, så har det tatt lengre tid enn nødvendig for å planlegge og utføre prosjektet-arbeidet. Grunnet dette så er det litt mangel på resultater som kan måles opp mot forventningene. Om vi hadde jobbet med dette prosjektet igjen, ville det vært et bedre forsøk å formulere krav og ønskelig funksjonalitet raskest mulig i planleggingsfasen, slik at mesteparten av prosjekttiden går til implementering, og gir oss mer tidsrom for feil og utforskning av andre muligheter om ting ikke går helt etter planen.

Et eksempel på dette er "trafikk-statistikk" funksjonen vår, som skal vise hvor travelt det er i trafikken når en bruker skal ta buss. Vårt møte med deres data-team ble arrangert fra MUST 19.mai. Dette var ganske sent, selv om vi hadde ferdig dokumentasjon på hvordan denne funksjonen skulle fungere på forhånd. Det hadde vært lurt å utforske mer om databruk selv i god tid før implementering, sånn at man hadde et bedre bilde over begrensninger relatert til data og muligheter for andre løsninger.

Applikasjonen ble utviklet i Android Studio ved bruk av Java programmeringsspråk. Vi kunne ha gjort applikasjonen mer tilgjengelig på flere enheter og nådd flere brukere med å utvikle en cross-platform applikasjon, men på grunn av mangel på kunnskap i cross-platform utvikling, hadde det ført til en del tidsforbruk for å lære oss dette, som ville ført til mindre tid for selve utviklingen. Nå som prosjektet er nesten ferdig, så tror vi at det ville ha blitt et lignende resultat med å gå for en cross-platform utvikling, siden det meste av tidsbruket gikk til andre ting enn selve utvikling. Bestemmelse for en Android applikasjon var på grunn av vår erfaring med Java og Android fra før av. Tanken bak dette var at vi kunne fokusere på implementering fra starten av prosjektet. Siden applikasjonen ikke skulle være et ferdig produkt, men var ment som *proof of concept,* så ble det tenkt at utviklingen av applikasjonen kunne fortsettes med utvidet funksjonalitet og iOS-støtte.

For å dele prosjektet i Android Studio mellom gruppemedlemmene ble det brukt Github. Vi brukte Github først og fremst for å sende hva som hadde blitt implementert i prosjektet til hverandre, som da gjorde det mulig for oss å samarbeide på det samme prosjekt i Android Studio. Vi hadde alle lite erfaring innenfor Github og slet derfor en del med "Push" og "Pull" metodene i begynnelsen og måtte derfor legge inn ekstra innsats for å lære om alt dette. Deretter ble det også brukt github sin prosjekt-fane som gjorde det enklere for oss å ha oversikt over arbeidet som skulle fullføres i de ulike iterasjonene. Denne prosjekt-fanen ble ikke tatt i bruk før litt senere innen prosjektutviklingen, siden vi ikke var klar over den, men ved utføring av prosjektet neste gang, anbefales det å ta i bruk dette allerede fra begynnelsen av.

Om prosjektet hadde blitt utført igjen, ville valg av verktøy forblitt det samme: Android Studio med Java under panseret. Hadde vi derimot brukt en del mer tid på å formulere oppgaven og kravene tidlig i startfasen, og utforsket tilgjengelig data og andre løsninger, så hadde det vært bedre tid til å fullføre alt av funksjonalitet og implementasjoner.

Utover dette så ser vi at produktet har kommet i live og det burde gi et klart bilde for brukere, som forhåpentligvis vil beregne det som et nyttig hjelpemiddel. For fremtidige utviklere så tenkes det at det er en ganske klar retning fremover med tanke på fullføring av den nåværende appen, men også for nye funksjoner som kan legges til senere.

7 KONKLUSJON OG VIDERE ARBEID

7.1 Oppsummering

For å hjelpe MUST med deres *living lab* bestemte vi på begynnelsen av prosjektet å utvikle en Android applikasjon som skulle fokusere seg rundt pilotprosjektene som de skulle gjennomføre. Gjennom applikasjonen skulle oppdragsgiver kunne gi ut informasjon angående MUST, og samtidig få inn tilbakemeldinger angående de aktivitetene rundt pilotprosjektene som de utfører. Mens brukerne ville få muligheten til å bidra i aktivitetene og påvirke disse pilotene som blir utført. Samtidig vil brukerne sitte igjen med en enkel applikasjon der de kan sende inn feilmeldinger, tilegne og dele informasjon angående trafikken. Til slutt skulle også applikasjonen brukes som en oppmuntring for brukerne til å velge kollektivtransport oftere.

