Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

BACHELOROPPGAVE

Studieprogram/spesialisering: Bachelor i ingeniørfag / Data	Vårsemesteret 2022 Åpen
Forfatter(e): Anders Skrøvseth Haugen, Edvin Grytnes, Theebthan Jeyakumaran	
Fagansvarlig: Erlend Tøssebro Veileder(e): Erlend Tøssebro	
Tittel på bacheloroppgaven: Virtuell stoppknapp for bussen Engelsk tittel: Virtual stop button for the bus	
Studiepoeng: 20	
Emneord: Flutter, Kart, Dart, API, Google Maps, Kolumbus AS, App	Sidetall: 57 + vedlegg/annet: 1 Stavanger 15. mai 2022

Innhold

Sammendrag

I denne oppgaven har vi laget en app i sammarbeid med Kolumbus AS som skal forbedre kollektivtransporten i Rogaland. Hovedmålet med appen er at det skal være mulig å stoppe en buss fra telefonen. Brukeren kan selv velge en bussholdeplass og hvilken buss brukeren ønsker at skal stoppe ved den bussholdeplassen.

Oppgaven ble løst ved å lage en mobilapp i Flutter og ved hjelp av Google Maps sin Api og vise et kart i appen med alle bussholdeplassene. Informasjonen vi trengte var offentlig data fra Kolumbus sine systemer. Det ble også brukt flere av mobilens funksjoner som GPS og mobildata/WiFI for å hente posisjonsdata og laste ned data fra Kolumbus.

Kolumbus sin plan med appen er at den skal være en prototype, som senere skal bli implementert med Kolumbus sin egen sanntidsapp. Ved å slå den sammen med Sanntidsappen vil dette kunne føre til enda bedre funksjonalitet som blant annet automatisk stopp av busser med reiseplanlegger.

Vi fikk implementert og fullført alle hovedmål og noen av de sekundærmålene vi satt oss i starten. Alt i alt er vi meget fornøyd med resultatet og appen vi nå har laget.

Kapittel 1

Introduksjon

1.1 Oppgavebeskrivelse

I samarbeid med Kolumbus AS, går oppgaven ut på å utvikle en mobilapplikasjon for en virtuell stoppknapp. Hovedfunksjonen i denne applikasjonen er å kunne stoppe bussen fra mobilenheten, med noen begrensninger. Denne teknologien vil endre brukeropplevelsen av kollektivtilbudet gjennom å gi passasjerene større kontroll over hvilken buss som stopper, spesielt i sistuasjoner der flere busser kommer samtidig og det kan bli usikkerhet hos både passasjer og sjåfør om hvilken buss man faktisk ønsker å stoppe.

1.2 Om bedriften

Kolumbus AS er Rogaland fylkeskommune sin mobilitetsleverandør, som sørger for at over 110 000 daglige reiser med kollektivtransport gjennomføres i Rogaland. De har ansvar for buss-og hurtigbåttrafikken og i tillegg jobber de nå for at tog, sykkel, gange og bildeling henger sømløst sammen med buss og båt, slik at hverdagen skal bli lettere for folket, uten bruk av egen bil. Kolumbus AS er et kundeorientert, ansvarlig og nytenkende selskap, som er opptatt av å utvikle en teknologi som gjør brukeropplevelsen bedre[13].

1.3 Mål og Motivasjon

For å jobbe strukturert og målrettet så ble det satt ned noen konkrete mål med oppgaven. Disse er separert inn i hovedmål og sekundærmål for å videre styre prioriteringen av rekkefølgen på arbeidet. Hovedmålene vil være det som kan defineres som "kjernen" av appen, eller det som er minimum funksjonalitet for å ha løst problemstillingen. Sekundærmålene vil være ekstra funksjonalitet eller utvidelse av hovedmålene for å gi en bedre og mer utfyllende brukeropplevelse.

1.3.1 Hovedmål

For at dette skal være en funksjonell app som brukeren kan gå inn på for å stoppe bussen, ble det satt opp en liste med det vi mente var det viktigste for at denne appen skulle fungere.

- Appen skal ha et kart som viser hvor brukeren er
- Appen skal ha en liste over holdeplasser og en liste over avganger som går igjennom holdeplassen.
- Det skal være mulig for en bruker å sende et signal til en bestemt buss om at den skal stoppe på en bestemt holdeplass.
- Validering av GPS data. En bruker får kun stoppe bussen hvis brukeren er innenfor en viss avstand fra bussholdeplassen
- En begrensning på hvor mange stoppsignal en bruker kan sende iløpet av en tidsperiode.

1.3.2 Sekundærmål

Det ble også satt opp en liste med sekundærmål. Denne listen er funksjoner som det enten var usikkert om de ville passe inn i appen, eller som kunne vise seg å være for tidkrevende å få til innen oppgaven skulle leveres. Disse er ikke de viktigste målene og er ikke nødvendig for at appen skal løse problemstillingen.

- Enkel reiseplanlegger. Brukeren legger inn hvor man reiser fra og hvor man skal av. Appen sender holdeplassene for start og ende til bussen. Bussen stopper dermed både på start og ende uten at brukeren gjør noe mer.
- En mer avansert validering som tar høyde for om brukeren vil rekke bussen en viss avstand unna på begrenset tid.
- Brukeren kan trykke på kartet for å velge en holdeplass isteden for å velge den fra lista for å stoppe bussen derfra.

1.3.3 Forventninger

Våres forventninger er at dette skal bli en prototype som Kolumbus kan ta med seg videre i utvikling av stopp knappen og få implementert det inn i sanntidsappen. Selv om det skal være en prototype, så forventer vi fortsatt at vi lager en fungerende app slik at stopp-funksjonaliteten enkelt kan testes uavhengig av om den er lagt til i Sanntids-appen eller ikke.

1.3.4 Motivasjon

Først og fremst har ingen i gruppen drevet med mobilutvikling tidligere. I tillegg var flutter et ganske nytt rammeverk. Så det å sette seg inn i det og få det startet opp var ganske utforderende og spennende. Men med hjelp av videoer og crash course på nettet, ble dette lettere å komme seg inn i Flutter.

Som sagt er Kolumbus ansvarlig for den daglige transporten for mange av innbyggerene i Rogaland. Ikke minst er mennesker avhengig av det for å komme seg fra A til Å. Det å få delta og være med på utviklingen av dette, er en stor glede og erfaring for oss som selv er avhengig av Kolumbus.

1.4 Arbeidsmetoder

Det å ha en effektiv og målrettet arbeidsmetode er noe av det viktigste for å lykkes med dette prosjektet. I tillegg jobber man med store mengder kode, så det er særdeles viktig å ha orden på dette. Ved å ha en tydelig arbeidsmetode fører det til at arbeidet blir orientert. Under står det litt mer om hvordan dette ble utført.

1.4.1 Github

For å samarbeide med kodingen ble GitHub brukt. GitHub er et versjonshåndteringsystem som brukes for å samarbeide om, håndtere og administrere filer. Fordelen med Github er at man kan gå inn og se og tilbakestille på endringer[17].

Det ble laget en repository for appen, der ble det lagt inn to branches, en main og en dev. Main branchen var kode som vi visste fungerte og ikke ville kræsje appen, mens i dev branchen ble det skrevet kode som var under utvikling. Meste parten av koden ble skrevet i dev branchen og hvis den nye koden fungerte og ikke kræsjet ble dette lagt inn i main branchen. Hvis det var noen enkel funksjoner som ble forsøkt å gjøre bedre eller noen funksjoner som var usikre på om skulle med i appen, laget vi en branch kun for denne ene funksjonen. Hvis funksjonen fungerte og gjorde det som den skulle ble den pushet videre inn i dev branchen også lukket vi denne brachen eller så ble branchen lukket uten å pushe til dev.

1.4.2 Ukentlige mål og loggføring

For å ha orden på arbeidet, ble det skrevet ned ukentlige mål, med hva som måtte bli gjort og når. På slutten av hver uke ble det loggført hva som ble gjort, og hva som måtte jobbes mer med. Dette førte til at vi hadde en oversikt over fremgangen og arbeidet. I tillegg var det lettere med å begynne på nye ting, enn å sitte med det samme, når det ikke trengtes. Vi hadde ukentlig møter med veileder på mandag morgen og etter disse møtene satt vi oss selv ned og bestemte hva vi skulle gjøre denne uka. På slutten av hver

uke så gikk vi igjennom hva vi fikk gjort og hva som ikke ble gjort av de målene vi hadde satt oss selv på starten av uka. Dette førte til at det ble en fin syklus på ukene hvor vi hadde bestemte mål og fikk diskutert hva som gikk bra og hva som kunne vært bedre.

Kapittel 2

Teknologier og rammeverk

2.1 Valg av rammeverk

Som med alle softwareprosjekter så må man starte med å velge hvilke(t) språk og/eller rammeverk eller andre teknologier man ønsker å bruke. Det er vanskelig å sette en fasit på hva som er best i en bestemt type prosjekt, men de fleste teknologier har fordeler og ulemper på forskjellige områder. Siden ingen av oss på gruppa hadde jobbet med noe relevante rammeverk for apputvikling tidligere så kunne vi prøve å gjøre en så objektiv vurdering som mulig ettersom ingen av oss hadde noen spesielle preferanser på et bestemt rammeverk fra før.

2.1.1 NativeScript

NativeScript er et åpent rammeverk som ble utviklet av Progress Telerik i 2014. Rammeverket lar deg bygge applikasjoner på en enkelt kode-base. Det er en av de mest kjente og brukte rammeverket innenfor frontend utvikling. NativeScript blir brukt for å lage applikasjoner på iOS og Android platformer. NativeScript et av rammeverkene man kan velge for et prosjekt som dette[16].

NativeScript applikasjoner er bygget opp av JavaScript eller andre programmeringspråk som kan transpileres til JavaScript, som f.eks. TypeScript. I tillegg støtter NativeScript, Vue og Angular JavaScript rammeverk. Ionic og NativeScript er ganske like rammeverk, men en av de viktigste forskjellen på de er at NativeScript apper kan kjøres direkte på native enheter, som er årsaken til at det heter NativeScript. Den trenger altså ingen krysskompilator eller nettinteraksjon. Derimot er Ionic avhenging av plugins for å dekke applikasjonen i native utforming[11].

2.1.2 Ionic

Ionic er en open-source SDK som lar deg lage en enkelt mobil-app på tvers av platformer, og i 2019 var det ca 5 millioner apper laget med Ionic. Ionic lar deg bruke eksisterende kunnskap og programmeringsspråk fra webprogrammering som HTML, CSS og JavaScript samt rammeverk til JavaScript som Angular. Dette kan gjøre Ionic veldig attraktivt for gode webutviklere etter som det er minimalt med ny kunnskap man må lære seg for å bygge apper i Ionic. I Ionic så lager man i prinsippet en web-app som gjennom en funksjonalitet i en mobil nettleser som heter WebView får en web-app til å fremstå som en native app for sluttbrukeren. Ionic er også avhengig av en plugin som heter Ionic Native som samler flere plugins som kan bruke enhetens hardware som blant annet kamera, GPS og gyroskop etter som en ren web-app ikke kan få tilgang til slik funksjonalitet[9].

