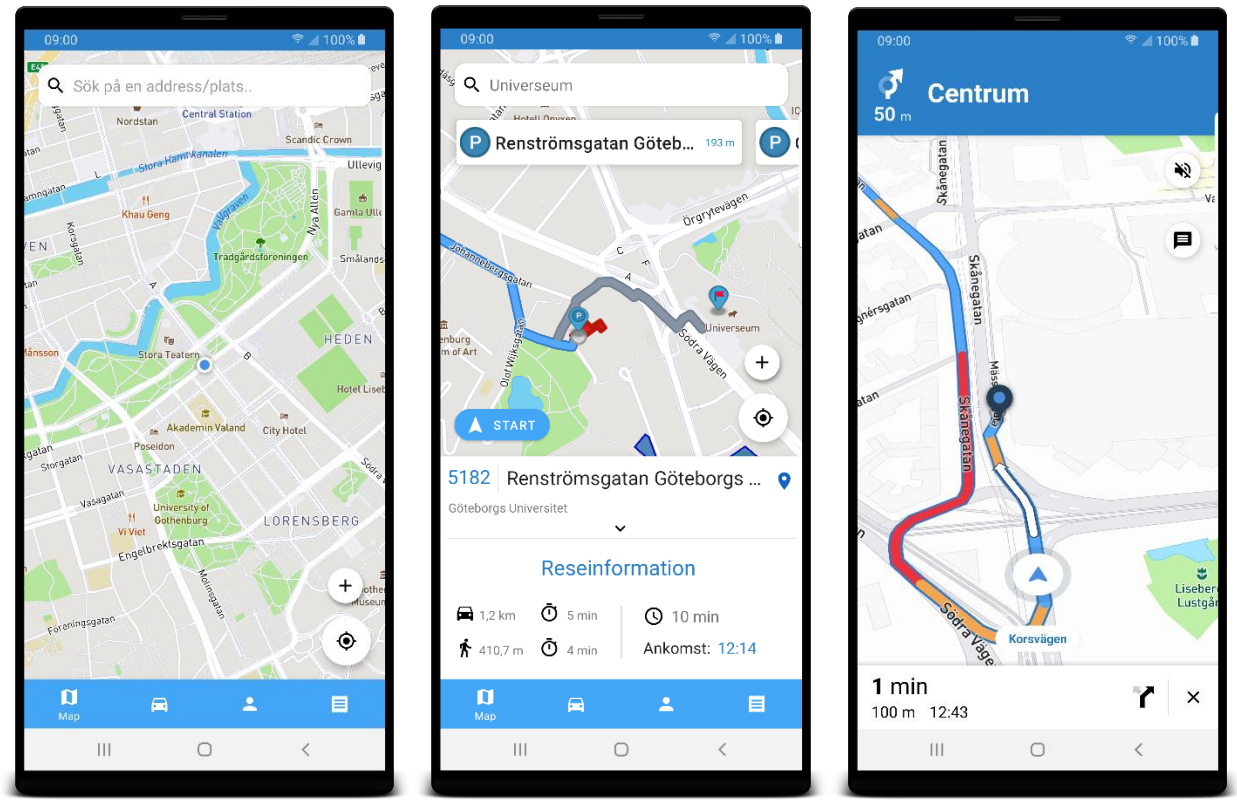
CHALMERS



Destinations-driven parkering

Android-applikation med guidance för komplett bilresa

Examensarbete inom Data- och Informationsteknik

Markus Saarijärvi

Albin Segestam

Institutionen för Data- och Informationsteknik

CHALMERS TEKNISKA HÖGSKOLA

GÖTEBORGS UNIVERSITET

Göteborg, Sverige 2019

EXAMENSARBETE 2019

Destinations-driven parkering

Android-applikation med guidance för komplett bilresa, med parkering i fokus

Markus Saarijärvi
Albin Segestam

CHALMERS

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg 2019

Destinations-driven parkering

Android-applikation med guidance för komplett bilresa, med parkering i fokus

MARKUS SAARIJÄRVI
ALBIN SEGESTAM

Examinator: Jonas Duregård

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola / Göteborgs Universitet
412 96 Göteborg
Telefon: 031-772 1000

Omslag: Bilder på applikationen som utvecklades under projektets gång.

Institutionen för Data- och Informationsteknik
Göteborg, Sverige 2019

Sammanfattning

Tekniken växer snabbt i dagens samhälle och bilister är en grupp som gynnats av detta. Tekniska lösningar som navigationstjänster kan hjälpa bilister att hitta till sin önskade destination. Det finns fortfarande en del i resans omfattning som kan förbättras; parkeringen. Då resa och parkera förknippas med varandra, är dessa moment åtskilda i vissa aspekter. En aspekt är att mobila applikationer hjälper med navigering från start till slutdestination, men i fråga att hitta en parkering, sker detta mer manuellt utan digitala hjälpmedel. Det kan betyda att parkeringen blir först påtänkt i slutet av resan då slutdestinationen börjar närma sig. Syftet med detta projekt, är skapa en lösning som skulle kunna gynna och bidra till en bättre parkeringsupplevelse för användaren genom att möjliggöra parkeringsvalet i ett tidigare skede för att sedan inkludera det i helhetsresan.

Under arbetets gång utvecklades en prototyp i formen av en Android-applikation. Prototypen förbättrar användarflödet genom processen som vi kallar destinations-driven parkering. Applikationen som utvecklades, möjliggör valet av parkering i ett tidigare skede. Användaren guidas sedan först med bil från en startpunkt till vald parkering. Vid parkeringen bekräftas start av parkerings-sessionen, för att därefter guida användaren till fots från parkering till slutdestinationen.

Nyckelord: Parkering, Destination, Android, Android-applikation, Navigation, Guidance

Abstract

Technology is growing rapidly today and technical solutions that can help motorists navigate to their desired destination exist. There is still some of the scope of travel that can be improved; parking. Travel and parking are easily associated with each other, but the steps are separated in some aspects. One aspect is that mobile applications help with navigation from start to destination, but in the case of finding a parking, this happens more manually without digital aids. This means that parking will first be considered at the end of the journey when the driver begins to approach their destination. The purpose of this work is to create a solution that could benefit and contribute to a better parking experience for the user and include the larger picture of how a trip in a whole could have been improved with the parking in mind.

What was developed during the project was an Android application that was designed according to what we came to call destination driven parking. The application that was developed can guide a user from a starting point to a parking lot by car, upon arrival to the parking the user can easily confirm the start of the parking session, in order to then guide the user on foot from parking to the final destination.

Keywords: Parking, Destination, Android, Android Application, Navigation, Guidance

Förord

Den här rapporten är vårt examensarbete för kandidatexamen på datateknikprogrammet på Chalmers tekniska högskola i Göteborg. Arbetet genomfördes av Markus Saarijärvi och Albin Segestam under vårterminen 2019.

Tack till

Vi vill börja att tacka vår handledare Sakib Sistek som trodde på oss under hela arbete samt för trevliga möten och roliga diskussioner. Vi vill rikta ett stort tack till att vi fick utföra vårt arbete på SMS-Park, Inteleon, och till alla härliga personer på företaget som gjorde tiden hos er till nått att minnas. Under vår tid på SMS-Park vill vi rikta ett extra stort tack till hela utvecklingsteamet Kaiju, vår handledare Mattias Markehed samt till Lisa Höjlund, UX-designer, för samarbetet och hjälp under hela arbetet.

Innehållsförteckning

Ordlista

SDK - Software Development Kit, ett utvecklingsverktyg skapat för att bygga applikationer mot ett annat ramverk eller plattform.

Vy - En vy i detta sammanhang är en del av det grafiska gränssnittet i en Android applikation, exempelvis en profilsida där man kan se sin personliga information är en vy.

API – Application Programming Interface är en gränssnitt för att kommunicera mellan program

IDE – Integrated Development Environment, ett utvecklings program med textredigering, kompilator med mera för att underlätta programmering.

Seperation of Concerns – Ett programmeringskoncept som menar att ett programs beståndsdelar bör va seperarade i distinkta sektioner, där varje sektion enbart har ett tydligt ansvar.

Design Pattern - Ett designmönster inom programvaruteknik innebär att man har designat ett mönster för att lösa ett typiskt problem inom programvarutveckling.

UI – User Interface, användargränssnitt.

Bottom Sheet - En UI-komponent i en applikation som kommer upp från botten av skärmen som kan svepas upp och ner för att visa mer eller mindre innehåll, och kan innehålla text, knappar, bilder med mera.

Boilerplate - Kod som måste användas på flera ställen utan någon större ändring. Vanligt i programmeringsspråk som är pratiga då man skriver mycket kod som inte gör någon större nytta utan bara för att man måste göra det.

Guidance – Vägledning, i kontexten till denna rapport menas vägledning genom en hel resa, från start till parkering och slutligen destination

Kapitel 1 Inledning

Tekniken växer i snabb takt i dagens samhälle och blir allt mer omfattande med tiden. Med den växande fordonsbranschen, framträder dagens teknik med nya möjligheter för att göra transportsystemet mer effektivt och lätthanterligt [1]. Det går att finna navigering med realtidstrafik, som kan vägleda fordon till mer lämpliga vägar vid trafikstörningar och därmed minimeras förseningar [2]. Det finns fortfarande förbättringspotential i bilresans helhet; parkeringen. Då resa och parkera förknippas med varandra, är de relativt åtskilda i vissa aspekter fast att de är givna med varandra. Idag hjälper mobilapplikationer till med att navigera, från start till slutdestination, men i fråga om att hitta en parkering, sker detta oftast mer manuellt utan digitala hjälpmedel.