Det var mye arbeid som gikk inn for å oppnå disse målene, spesielt en del forarbeid. Mye av tiden i prosjektet vårt gikk ut på å spesifisere selve oppgaven og planlegging av funksjonalitetene, som da førte til at vi satt igjen med mindre tid på selve utviklingen av applikasjonen. Vi fikk til slutt oppnådd de fleste målene våre, hvor vi endte opp med en grei og nyttig prototype, men det var likevel noe vi ikke fikk fullført. Det som hindret oss for å oppnå vårt fulle mål var altså reiseplanlegger-statistikk funksjonen som vi ikke fikk ferdig utviklet, som førte til at applikasjonen ikke ble komplett. Årsaken til at dette var noe mangel av oppdragsgivers ressurser som førte til mye ventetid for undersøking av andre muligheter, og dermed satt vi igjen med begrenset tid for resterende arbeid. I tillegg har ikke applikasjonen blitt testet mot en tilfredsstillende pilotgruppe ennå, så vi kan ikke være helt sikre om de resterende målene er blitt møtt.

Resultatene våre kan være nyttig for andre grupper som vil utføre lignende aktiviteter innenfor andre områder som trengs å testes av folkemengder. Applikasjonen vår vil gjøre det mulig for brukerne å gi tilbakemeldinger angående disse aktivitetene. På denne måten kan man samle inn brukernes meninger og utføre ulike undersøkelser ved å bruke applikasjon slik. Samtidig som at det er mulighet for brukerne å dele informasjon direkte til hverandre på en enkel og brukervennlig måte.

7.2 Videre arbeid

Dersom dette prosjektet skulle bli bygget videre på, så hadde vi først og fremst anbefalt å være tidlig ute med å finne ut om man har alle ressursene som trengs før man utvikler det som er ønskelig. Slik sparer man mye tid og energi for utvikler teamet, som heller kan bli brukt på å finne alternative løsninger som er effektive. Det mest åpenbare som må bygges videre på er reiseplanlegger/statistikk funksjonen. Det burde være en relativ lett oppgave å fullføre denne ettersom vi har et design på det, og data-teamet til MUST kan sannsynligvis bygge APIen gitt mer tid. Om det ikke skulle være mulig, så finnes det tilgjengelig data og API på internett, f.eks "enTur" sitt register for offentlig transport. Vi var for sen med å utforske dette nærmere, ettersom begge parter var enig om å bruke MUST sine data.

Det kan naturligvis legges til nye funksjoner. Vi har en ide hvor brukerne kan registrere seg og ha en profil, som holder oversikt over hvor aktivt en tar i bruk kollektive løsninger på transport. Brukere som f. eks tar mye buss, eller deltar på MUST sine pilotprosjekter, vil kunne bygge opp poenger. Disse poengene kan bli brukt til å gi hyppige brukere rabatt på relaterte produkter/tjenester, eller være med i trekningen på å vinne diverse priser. Utenom dette så blir det sikkert et behov for å utvide områdene hvor appen tas i bruk (om den blir en suksess). I første omgang er det Bergen. Det vil bli naturlig å utvide til flere storbyer, og etterhvert til flere fylker/kommuner.

Layouten til applikasjonen kan alltids bli bedre. Vi har et konsekvent og matchende design på de ulike skjermene på appen, men det er et noe forenklet og generisk design som kan forbedres videre på.

Applikasjonen er nå i første omgang bare tilgjengelig for Android, men den kan senere bli utviklet i Cross-platform som vil gjøre det mulig for iOS-brukere til å få tilgang til applikasjonen. Siden iPhones er populære i Norge, så vil dette være en bra måte for å nå flere brukere (iOS enheter).

(Plikk,, 2020) Figur .7.1:Oversikt over topp 10 mest solgte mobiler i 2019 hos Telenor

8 REFERANSER

8.1 Litteratur

- Annuzzi, J.Jr., Darcey, L. & Conder, S. (2016) *Introduction to Android Application Development (5. edition)*,Boston:Addison-Wesley Professional

- Bergen Kommune (u.å) *Digitalisering og innovasjons konsern*

  Tilgjengelig fra:
  https://www.bergen.kommune.no/omkommunen/avdelinger/digitalisering-og-innovasjon-konsern (Hentet: 02.april.2020)

- Android Developer (2020) *Documentation for app developers*
  Tilgjengelig fra:
  https://developer.android.com/docs
  (Hentet: 15. januar 2020).