2.1.3 Flutter

Flutter er et moderne rammeverk utviklet av Google og fungerer som et rammeverk til språket Dart (også utviklet av Google). Når man skriver en app med Flutter så består det hovedsakelig av en trestruktur med widgets. Alt i Flutter er en widget, og hver enkelt widget er objekter på samme måte som i objektorientert programmering. At widgets i prinsippet er objekter gjør at hver type widget har noen default egenskaper og grafiske stiler. Widgets kan også tilpasses akkurat slik man selv ønsker. Om man er komfortabel med objektorientert programmering tar det veldig kort tid å bli vant med widgets i Flutter. Siden widgets i Flutter settes opp i en trestruktur så er også hver widget en node, og mange widgets kan ha enten

en eller flere barnenoder, men noen kan ikke ha barnenoder. Mange widgets krever minst en annen widget for å konstrueres. Det er også mulig å konstruere egne widgets, akkurat som man skulle laget egne klasser i objektorientert programmering, dette gir god kontroll og separasjon av ansvar samt at widget-treet som bygges i main-fila ser ryddig ut når mesteparten av logikken til en widget flyttes vekk fra selve widget-treet.

Kanskje den største styrken til Flutter er at det gir nærmest like god ytelse som en ordentlig native app for hver av de separate operativsystemene, til tross for at det skrives som en enkelt kryssplatform kodebase. Flutter gir også muligheten til å utnytte operativsystemspesifikke funksjoner samt å sjekke hvilket operativsystem man er på slik at man sømløst kan bruke spesifikk funksjonalitet selv om mesteparten av koden brukes for begge operativsystem. At alt kompileres til native maskinkode med Flutter gir også den fordelen at Flutter kan styre alt innhold på skjermen, slik at man kan sørge for at appen ser lik ut uansett operativsystem, men ha det forskjellig om man ønsker. Noen andre rammeverk vil bruke operativsystemets native design på flere widgets. Hva som er best kommer nok an på hvem man spør, men med Flutter har man hvertfall en stor grad av frihet[3].

```
1  import 'package:flutter/material.dart'
2
3  void main() => runapp( HelloWorldApp());
4
5  class HelloWorldApp extends StatelessWidget {
6      @override
7      widget build(BuildContext) {
8          return MaterialApp(
9              home: MaterialApp(
10                 child: Text('Hello World!'),
11             ),
12         ),
13     }
14 }
```

Kode 2.1: Hello World eksempel i Flutter

2.1.4 Separate native apper for iOS og Android

Det som menes med seprate apper for IOS og android er at man da må ha to forskjellige kodebaser, en til IOS og en til android. IOS sitt operativsystem er basert på programmeringsspråket Objective-C, mens Android er basert på programmeringsspråket Java. Dette er to veldig forskjellige språk som ikke snakker så godt sammen, derfor er man nødt til å skrive app'er i forskjellige kodebaser eller ha et rammeverk som Flutter som beskrevet tidligere i kapittelet 2.1.3. Det mest populære programmeringspråket for IOS apper er Swift, og det mest populære for android er Kotlin. Så det er disse to programmeringspråkene som ble vurdert for prosjektet vårt. Fordelene med å bruke native apper er at det er enklere å utforme og man kan se at stilen ligner veldig på hvordan plattformens apper ser ut. Problemene med native apper er at det er dyrt og man trenger flere utviklere, gjerne et team per plattform. Under kommer et avsnitt om hvordan swift og kotlin fungerer, og fordeler og ulemper med de to. [12]

2.1.5 Swift

Swift er et programmeringsspråk utviklet hos Apple for å ha et moderne språk spesielt rettet mot utvikling av applikasjoner mot Apple sine forskjellige operativsystemer og enheter som iOS, MacOS, watchOS og tvOS for iPhone, Mac, Apple Watch og Apple Tv.

Swift er ment å være en erstatning for eldre C-baserte språk som C, C++ og Objective-C, og det var dermed viktig for Apple at det hadde tilsvarende ytelse som disse språkene samtidig som det kunne tilby et enklere språk med moderne funksjonalitet.

Swift har også høyt fokus på sikkerhet gjennom blant annet null-safety (objekter kan ikke være *nil* uten å eksplisitt tillate det for den enkelte variabel), alltid initialisering av variabler før bruk, automatisk overflow sjekk på arrays, og automatisk memory management. [1]

Swift-koden som vises i Kode 2.2 er et eksempel på en helt enkel Hello World-app for iOS.

```
1  import UIKit
2
3  class ViewController: UIViewController {
4
5      @IBOutlet var displayLabel: UILabel!
6
7      @IBAction func saySomethingTapped(_ sender: UIButton) {
8          displayLabel.text = "Hello World!"
9      }
10 }
```

Kode 2.2: Hello World eksempel i Swift

[10]

2.1.6 Kotlin

Kotlin er et ganske nytt programmeringspråk, og ble laget fordi Java er et vanskelig språk å lære seg. Kotlin ble laget for at det skal være ganske likt Java, men at det er enklere å skrive i enn Java og at det ikke tar så lang tid å lære seg. Java er et objekt orientert programmeringspråk som krever mye kode for å kjøre et enkelt program, mens Kotlin er en miks av objektorientertprogrammering og funksjonell programmering. Under blir det vist to eksemepler på en helt enkel applikasjon i både Java og Kotlin, her ser man at Kotlin har fokusert på å gjøre det så enkelt som mulig og at brukeren skal skrive minst mulig kode for å kunne kjøre applikasjonen.[15] Se kodeeksempel 2.3 og 2.4

```
1  public class Helloworld
2  {
3      public static void main(string[] args)
4      {
5          System.out.print("Hello, World");
6      }
7  }
```

Kode 2.3: Hello World eksempel i Java

```
1  fun main(args : Array<String>) {
```

```
2       println("Hello, World!")
3   }
```

Kode 2.4: Hello World eksempel i Kotlin

Siden kotlin og java er to ganske like språk så er framgangsmåte for å starte en app på telefonen ganske så lik se kode eksempel 2.5. Det kode eksemplet gjør er at det setter opp en bakgrunn på telefonen med en tekst midt på skjermen som sier "Hello World". Som man ser er dette mer kode og vanskeligere å forstå enn kodeeksemplet 2.1 som viser en enkel app i Flutter.

```
1  Activity.xml
2  <?xml version="1.0" encoding="utf-8"?>
3  <androidx.constraintlayout.widget.ConstraintLayout ...
       xmlns:android="https://schemas.android.com/apk/res/android"
4      xmlns:app="http://schemas.android.com/apk/res-auto"
5      xmlns:tools="http://schemas.android.com/tools"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      tools:context=".MainActivity">
9      <TextView
10         android:layout_width="wrap_content"
11         android:layout_height="wrap_content"
12         android:text="Hello World!"
13         app:layout_constraintBottom_toBottomOf="parent"
14         app:layout_constraintLeft_toLeftOf="parent"
15         app:layout_constraintRight_toRightOf="parent"
16         app:layout_constraintTop_toTopOf="parent" />
17 </androidx.constraintlayout.widget.ConstraintLayout>
18 Default main activity code
19 package com.example.helloworldapplication;
20 import androidx.appcompat.app.AppCompatActivity;
21 import android.os.Bundle;
22 public class MainActivity extends AppCompatActivity {
23     @Override
24     protected void onCreate(Bundle savedInstanceState) {
25         super.onCreate(savedInstanceState);
26         setContentView(R.layout.activity_main);
27     }
28 }
```

Kode 2.5: Hvordan starte opp en mobil app i Java og Kotlin

2.1.7 Endelig valg av Flutter

Det endelige valget av rammeverk falt på Flutter av flere grunner. En av grunnene var at Flutter fremsto veldig enkelt å jobbe med, spesielt etter som to av oss på gruppa hadde jobbet mye med C# kort tid før denne oppgaven, og Dart som brukes med Flutter virket veldig naturlig om man var vant med C# fra før.

Flutter gav også inntrykket av å være det rammeverket blant de som ble sett på som gav størst frihet til design ved at man ikke er begrenset til native UI på Widgets. Etter som vi testet app'en på både iOS og Android var det på en måte fint å ha en jevn og konsistent opplevelse av app'en på tvers av plattformer, men andre vil selvfølgelig foretrekke en langt mer native UI på app'er.

En annen viktig grunn til at Flutter ble valgt er at det er det nyeste av de rammeverkene som ble vurdert, samt at det har vokst veldig raskt, dermed kan det være veldig relevant å ha kjennskap til Flutter.

2.2 Valg av kart

2.2.1 TomTom

TomTom er et kartsystem mange har hørt om da det er brukt mye som GPS i bilen. Det var derfor dette fort ble et kartsystem som ble vurdert. TomTom har en gratis karttjeneste for mobil hvor man kan ha 2500 request daglig til TomTom sin API. Siden denne aplikasjonen er en prototype som ikke skal ut til offentligheten er 2500 request per dag mer enn nok for å teste appen. Etter å ha undersøkt hvordan kartsystemet fungerer for mobil så ble det tydelig at de satser mer på navigasjon og GPS til biler da de har en ikke veldig god karttjeneste for mobil, og som heller ikke er veldig kompatibelt med Flutter. Derfor ble det lite aktuelt å bruke TomTom da det finnes mange andre og enklere karttjenester tilgjengelig.

2.2.2 Google maps

Google maps er den mest kjente karttjenesten og følger med som standard på alle Androidtelefoner. Google maps tilbyr en rekke tjenester og api'er som man kan bruke for å utvikle mobilapper med karttjenester. Mange av tjenestene til Google koster penger og følger en strategi som går ut på at prisen øker med antall brukere av appen. Man kan opprette en gratis konto med det enkleste kartet uten noen ekstra funksjoner. Her har man en begrensning på 2000 request i døgnet, som er mer en nok for denne appen da den skal bare testes lokalt og ikke publiseres.

2.2.3 Mapbox

Mapbox er et selskap som ble grunnlagt i 2010 og lager digitale kart som man kan bruke på applikasjoner. De har i de siste årene vært en stor konkurrent til Google Maps. Mapbox blir brukt av flere store selskaper i USA, som blant annet Strava og Snapchat [2]. Mapbox tilbyr også gratis versjoner av det helt enkle kartet og her kan man ha opptil 25 000 månedlige brukere som er ganske mange flere brukere enn det Google tilbyr. Mapbox har også flere tjenester som navigasjon, men dette koster penger fra første request. Det ser ut som Mapbox satser mest på navigasjon. Mapbox tilbyr også en integrasjon med Flutter som man kan installere via en plugin, men denne er ikke like enkel å bruke som Google sin.[14]

2.2.4 Endelig valg av Google Maps

Det endelig valget av kart falt på Google Maps av flere grunner. En av de grunnene var at Google Maps fremsto som veldig enkelt å jobbe med og var kompatibelt med Flutter, spesielt ettersom Flutter hadde egen plugin (les mer her 2.3.2) og man kan bruke en egen widget til å styre innstillingene på kartet. Google maps er også det mest kjente kartet og ga mest frihet til design etter som man kan bruke det som en widget. Etter som appen skal fungere på både iOS og Androd var Google maps veldig like på begge de operativsystemene og fungerte på samme måte for begge to.

2.3 Plugins for kart og GPS

2.3.1 Geolocator

Geolocator er en tredjepartstjeneste man kan laste ned fra Flutter sin offisielle utviklerside. Geolocator er en plugin som gir tilgang til stedstjenester på de forskjellige plattformene som telefoner bruker. Noen av funksjonene denne pluginen tilbyr er at den kan hente siste kjente posisjonen, hente nåværende posisjon med flere nivå av presisjon, sjekke om stedstjenester er slått på og hvis stedstjenster ikke er slått på tilbyr denne pluginen en tjeneste som sender ut et varsel som spør om brukeren tilater at appen bruker stedstjenester. Den inneholder også en klasse, *Position*, som inneholder all posisjonsdata som hentes ut fra GPS (se kapittel 2.4.1 for mer info) , men det er kun koordinater i lengde- og breddegrader som blir brukt i denne app'en.