1.1 Syfte

Syftet med detta projekt, är skapa en lösning som skulle kunna gynna och bidra till en bättre parkeringsupplevelse för användaren genom att möjliggöra parkeringsvalet i ett tidigare skede för att sedan inkludera det i helhetsresan.

1.2 Mål

Målet med detta projekt är att skapa en prototyp i form av en Android-applikation, som kan göra valet av parkeringen mer proaktivt i ett tidigare skede. I målet inkluderas även, att om valet av parkering kan göras mer proaktivt i ett tidigare stadie, skall applikationen även kunna erbjuda guidance först till parkering via bil, sedan starta parkeringen för att till slut erbjuda navigering via gång till slutdestination.

1.3 Avgränsningar

Det här arbetet kommer att beröra olika aspekter kring parkeringar samt beteenden som bilister följer. Det bör förtydligas att det inte kommer att utföras nya undersökningar eller studier i detta arbete. Arbetet kommer att bygga på tidigare studier/undersökningar.

Arbetet kommer att inkludera utveckling av en applikation som skall lösa eller påvisa hur problematiken kring parkeringar skulle kunna hanteras bättre. Applikationen skall kunna drivas på Android-enheter och det skall förtydligas att det enbart är mjukvara som skall utvecklas. Specifika hårdvarurelaterade aspekter för de olika enheter som applikationen kommer att kunna drivas på, hanteras av operativsystemet Android.

Applikationen som skall utvecklas kommer inte att hantera betalningar eller kunna starta en parkeringssession på riktigt utan den kommer bli en prototyp som påvisar ett koncept, hur den hade kunnat hantera en helhetsresa bättre.

Kapitel 2 Teoretisk Bakgrund

Under detta kapitel presenteras de olika delarna av målet med en parkering, hur bilister väljer parkering samt hur bilister betalar. Sedan kommer programmeringsspråket Kotlin samt mjukvarumönstret Model-View-ViewModel att presenteras.

2.1 Parkering idag

Här nedan kommer målet med parkering, hur man väljer parkering, hur man betalar samt hur en vanlig bilresa kan se ut att presenteras.

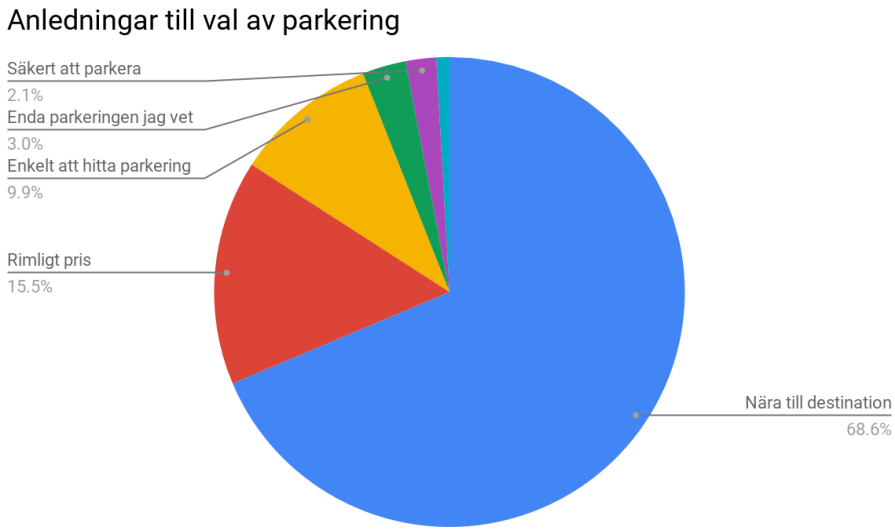
2.1.1 Målet med parkering

Som man kan se på figur 2.1 är bilisters ändamål till att parkera i centrala Göteborg främst shopping eller nöjessammanhang, dessa två ändamål tar upp hela 67% av parkeringar i Göteborg [3]. Dessa ändamål ökar även i andel hos kranskommunerna, vilket inte är underligt, då de inte bor i staden, ökar andelen arbetspendling, shopping och nöje hos dessa bilister. Liknande resultat framgår i en studie i Indonesien där 62,5% av bilisternas ändamål var shopping/nöje och 37,5% var arbetsrelaterat eller affärsrelaterat [4].

Figur 2.1 Sifo-undersökning med uppdrag från Parkering Göteborg [3]

2.1.2 Hur väljs parkering?

Hur trafikanter väljer parkering vägs emot ett flertal olika faktorer, där de största faktorerna är avstånd till destination, pris samt söktid. Figur 2.2 visar procentuellt fördelningarna till val av parkering där avstånd till destination är den största anledningen [5]. En liknande studie från Beijing visade på att 88% av bilisterna valde parkering baserat på avstånd till destination och under 1% baserat på priset [6]. Sammanfattningsvis framgår att en stor majoritet av bilister, väljer parkering baserat på avståndet till deras destination och att priset inte är den viktigaste faktorn. Dessa faktorer varierar mellan städer och länder men generellt sett är bilister inte allt för priskänsliga.



Figur 2.2 Fallstudie från Cardiff [5]

2.1.3 Hur man betalar

Sättet att betala för sin parkering idag, kan variera baserat på var du befinner dig och vilka tekniska lösningar den specifika parkeringen tillhandahåller. Ett betalsätt är parkeringsautomaten där du som bilist går ut ur din bil, går till en automat och betalar för den beräknade parkeringstiden. Bilisten får då en parkeringsbiljett, som påvisar betalning och hur länge parkeringen är giltig. Ett annat alternativ som blir mer populärt är mobilbetalning, antingen via SMS eller via en mobilapplikation. Här kopplas

parkeringar till olika zoner som har information om pris och tid. Sedan väljs zonen som fordonet står på, betalning sker via ett SMS eller via en applikation. Tiden är mer flexibel och det är enklare att avsluta/utöka parkeringssessionen. Även om det är svårt att veta hur många som använder sig av mobilbetalning kontra parkeringsautomat, går det se att mobilbetalningar har ökat kraftigt. År 2016 ökade exempelvis mobilbetalningar i Västerås med 67% jämfört med året innan [7].

2.1.4 Parkeringsflöde

Ett scenario där resan inte är rutinbaserad, till exempel en resa till en nöjespark, ett nytt område eller liknande, ökar chansen för att en resas flöde hade kunnat representeras av följande tillvägagångssätt.

1. Bilisten sätter sig i bilen och planerar sin resa genom att slå upp sin destination i en navigationsapplikation och startar upp navigeringen.
2. Bilisten börjar köra mot sin destination med hjälp av navigering.
3. Bilisten närmar sig sin destination och börjar nu tänka på att hitta en parkering.
4. Efter sökande, hittar bilisten en parkering och parkerar sin bil. Om individen har en av de mobilapplikationer som kan ta betalt på just denna parkering, öppnar hen applikationen och påbörjar parkeringssession. Ett annat alternativ kan vara att användaren behöver ladda ner ytterligare en applikation och sätta upp betalningsalternativ för att sedan kunna starta betalningen av parkeringen. Om inte tidigare alternativ valts, kan bilisten gå till parkeringsautomaten och betala manuellt, för att sedan gå tillbaka till bilen för att lägga dit sin parkeringsbiljett.
5. Slutligen går individen sista biten till önskad destination.

2.2 Kotlin

Java har varit standard språket för Android sedan början av plattformen, men på senare år har valet av språk ökat. Idag stödjer Android tre utvecklingsspråk, Java, Kotlin och delvis C++ [8].
Under Googles utvecklarkonferens Google I/O 2017 meddelade Google att Android nu stödjer Kotlin som ett officiellt språk [9]. Sedan dess har användandet av Kotlin ökat drastiskt, under Google I/O 2019 nämner de att över 50% av alla professionella Android-utvecklare använder Kotlin och det är det snabbaste växande språket på Github [10].

Kotlin började utvecklas 2011 av JetBrains, skaparna av den populära IDE IntelliJ, och släpptes officiellt i februari 2016 [11]. Det är ett objektorienterat språk men kan även skrivas i en funktionell stil, och är inspirerat av språk som Java, C#, Javascript, Groovy och Scala [11]. En av dess säljpunkter är att det är ett väldigt koncist språk, med språkfunktioner som fokuserar på att minska boilerplate-kod och förbättra läsbarheten och produktivitet. Enligt JetBrains själva är en grov uppskattning att Kotlin minskar antalet kodrader jämfört med Java med 40% [11]. En annan stor del av Kotlin är dess null-säkerhet där dess typ-system skiljer mellan typer som kan vara null och typer som inte kan vara null. Detta gjordes för att minska eller totalt ta bort NullPointerExceptions från sin kod, vilket är ett vanligt problem i Java [12].

För att ge en större uppfattning om hur Kotlin ser ut och fungerar kan man nedan se tre korta kodexempel i Kotlin.
Exempel 1:

```
val a = "a"
var b: String? = null
println(a.length)
println(b?.length)
```

I detta exempel deklareras en variabel vid namn ’a’ med ’val’ som gör variabeln omutbar, vilket betyder att värdet inte kan ändras. Sedan deklareras en variabel vid namn ’b’ med ’var’ som gör variabeln mutbar samt att den är markerad nullbar med ’?’. Sedan försöker man skriva ut längden på dessa strängar. Första utskriften kommer att

skriva längden på strängen vilket är 1, andra kommer skriva ut null om den är null annars längden på strängen med hjälp av ’?’ operatorn.