- Statistisk Sentralbyrå (2020) *Bil og transport*
  Tilgjengelig fra:
  https://www.ssb.no/transport-og-reiseliv/faktaside/bil-og-transport
  (Hentet: 01. mai 2020).

- Agile Alliance (u.å.) *What is Scrum?*
  Tilgjengelig fra:

https://www.agilealliance.org/glossary/scrum/
(Hentet: 07. mai 2020)

- Statistisk sentralbyrå (u.å.) *Ordforklaring*
  Tilgjengelig fra:
  https://www.ssb.no/ajax/ordforklaring?key=315602&sprak=no
  (Hentet: 01. mai 2020)

- Android Developer (2020) *Maps SDK for Android*
  Tilgjengelig fra:
  https://developers.google.com/maps/documentation/android-sdk/intro
  (Hentet: 10. april 2020)

- Android Developer (2020) *Fused Location Provider API*
  Tilgjengelig fra:
  https://developers.google.com/location-context/fused-location-provider/
  (Hentet: 20. mai 2020)

- Github (2020) *Google Gson library*
  Tilgjengelig fra:
  https://github.com/google/gson
  (Hentet: 13. mai 2020)

- W3School (u.å.) *JSON Introduction*
  Tilgjengelig fra:
  https://www.w3schools.com/js/js_json_intro.asp
  (Hentet: 13 mai 2020)

- Colliander,A. (2014) *Skyss lanserer ny reise app.*

  Tilgjengelig fra:

  https://www.bt.no/nyheter/lokalt/i/AE2n5/skyss-lanserer-ny-reise-app

  (Hentet: 02.april.2020).

- Skyss(2016) *Sanntid og reiseplanlegger i samme app*

  Tilgjengelig fra:
  https://www.skyss.no/Verdt-a-vite/Nytt-fra-Skyss/sanntid-og-reiseplanleggar-i-samme-app/(Hentet:02.februar.2020)

- Plikk, N. (2020) *De mest solgte mobilene i 2019*

  Tilgjengelig fra:
  https://www.tek.no/nyheter/nyhet/i/kJL5eL/de-mest-solgte-mobilene-i-2019?fbclid=IwAR0cnJYulCMtdIu3cQaHqHKxYAD1w4Om5Zae4fPK0VxYx0DpYYFiL6tsdF0

  (Hentet: 29.mai.2020)

8.2 Verktøy

- Android Developer (2020) *Android Studio Download*
  Tilgjengelig fra:
  https://developer.android.com/studio/?gclid=CjwKCAjwztL2BRATEiwAvnALcvOadxMlIFNk5CKGyT6eY2Tp6tyH8JkeQs2eTCKXyBgAddfgfpN2LxoC2HcQAvD_BwE&gclsrc=aw.ds (Hentet: 5. januar 2020).

- GitHub, Inc. (2020) *GitHub*
  Tilgjengelig fra:
  https://github.com/ (Hentet: 8. januar 2020).