Denne pluginen blir brukt mye i dette prosjektet da den brukes for å posisjonere kartet. Den blir også hyppig brukt for å validere brukerens posisjon og da spesielt å sjekke om brukeren faktisk kan stoppe bussen, dette forklares i mer detalj i kapittel 3.5.

2.3.2 Google Maps for Flutter

Google Maps for Flutter en plugin som gir oss en widget av et Google Maps kart. For å få dette til å fungere trenge vi en API-key fra Google som man enkelt kan sette opp uten kostnader. Ved at dette er en widget kan man enkelt sette dette inn med resten av koden. Den gir også mye valgfrihet når det kommer til hva vi vil at kartet skal inneholde da pluginen kommer med mange forskjellige metoder og funksjoner for å styre kartet. Google sin SDK (software devolpment kit) for iOS og Android inneholder en egen integrasjon for Flutter som gjør at det blir enklere å bruke enn for eksempel TomTom (2.2.1) som ikke hadde det. Denne integrasjonen med Flutter gjør det veldig enkelt å sette opp et standard prosjekt da denne tilbyr en egen widget som man kan legge inn i Flutter, noe som var vanskeligere med Mapbox da de ikke tilbyr en egen widget.

2.4 Hardware

Dette er primært et softwareprosjekt, men for at det skal fungere er det nødvendig med bruk av noe hardware. Selvfølgelig må man ha en smarttelefon med muligheten for å laste inn app'en, men den må også ha bygget inn mottaker og sender for Wi-Fi/Mobildata samt en mottaker for GPS-signal. Dette er standard på så og si alle mobiltelefoner i dag, og har vært det lenge. Hvordan dette fungerer vil bli beskrevet i dette kapittelet.

2.4.1 GPS

GPS, eller satellittnavigasjon, er et globalt nettverk av satellitter som bidrar til at enheter med kompatible mottakere kan regne ut sin egen posisjonsdata. Begrepet GPS brukes ofte synonymt med satellittnavigasjon, men GPS er ett av fire satellittsystem globalt. GPS er et amerikansk nettverk, i tillegg til at Russland, Kina og EU har sine tilsvarende nettverk av satellitter.

I dette prosjektet brukes satellittposisjonering nærmest konstant da det er veldig viktig å finne ut hvor en bruker er for å posisjonere kartet i appen riktig, samt å validere posisjonen mot tillatte holdeplasser. Dette vil utdypes mer senere i prosjektet.

Satellittnavigasjon fungerer ved at satellittene sender ut radiosignaler med forskjellig informasjon som blant annet tidspunktet signalet sendes ut og satellittens egen posisjon. Om en mottaker plukker opp signalet fra 4 unike satellitter, kan mottakeren regne ut sin egen posisjon, høyde over havet og tid. Om en mottaker har en innebygd atomklokke, en form for ekstremt presis klokke som også brukes i satellitter, så kan man klare seg med kun tre signaler, men ved å bruke signalet fra fire satellitter slipper mottakeren å ha en egen atomklokke, men kan heller beregne tidspunkt basert på signalet fra den fjerde satellitten [6].

Med ordinær satellitttjeneste kan man oppnå presisjon på omtrent 7 meter 95% av tiden nærmest overalt på planeten.

2.4.2 Wi-Fi/Mobildata

I dette prosjektet brukes det en REST API til å hente data fra Kolumbus. Dette krever at brukeren er koblet til enten Wi-Fi eller mobildata. Mobildata og Wi-Fi er to ganske like konsepter og man har som regel ikke tilgang til begge to samtidig, gitt at begge er tilgjengelig er det opp til brukeren hvilken vedkommende vil bruke. Største forskjellen mellom Wi-Fi og mobildata er at Wi-FI har raskere overføring av data og koster mindre, men fungererer kun er for et avgrenset område. Mobildata fungerer innenfor dekningsområde til mobiloperatøren man bruker, som i 2022 vil si nesten over alt.

For at mobildata skal fungere må telefonen koble seg på en radiosender som enten står på et tak eller en øde posisjon som sender. I Norge er det tre mobiloperatører som har eget mobilnett, dette er Telia, Telenor og Ice [5].

Wi-Fi fungerer ved at brukeren har satt opp en ruter hjemme eller ved at brukeren kobler seg på et offentlig nettverk. Wi-Fi fungerer ved at det sender ut radio frekvenser for å sende informasjon mellom brukerens enhet og ruteren [4]. Ruteren mottar signaler fra internett operatøren enten via antenner eller via fiberoptiske kabler og oversetter disse signalene til radio frekvenser som kan snakke med telefonen.[7]

2.5 Kolumbus REST API

Denne appen handler jo om å finne bussholdeplasser i nærheten av brukeren og stoppe en bestemt buss som kjører forbi. Da trengs informasjon om både holdeplasser, bussruter og busser, noe som Kolumbus har offentlig tilgjengelig gjennom sin egen REST API. Her kan man hente ut relevant informasjon som blant annet navn, id, og geografisk posisjon, der spesielt geografisk posisjon er veldig relevant for denne oppgaven. Man kan også finne ut hvilke ruter som kjører forbi hvilke holdeplasser, noe annet som appen vil ha behov for.

Kapittel 3

Design og konstruksjon av programvare

I dette kapittelet vil det presenteres hvordan prosjektet ble opprettet og gjennomført. Her fokuserer det på enkeltområder av applikasjonen før det blir presentert en oversikt over hele systemet.

3.1 Oppbygning

3.1.1 Opprettelse og kjøring av prosjektet

Når man oppretter et nytt prosjekt med Flutter har man noen alternativer avhengig av om man vil opprette det i Android Studio, Visual Studio Code eller direkte i terminalen på PC'en. Alle resulterer i det samme; et standard Flutter prosjekt som man kan bygge videre til sitt eget, det er derfor ganske vilkårlig hvilken metode man går for. Et standard prosjekt består av en mappe med et valgfritt navn som igjen inneholder flere mapper og filer som konstruerer en enkel applikasjon med kun en knapp og tekst. Mer om denne filstrukturen samt hvordan man går frem for å bygge en egen app fra dette forklares i kapittel 3.1.2.

3.1.2 Filstruktur

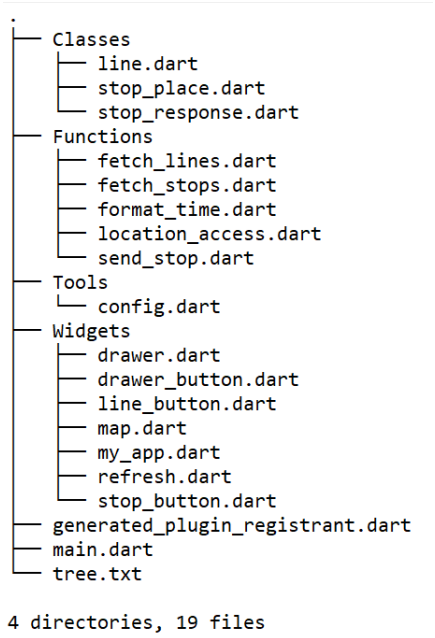
Ved opprettelse av et standard prosjekt i Flutter vil man få flere mapper som blant annet egne mapper for iOS og Android som har de nødvendige filene for å kompilere appen til native kode for hvert operativsystem, en build mappe og viktigst; en lib mappe der mesteparten av arbeidet foregår. I lib mappen vil man ha en main.dart fil som Flutter leter etter for å starte app'en og den fila vil være utgangspunktet for all koden. Inne i lib-mappa vil man også kunne strukturere koden slik man selv vil med mapper og filer. For å ha en god struktur og separere ansvar for forskjellige typer filer ble det satt opp fire mapper; Classes, Functions, Tools og Widgets.

I Classes ble alle filene som inneholder klasser lagt inn, primært klasser for å representere data som hentes inn fra Kolumbus sin API for å strukturere holdeplasser og bussruter på et format som passer i Dart fremfor å bare lagre en JSON-respons direkte.

I Functions ble filer som inneholder funksjoner som er uavhengige fra Widgets plassert. Dette er primært funksjoner knyttet til fetching av data fra Kolumbus API, men også funksjoner som bl.a konverterer tid fra formatet som ligger i API'en til det støttede DateTime-formatet til Dart, en funksjon som sjekker om man har tillatelse til å bruke posisjonstjenester på enheten, og en funksjon for å sende stopp-signal til bussen.

I Tools ligger det kun én fil, config.dart, som inneholder konfigurasjonsvaribler som det forklares mer om i 3.1.3.

I figur 3.1 så vises filstrukturen innad i lib-mappa.



Figur 3.1: Filstrukturen til prosjektet

3.1.3 Config-fila

Tidlig i utviklingsarbeidet ble det opprettet en config-fil som inneholder parametre som kan trenge finjustering før endelig utgave av app'en. Dette er spesielt med tanke på begrensningene som nevnes i 3.6, men også ting som for eksempel API tokens/nøkler. Her kan parametre som maksimal avstand til holdeplasser, hvor langt frem i tid man kan stoppe busser med mer endres lett. Fordelen med en slik config-fil er at etter hvert som programmet blir større, vil også mengden parametre bli større, og hvor disse parametrene brukes vil være på veldig forskjellige deler av koden, og noen ganger vil man også bruke samme parameter flere ganger.

Ved å legge det i en config-fil kan man samle alle slike parametre på ett enkelt sted for å ha oversikt, samt at det er lett å endre dem etter behov. Disse parametrene er veldig greit å ha muligheten til å endre raskt etter som det er vanskelig å bare velge en verdi for disse uten å ha testet ut hva som fungerer best. Det er i utgangspunktet ikke meningen at disse parametrene skal kunne endres av brukeren som innstillinger i app'en, men heller for oss

som utviklere å kunne endre de under veis. Endringen av disse parametrene vil kanskje være mest aktuelt i det tilfelle at app'en blir lansert offentlig, og at man kan få responser fra brukere. Om man får tilbakemeldinger knyttet til noen av tingene man allerede har en parameter på så er det enkelt å justere det.

3.2 Widgets

Som beskrevet i 2.1.3, så er en Flutter app bygd opp av widgeter. I kapittelet under beskrives de ulike widgetene som har blitt laget for at appen skal fungere. Hovedwidgeten er kalt My app som så kaller på Google maps widgeten slik at kartet kommer opp når man starter appen. Oppe i høyre hjørnet er knappen til Draweren som kaller på widgeten Drawer. Inne i Drawer blir Holdeplass-widgeten kalt på og når man trykker på en holdeplass widget så kommer Linje-widgeten opp. De neste underkapitlene gir en oversikt over hver enkelt widget, og den endelige widgetstrukturen kan man se i kapittel 3.2.6.