Exempel 2:

```
var person = Person(”John Doe”, 45)
person?.let {
	val name = person.name
	if(person.age > 18){
		println(name + “is an adult”)
	}
	else{
		println(name + “is not an adult)
	}
}
```

I detta exempel har vi ett objekt ’person’, tänk dig att detta objekt innehåller personinformation som namn och ålder. Person-objektet här är mutbar samt nullbar, om man enbart kollar om person inte är null så kan värdet på person ändras efter man har kollat om den är null. Kotlin har ’Scoping functions’ vars funktion är att utföra ett kodblock på ett objekt [13]. En av dessa funktioner är ’let’ som först kollar om person-objektet är null med hjälp av ’?’ operatorn. Om person är null utförs inte kodblocket utan den ignoreras helt. I en vanlig null-koll hade värdet på ’person’ kunnat ändras mitt i kodblocket. Det som är fördelaktigt med ’let’ är att funktionen behåller objekt-referensen som funktionen användes på och kan då användas som om den inte vore null genom hela kodblocket.

Exempel 3:

```
val name = getName() ?: "Albin"
```

Här deklarerar man variabeln ’name’ med get-metoden ’getName’, om värdet som ges av ’getName’ är null sätter den värdet till det som är efter den så kallade Elvis-operatorn ’?:’, det vill säga ’Albin’ istället. Detta kan vara hjälpsamt om man exempelvis har ett standardvärde för en variabel samt för att undvika att variabeln name är null.

I Android betyder detta att en applikation skriven i Kotlin kan minska risken att applikationen kraschar på grund av en null-referens, vilket leder till en mer stabil och användbar applikation.

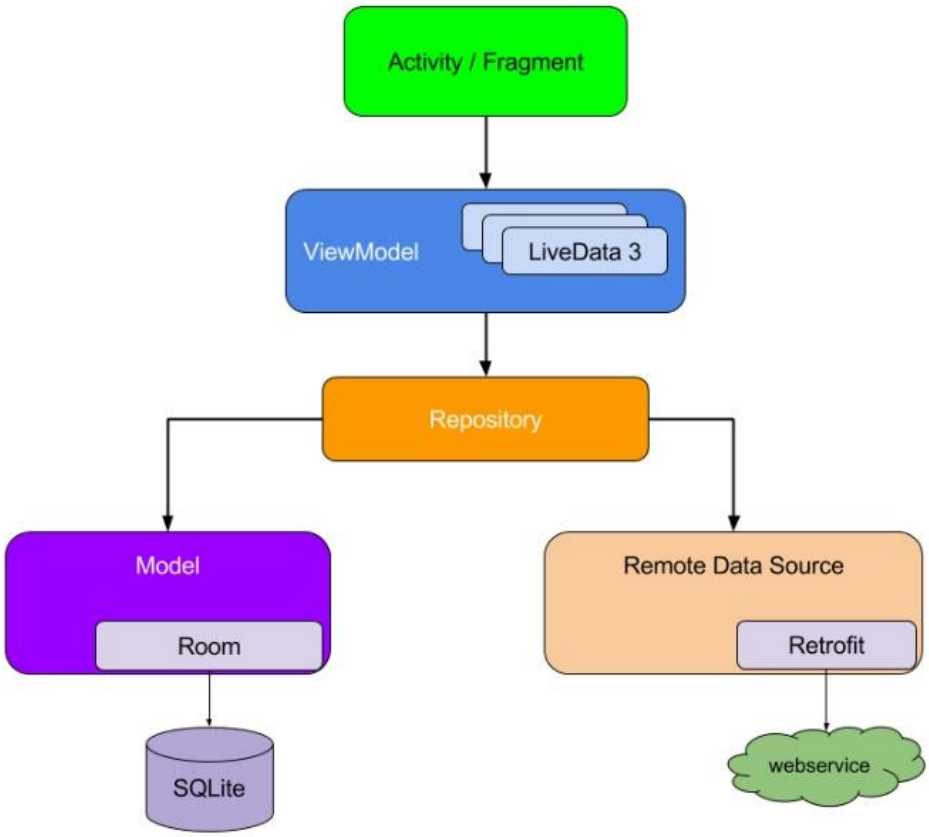
2.3 Mjukvarumönster

I mjukvaruutveckling används det allt för ofta tekniker för att återanvända lösningar i specifika sammanhang. Inom mjukvaruutveckling har det formats så kallade ’mjukvarumönster’ för att förenkla eller hantera problematik som ofta kan uppstå i utveckling av mjukvara. Dessa mjukvarumönster som följs är inga konkreta lösningar som kan transformeras direkt till källkod utan mer som mallar och guidningar för hur man bör hantera exempelvis relationer mellan olika objekt och klasser.

2.3.1 Model-View-ViewModel

Model-View-Viewmodel, eller förkortat ’MVVM’ är ett mjukvarumönster vars uppgift är att förenkla separationen av koppling mellan användargränssnitt och affärslogik. Fördelarna med separationen av gränssnitt och logik leder till att system blir enklare att testa, underhålla och vidareutveckla [14].

Mönstret bygger på tre huvudkomponenter som är View, ViewModel och Model. Repository mönstret lägger till ytterligare ett lager abstraktion av datalagret genom att frikoppla de andra komponenterna från nätverks- och databashantering. I figur 2.3, kan man se ett generellt diagram över relationerna mellan komponenterna [15].



Figur 2.3 Diagram över en rekommenderad applikationsarkitektur från Google [15]

Det som bör förtydligas med MVVM innan ytterligare förklaring av de individuella komponenterna är hur de interagerar med varandra. Kollar man på figur 2.3 kan man se att på högsta nivån befinner sig ’Activity’ eller ’Fragment’ som kommer att känna till ’ViewModel’. ViewModel kommer i sin tur att känna till ’Repository’ men inte omvänt och ’ViewModel’ kommer vara omedveten att ’Activity’ eller ’Fragment’ existerar. ’Repository’ känner till ’Model’ samt ’Remote Data Source’.

2.3.1.1 View

I figur 2.3 refereras ’View’-komponenten som ’Activity/Fragment’ och kommer refereras som ’vyn’ eller ’View’ i resterande text. Ansvaret som vyn har är att definiera strukturen och hur applikationen skall presenteras till användaren via en skärm. Vyn kommer inte att hantera någon affärslogik utan den kommer endast att presentera

innehåll och reagera på interaktion från användaren [15]. Exempel på användar-interaktioner kan vara tryck av knappar eller svep/drag på skärmen.

2.3.1.2 Model

’Model’-komponenten används för att implementera de olika delarna av applikationen som har med data och applikationstillstånd att göra. Exempelvis kan ’Model’ vara en lokal databas där lagrar användardata eller applikationsdata [15].

2.3.1.3 Viewmodel

Rollen som ’ViewModel’ har är att förse vyn med specifika data för användargränssnittet. ’ViewModel’ kan även vid användarförfrågning kommunicera med andra komponenter föra för att förse vyn med ny data [15].

2.3.1.4. Repository

’Repository’-komponenten agerar som en mellanhand för nätverks- och databashanteringen och ’ViewModel’. Komponenten vet då om hur man hämtar data och hur denna data ska hanteras genom insättning och borttagning [15].

2.3.1.5 Remote Data Source

’Remote Data Source’-komponentens ansvar är att hämta data från en extern server. Denna komponent använder sig av ’Retrofit’, som är en HTTP-klient för Android och Java som gör om ett HTTP-API till ett gränssnitt [16]. Detta gör det enkelt och smidigt att konsumera ett API direkt i sin applikation.

Kapitel 3 Metod

Under det här avsnittet, kommer vi att presentera de olika faserna som arbetet gick igenom samt vilken utvecklingsmiljö som användes under arbetets gång.

3.1 Utforskning

När problemformulering samt syfte med detta arbete blev klarare, blev det tydligare hur applikationen skulle formas för att möta samt stärka helhetsresan för bilister. Första stadiet i arbetet blev att hitta lösningar för att visualisera en karta samt hitta en lösning för navigation.

Ett stort val var att välja ett API samt SDK för kartan som skulle integreras i applikationen som skulle utvecklas. För Android finns flertalet tjänster att välja bland men generellt kan man säga att det finns Google Maps och sedan en uppsjö av Open-Street-Maps-baserade API:er. För applikationen, som utvecklades, behövdes:

1. en karta som kunde visualisera parkeringar,
2. en sökfunktion för att hitta adresser/platser och
3. navigation.

De två förstnämnda av dessa komponenter, kan fullgöras med liknande SDKs som Google Maps, men navigation går inte att implementera med Google Maps internt i en egenutvecklad applikation.

När fler tjänster jämfördes hittades Mapbox vilket blev ett alternativ. Mapbox är en dataplattform för webbapplikationer samt mobila applikationer som bygger på Open Street Maps. De erbjuder tjänster såsom kartor, navigation och platssökning. Deras tjänster når idag runt 300 miljoner användare varje månad. Det som gör att Mapbox sticker ut, är deras utmärkta datavisualisering och hur anpassningsbara deras tjänster är. Mapbox är även mycket effektivt att utveckla med, då deras tjänster ger utvecklare mycket grundfunktionalitet, vilket gör att utvecklare kan fokusera mer på övergripande funktionalitet istället.