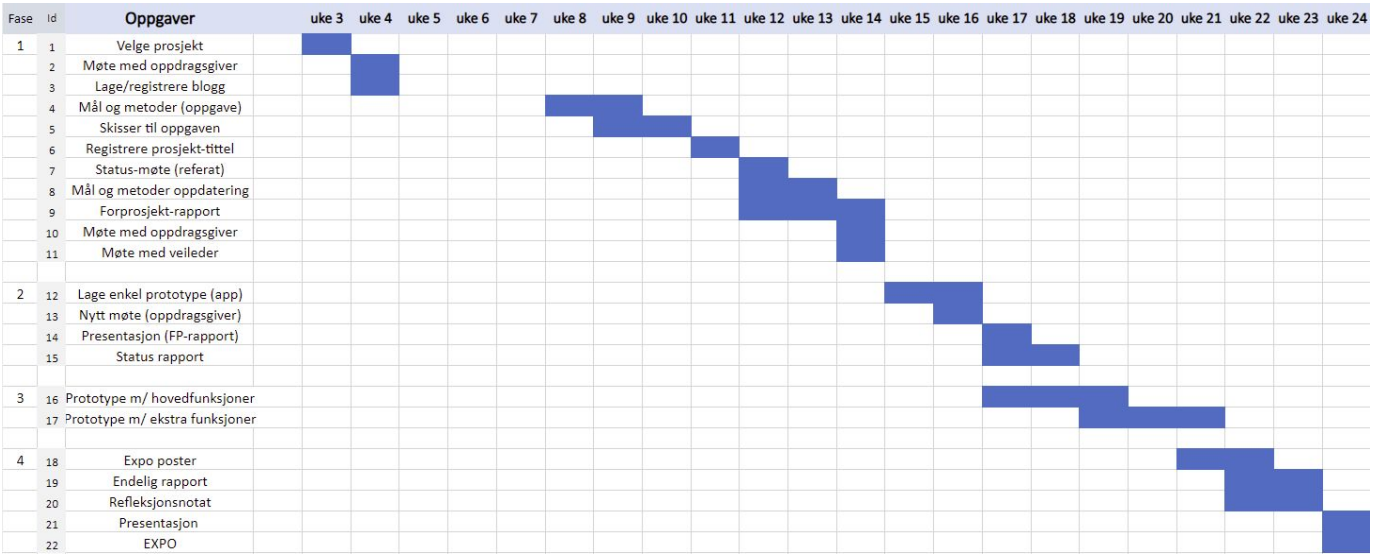
- Google Docs (2020) *Google Documents*
  Tilgjengelig fra:
  https://www.google.com/docs/about/ (Hentet: 11. januar 2020).

9 APPENDIX

9.1 Risikoliste

Risiko	Sannsynlighet (1-5)	Konsekvens (1-5)	Total risiko (1-25)	Skadebegrens-ning
Overskride tidsfrist	2	4	8	Jobbe iterativt og fokusere på hoved-funksjon er
Mangel på kompetanse	2	5	10	Vi får en API å jobbe med. Hovedfunksjone ne regnes som oppfyllbar.
Corona-virus	5	1.5	7.5	Vi er ikke avhengig av fysisk kontakt. Arbeid og møter utføres på egen PC og digitalt.
Manglende sluttprodukt	2	4.5	9	Vi tester mye underveis og kommuniserer mye med oppdragsgiver

9.2 GANTT diagram



Fase	Id	Oppgaver	uke 3	uke 4	uke 5	uke 6	uke 7	uke 8	uke 9	uke 10	uke 11	uke 12	uke 13	uke 14	uke 15	uke 16	uke 17	uke 18	uke 19	uke 20	uke 21	uke 22	uke 23	uke 24
1	1	Velge prosjekt																						
	2	Møte med oppdragsgiver																						
	3	Lage/registrere blogg																						
	4	Mål og metoder (oppgave)																						
	5	Skisser til oppgaven																						
	6	Registrere prosjekt-tittel																						
	7	Status-møte (referat)																						
	8	Mål og metoder oppdatering																						
	9	Forprosjekt-rapport																						
	10	Møte med oppdragsgiver																						
	11	Møte med veileder																						
2	12	Lage enkel prototype (app)																						
	13	Nytt møte (oppdragsgiver)																						
	14	Presentasjon (FP-rapport)																						
	15	Status rapport																						
3	16	Prototype m/ hovedfunksjoner																						
	17	Prototype m/ ekstra funksjoner																						
4	18	Expo poster																						
	19	Endelig rapport																						
	20	Refleksjonsnotat																						
	21	Presentasjon																						
	22	EXPO																						

9.3 Dokumentasjon på algoritme

Algoritme for busstrafikk

Målsetting:
Lage en algoritme som kan fortelle hvor travelt det er på bussen, til ethvert tidspunkt.
Med våres ressurser, så kan vi ikke gi et helt nøyaktig resultat.
Målet er å vise om det er generelt lite eller mye folk på et gitt tidspunkt.

Utfordringer:
Ferie- og helligdager kan forekomme på ulike ukedager fra år til år. (selv med samme dato).
Påsken kan forekomme på ulike datoer år til år.

Løsning?

Vi identifiserer alle spesielle dager som påvirker trafikken (ved å se på tidligere data), og markerer disse dagene på forhånd for nåværende år. På den måten glemmer vi ikke en spesiell dag.