3.2.1 MyApp

I denne appen så fungerer MyApp som en slags container for alle andre widgets samt et ledd i kommunikasjonen mellom widgets. Spesielt informasjon fra kartet til drawer vil skje gjennom at en verdi sendes fra et trykk på en markør i kartet til MyApp, og så sørger MyApp for å sende denne verdien ned igjen til Drawer. MyApp inneholder også flere variabler for å holde styr på alle holdeplasser, samt hvilke holdeplasser som vises til brukeren. Det er også i MyApp at den første automatiske hentingen av brukerens posisjon hentes.

```
1   Widget build(BuildContext context) {
2     return MaterialApp(
3       debugShowCheckedModeBanner: false,
4       home: Scaffold(
5         key: scaffoldKey,
6         appBar: AppBar(
7           title: const Text('Stopp bussen'),
```

```
8             backgroundColor: const Color.fromRGBO(60, 180, ...
                  84, 1),
9           ),
10          body: hasLocation
11              ? Map(_currentLocation, closeStops, ...
                    onMarkerTap, markers)
12              : null,
13          drawer: hasLocation
14              ? Drawer(
15                  child: AppDrawer(
16                      _updateLocation,
17                      _currentLocation,
18                      closeStops,
19                      accuracy,
20                      stopPlaceLines,
21                      pressedId,
22                      autoFetch,
23                      autoDrawerResponse),
24                )
25              : null,
26        ),
27      );
28    }
```

Kode 3.1: Hovedwidgeten MyApp

3.2.2 Google Maps

I kodeutsnitt 3.2 ser man widgeten som er ansvarlig for å vise Google Maps kartet på telefonen. Under er de forskjellige parameterne som blir sendt inn til GoogleMap sin klasse. Disse parameterne blir brukt for å styre hva som blir vist på telefonskjermen. `initialCameraPosition` setter kamera posisjonen på kartet til brukerens lokasjon og dette skjer helt autmoatisk når brukeren åpner appen og denne widgeten kjører. En av de andre parameterne som blir sendt inn her er `markers` som setter på alle markørene i kartet som viser bussholdeplasser i nærheten.

```
1   Widget build(BuildContext context) {
2     return GoogleMap(
3       initialCameraPosition: CameraPosition(
4         target: LatLng(widget.location.latitude, ...
              widget.location.longitude),
5         zoom: 15,
```

```
6         ),
7         myLocationEnabled: true,
8         onMapCreated: (GoogleMapController controller) {
9           controller.setMapStyle(
10              '[{"featureType": "poi","stylers": ...
                    [{"visibility": "off"}]}]');
11          _controller.complete(controller);
12        },
13        mapType: MapType.normal,
14        markers: widget.markers,
15      );
16    }
17
18  }
```

Kode 3.2: Map-widgeten

3.2.3 Drawer

Drawer widgeten er bindeleddet mellom holdeplasser og busslinjer og MyApp. Draweren inneholder en header og body som vist i widgettreet i figur 3.2. Draweren sin header vist på figur 3.3 ved `DrawerHeader` inneholder en overskrift og en refresh knapp som gjør at brukeren kan oppdatere sin posisjon. I drawer headeren vil også brukeren også ha en stopknapp som kun er aktiv når variabelen `isStopped` er false. Drawer body begynner ved linje 30 og inneholder flere if for å sørge for at brukeren blir vist riktig informasjon om holdeplassene og at når brukeren trykker på en av holdeplassene at riktig informasjon blir sendt videre til holdeplass widgeten.

```
1   Widget build(BuildContext context) {
2     return SingleChildScrollView(
3       child: Column(
4         children: [
5           SizedBox(
6             height: 240.0,
7             child: DrawerHeader(
8               child: Column(
9                 children: [
10                  const Text('Finn holdeplass'),
11                  Text('Presisjon: ${widget.accuracy}'),
12                  Refresh(onRefresh, isStopped),
13                  if (!isStopped) Stop(selectedLine, onStop),
```

```
14                    if (isStopped) Cancel(selectedLine, ...
                          onCancel),
15                    if (message != '') Text(message)
16                  ],
17                ),
18              ),
19            ),
20            if (widget.stopPlaces.isEmpty) const ...
                  CircularProgressIndicator(),
21            ...widget.stopPlaces.map((stopPlace) {
22              if (widget.autoFetch && stopPlace.id == ...
                    widget.pressedId) {
23                fetchLines(
24                    stopPlace.id.toString(),
25                    stopPlace.distanceTo(widget.position),
26                    stopPlace.name.toString());
27              }
28              return DrawerButton(
29                  stopPlace,
30                  lines,
31                  widget.position,
32                  onStopPlaceClick,
33                  widget.pressedId,
34                  onLinePressed,
35                  selectedLine,
36                  isStopped);
37            })
38          ],
39        ),
40      );
41    }
```

Kode 3.3: Drawer widget

3.2.4 Holdeplass

Holdeplass widgeten gjør det mulig å se alle holdeplasser når brukeren har trykket på draweren. Under kan du se at widgeten består av en `ElevatedButton` som viser holdeplass navnet og hvor langt unna holdeplassen her. Denne knappen gjør det så mulig å se alle linjene, når knappen blir trykket på vil `visibility` endres slik at alle busslinjer vil vises. For at busslinjene fra valgt holdeplass skal vises blir det satt opp en sammenligning av to id'er og hvis de er like vises busslinjene.

```
1   Widget build(BuildContext context) {
2     return Column(
3       children: [
4         ElevatedButton(
5           child: Text(
6               '${stopPlace.name}: ...
                    ${stopPlace.distanceToAsString(position)}m'),
7           onPressed: isStopped ? null : () async {
8             await onStopPlaceClick(stopPlace.externalId,
9                 stopPlace.distanceTo(position), ...
                      stopPlace.name);
10          },
11          style: ElevatedButton.styleFrom(
12            minimumSize: const Size.fromHeight(40),
13            primary: const Color.fromRGBO(60, 180, 84, 1),
14          ),
15
16        ),
17        Visibility(
18          child: lines.isNotEmpty
19              ? Column(
20                  children: [
21                    ...lines.map((line) {
22                      return LineButton(line, ...
                            onLineSelect, selectedLine, ...
                            isStopped);
23                    }),
24                  ],
25                )
26              : const CircularProgressIndicator(),
27          visible: pressedId == stopPlace.externalId,
28        ),
29      ],
30    );
31  }
```

Kode 3.4: Holdeplass-widgeten

3.2.5 Linje

Linje widgeten har ansvaret for å vise busslinjene samt når bussen kjører. Kolumbus tilbyr to alternativer for busstider. Det ene er estimert tid for busser som allerede har begynt å kjøre, og det andre er den planlagte rutetiden. Om ruten har en estimert tid er det denne som vil vises til brukeren

siden den vil være oppdatert med tanke på om bussen er forsinket eller ikke. Ruter som har estimert tid vises med grønn tekst. Under på figur 3.5 kan man se de forskjellige tekstlinjene som blir vist til brukeren. Den inneholder også en egen funksjon ved `onPressed` som forteller Draweren hvilken busslinje brukeren har valgt. Den sender hele linje objektet som vist i 3.4.3 tilbake til draweren.

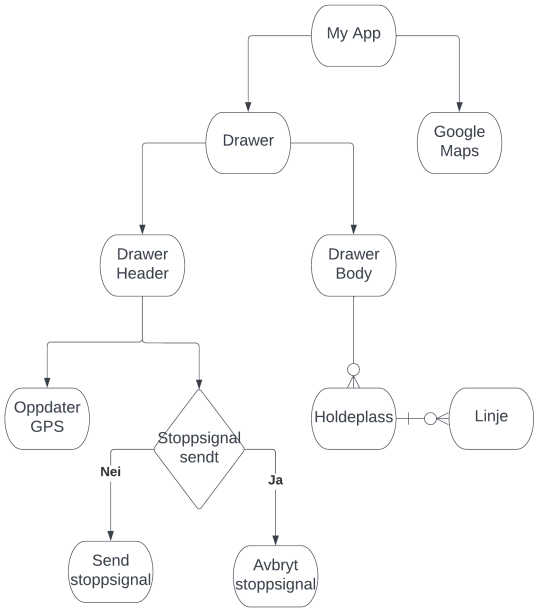
```
1   Widget build(BuildContext context) {
2     return ElevatedButton(
3       child: Column(
4         children: [
5           Text(
6             line.expectedDepartureTime != null
7                 ? '${line.lineName} mot ...
                      ${line.destination}\nAvgang:
8                 ${line.formattedTime( ...
                      line.expectedDepartureTime.toString())}'
9                 : '${line.lineName} mot ...
                      ${line.destination}\nAvgang:
10                ${line.formattedTime( ...
                      line.scheduleDepartureTime.toString())}',
11            textAlign: TextAlign.center,
12            maxLines: 3,
13            style: TextStyle(
14                fontSize: 15.0,
15                color: isStopped == true ? Colors.grey: ...
                      line.expectedDepartureTime != null
16                     ? Colors.green
17                     : Colors.black),
18          ),
19        ],
20      ),
21      onPressed: isStopped ? null : () {
22        onLineSelect(line);
23      },
24      style: ElevatedButton.styleFrom(
25        minimumSize: const Size.fromHeight(40),
26        primary:
27            line.id == selectedLine?.id ? ...
                  Colors.blueGrey[200] : Colors.white,
28      ),
```

Kode 3.5: Linje-widgeten

3.2.6 Endelig Widgetstruktur

For å vise hvordan widgetene i appen henger sammen ble det tegnet opp et widgettre (figur 3.2). Her vises de mest sentrale containerene, altså widgets som i seg selv ikke er synlige men som fungerer som en ramme for andre widgets. Widgetene som viser den viktigste informasjonen til en bruker eller som en bruker kan samhandle med. Et tre som dette kunne blitt tegnet i mye større detalj der man kunne gått ned på hver eneste lille widget, men siden alle widgetene allerede er forklart i 3.2 så vil dette treet primært gi et overblikk over sammenhengen i større grad enn å vise de små detaljene.

Det første er MyApp som fungerer som en container for de to primære delene av brukergrensesnittet, Drawer og Google Maps. Som vist i treet så inneholder Drawer igjen en header og en body. I headeren er knappen for å oppdatere posisjon og enten knapp for å bestille stopp, eller en knapp for å avbryte stopp. Her er det brukt symbolet for valg slik som i flytdiagram for å vise at en av disse knappene vises, men ikke begge. I bodyen er det brukt piler slik de som vil brukes i et relasjonsdiagram over databaser for å vise at det kan være null eller flere holdeplasser som holder til i drawer body. Det samme brukes for linje der samme pilen som til holdeplasser for å vise at en holdeplass kan inneholde null eller flere linjer.

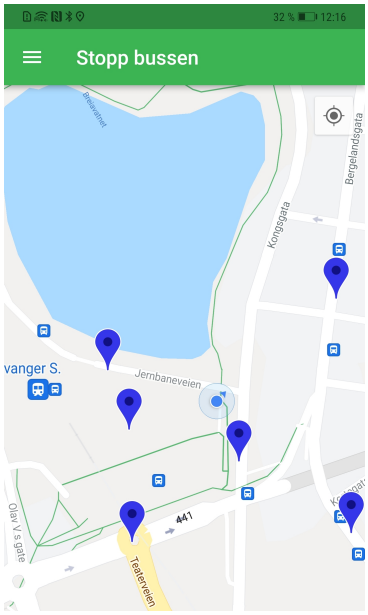


Figur 3.2: Oversikt over Widget Strukturen i appen

3.3 UI/UX

I dette kapitlet skal UI/UX av appen bli presentert. UI/UX står for for user interface og user experience som i hovedsak går ut på brukergrensesnittet til appen. Her var det mange muligheter for hvordan designet av appen skulle være, da det ikke var fastsatt så mange krav om design i oppgavebeskrivelsen.