Med tjänstens fördelar, bestämdes det att gå vidare med att utveckla och testa Mapbox. Grundfunktioner som att visualisera en karta med olika objekt på och rita resvägar mellan olika punkter testades bland annat. Vi som utvecklare var nöjda med hur det var att arbeta med Mapbox samt vad tjänsten kunde erbjuda, vilket gjorde att det blev det slutgiltiga valet.

3.2 Applikationsarkitektur

När erfarenhet skapats med att arbeta med Mapbox tjänster, gick utvecklingen in i en fas där skalbarheten kom in i bilden. För att under en längre tid kunna utveckla utan att stöta på större problem gällande låg kohesion eller hög koppling i applikationen, valde vi att tillämpa en applikationsarkitektur. Innan arbetet med att refaktorera om de redan utvecklade grundfunktionerna, jämfördes olika systemarkitekturer för att välja en lämplig metod att strukturera upp arbetet i. Hade vi inte valt en systemarkitektur hade det kunnat leda till problematik i formen av komplexitet, ju mer applikationen växer. Problem som vi ville undvika med en applikationsarkitektur, var bland annat att ta hänsyn till ’separation of concerns’ samt kunna driva användargränssnittet via en modell, vilket även rekommenderas av Google [15]. Valet föll på lösningen Model-View-ViewModel, som skulle göra det enklare att skapa en separation mellan användargränssnittet och den bakomliggande logiken som kan återfinnas i applikationen. Denna applikationsstruktur diskuteras under kapitel 4.

3.3 Utvecklingsfas

Efter att systemarkitekturen av systemet var utformad, kunde utvecklingen fortgå med att hämta in riktiga parkeringar och skapa funktionalitet i applikationen. Det var i detta stadie som applikationens värde för användaren började växa, då resväg från vald startpunkt till destination kunde skapas för att sedan även inkludera parkeringen som ett delmoment i resan. Enklare sagt var det i detta skede som processen av destinations-driven parkering utvecklades in i applikation.

3.4 Utvecklingsmiljö

I detta projekt, användes Android Studio version 3.3 till 3.5 vilket är ett standard IDE för Android-utveckling. Projektet är utvecklat i Kotlin, som har beskrivits i kapitel 2. Versionshantering gjordes via Github.

Kapitel 4 Konstruktion

I detta avsnitt kommer problematiken med dagens parkering att lyftas fram. Sedan presenteras en föreslagen lösning, som kom att gå under namnet ’destinations-driven parkering’. Slutligen kommer projektets implementation av MVVM-mönstret att presenteras och vad varje komponents roll och funktion.

4.1 Problematik med dagens parkeringsflöde

Vanligtvis är inte alltid parkeringen planerad in i resans helhet, utan den blir mer som en eftertanke, då individen börjar närma sig sin slutdestination. Genom att användaren inte börjar tänka på sin parkering förens vid slutet av bilresan kommer valet av parkering av största sannolikhet ske reaktivt. Det kan då uppstå problem i sökandet av en parkering. Var parkeringar är lokaliserade kan vara okänt för föraren vilket kan leda till frustration samt stress hos personen.

Resa och parkera kan idag ses som två skilda fenomen, dessa två saker hör naturligt ihop och bör då behandlas som sådant. Idag kan resan bli hjälpt av digital assistans men att hitta/välja parkering är en manuell och oassisterad uppgift som är mer som en eftertanke. Ytterligare problematik för bilister, är användandet av flertalet applikationer i en och samma resa. Förutom en navigeringsapplikation samt en parkeringsapplikation, kan parkeringar ha olika företag, som står för betaltjänsten. Det kan leda till att användare har flera olika betalningsapplikationer beroende på var man parkerar, eller att vid olika tillfällen behöva ladda ner nya applikationer. Problem som kan uppstå är att vid tillfället av parkering, måste användaren konfigurera den nya applikationen med betalningsmetod och fordon vilket kan ytterligare stressa samt göra användaren frustrerad i sin resa som helhet.

4.2 Destinations-driven parkering

Det hade varit fördelaktigt för användare om hela resans omfång kan hanteras inom en och samma applikation, som hjälper användare med hela resan. Flödet som vi presenterar här som vi vill referera till som destinations-driven parkering följer som nedan:

1. När bilisten skall resa kommer den att starta och söka efter önskad adress/plats. När adressen lokaliserats och påvisas för användaren i applikationen, skall parkeringar visualiseras samtidigt, så att det blir enkelt att välja parkering i ett tidigare skede.
2. Bilisten kan börja köra med hjälp av navigation i applikationen som leder föraren mot sin destination via den valda parkeringen.
3. När bilisten kommit fram till den valda parkeringen skall det via en enkel bekräftelse kunna starta betalningen av parkeringen.
4. I det sista steget skall applikationen presentera ny navigering i form av gång från parkering till slutdestination och resan är genomförd med hänsyn till parkeringen.

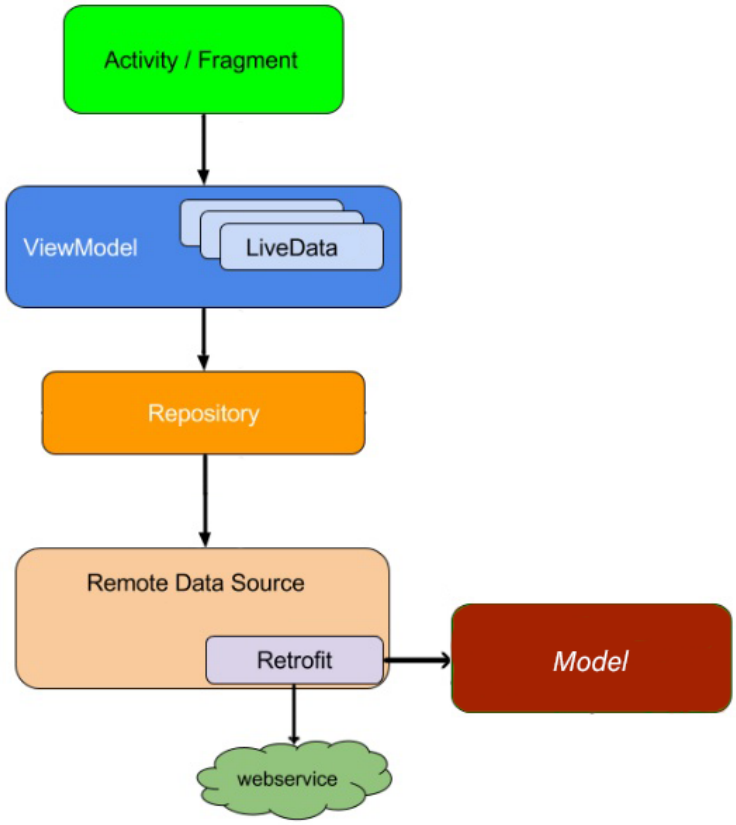
Fördelar med detta flöde, är att valet av parkering inte längre är reaktivt utan är proaktivt. För bilister kommer det bli lättare att lokalisera och överblicka parkeringar i ett tidigare skede innan man åker iväg som kan förhindra frustration/stress för bilister om valet annars skett reaktivt på plats [2]. Ytterligare en fördel med att kunna överblicka parkeringar i förväg, är att chansen till att önskad parkering i relation till slutdestination kommer att matchas bättre om valet sker proaktivt.
Dessutom kommer betalningen av parkeringen ske via samma applikation som navigationen. Detta underlättar för användaren att slippa använda olika applikationer för de olika delmomenten i helhetsresan.

4.3 Implementation av Model-View-ViewModel

De huvudfunktioner som behövdes implementeras i applikationen för att uppnå det som presenterades i 4.2 var då

- Ett nätverks-API som kan hämta parkeringsdata
- En karttjänst som kan visualisera parkeringar
- En platssökningsfunktion så användaren kan söka på en adress eller plats
- Navigering som kan guida användaren till sin parkering samt slutdestination

I figur 4.1, kan man se ett generellt diagram över hur MVVM-mönstret implementerades i applikationen. Det som skiljer sig från diagrammet, som visades i figur 2.3, är att denna applikation inte behöver en lokal databas för att lagra data. Applikationen behöver enbart en ’Model’ som tillfälligt kan lagra parkeringsdata.



Figur 4.1 Diagram som visar de olika delarna av det implementerade MVVM-mönstret

4.3.1 Model

’Model’-komponenten i applikationen innehåller klasser som har information om parkeringar. Modellen hanterar även uppgifter som formatering av distans och tid.

4.3.2 View

En vy i applikationen är då en UI-komponent som enbart hanterar UI-relaterade ansvar. Ett typexempel på en vy i applikationen är kartvyn, som visar kartan som användaren interagerar med. Kartan visar exempelvis användarens position, alla parkeringar samt resväg till parkering samt destination.

4.3.3 ViewModel

I applikationen finns det ett flertal ’ViewModel’-komponenter, där alla har olika typer av data, som den innehåller och förser till vyn. Varje ’ViewModel’ har observerbara datahållare som i Android kallas för ’LiveData’. Genom att skapa observerare på dessa datahållare, kan observerarna sedan reagera när datan inuti ändras [17].
Exempel på ’ViewModel’-komponenter som används i applikationen är:

- *ZoneViewModel* – innehåller parkeringsdata i form av enskilda zoner där varje zon har en latitud- och longitudpunkt, alternativt en lista med punkter för att representera en parkeringsyta.
- *RouteViewModel* – innehåller objekt som representerar resvägar från startposition till parkeringen samt från parkeringen till slutdestinationen.
- *SelectedZoneViewModel* – innehåller parkeringsdata om senast vald parkering.