Tidsblokker:
Data kan bli delt inn i 24 timers blokker for hver dag. Eventuelt 12 timer per dag. På denne måten blir det enkelt å gjøre utregninger. Om en buss går f.eks 16:52 så trenger vi ikke søke opp samme buss-avreise fra forrige år. Den trenger kun å få tilbake gjennomsnittet fra tidsintervallet 16:00-17:00.

Identifisere avvik:
En vanlig mandag blir regnet ut ved å hente den samme mandagen de siste 3 årene, og gi tilbake gjennomsnittet. Om en av mandagene gir et veldig annerledes resultat (la oss si et avvik på f.eks 50%), så vil algoritmen si ifra hvilken mandag som skiller seg ut, og hvor mye. På denne måten kan vi identifisere f.eks helligdager som har falt på en annen ukedag.

Oppbygging av algoritmen:
Antagelse: Vi har data for hver dag/uke/måned de siste 3 årene.

En vanlig mandag blir regnet ut ved å hente den samme mandagen de siste 3 årene, og gi tilbake gjennomsnittet. Mandagen sin data vil bli delt inn i 12 eller 24 tidsblokker, og riktig tidsblokk hentes ut etter brukerens søk.

Vi kan inkludere et gjennomsnitt fra de siste 3 mandager generelt.

Spesielle dager blir markert på forhånd. Slik blir det enkelt å identifisere og sammenligne med spesielle dager fra tidligere år. F.eks hvordan ser 17.mai ut?

Resultat:
Resultater som kommer opp på appen kan f.eks deles inn i:

- Minimalt
- Lite
- Medium/vanlig
- Mye

Siden dataene vi jobber med er gamle data, og ikke live data, så vil ikke resultatet bli særlig mer nøyaktig enn dette.

Covid-19 påvirkning?
De høyest trafikkerte tidspunktene, vil nok være de samme både med og uten viruset. (rushtid er rushtid). Det samme gjelder nok for de minst trafikkerte tidspunktene. Er fortsatt lite folk klokken 07.00 på en søndag.

Løsning?
Vi kan informere om at dette er generelle data som ikke tar covid-19 i betraktning.

Algoritme #1:

$$R = \frac{Gå+Gu}{2}$$

R = Resultat
T = Tidsblokk
Gå = Gjennomsnitt for gitt ukedag siste 3 år

Gu = Gjennomsnitt for gitt ukedag, siste 3 uker

Først henter vi Gå basert på en gitt tidsblokk(T). Dette vil gi et svar på 1-10
Deretter henter vi Gg basert på samme tidsblokk(T). Dette vil gi et svar på 1-10

Resultat (R) vil variere fra 1-10.
10 vil være full buss. 1 vil være tom.

Algoritme #2:

$$R = \frac{(Gå + Gu)*Dg}{2}$$

R= Resultatet
T= Tidsblokk
Dg = Differansen i prosent fra år til år (gjennomsnitt av 2 siste år)
Gå = Gjennomsnittet av gitt ukedag i de 2 siste årene (på gitt tidsblokk)
Gu = Gjennomsnittet av gitt ukedag basert på de siste 4 ukene (på gitt tidsblokk)

Samme algoritme, men tar hensyn til vekst fra år til år (Dg).

Algoritme #3

$$R = \frac{MvT}{Ma/100}$$

Utrekning
R = Resultat, vist i prosent
Ma = Maks antall av passasjerer på ein bestemt buss
MvT = Median for gitt tidsblokk og gitt vekedag, siste 3-4 veker

Visning av data
Ide #1
Visning blir fordelt i 4 bars. Som kvar av dei skal tilsvarer 25%.
0%-25%
25%-50%
50%-75%
75%-100%

Ide #2 (?)
Det blir vist med same 4 bars, men her kvar av dei kjem å tilsvarer forskjellig tilstand i bussen.
1Bar - Bussen er nesten tom
2Bars - Ledige sitteplasser
3Bars - Kun ståplasser
4Bars - Stappfult

Håndtering av forskjellige størrelse av kjøretøy
Antagalese: Data som er samla inn er antall av passasjerer på ein bestemt buss.

Det finst forskjellige busstørrelse med forskjellige antall av sitteplasser og ståplasser. Kvar av vår bestem visningstype må ta utgangspunktet i dette.
Det enklaste og kanskje smartaste blir å dele maks antall av passasjerer som kan være inn på bussen med 100%, dette skal gi oss antallet som tilsvarer 1% som vi kan da dele utrekna antall av passasjerer med.