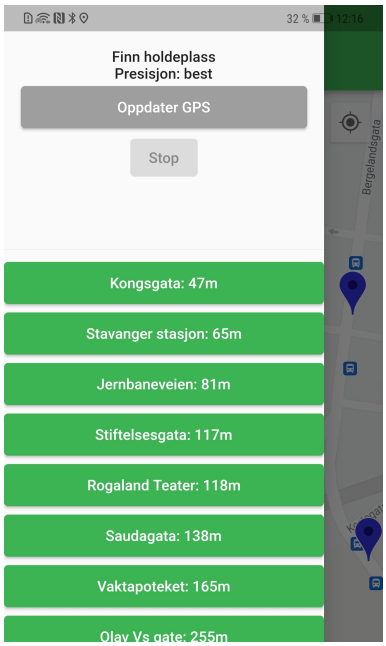
Det er to hovedsider brukeren kommer til å se. Den første siden brukeren ser når appen blir åpnet er et kart som i figur 3.3 denne viser Google Maps widgeten som beskrevet i kapittel 3.2.2. Her ser brukeren en blå markør som indikerer hvor brukeren er og man ser blå markører i kartet som er bussholdeplassene i nærheten av brukeren. På denne siden kan brukeren dra kartet rundt og se de forskjellige bussholdeplassene. Her kan brukeren trykke på en markør for å se alle holdeplasser. Brukeren får også se en knapp oppe i venstre hjørne som vil ta brukeren til neste hovedside.



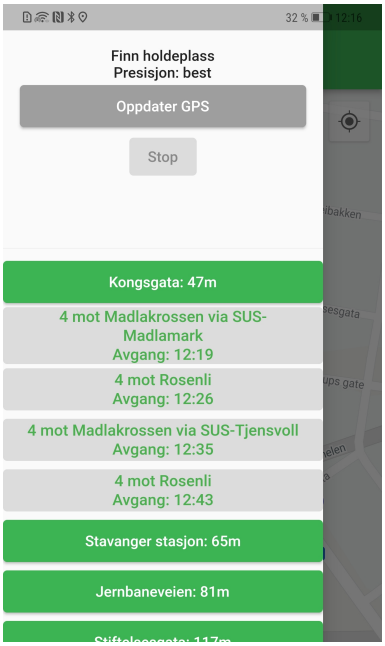
Figur 3.3: Hva brukeren møter når appen blir åpnet

I figur 3.4 ser man appen's andre hovedside. Her ser brukeren en liste med holdeplasser i nærheten av seg selv. Her vises Drawer-widgeten som beskrevet i kapittel 3.2.3. Det vises også en knapp med oppdater GPS som vil oppdatere presisjonen og brukerens GPS denne funksjonaliteten kan du lese mer i kapittel 3.5. Ved å trykke på en holdeplass vil brukeren se en liste over busser som går fra dette stoppet som vist i figur 3.5. Hvis teksten som vises på busslinjene er grønne vil det si at appen bruker en egen tid over når bussen er forventet å ankomme bussholdeplassen. Er fargen på teksten grå har ikke appen forventet tid og vil bruke tiden fra ruteinformasjonen.

3.3 UI/UX



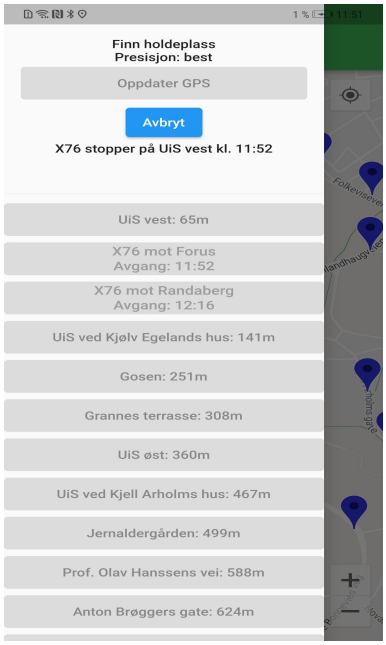
Figur 3.4: Liste over holdeplasser



Figur 3.5: Liste over busser som går fra valgt bussholdeplass

3.3 UI/UX

I figurene 3.6 og 3.7 ser man hvordan man kan sende et stopp signal og hvordan brukeren kan kansellere stopp bestillingen. For å sende et stopp signal velger brukeren en av busslinjene fra en holdeplass og deretter trykker på stoppknappen som nå har blitt aktivert. Da brukeren får en melding om at bussen stopper på valgt holdeplass vil alle knappene i draweren bli deaktivert og det eneste som er mulig for brukeren er å avbryte stoppsignalet. I kapittel 4.2.2 blir det diskutert mer rundt automatisk deaktivering av stoppsignal. Når brukeren kansellerer bestillingen om stopp vil det komme opp en melding som bekrefter at bestillingen er kansellert. Draweren vil aktivere knappene og funksjonaliteten igjen og brukeren kan nå sende inn ett nytt stoppsignal.



Figur 3.6: Viser kansellering av stopp bestiling

Figur 3.7: Viser en buss som skal stoppe

3.4 Hente data fra Kolumbus

Som nevnt i 2.5 så har Kolumbus en egen offentlig tilgjengelig REST API som kan brukes til å hente ut alt av data knyttet til holdeplasser, bussruter etc. I programmet bruker vi det slik at en liste over alle holdeplasser hentes inn når app'en åpnes og lagres til en variabel i minnet. Dataen filtreres videre i programmet basert på kriterier som diskuteres i 3.6. Den dataen som blir brukt i denne app-en er holdeplasser og avganger, og hvordan de hentes og brukes videre vil bli forklart i de neste underkapitlene.

3.4.1 Holdeplasser

Tidlig i utviklingen hentet programmet inn denne listen over holdeplasser hver gang man tok en refresh av GPS-posisjon og deretter filtrert før lista ble lagret. Med tanke på databruk, responstid og ytelse så ble dette fort endret til å hente inn og lagre all data kun én gang ved åpning av appen. Om man lukker appen, men lar den kjøre i bakgrunnen så vil fortsatt listen over holdeplasser være lastet inn. Dataen som vises til brukeren derimot kan endre seg, etter som hver gang man oppdaterer posisjon vil app'en gå gjennom alle holdeplasser og hente ut de som er innenfor aktuell avstand fra brukerens posisjon. Se kode 3.6 for hvordan holdeplasser hentes i appen.

```
1  Future<List<StopPlace>> fetchAllStopPlaces() async {
2    try {
3      final response = await http.get(
4          Uri.parse('https://api.kolumbus.no/api/
5          stopplaces?transportMode=bus'));
6      if (response.statusCode == 200) {
7        var jsonStopPlaces =
8            List<Map<String, ...
                dynamic>>.from(jsonDecode(response.body));
9        List<StopPlace> stopPlaces = [];
10       for (var i = 0; i < jsonStopPlaces.length; i++) {
11         var newStopPlace = ...
               StopPlace.fromJson(jsonStopPlaces[i]);
12         if (newStopPlace.modification != 'delete') {
13           stopPlaces.add(newStopPlace);
14         }
15       }
16       return stopPlaces;
```

```
17      } else {
18        throw Exception('Failed to load stop places');
19      }
20    } on Error {
21      throw Exception('No location data yet');
22    }
23  }
```

Kode 3.6: API kall for henting av holdeplasser

3.4.2 Avganger

I programmet har man også muligheten til å trykke på en bestemt holdeplass og hente inn avganger fra den. Ettersom denne dataen kan filtreres i forhold til hvilken holdeplass man vil ha avganger for direkte i API-kallet, så hentes denne dataen hver gang man ønsker å hente ut informasjon om avganger. Denne dataen hentes også hver gang man trykker på en holdeplass for å sørge for at tidspunktene for avgang stemmer, og at filtrering i forhold til nåværende tid er riktig. Spesielt på busser som allerede har begynt å kjøre så vil de ha en estimert tid i tillegg til den planlagte tiden og denne kan endres ofte, dermed er det lurt at denne informasjonen er så ny som mulig når man henter inn avganger. Se kode 3.7 for for hvordan avganger hentes ut i appen.

```
1  Future<List<Line>> fetchStopPlaceLines(String id, double ...
       distance) async {
2    try {
3      final response = await http.get(
4          Uri.parse('https://api.kolumbus.no/api/
5          stopplaces/$id/departures'));
6      if (response.statusCode == 200) {
7        var jsonLines =
8            List<Map<String, ...
                 dynamic>>.from(jsonDecode(response.body));
9        List<Line> allLines = [];
10       for (var i = 0; i < jsonLines.length; i++) {
11         if (jsonLines[i]['schedule_departure_time'] != ...
               null) {
12           var newLine = Line.fromJson(jsonLines[i]);
13           if (_validLine(newLine, distance)) {
14             allLines.add(newLine);
15           }
```

```
16          }
17        }
18
19        return allLines;
20      }
21      if (response.statusCode == 204) {
22        List<Line> Line = [];
23        return Line;
24      } else {
25        throw Exception('Failed to load lines');
26      }
27    } catch (e) {
28      print(e);
29      throw e;
30    }
31  }
```

Kode 3.7: API kall for henting av avganger

3.4.3 Interne objekter for avganger og holdeplasser

For å lettere håndere informasjonen knyttet til holdeplasser og avganger i andre deler av programmet blir JSON-objektene som hentes inn lagret til dedikerte objekter kalt Line og StopPlace. Ved å bruke en *factory* i Dart kan man sende inn et JSON-objekt og gjøre dette om til et objekt av en forhåndsdefinert klasse, og denne implementasjonen vises for henholdsvis Line og StopPlace i kode 3.8 og 3.9.

```
1  factory Line.fromJson(Map<String, dynamic> json) {
2      return Line(
3          id: json['id'],
4          lineNumber: json['line_number'],
5          lineName: json['line_name'],
6          destination: json['destination'],
7          platformExternalId: json['platform_external_id'],
8          scheduleDepartureTime: ...
               json['schedule_departure_time'],
9          expectedDepartureTime: ...
               json['expected_departure_time'],
10         transportSubMode: json['transport_sub_mode'],
11         boarding: json['boarding'],
12         tripId: json['trip_id'],
13         order: json['order']);
```

```
14 }
```

Kode 3.8: JSON Factory innad i Line-objektet

```
1  factory StopPlace.fromJson(Map<String, dynamic> json) {
2      return StopPlace(
3        id: json['id'],
4        externalId: json['id'],
5        nsrId: json['nsrId'],
6        name: json['name'],
7        description: json['description'],
8        latitude: json['latitude'],
9        longitude: json['longitude'],
10       modification: json['modification'],
11       type: json['type']
12     );
13 }
```

Kode 3.9: JSON Factory innad i StopPlace-objektet

3.5 Hente brukerens posisjon

For å hente inn brukerens posisjon brukes pluginen Geolocator som ble nevnt i kapittel 2.3.1. Det som brukes i den primære funksjonen for å hente inn brukerens posisjon er Geolocators innebygde metode `getCurrentPosition()` som asynkront henter inn brukerens posisjon fra enhetens posisjoneringstjeneste.

For best opplevelse ønsker man å ha så høy presisjon som mulig, men under tidlig testing ble det oppdaget at i noen situasjoner tok det veldig lang tid å hente posisjon, ofte 10-20 sekunder eller mer, og dette skapte en lite smidig brukeropplevelse. Derfor ble det gjort et valg om å sekvensielt hente inn posisjonen i flere omganger, der posisjon hentes med lav presisjon først og deretter blir bedre. Tanken bak dette var at lav presisjon kanskje var raskere å hente inn, noe som viste seg å stemme.

I kode 3.10 ser man hvordan funksjonen `_updateLocation()` består av en for-løkke som går gjennom verdiene 0, 2 og 4, som henviser til laveste, middels og høyeste presisjon i Geolocator. For hver iterasjon hentes posisjon

med spesifisert presisjon og når en posisjon returneres blir denne satt til variabelen `_currentLocation`.