4.3.4 Repository

’Repository’-komponenten agerar som en mellanhand för nätverks-API:et och ’ViewModel’-komponenterna. Komponenten innehåller metoder för att kommunicera med nätverks-API:et och hämta parkeringsdata eller resvägar.

4.3.5 Remote Data Source

Den här komponenten kommunicerar med ett externa API:er för att hämta parkeringsdata samt resvägar. Data om vanliga parkeringar hämtas via SMS-Parks API och parkeringar för rörelsehindrade hämtas från ’Göteborgs Stads Öppna Data’. Resvägar hämtas från ’Mapbox Directions API’.

4.3.6 Applikationsflöde

För att förstå vad varje komponent gör i applikationen kommer ett typiskt applikationsflöde att presenteras. I varje steg i detta flöde kommer sedan varje komponents funktion att förklaras.

Kommande flöde sker efter att användaren har sökt på en destination.

1. Kartvyn, som är applikationens huvudvy kommunicerar med ’ZoneViewModel’ att användaren har sökt på en destination.
2. ’ZoneViewModel’ kommunicerar vidare detta till ’Repository’ och ber den att hämta parkeringsdata runt den destination som söktes på.
3. ’Repository’ har en nätverks-API-komponent och använder den för att hämta parkeringsdata.
4. Efter en kort stund returnerar det externa API:et ett svar i form av ett JSON-objekt.
5. För att kunna använda parkeringsdatan i applikationen behandlar ’Retrofit’ svaret inuti ’Model’-komponenten.
6. Parkeringsdatan som efterfrågades i steg 2 skickas sedan tillbaka till ’ZoneViewModel’ som uppdaterar sin ’LiveData’.
7. Kartvyn som observerar ’ZoneViewModel’s ’LiveData’ kan reagera på de nyhämtade parkeringarna och visualiserar de på kartan.

Ett liknande flöde sker då användaren sedan väljer en parkering på kartan och en resväg behöver hämtas.

Kapitel 5 Resultat

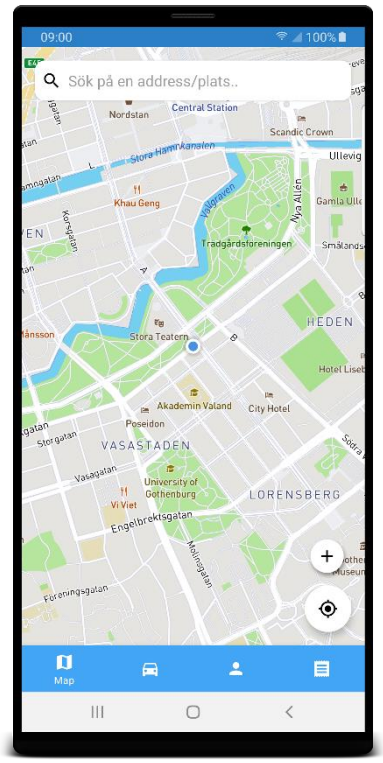
Den tekniska lösningen som utvecklades i detta arbetet kom att bli en applikation för Android-enheter. Den förbättrar resan i helhet genom att göra parkeringen mer relevant i ett tidigare skede. Nedan kommer den utvecklade applikationen att beskrivas.

5.1 Applikationen

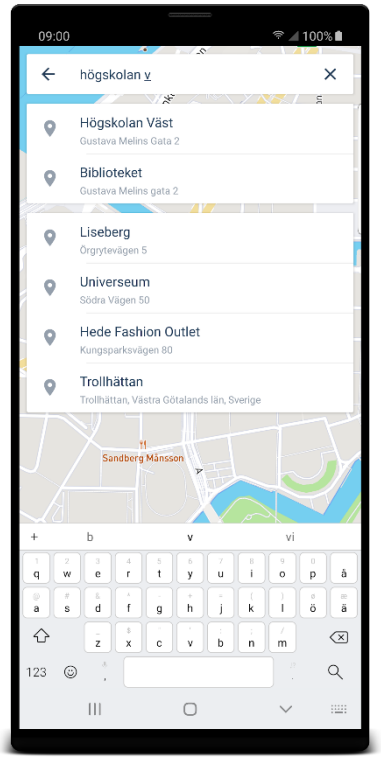
I detta avsnitt kommer vi presentera vår tekniska lösning, hur den fungerar och hur den ser ut visuellt. Applikationen kommer att presenteras genom att visa var den passar in i processen av destinations-driven parkering, steg för steg och hur den bidrar till en bättre helhetsresa.

5.1.1 Startpunkt

I början av resan när en person vill påbörja en ny resa, kan personen söka och ta fram sin önskade destination. Vår tekniska lösning gör det möjligt att söka efter adresser eller platser för att sedan presenteras visuellt. Ur processen av destinations-driven parkering kan användaren här söka efter adress eller plats som användaren vill resa till.



Figur 5.1 Startvyn i applikationen.

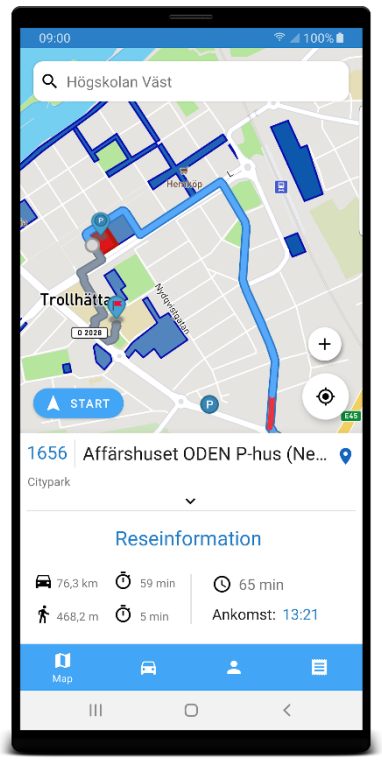


Figur 5.2 Användaren söker på en destination.

I detta steg söker användaren efter en destination, man kan söka efter en adress eller plats, vilket kan ses i figur 5.2. Det som bör förtydligas med figur 5.2 är att komponenten under sökrutan är förslag på platser som baseras på vad användaren skriver in. Komponenten under adress- och platsförslag är historik över tidigare sökta destinationer.

Det som applikationen sedan gör är att när destinationen presenteras med en markör med en flagga i, kommer det även visualiseras var parkeringar är lokaliserade i närhet till vald destination. Parkeringar visualiseras som mörkblåa zoner eller runda ikoner, men även i en lista som är sorterad efter avstånd, som kan ses i figur 5.3. Användaren väljer parkering via klick på kartan, alternativt via klick i parkeringslistan i den övre delen av skärmen. Det är i detta skede som steg 1 av destinations-driven parkering utnyttjas, parkeringen blir relevantare tidigare i resan.

Figur 5.3 Användaren har valt destination.

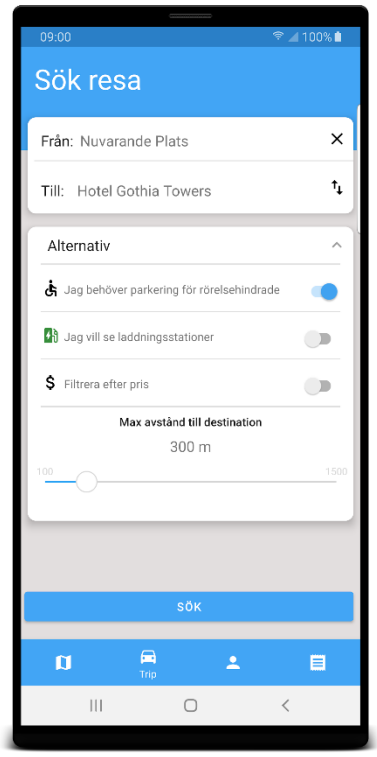


Figur 5.4 Användaren har valt parkering

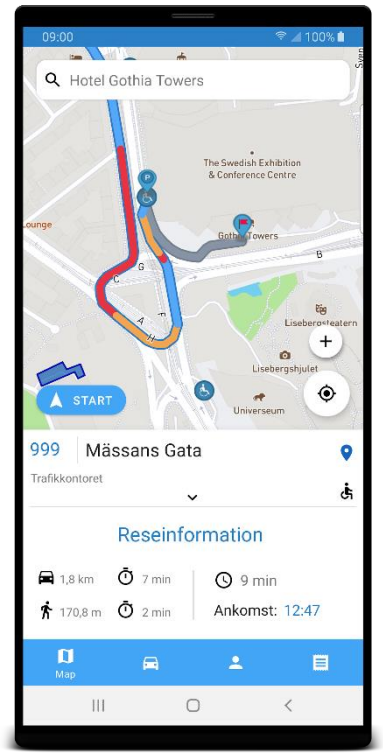
När användaren har valt eller är i skedet att välja mellan olika parkeringar, kan användaren få extra information om resan i ett så kallat ’Bottom Sheet’ som man kan se i figur 5.4.

Det som presenteras här är information om parkeringen, dess namn, zon kod samt ägare. Ytterligare reseinformation presenteras i form av hur lång bilvägen är, hur lång gångvägen är samt korresponderande tider och när beräknad ankomst är beräknad. Under detta skede visualiseras resan via en navigations-rutt från start sedan via parkeringen och till sist slutdestinationen.

För användare som vill slippa att tänka på vilken parkering som skall väljas, finns även en vy där användaren endast väljer start, slutdestination samt övriga alternativ som kan ses i figur 5.5. När användaren klickar på sökknappen i figur 5.5, tilldelas sedan en utvald parkering för användaren.



Figur 5.5 Alternativ sökvy

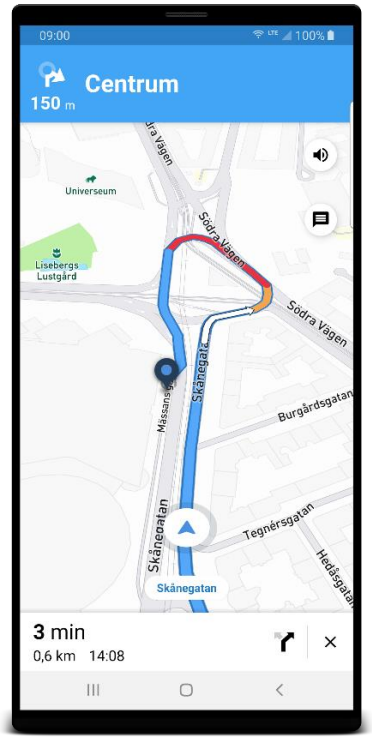


Figur 5.6 Användaren blir presenterad utvald parkering

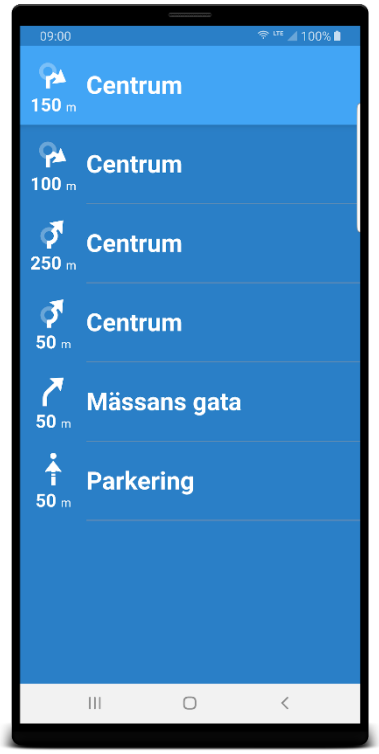
Efter att användaren använt den alternativa sökvyn, presenteras utvald parkering på kartan samt reseinformation, rutt och den nuvarande trafiksituationen som kan ses i figur 5.6.
Nu kan användaren fortsätta processen i resan genom att starta navigationen via startknappen i figur 5.6.

5.1.2 Under färd

För bilisten kommer det under resans gång finnas navigeringsstöd, som hjälper föraren att guidas i sin resa. Det som bör poängteras är ju att navigeringen sker med hänsyn till helhetsresan. Först kommer bilisten bli hänvisad och guidad till parkeringen som har valts i ett tidigare skede som visas i figur 5.7 som är steg 2 i destinations-driven parkering.



Figur 5.7 Första steget i navigationen mot vald parkering



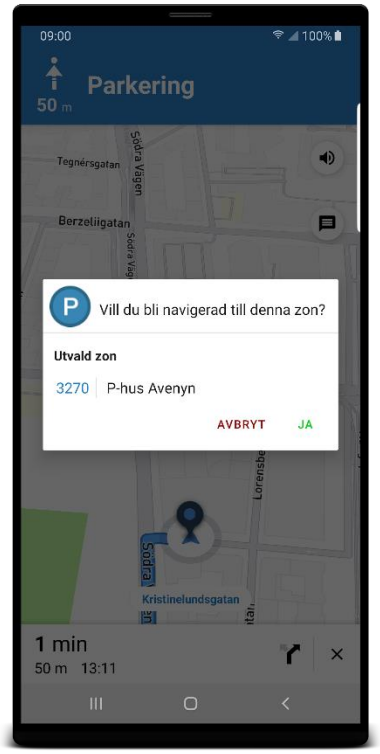
Figur 5.8 Navigeringsinstruktioner

Under navigeringen kan användaren överskåda de kommande instruktionerna av guidningen samt parkeringen i slutet av bilfärden, detta kan ses i figur 5.8.

5.1.3 Parkeringen

När ankomsten till parkeringen är fullbordad kommer applikationen att presentera en dialog där man begär en bekräftelse att starta parkeringen för vald zon och fordon, detta kan ses i figur 5.9, vilket är steg 3 i destinations-driven parkering. En bekräftelse skall endast vara nödvändigt då applikationen redan vet vilken parkering användaren befinner sig vid.

Figur 5.9 Parkeringsdialog



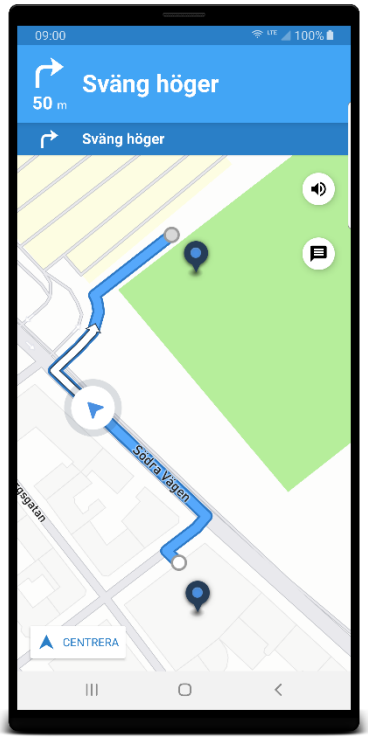
I figur 5.10 kan man se ett scenario där parkeringen som användaren kom till är full, när användaren klicka på knappen ’Full parkering’ som kan ses i figur 5.9. Vid klick av denna knapp, blir användaren tilldelad en ny närliggande parkering. Den nya parkeringen har genererats genom att välja den parkering som är närmast slutdestinationen.

Det bör förtydligas att användare inte kommer guidas till parkeringar, där de redan klickat på knappen ’Full parkering’. Om parkeringen accepteras kommer användaren att guidas till den nya parkeringen där det kommer presenteras en parkeringsdialog som i 5.9 med uppdaterad information.

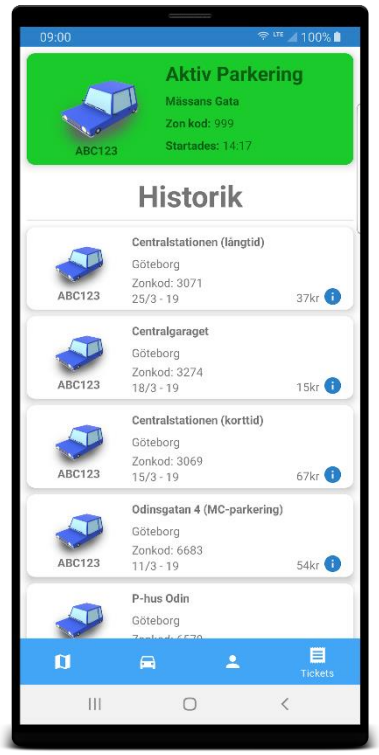
Figur 5.10 Omvald parkeringsdialog

5.1.4 Slutdestination

När bilisten har startat sin parkering kommer nästa steg bli att gå sista biten till sin slutdestination, vilket är steg 4 i destinations-driven parkering. Efter att bekräftelsen har skett av användaren gällande starten av parkering, kommer applikationen att gå vidare med att guida användaren via gång-anvisningar från parkeringen till sin slutdestination. I figur 5.11 kan man se att användaren blir guidad via gång från parkeringen P-Hus Avenyn Göteborg, som var den omvalda parkeringen i detta exempel, till slutdestinationen Heden Göteborg.



Figur 5.11 Navigation från parkering till destination



Figur 5.12 Biljettvyn som visar aktiva samt gamla biljetter

När användaren lämnar navigations-vyn, antingen genom att gå ur manuellt eller om användaren anländer till destinationen, kan man enligt figur 5.12 se sin aktiva samt gamla parkeringsbiljetter i biljett-vyn. Det skall förtydligas att detta enbart är en visuell del av applikationen som visar hur det hade kunnat se ut.

Kapitel 6 Diskussion

6.1 Vidareutveckling

I detta kapitel kommer tankar och idéer om vidareutveckling av applikationen att diskuteras. Det kan vara förbättringar till existerande funktionalitet eller helt ny funktionalitet, som skulle vara intressant att implementera.

6.1.1 Full parkering

Ett problem som skulle kunna uppstå med den utvecklade applikationen, är att hanteringen av att parkeringen som användaren har valt, är full då hen anländer till den. I kapitel 5 visade vi på en lösning på problemet genom att användaren har möjlighet att säga till att parkeringen är full. Nedan kommer det att presenteras olika lösningar som hade kunnat implementeras för att hantera eller att undvika detta moment nedan på ytterligare sätt.

6.1.1.1 Data om parkeringsplatser tillgänglighet

Tillgång till data om lediga parkeringsplatser samt hur många parkeringar varje parkeringsplats har, hade det kunnat finnas ytterligare lösningar till problemet gällande ankomst till full parkeringsplats, alternativt hur det hade kunnat förhindras tidigare.