Videre brukes funksjonen `getCloseStopPlaces()` som ble laget for å finne holdeplasser innefor gyldig radius fra brukeren. Parametrene som sendes inn er brukerens posisjon samt listen over alle holdeplasser og vil da returnere til variabelen `closeStops` en liste med de nærmeste holdeplassene som vises til brukeren. Det siste som gjøres er å fjerne gamle markører fra kartet samt å legge inn de nye markørene basert på de nye aktuelle holdeplassene.

```
1  Future _updateLocation() async {
2      hasLocation = false;
3      for (var i = 0; i < 5; i += 2) {
4        await Geolocator.getCurrentPosition(
5                desiredAccuracy: LocationAccuracy.values[i])
6            .then((Position position) {
7          setState(() {
8            hasLocation = true;
9            _currentLocation = position;
10           pressedId = '';
11           accuracy = LocationAccuracy.values[i]
12               .toString()
13               .replaceAll('LocationAccuracy.', '');
14           closeStops = ...
                 getCloseStopPlaces(_currentLocation, allStops);
15         });
16       });
17     }
18     markers.clear();
19     setMarkers(closeStops);
20 }
```

Kode 3.10: Innhenting av posisjon

Denne funksjonen vil alltid kjøre automatisk ved oppstart av app'en, og deretter vil den kjøre kun hvis en bruker trykker på knappen for å oppdatere posisjon. Det ble vurdert å utvide dette til en kontinuerlig eller periodisk oppdatering av posisjonen, men det ble vurdert at til dette formålet var en manuell oppdatering kanskje bedre. Tankegangen bak dette var at siden listen over tilgjengelige holdeplasser vil oppdateres hver gang posisjonen oppdateres så kunne det føre til forvirring eller frustrasjon hos en bruker om den holdeplassen de skulle trykke på plutselig går ut av listen eller flytter på seg som et resultat av oppdateringen. En annen del av denne

vurderingen var også at på grunn av begrensingen på avstanden man kan være fra holdeplassen så vil nok en bruker allerede være ganske nære den holdeplassen de ønsker å stoppe en buss på når appen åpnes. Det vil bety at det ikke egentlig betyr så mye om det er noen holdeplasser til eller fra som kanskje ikke stemmer 100% med posisjon og begrensinger, så lenge brukerens ønskede holdeplass er i listen.

3.6 Begrensinger/Validering av bruk

En sentral del av denne appen er å begrense hvilke og hvor mange busser en bruker kan stoppe. Det er to primære grunner til at det gjøres noen begrensninger og det er for det første å hindre misbruk ved å blant annet stoppe mange busser man ikke skal gå på; og for det andre er det å vise brukeren de mest relevante bussene å stoppe.

Den første begrensingen som blir gjort er å filtrere holdeplassene til å kun vise brukeren holdeplasser innenfor en viss radius fra der brukeren er. Denne begrensingen gjøres fordi man kun skal ha muligheten til å stoppe en buss på en holdeplass man er nærme nok til å være til stede på innen bussen kommer.

En annen begrensning er å kun vise busser som kommer på holdeplassen innenfor en viss tid. Dette er igjen knyttet til å i størst mulig grad sørge for at en bruker mest sannsynlig skal og kan gå på den bussen som stoppes. En situasjon som kan oppstå uten noen begrensning er for eksempel at en bruker stopper en buss noen timer frem fordi de tenker de skal gå på den, men ikke fjerner signalet om de ombestemmer seg. Det er selvfølgelig mulig å si at dette er et lite problem, og at sjansen er stor for at noen andre skal på den bussen på det stoppet uansett, men en app som dette skal kunne bidra til en optimalisering av busstilbudet, slik at selv små problemstillinger kan være verdt å ta hånd om.

Videre ble valideringen utvidet for å ta hensyn til hvor lang tid det er forventet at man trenger for å gå til holdeplassen. For å gjøre dette ble det tatt utgangspunkt i en gangfart på 1.42m/s [8], som deretter blir brukt sammen med avstand til holdeplassen for å estimere tiden det vil ta å gå til holdeplassen. Tiden finner man som vist i figur 3.1 der d er avstand til

holdeplass, v er hastighet, altså 1.42m/s og tiden t er resultatet man får ut. Dette vil være i sekunder da det brukes m og m/s i utregningene. Denne utregningen vil videre brukes i filtreringen av hvilke busser man ser for hver holdeplass. Tiden vil sette en grense fra tidspunktet man laster inn bussene på og frem t sekunder hvor disse bussene ikke vises siden det er sannsynlig at man ikke rekker disse.

Hvis spørsmålet er om brukeren rekker en buss som kommer om 2 minutter og bussholdeplassen er 350 meter unna, så er det flere faktorer som spiller inn. Har brukeren en standard gangfart på 1.42m/s blir det litt vanskelig å få det til. For å rekke denne bussen, må man ha en gangfart på omtrent 2,91 m/s, som vil si at brukeren må jogge/løpe for å rekke denne bussen. Så selvom det er mulig å rekke bussen, så er det ikke sikkert man får det opp eller kan stoppe bussen på applikasjonen. Men når vi tar hensyn til valideringen, tar vi i utgangspunkt til et vanlig gangfart.

$$\frac{d}{v} = t \tag{3.1}$$

3.7 Stoppsignal

Den mest sentrale funksjonaliteten i app'en vil selvfølgelig være muligheten til å sende et signal til en buss om å stoppe. Den tekniske implementasjonen av dette innebærer å sende en HTTP request til et dedikert API. Dette API-et har blitt konstruert av Webstep på oppdrag fra Kolumbus, og vil sørge for å sende signal om stopp til bussen når forespurt holdeplass er den neste på ruta. Fordelen med det fra perspektivet til app-en er at den bare kan sende ut signalet om å stoppe en buss i det brukeren trykker på stopp, og så vil systemet til Webstep sørge for resten.

Ved bruk av dette API-et trenger man en unik API token for å ha tillatelse til å sende inn requests. Vi fikk for dette prosjektet tildelt en unik token som kunne brukes til å sende inn requests, og denne nøkkelen ble lagret til config-fila slik at det var en lett tilgjengelig variabel.

Dette API-et har to alternativer; en POST request som tar inn informasjon om hvilken buss som skal stoppe og hvor den skal stoppe, og en DELETE

request som sørger for å kansellere en bestilling om stopp. Hver av disse endepunktene vil utdypes mer i de kommende underkapitlene.

3.7.1 Bestille stopp

For å bestille et stopp hos en buss så er det tre forskjellige typer med informasjon som må sendes inn til API-et. Den første er Trip ID som er en ID som ikke bare er unik for bussen eller ruten, men for den spesifikke turen en buss er på. Dette er fordi API-et kan i prinsippet motta bestilling om stopp i ubestemt lang fremtid, slik at det er viktig å vite eksakt hvilken avgang det er snakk om. Det andre som trengs er Stop Sequence som refererer til den holdeplassen man ønsker å stoppe på. Stop Sequence betyr i denne sammenhengen hvilket nummer en holdeplass er i rekkefølgen på alle holdeplasser for turen bussen er på. Dette er informasjon som hentes for avganger, og heter `order` i objektet for avganger, vist i kode 3.8. Dette brukes i steden for den unike ID'en til en holdeplass ettersom noen bussruter kjører innom samme holdeplass flere ganger, og det ville derfor ikke vært tydelig hvilken av gangene den skulle stoppet der, men med Stop Sequence er dette tydelig. Det siste som trengs er Active Date som sier hvilken dato den aktuelle ruten hører til.

Disse tre delene med informasjon samles i et JSON-format og sendes til API-et, og den eksakte implementasjonen i app-en vises under i kode 3.7.1.

```
1  Future<StopResponse> sendStop(Line line) async {
2    try {
3      final response = await http.post(
4          Uri.parse(
5              'https://sanntidng-public-apim-test.azure-api.net/
6              V1-2/digital-stop'),
7          headers: <String, String>{
8            'Content-Type': 'application/json',
9            'Ocp-Apim-Subscription-Key': API_KEY,
10         },
11         body: jsonEncode({
12           'tripId': line.tripId,
13           'stopSequence': line.order,
14           'activeDate': ...
                 formatTime(line.scheduleDepartureTime, ...
                 'yyyy-MM-dd'),
```

```
15          }));
16      return StopResponse.fromJson(
17          Map<String, dynamic>.from(jsonDecode(response.body)));
18    } on Error {
19      throw Exception();
20    }
21  }
```

Kode 3.11: kode for bestilling av stopp

3.7.2 Kansellere stopp

Kansellering av stopp bruker nøyaktig den samme informasjonen som for å bestille stopp, slik at det bare er å matche innkommende kansellering mot en eksisterende bestilling for å se om man skal avbryte eller ikke. I prinsippet er den eneste forskjellen at ved kansellering så er det selvfølgelig en delete request isteden for en post request, og så legges all informasjon direkte i URL-en i steden for som et JSON-objekt. Igjen vises den eksakte implementasjonen under i kode 3.7.2.

```
1  Future<bool> cancelStop(Line line) async {
2    try {
3      var activeDate = ...
           formatTime(line.scheduleDepartureTime, 'yyyy-MM-dd');
4      var tripId = line.tripId;
5      var stopSequence = line.order;
6      final response = await http.delete(Uri.parse(
7          'https://sanntidng-public-apim-test.azure-api.net/
8          V1-2/digital-stop/$activeDate/$tripId/$stopSequence'),
9          headers: <String,String>{
10           'Ocp-Apim-Subscription-Key': API_KEY
11         });
12     bool validCancel;
13
14     response.statusCode == 200 ? validCancel = true : ...
           validCancel = false;
15     return validCancel;
16   } on Error {
17     throw Exception();
18   }
19 }
```

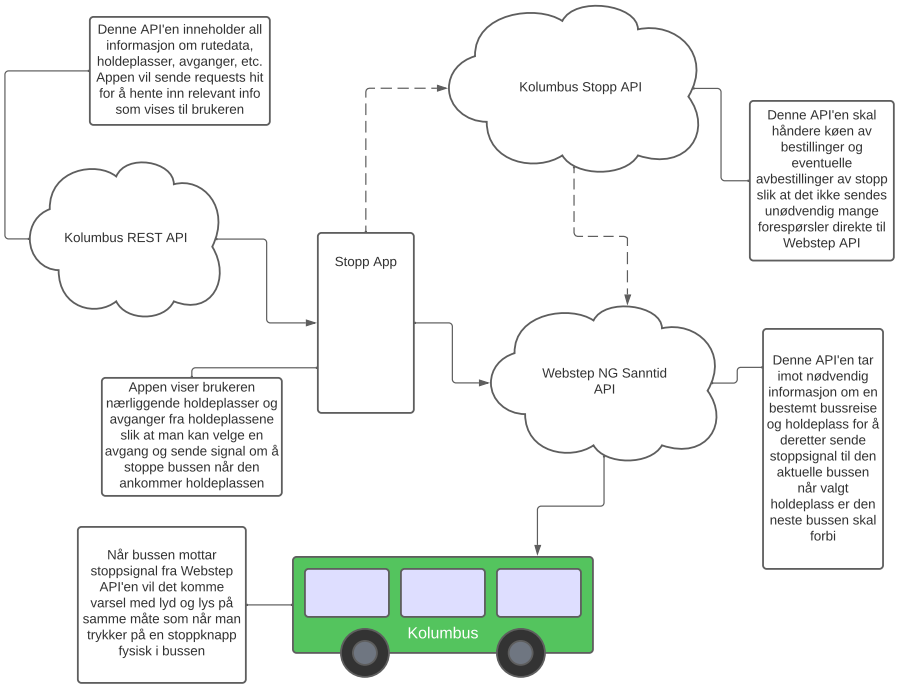
Kode 3.12: kode for kansellering av stopp

3.8 Komplett systemarkitektur

Dette prosjektet har bestått av å utvikle selve appen som lar en bruker finne en aktuell buss og sende signal for å stoppe den, og for å få dette til har det vært nødvendig å bruke andre tjenester/API'er enten utviklet internt i Kolumbus eller hos andre. De tjenestene har blitt forklart tidligere i rapporten, slik at dette kapitlet vil hovedsakelig ta et mer overordnet blikk og se på hvordan de forskjellige tjenestene, appen inkludert, henger sammen for å muliggjøre hele løsningen.