Maskininlärning

För att få en bättre uppfattning av vad maskininlärning är, beskriver Ethem Alpaydin det på följande sätt:

"Maskininlärning är att programmera datorer för att optimera prestandakriterier genom att använda exempeldata eller tidigare erfarenheter. Vi har en modell som är definierad på ett antal parametrar, och lärandet är exekveringen av ett datorprogram för att optimera parametrarna av modellen genom att använda träningsdata eller tidigare erfarenheter. Modellen kan vara förutsägbar för att försöka förutsäga framtiden, eller beskrivande för att få kunskap från data, eller båda." [18]

Ett av alternativen hade kunnat vara att tillämpa maskininlärning för att statistiskt kunna överväga valet av parkering beroende på hur stor chansen är att parkeringen är tillgänglig vid ankomst. Funktionaliteten som applikationen nu besitter med att kunna hjälpa användaren ytterligare om parkeringen som var utvald var full samt vid start av parkering hade kunnat utnyttjas vidare. Det som hade varit möjligt att göra, är att föra statistik på när användare startar en parkering eller använder funktionen 'Full parkering'. Sedan hade en modell kunnat tränas att försöka förutsäga om parkeringen är ledig vid användarens beräknade ankomsttid. Hade denna typen av lösning funnits tillgänglig, hade det i ett tidigare skede, kunna påvisas för användaren om hur stor chansen är att vid ankomst att det finns en ledig parkering.

Under färd

Nästa sätt att hantera fulla parkeringar med att hjälpa av tillgången till data angående lediga platser kan vara att under färd kolla om parkeringen fortfarande är tillgänglig. Om den valda parkeringsplatsen under blir full under färden, skulle det vara attraktivt att automatiskt i bakgrunden kunna leda om användaren till en ledig parkering i närområdet som vi redan gör i vår applikation. När sensorer av dess olika slag blir billigare, hade det kanske kunnat vara möjligt att applicera dom på parkeringsplatser i större utsträckning som man redan kan se i nyare parkeringshus.

Reservation av parkering

Sista alternativet som hade kunnat vara aktuellt i framtiden, är som bilist kunna reservera en parkering för att säkerställa att ledig parkering finns tillgänglig på utvald plats. I länder som USA (77%), Tyskland (70%) och Storbritannien (71%), har man sett i studier att människor skulle vilja ha tillgång till mer avancerade reservations-metoder för parkeringar [2]. Skulle det finnas möjlighet att reservera sin parkering i framtiden hade det kunnat integreras och vidareutvecklas. En idé hade kunnat vara att i momentet när användaren väljer en specifik parkering ge möjligheten att reservera och garantera att en parkering finns tillgänglig vid ankomst. Undersökningar har påvisat att användare är villiga att betala cirka 5.12 kronor (£0.42) extra per resa med reservation för att sedan bli guidad till vald parkering, där två tredjedelar är för platsgarantin och en tredjedel för guidance [19].

6.1.2 Utveckla teknisk lösning till bilar

Undersökningar har påvisat att när frågan ställs var tekniska lösningar som skall hjälpa till med momentet att parkera, eftersöks det lösningar till navigeringssystemet i bilen [2]. Ett naturligt steg hade kunnat vara att vidareutveckla prototypen till att fungera i bilar. Det som skulle kunna vara aktuellt hade kunnat vara är att utöka och utveckla kompatibilitet för biltillverkares navigationssystem eller till liknande lösningar som Android Auto eller Apple CarPlay.

6.1.3 Integrera kollektivtrafik

Tanken att kunna erbjuda en integrering av kollektivtrafik i vår tekniska lösning hade varit intressant. Om parkeringar i närområdet inte finns eller valet att kunna integrera kollektiva medel i den tekniska lösningen skulle ge ytterligare friheter samt möjligheter när man ser på resan i helhet. Projekt med att göra urbana städer mer miljövänliga samt effektivare på att använda stadens ytor och minska antalet parkeringar skulle en vidareutveckling med integrering av kollektivtrafik vara intressant. Det hade även varit fördelaktigt om data angående tillgängligheten av parkeringar hade kunnat påvisas för att presentera alternativa färdmedel om parkeringar i närheten till destinationen är fulla.

6.1.4 Komplexa och långa resor

Den skapade tekniska lösningen som har gjorts i detta arbete fyller syftet att kunna lättare planera och överblicka enklare resor. Det skulle vara fördelaktigt att utveckla stöd till vägpunkter som användare längst med vägen vill åka via. Skulle resan vara ännu större som i en bilsemester eller när bilisten korsar gränser, tullar, åker färja eller dylikt skulle ytterligare utveckling behövas. Tanken är spännande att kunna planera för tull i förväg, planera in färjor eller annat för att förvandla denna tekniska lösning till att ytterligare kunna stödja mer än en enkel resa till att kunna hjälpa användare med att kunna planera en semester eller längre bilresor.

6.1.5 Hitta min bil samt navigation hemåt

En annan intressant funktion hade varit en ’hitta min bil’-funktion, då applikationen redan vet var du parkerade, hade det varit möjligt att se bilens position. När användaren sedan har hittat sin bil och börjar köra vidare hade applikationen kunnat fråga användaren om hen vill åka hem. Då kan applikationen navigera direkt hemåt alternativt till en ny destination om användaren önskar.

6.1.6 Hantera djup länkning via sökmotor

Att tekniska lösningar finns men inte alltid används kan vara en problematik som skulle kunna uppstå. En funktion som skulle kunna få användare att använda tjänsten, hade kunnat vara om applikationen hade kunnat hantera djupa länkar till applikationen. Det som menas är, att när man exempelvis söker på olika platser i en sökmotor, kan man klicka på dess adress, när adressen är klickad kan alternativ på vilken applikation man

vill öppna adressen med presenteras. Kan man i detta stadie välja vår applikation tas användaren in direkt i applikationen, adressen visas på kartan och närliggande parkeringar visas direkt.
Om vi utvecklat applikationen för att hantera denna typ av användarflöde skulle det kunna öka utnyttjandegraden av applikationen. Om adressen i form av en djup länk från sökmotorn hade kunnat bli tillgänglig samt hanterad av applikationen, hade det kunnat bli mer intuitivt för användare att hitta samt ytterligare använda applikationen. Tanken är då att processen med destinations-driven parkering blir kompletterad i momentet innan man är i bilen. Det behöver inte vara självklart att användare söker efter destinationer i applikationen direkt utan att användaren istället använder en sökmotor i första skedet av att planera sin resa. Då skulle det vara lämpligt med vidareutveckling för att hantera den typen av scenarion.

6.2 Applikationens nytta

6.2.1 För individen

Problematiken kring momentet att parkera kan summeras i två olika sidor, ekonomiska samt icke-ekonomiska konsekvenser. Att börja med icke-ekonomiska konsekvenser gällande parkeringar kan ett exempel vara upplevelsen för bilister; om önskad parkering inte hittas lätt kan det byggas upp en känsla av frustration samt stress. I den ekonomiska aspekten för bilister i sökandet av en lämplig parkering ödslas tid och bränsle i onödan.

I den skapade lösningen som utvecklades i detta projekt kan både icke-ekonomiska samt ekonomiska konsekvenser minskas. Med icke-ekonomiska konsekvenser hjälper applikationen användaren med helhetsresan vilket kan minska chansen till frustration som hade uppkommit vid letande av parkering. Frustrationen med momentet att parkera kan vara att hitta parkeringar i närheten till slutdestinationen, vilket vår applikation hjälper användaren med. Om söktiden minskas påverkas även ekonomiska konsekvenser positivt genom minskat bränsle och tidsåtgång.

6.2.2 För samhället

Det är inte bara individen som söker en parkering som påverkas av sökandet av parkeringar. I ett samhällsperspektiv blir det större utsläpp av växthusgaser, risk för köbildning samt en risk att individer inte hittar en parkering, vilket kan leda till att resan avfärdas och möjligheten att gynna lokal handel även går miste [2].
Allt flera urbana städer formas till att bli mer miljövänliga samt yt-effektiva eller som Ngai Weng Chan säger "a city should also be able to maximize its usage of limited land for buildings, housing, agriculture, transportation, recreation and other need" [20]. Att minska på antalet parkeringar i en stad och maximera utnyttjandet av mark skulle kunna leda till större söktid vilket i sig skulle leda till mer utsläpp, men hade en teknisk lösning funnits tillgänglig hade antalet parkeringar kunnat minskas men söktiden inte påverkas i samma grad. En annan nackdel i storstäder är att köbildning är en faktor när det blir många fordon. Hade sökandet av parkering minskat skulle antalet fordon i rotation på vägarna minskas vilket skulle minska tiden fordon står på tomgång vilket är en fördel för miljön.

I så kallade ekostäder försöker man att motverka användandet av privata bilar och minska på växthusgaser. Ett exempel på en sådan stad är Vancouver, Kanada, som har förbättrat möjligheterna genom bland annat förbättrad kollektivtrafik [20]. När städer vill minska på biltrafiken samt utöka kollektivtrafiken hade integreringen av kollektivtrafik blivit mycket intressant. Hade applikationen kunnat vidareutvecklas för att kunna integrera kollektivtrafiken in i helhetsresan samt göra det mer intuitivt för bilister att åka kollektivt, skulle det vara gynnsamt för miljön.

Samtidigt som man vill minska på biltrafiken i städer samt maximera ytor, vill man fortsätta stötta den lokala handeln i städer. Av alla parkeringar som skedde i Göteborg var 67% gjorda för shopping och nöje vilket är viktigt för lokal handel [3]. Vid minskande av parkeringar blir det ännu svårare för bilister att hitta parkering vilket kan leda till frustration. Frustrationen kan leda till att bilisten avfärdar sin avsikt med resan och åker vidare vilket drabbar lokal handel. Hade städer kunnat maximera sina ytor och

avlägsna parkeringar samtidigt som det är lätt för användaren att hitta till parkeringar kan oönskade effekter som kan drabba lokal handel kanske undvikas.

6.2.3 För Företag

Det finns en del branscher som hade kunnat utnyttja denna typen av slag där guidance genom hela resan hade kunnat förbättras. Vi kommer nedan att presentera olika branscher som hade kunnat utnyttja liknande koncept som skapad prototyp i detta projekt.

Parkering och betalningstjänster

För företag som skulle tillämpa denna typ av tekniska lösning för att hjälpa sina kunder med guidance, finns det även här ekonomiska samt icke-ekonomiska fördelar. Om valet av parkering sker mer proaktivt snarare än reaktivt för användare kan bolag presentera sina egna parkeringar. Det kan öka chansen att en av företagets parkeringar väljs istället för att valet blir reaktivt och chansen att en konkurrents parkering annars hade kunnat väljas. När det kommer till icke-ekonomiska fördelar kan användarupplevelsen bli positivare vid användandet av företagets parkeringar. Frustration i momentet att hitta en parkering minskas i ett tidigare skede, kan bidra till hela upplevelsen kring att parkera blir positivare. Det kan anses vara en icke-ekonomisk vinst men samtidigt på sikt kanske kunna få användare att använda tjänsten flertalet gånger; vilket hade kunnat leda till ekonomisk vinning.

Taxibranschen

En tanke hade kunnat vara att taxibranschen hade kunnat utnyttja denna typen av lösningen för att göra om processen av destinations-driven parkering till att mer passa en process för en kund i deras sammanhang. Tanken är att användarupplevelsen för en person som färdas med taxi kunnat förbättras. Tanken är att när en kund som vill ha en taxi beställer en resa hade man kunnat ha startpunkt samt slutdestination i åtanke för att förbättra helhetsupplevelsen av resan. Ett flöde som hade kunnat tillämpas till en kund som beställer en taxi hade kunnat ske på följande sätt:

1. Kund beställer taxi med önskad start och destination. Kostnad, restid, ankomsttid, upphämtningsplats, avlämningsplats samt att resvägen kan överblickas med avseende till start och slutdestination.
2. Taxi ankommer till upphämtningsplats. Kund informeras samt kan få guidance till upphämtningsplats om så önskas.
3. När taxin färdas mot avlämningsplats kan kunden överblicka resan om så önskas via en karta.
4. När taxi ankommer till avlämningsplats skall betalning ske enkelt via en bekräftelse, för att sedan erbjuda kunden guidance från taxi till slutdestination.

Leveransombud

Ytterligare en bransch som kanske hade kunnat utnyttja denna typen av guidance hade kanske kunnat vara för leveransombud. Hade en förares leveransadresser samlats in samt att en optimerad resväg hade skapats hade processen kunnat ske på följande sätt:

1. Föraren får visualiserat för sig över dagens leveranser genom att resvägarna visas på en karta. När resvägen visualiseras hade det kunnat vara lämpligt att visa beräknad ankomsttid till varje leveransadress som blir som vägpunkter längst med resan. Dessa beräknade ankomsttider skickas till var kund för beräknad ankomsttid.
2. Föraren får guidance till första leveransadress.
3. Vid ankomst i närhet till leveransadress erbjuds guidance från fordon till leveransadress via gång. I detta steg kan även kund få avisering om att ombud snart är på plats.
4. Förare skall enkelt kunna bekräfta överlåtande av paket vid ankomst samt signatur av kund i samma applikation.
5. Guidance tillbaka till fordon kan erbjudas.
6. Sedan blir föraren guidad till nästa leveransadress och steg 3,4,5 utförs igen.
7. Steg 6 upprepas tills alla leveransadressen är utförda för att sedan kunna bli guidad tillbaka till förarens utgångspunkt tidigare av dagen.

Kapitel 7 Slutsats

Det som vi har skapat är en prototyp som visar hur helheten i en resa kan hanteras bättre för att gagna användare. Med tanken på att användare tidigare har valt parkering reaktivt, kan det med det vi vill kalla destinations-driven parkering erbjuda en teknisk lösning som underlättar helhetsresan för användare. Valet av parkering sker nu proaktivt samt mer inkluderat i helheten av resan, genom att göra parkeringen mer relevant i ett tidigare skede.

Applikationen som utvecklades hjälper och kan guida användaren genom hela sin resa från start till slutdestination via en parkering. Att kunna få ett tekniskt hjälpmedel för vad som tidigare har gjorts mer manuellt och reaktivt kan bidra till en positivare parkeringsupplevelse. Nu när en prototyp har skapats och presenterats är det lägligt med en fallstudie där frågor kring hur helhetsresan upplevdes med destinations-driven parkering, för att sedan göra nya avvägningar för hur applikationen ytterligare hade kunnat vidareutvecklas. Frågor som hade velat besvaras är bl.a. värdet att slippa byta applikation för alla delmoment i resan, upplevelsen av att resan är guidad från start till slut, samt om lösningen minskar stress och frustration som annars hade kunnat ske reaktivt vid valet av parkering på plats.

Kapitel 8 Referenser

[1] E. Andrén, ”Nu finns det en miljard bilar i världen,” Teknikens Värld, 19 08 2011. [Online]. Available: https://teknikensvarld.se/nu-finns-det-en-miljard-bilar-i-varlden-122180/. [Använd 05 06 2019].

[2] G. Cookson och B. Pishue, ”The Impact of Parking Pain in the US, UK and Germany,” *INRIX Research,* vol. 1, nr 1, pp. 6-36, 07 2017.

[3] Göteborgs stads parkeringsaktiebolag, ”P-bolaget göteborg,” 5 2017. [Online]. Available: https://www.p-bolaget.goteborg.se/globalassets/foto/pdf-filer-allmant/parkeringsrapport-februari-2017.pdf. [Använd 20 5 2019].

[4] K. Teknomo och H. Kazunori, ”Parking Behavior in Central Business District - A Case Study of Surabaya, Indonesia.,” *EASTS,* vol. 2, nr 2, pp. 551-570, 1997.

[5] C. Qi, ”Short-Stay Car Parking Choice Behaviour: A Case Study of Cardiff City Centre,” Cardiff University, Cardiff, 2014.

[6] M. Xiaolong, S. Xiaoduan, H. Yulong och C. Yixin, ”Parking choice behavior investigation: A case study at Beijing Lama Temple,” *Procedia Social and Behavioral Sciences,* vol. 96, nr 1, p. 2640, 2013.

[7] J. Widerberg, ”Västeråsarna betalar oftare med mobilen – nästan 70 procents ökning,” Vestmanlands Läns Tidning, 25 01 2017. [Online]. Available: https://www.vlt.se/artikel/vasterasarna-betalar-oftare-med-mobilen-nastan-70-procents-okning. [Använd 20 05 2019].

[8] Android Developers, ”Application Fundamentals,” Google, [Online]. Available: https://developer.android.com/guide/components/fundamentals?hl=en. [Använd 20 05 2019].

[9] Android Developers, ”Google I/O 2017,” Google, [Online]. Available: https://android-developers.googleblog.com/2017/05/google-io-2017-empowering-developers-to.html. [Använd 20 05 2019].

[10] Google, ”Google I/O 2019,” Google, [Online]. Available: https://youtu.be/LoLqSbV1ELU?t=487 . [Använd 20 05 2019].

[11] JetBrains, ”Kotlin Language FAQ,” JetBrains, [Online]. Available: https://kotlinlang.org/docs/reference/faq.html. [Använd 20 05 2019].

[12] JetBrains, ”Kotlin Language Null-safety,” JetBrains, [Online]. Available: https://kotlinlang.org/docs/reference/null-safety.html. [Använd 20 05 2019].

[13] JetBrains, ”Scoping Functions,” JetBrains, [Online]. Available: https://kotlinlang.org/docs/reference/scope-functions.html. [Använd 16 06 2019].

[14] Microsoft, ”The MVVM pattern,” Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10). [Använd 20 05 2019].

[15] Android Developers, ”Guide to app architecture,” Google, [Online]. Available: https://developer.android.com/jetpack/docs/guide. [Använd 20 05 2019].

[16] Square, ”Retrofit,” Square, [Online]. Available: https://square.github.io/retrofit/. [Använd 03 06 2019].

[17] Android Developers, ”Livedata,” Google, [Online]. Available: https://developer.android.com/topic/libraries/architecture/livedata. [Använd 20 05 2019].

[18] E. Alpaydın, Introduction to Machine Learning Second Edition, Second Edition red., Cambridge, Massachusetts: Massachusetts Institute of Technology, p. 3.

[19] P. Wockatz och P. Schartau, ”Traveller Needs and UK Capability Study,” *Transport Systems Catapult,* vol. 1, nr 1, p. 27, 10 2015.

[20] N. W. Chan, ”URBANIZATION, CLIMATE CHANGE AND CITIES: CHALLENGES AND OPPORTUNITIES FOR SUSTAINABLE DEVELOPMENT,” Penang, 2017.

[21] Mapbox, ”Maps,” Mapbox, [Online]. Available: https://www.mapbox.com/maps/. [Använd 20 05 2019].

[22] Mapbox, ”About,” Mapbox, [Online]. Available: https://www.mapbox.com/about/. [Använd 20 05 2019].

[23] JetBrains, ”Scope Functions,” 16 06 2019. [Online]. Available: https://kotlinlang.org/docs/reference/scope-functions.html.

[24] Google, ”Guide to app architecture,” Google, [Online]. Available: https://developer.android.com/jetpack/docs/guide. [Använd 16 06 2019].