Figur 3.8 er en oversikt over hele systemet, der de skyformedeboblene er skytjenester/API'er som appen enten sender data til eller mottar data fra, rektangelet markert med *Stop App* representerer en enhet som appen kjører på, og nederst på figuren er en buss. Streker uten pil henviser til en faktaboks og streker med pil viser hvilken vei informasjonen går. De stiplede pilene viser hvordan dataen vil flyte i hele systemet ved endelig produksjon, men grunnet at systemet markert *Kolumbus Stop API* ikke er klart til bruk så kan ikke det brukes enda. Dermed sendes signal direkte til *Sanntid API* som sender stoppsignalet videre til en buss. Dette ville vært en dårlig løsning ved endelig produskjon fordi denne tjenesten har ingen mulighet for å registrere hvor signalet ble sendt fra, og vil kun reagere på sist sendte bestilling eller avbestilling av stopp, altså om noen sender avbestilling et sted flere har bestilt stopp så vil alle bestillinger for det stoppet avbrytes. Dette er ting som *Kolumbus Stop API* vil ta seg av når det er ferdig, men så lenge det kun er i testfase med på det meste 2-3 enheter om gangen så er det ikke en stor problemstilling om denne snarveien tas.

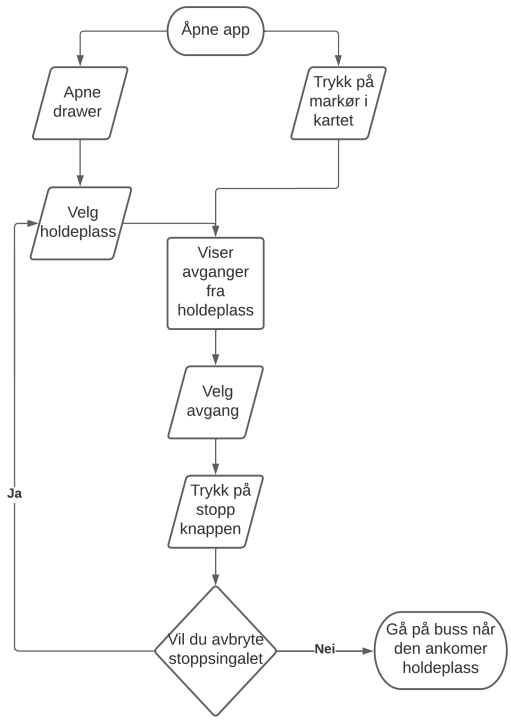
3.8 Komplett systemarkitektur



Figur 3.8: Oversikt over systemet

3.9 Flytdiagram

Med tanke på at mange andre kapitler i denne rapporten fokuserer mye på tekniske detaljer så ble det gjort en vurdering på at det kunne passe inn med et flytdiagram som ser det fra den andre siden, nettopp brukerens perspektiv. Figur 3.9 viser flytdiagrammet for den tenkte bruken av appen, og kan kanskje minne litt om en User Story på måten den er lagt opp.



Figur 3.9: Flytdiagram sett fra brukerens perspektiv

Kapittel 4

Resultater og diskusjon

4.1 Resultater

I dette kapittelet vil resultatene fra prosjektet legges frem og sammenliknes med målene som ble satt før arbeidet begynte. Dette vil bli delt opp i hovedmål og sekundærmål som nevnt i underkapitlene 1.3.1 og 1.3.2. For hver gruppe med mål vil det vurderes om målene ble nådd eller ikke og en kort kommentar på hvorfor det ble vurdert slik. En større refleksjon rundt resultatene, spesielt de som eventuelt ikke ble nådd, kommer i kapittel 4.2.

4.1.1 Hovedmål

For dette prosjektet var det en liste med 5 hovedmål som var de viktigste punktene for at app'en skulle ha nok funksjonalitet for å fungere som tenkt. I dette underkapittelet vil hver av de sees på med tanke om de ble gjennomført eller ikke.

1. *Appen skal ha et kart som viser hvor brukeren er.* Dette er absolutt gjennomført og er det første en bruker ser når appen åpnes.

2. *Appen skal ha en liste over holdeplasser og en liste over avganger som går*

gjennom holdeplassen. Dette er gjennomført og ligger i drawer'en i appen, og er løst med at man ser en liste med holdeplasser, og ved valg av en holdeplass ser man avganger fra holdeplassen.

3. *Det skal være mulig for en bruker å sende et signal til bussen om at den skal stoppe på en bestemt holdeplass.* Dette er utført og løst slik at man kan velge en holdeplass, deretter en avgang og så når man trykker på stopp vil signalet om dette sendes videre. Bussen mottar signalet når valgt holdeplass er den neste på ruta.

4. *Validering av GPS data. En bruker får kun stoppe bussen hvis brukeren er innenfor en viss avstand fra bussholdeplassen.* Dette er utført gjennom å kun vise brukeren holdeplasser innenfor en bestemt avstand, satt til 700m under testing. Denne avstanden kan endres fra config-fila.

5. *En begrensning på hvor mange stoppsignal en bruker kan sende i løpet av en tidsperiode.* Det er satt inn en slags begrensing på dette, men den er ikke basert på tid. Om man sender et stoppsignal vil muligheten for velge ny avgang og sende stoppsignal være deaktivert frem til man eventuelt avbryter det første stoppsignalet. Dermed er det en implisitt begrensning på maks en aktiv stopp om gangen, men vi valgte altså ikke å koble den mot et bestemt tidsrom.

4.1.2 Sekundærmål

Videre var det tre sekundærmål som var knyttet til utvidet eller mer avansert funksjonalitet som sannsynligvis kunne bli implementert i prosjektet.

1. *Enkel reiseplanlegger. Brukeren legger inn hvor man reiser fra og hvor man skal av. Appen sender holdeplassene for start og ende til bussen. Bussen stopper dermed både på start og ende uten at brukeren gjør noe mer.* Dette ble ikke implementert av to hovedgrunner. For det første tok det litt mer tid enn forventet å få tilgang til API'et som gjorde det mulig å sende signaler til buss, og når dette endelig var på plass var det usikkerhet rundt hvor mye tid en slik løsning ville ta å utvikle. For det andre var det heller ikke mulig å faktisk stoppe busser i starten, bortsett fra noen få som var utstyrt med testenheter, dermed ville det være ekstremt vanskelig å faktisk teste en slik løsning.

2. *En mer avansert validering som tar høyde for om brukeren vil rekke bussen en viss avstand unna på begrenset tid.* Dette ble det implementert en enkel variant av som måler avstanden mellom brukerens posisjon og holdeplasser i luftlinje. Deretter ved å regne ut en estimert tid man vil bruke på å gå denne avstanden så vil avgangene som går før man er estimert til å komme fra ikke vises. Denne løsningen kunne blitt utvidet til å gi mer eksakte estimat, men denne løsningen fungerte til formålet.

3. *Brukeren kan trykke på kartet for å velge en holdeplass isteden for å velge den fra lista for å stoppe bussen derfra.* Dette ble utført og var primært for å gjøre det mer intuitivt for brukere som ikke nødvendigvis legger merke til ikonet for drawer'en umiddelbart. Det ble løst gjennom å plassere markører i kartet ved holdeplassene som man kan trykke på, disse vil automatisk åpne drawer'en med den holdeplassen man trykket på markert.

4.2 Diskusjon og videre utvikling

Her vil det gjennom flere underkapitler reflekteres rundt hva som var grunnen til at noen mål ikke ble nådd samt hvor godt vi selv tenker vi løste de målsettingene vi oppnådde. Det vil også komme vurderinger på økonomisk og miljømessig kostnad av programmet og en drøfting av hvilken videre utvikling som kan gjøres med programmet. På punktene som omhandler videre utvikling vil noen av dem være mål vi ikke nådde, andre vil være eventuelle utvidelser av eksisterende funksjonalitet. Noen andre punkter igjen vil være utvidelser som ikke var mulig å få til på dette tidspunktet grunnet ytre begrensninger. Det vil også reflekteres rundt sikkerhetsmessige aspekter som kan oppstå ved en fullstendig utrulling av appen til brukere.

4.2.1 Videre utvikling av sekundærmål

To av sekundærmålene er det allerede implementert en løsning på, men det er også på sekundærmålene det er mest rom for potensiell videre utvikling om man hadde mer tid å utvikle appen på.

For å starte med det første sekundærmålet som handler om å kombinere

stopp-appen med en reiseplanlegger så er dette noe som ikke ble implementert i dette prosjektet. Det er derimot noe som kan være veldig aktuelt på et senere stadie om Kolumbus integrerer stopp-app funksjonaliteten inn i Sanntid-appen, etter som den allerede har en reiseplanlegger. I prinsippet er ikke det alt for avansert å legge til, men det krever en mulighet for å velge to holdeplasser for samme buss, hvorav en av dem sannsynligvis er utenfor den lovlige radiusen som er satt opp i denne versjonen av appen. Det vil også være mest nyttig om reisen innebærer bussbytte at også den neste bussen man skal gå på får stoppsignal uten at brukeren må gjøre noe mer enn å velge hvor man skal fra og til. Dermed vil stopp-funksjonaliteten sammen med Sanntidsappen fungere som en virtuell reiseguide. Det andre sekundærmålet med å vurdere om man vil rekke en buss eller ikke er implementert i en enkel variant, men det største problemet"med den løsningen er at den måler avstand i luftlinje. Det kan fungere greit nok for å teste at filtreringen fungerer, men vil nok gi feil avstand og tid i virkeligheten. Den mest naturlige videreutviklingen av denne funksjonaliteten er å implementere en mer avansert vurdering av avstand og tid, for eksempel ved å bruke funksjoner i Google Maps eller et annet API for å finne gangavstand i kart. I stor grad ble dette ikke tatt med i oppgaven da denne funksjonaliteten i Google Maps vil koste penger å bruke, og ved å bruke en annen gratis tjeneste ville det bety å blande inn enda flere tredjepartstjenester.

4.2.2 Automatisk deaktivering av stoppsignal

Det som kanskje er det mest åpenbare problemet med appen slik den er nå er at om man har sendt signal for å stoppe en buss så vil muligheten for å velge et nytt stopp bli deaktivert med mindre man avbryter det signalet man har sendt. Det er gode grunner knyttet til potensiell feil bruk eller misbruk som forklarer hvorfor dette valget ble tatt, men problemet oppstår om man har stoppet en buss og deretter skal stoppe en ny buss senere. Per nå så vet ikke appen om bussen har kjørt forbi stoppet eller ikke, og dermed vil muligheten for å velge en ny avgang på et senere tidspunkt være permanent deaktivert. Det løses relativt enkelt ved å bare lukke appen og åpne den på nytt, men det skaper ikke den mest sømløse brukeropplevelsen. Dette er nok også noe som kan bli bedre ved integrasjon inn i Sanntids-appen etter som den har posisjonen til bussene i sanntid, og dermed kan gi en respons når en buss stopper på valgt holdeplass.

4.2.3 Testing

I løpet av dette prosjektet har vi lagt vekt på høy grad av manuell testing for å se om nye widgets, funksjoner, etc. fungerer slik de skal. Det har fungert godt nok og så langt har ingen sentrale funksjoner hatt problemer med å fungere, samt at feil som har oppstått har blitt debugget og fikset manuelt. Testingen har dermed gått ut på å prøve ut funksjoner i appen, se om resultatet er slik det skal være, og om det ikke fungerte så har det blitt manuell code review med bruk av `Console.WriteLine()` for å skrive ut verdier som var ønskelig å undersøke.

Ved å se tilbake på prosjektet etter det er ferdigstilt ville kanskje større bruk av automatiske tester gitt raskere oppdagelse av feil samt bedre løsninger på noen områder. Det som nok er den største grunnen til at automatiske tester ikke ble brukt på dette prosjektet er for lite erfaring med testing blant oss på gruppa. Dette fører til en arbeidsmetode som går mer ut på å starte med et enkelt utgangspunkt og så teste dette manuelt til den grunnleggende infrastrukturen fungerer. Deretter blir prosessen med å legge til ny funksjonalitet, teste manuelt og deretter fikse eventuelle feil repetert for nye funksjoner. Det hadde nok også vært lettere å legge til automatiske tester om det ble gjort et større arbeid før programmeringen begynte med å planlegge den tekniske arkitekturen slik at tester kunne blitt laget mot de widgetene og funksjonene som skulle være med.

4.2.4 Sikkerhet og pålitelighet

Som nevnt i kapittel 3.8 fungerer appen mot en back-end som videresender stoppsignal til bussen, og dette er en løsning som har fungert fint for å teste at det er kontakt, men det gir en implementasjon med lav grad av sikkerhet med tanke på en offentlig utrulling av appen. Dette er fordi det nå ikke er noen formening om brukere eller innlogging i systemet, samt at det ligger en API nøkkel for Stopp API'et direkte i kildekoden for appen. Dermed vil det være spesielt to ting som kan gjøres for å øke sikkerheten ved endelig produksjon; nemlig å implementere en løsning for å registrere og logge inn brukere, og å sette opp enda en back-end som håndterer validering av brukere før stoppsignalet sendes videre til Stopp API.

Noe annet som kan bidra til loggføring og oppfølging av bruk på stopp-funksjonaliteten er å koble det opp mot APC-systemet (Automatic Passenger Counter) til bussene. Dette er et kamerasystem som kan registrere antall passasjerer og om noen går av eller på bussen på en holdeplass. Ved å finne et forhold mellom hvor mange ganger det faktisk det er noen som går av eller på bussen og hvor mange ganger stoppsignalet kommer gjennom app så kan man begynne å finne et tall på graden av misbruk eller feilbruk. Hvis man på dette tidspunktet også har registrering av brukere vil det bety at man kan legge inn en mulighet for å eventuelt svarteliste brukere som ved gjentatte anledninger feilaktig stopper busser.

Noe som naturlig henger litt sammen med sikkerhet i et program er pålitelighet. Under utviklingen av denne appen har det i all hovedsak basert seg på en "happy flow", altså det ønskede scenariet der alt fungerer som forventet. Det handler i stor grad om at Stopp API'et ikke har vært koblet opp mot et stort nok antall busser til å oppleve noen situasjoner der ting ikke har fungert så langt. Det man dermed kan gjøre er å spekulere i potensielle områder systemet kan feile på, og hvordan man kan håndtere dette på en best mulig måte for sluttbrukerne. Kanskje det enkleste stedet å starte er om signalet som sendes til Stopp API enten skulle være feil eller ikke komme gjennom. Da vil ikke appen motta en successrespons, noe som vil være enkelt å håndtere innad i appen ved å vise et varsel til brukeren som oppfordrer til å stoppe bussen på den gamle"måten ved å trykke på fysisk knapp i buss eller vifte med armen. Noe annet som kan skje er hvis Stopp API mister kontakt med en eller flere busser. Da hadde det vært en fordel å utvide kommunikasjon mellom app og Stopp API slik at API'et kan varsle brukere som har sendt stopp til en buss som mangler kontakt. Om det også skulle blitt oppdaget en større systematisk feil burde alle brukere motta et varsel som oppfordrer å stoppe bussen på det gamle"måten inntil videre. Det siste som kan bidra til god opplevd pålitelighet hos sluttbrukere er om bussen kan sende et signal tilbake til brukere når stoppen i bussen aktiveres. Da vil brukerene få en konkret bekreftelse på at bussen stopper der man ønsker. Dette er foreløpig ikke mulig slik teknologien i bussene er.

4.2.5 Utvidede parametere som blir sendt til bussen

En mulighet man kunne hatt ved videre utvikling av appen er å ha flere parametre man sender inn sammen med stoppsignalet som er av betydning

for busssjåføren. Kanskje det mest aktuelle her er om man for eksemel bruker rullestol eller av andre grunner trenger at en rampe i bussen legges ut for å gå om bord. Om man kunne sagt fra om dette når stoppsignalet sendes vil det kunne føre til at bussjåføren raskere får rampen ut siden sjåføren allerede er opplyst om det.

4.2.6 Økonomi og miljø

Dette er et hypotetisk spørsmål siden denne appen ikke skal kjøre som en egen app, men integreres med sanntidsappen. Det ville nok hatt stor betydning på antall sessions. Her gjør vi et estimat på hvor mye det hadde kostet når vi bruker tall fra Kolumbus. I februar måned 2022 hadde Kolumbus ca 65 000 brukere av sanntidsappen og hver av disse brukerne hadde i snitt ca 17 sessions hver. Det gir oss et totalt antall sessions på ca 775 000. Det er nå det vanskelig spørsmålet kommer inn da vi ikke vet hvor mange av disse sessions som ville ha brukt en eventuell stoppknapp, vi vil derfor gå ut ifra at ca 500 000 ville brukt stopp funksjonaliteten i appen for å få et grovt estimat av hvor mye det vil koste å kjøre appen med karttjenesten Google Maps. Kolumbus sin sanntidsapp har allerede kart tjenester så derfor må vi tenke at vi skal kjøre dette som en egen app. Utifra google api sin prisliste ligger prisen på 500 000 sessions per måned ligge på ca 3000 dollar som i dag er ca 27 000 nok. Etter 500 000 sessions oppgir ikke google maps noe pris og man bes om å ta kontakt med google for å få et prisestimat, ved en så stor app så er det nok lurt for å få en bedre pris.

Som nevnt i 4.2.4 er sikkerhet veldig viktig for appen, hvis det er noen som klarer å få tilgang til appen og stoppe alle bussene en dag, vil det få en negativ innvirkning på miljøet da alle bussene vil stoppe på alle stoppene. Ved at appen fungerer slik som det er tenkt vil appen ha et positiv innvirkning på miljø ved at bussene stopper kun ved de riktige holdeplassene og ved at folk slipper å vinke inn feil buss. Dette har nok ikke veldig stor betydning på miljøet, men vi tenker det er greit å få med et lite avsnitt om det da klima stadig blir viktigere ved de store klimaendringene idag.

4.2.7 Stopp-app sammen med selvkjørende buss

For å tenke videre utvikling på lang sikt så vil muligheten til å stoppe en buss fra app fungere veldig godt sammen med selvkjørende busser. På dette tidspunkt har Kolumbus én selvkjørende buss i rute som et supplement til det ordinære kollektivtilbudet, men dette er ikke utenkelig at vil bli utvidet til å bli en større del av kollektivtrafikken både her og mange andre steder i løpet av de neste årene. Slik som den selvkjørende bussen fungerer nå så stopper den på alle holdeplasser uavhengig av om noen skal på eller ikke. Ved en eventuell utvidelse av tilbudet av selvkjørende busser og stor nok bruk av stopp-app hos passasjerer vil det på sikt være gunstig å kun la en selvkjørende buss stoppe om den mottar et signal fra app om å gjøre det.

Kapittel 5

Konklusjon

For å konkludere dette prosjektet vil det bli tatt et siste tilbakeblikk på resultatene og diskusjonen for å vurdere arbeidet som har blitt gjort. Konklusjonen vil deles opp i hovedmål og sekundærmål slik som i presentasjonen av resultatene i kapittel 4.1

5.0.1 Hovedmål

Vi tenker at alle hovedmålene er nådd, kun med en liten endring på det siste målet, som gikk ut på begrensing av hvor mange stoppsignal brukeren kan sende ut iløpet av en tidsperiode, ble endret slik at det som virket mest naturlig ble implementert etter utviklingen hadde begynt. Vi er fornøyd med resultatet og appen ble slik som vi hadde sett for oss i starten og vi fikk implementert alle hovedmålene som vi hadde satt oss.

5.0.2 Sekundærmål

Som presentert i resultatene så ble to av tre hovedmål fullført, noe vi kan si oss fornøyd med. Løsningene på disse målene ble implementert på en enkel, men fungerende, måte og dette åpnet for mye diskusjon rundt videreutvikling av disse punktene. Mulige videreutviklinger er blant annet å

slå stopp-funksjonen sammen med en reiseplanlegger og å bedre vurdere om en bruker vil rekke bussen eller ikke.

Bibliografi

[1] About swift. https://www.swift.org/about/.

[2] Built with mapbox. https://www.mapbox.com/showcase.

[3] Flutter architectural overview. https://docs.flutter.dev/resources/architectural-overview.

[4] How does wi-fi work. https://www.britannica.com/story/how-does-wi-fi-work.

[5] Mobildekning. https://www.nkom.no/telefoni-og-telefonnummer/informasjon-om-telefoni-for-sluttbruker/mobildekning.

[6] Satellite navigation - gps - how it works. https://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/navservices/gnss/gps/howitworks.

[7] Wi-fi ruter. https://www.xfinity.com/hub/internet/modem-vs-router.

[8] Effects of obesity and sex on the energetic cost and preferred speed of walking. 2006. https://doi.org/10.1152/japplphysiol.00767.2005.

[9] Pros and cons of ionic mobile app development, 2019. https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-ionic-mobile-development/.

[10] Hello world! your first ios app in swift, 2020. https://www.codingexplorer.com/hello-world-first-ios-app-swift/.

[11] Dheeraj Jaiswal. Ionic vs nativescript, April 28, 2020. https://medium.com/@DheerajJaiswal6/ionic-vs-nativescript-82d12149c889.

[12] Uzair Khan. The pros and cons of native apps. https://clutch.co/app-developers/resources/pros-cons-native-apps.

[13] Kolumbus. Om kolumbus. https://www.kolumbus.no/om-kolumbus/om-kolumbus/.

[14] mapbox. Maps, geocoding, and navigation apis & sdks. (n.d.). https://www.mapbox.com/.

[15] Rute Figueiredo Mariana Berga. Kotlin vs java: the 12 differences you should know. https://www.imaginarycloud.com/blog/kotlin-vs-java/.

[16] tutorialspoint. Nativescript-quick guide. https://www.tutorialspoint.com/nativescript/nativescript_quick_guide.htm.

[17] Uio. Hva er github. https://www.uio.no/studier/emner/matnat/ifi/IN2001/v18/forelesninger/in2001.2018.02.06.git_intro.pdf.

Figurer

Kodeliste

Vedlegg A

Github Repo

https://github.com/BlockBuster453/DATBAC-Lib