ARCADA

# Utveckling av en mobil applikation som användargränssnitt mot en webbtjänst för automatisk tolkning av parkeringsmärken

Niklas Knappe

Lärdomsprov

Informationsteknik

2023

# Lärdomsprov

Niklas Knappe

Utveckling av en mobil applikation som användargränssnitt mot en webbtjänst för automatisk tolkning av parkeringsmärken

Yrkeshögskolan Arcada: Informationsteknik, 2023.

## Identifikationsnummer:

9189

## Uppdragsgivare:

## Sammandrag:

En vardaglig sak för många är att kunna känna igen olika parkeringsmärken och om märkena tillåter en att parkera på en viss plats. Den teoretiska delen av arbetet beskriver vilka alternativ det fanns och varför de valdes för att utveckla Android-applikationen, hur applikationen fungerar och vad som kunde göras i framtiden för att utveckla den vidare. Syfte med den praktiska delen av arbetet var att utveckla en Android-applikation med hjälp av Android Studio, Jetpack Compose och Kotlin för att fungera som användargränssnitt för en molntjänst som känner igen parkeringsskyltar. Molntjänsten utvecklades av en annan studerande på Yrkeshögskolan Arcada, Joakim Mutka, som en del av hans examensarbete. För tillfället fungerar prototypen bara på Android-telefoner och kan bara känna igen vissa märken. Till framtidsplanerna hör att utveckla mera funktionalitet i applikationen och en iOS-version av applikationen för att kunna nå så många användare som möjligt.

## Nyckelord:

Android, Jetpack Compose, Kotlin, Datorseende

# Degree Thesis

Niklas Knappe

Development of a mobile application as a user interface that uses a web service to automatically recognize parking signs

Arcada University of Applied Sciences: Information technology, 2023.

## Identification number:

9189

## Commissioned by:

## Abstract:

An everyday task for many is being able to recognize different parking signs and whether the signs allow you to park your car in a certain place at a certain time. The theoretical part of the thesis describes what options there were and why they were chosen to develop the Android application, how the application works and what could be done in the future to develop it further. The purpose of the practical part of the thesis was to develop an application with the help of Android Studio, Jetpack Compose and Kotlin to serve as a user interface for a web service that recognizes parking signs. The web service was developed by another student at Arcada University of Applied Sciences, Joakim Mutka, as a part of his thesis. At the moment the prototype only works on Android phones and can only recognize a few signs. Future plans is to add more functionality in the application and make an iOS version of the application to reach as many users as possible.

## Keywords:

Android, Jetpack Compose, Kotlin, Computer vision

# Innehåll

# Figurer

# 1 Inledning

## 1.1 Bakgrund

Detta examensarbete har gjorts i formen av ett samarbetsprojekt med Joakim Mutka, en annan studerande i informationsteknik vid Yrkeshögskolan Arcada.

En vardaglig och allmän utmaning är att det är svårt att tolka parkeringsskyltar och parkeringsförbud när de är utrustade med tilläggsskyltar. Iltalehti hade ett test på sin nätsida var de frågade vad dessa två märken betyder. (se figur 1)


*Figur 1. Trafikmärkena C38: Parkering förbjuden, E2: Parkeringsplats och H18: Tidsbegränsning*

84 % trodde att parkeringsförbudsmärket utrustad med en 24 h tilläggsskylt betydde parkering förbjuden dygnet runt men i verklighet betyder det att man får parkera för 24 timmar. 78 % visste att parkeringsskylten utrustad med en 24 h tilläggsskylt betydde att man fick parkera för 24 timmar men 44 % trodde att det var förbjudet att parkera på lördag och söndag men det är det inte (Rönkkö, 2018).

I olika medier har det diskuterats om behovet för en mobilapplikation som kunde berätta åt användaren om och hur man får parkera på en specifik parkeringsplats. Joakim nämnde det till mig och vi började diskutera om att det låter inte så svårt och det måste väl redan finnas en applikation som kan göra det men efter att vi hade sökt en stund så hittade vi inget. Det finns ett par applikationer, till exempel Parkopedia och JustPark

som du kan leta parkeringsplatser i men i dem syns inte alla möjligheter. Då kom idén att bygga en mobilapplikation som kunde göra det.

## 1.2 Syfte och mål

I detta arbete dokumenteras utvecklingsprocessen av användargränssnittet, vilka utvecklingsverktyg som användes, förverkligande av applikationens tekniska struktur samt testning. Syftet med praktiska delen av detta arbete är att skapa en prototyp av en mobilapplikation som skickar bilden användaren tagit av en parkeringsskylt till en server var bilden analyseras och sedan skickas information om skylten till applikationen som visar relevant information om den. Syftet med den teoretiska delen är att ge en insikt i olika teknologier som används i utveckling av en Android applikation samt beskriva hur denna applikation fungerar och är uppbyggd.

## 1.3 Metoder

Vi har med Joakim använt den agila systemutvecklingsmetoden då vi arbetat tillsammans. Agil systemutveckling går ut på att projektet görs inkrementellt. Projektet görs i mindre delar med flera regelbundna möten som orsakar att projektet görs i ett nära samarbete mellan utvecklare och den som projektet görs åt. Detta gör det lätt att hålla koll på att alla delar av projektet framskrider i samma takt så att det inte finns delar av projektet som är klara och sådana som inte. Det gör det också lätt att testa att allt funkar tillsammans. Den som projektet görs åt ser också att projektet framskrider. Android Studio valdes som utvecklingsmiljö för mobilapplikationens utveckling på grund av att den är den mest använda utvecklingsmiljön och har allting inbyggt som man behöver för att utveckla en Android-applikation. Programmeringsspråket som mobilapplikationen skrivs i är Kotlin eftersom det idag är det vanligaste språket för applikationsutveckling i Android Studio. Över 60 % av professionella Android-utvecklare i världen använder redan Kotlin (Android Developers, 2023a). Användargränssnittet är byggt med hjälp av Jetpack Compose som gör utvecklande väldigt snabbt och lätt. Vad än för element du vill bygga så kräver det mindre kod än med XML som man använder om man inte använder Jetpack Compose.

# 2 Verktyg

Detta kapitel omfattar vilka verktyg användes för att bygga upp användargränssnittet samt några alternativ som kunde ha använts.

## 2.1 Android Studio

Det finns några alternativ att välja mellan när man väljer sin utvecklingsmiljö för att utveckla en Android applikation, till exempel Eclipse och Xamarin men de flesta rekommenderar Android Studio när man letar efter en utvecklingsmiljö för Android-applikationer. Android Studio är öppen källkod och gratis, det finns inga gömda kostnader eller begränsningar för att använda programmet. Det är den officiella integrerade utvecklingsmiljön menad för att utveckla Android-applikationer (Android Developers, 2023b). Den är utvecklad av Google och släpptes ut i slutet av 2014. Android Studio erbjuder många funktioner för att underlätta utvecklande av Android applikationer såsom enhetstestning, emulator, versionshantering, profileringsverktyg och mycket mera.

## 2.2 Jetpack Compose vs XML

Jetpack Compose är ett verktyg för att bygga grafiska användargränssnitt för Android applikationer. I stället för att använda XML-filer som i traditionell Android utveckling så skriver man Compose-funktioner i Kotlin programmeringsspråket för att skapa element i användargränssnittet. (se figur 2)

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button"
    android:textColor="#FF0000"
    android:textSize="16sp"/>
```

```
@Composable
fun ButtonExample(){
    Button(onClick = { /*TODO*/ }) {  this: RowScope
        Text(text = "Button", Color( color: 0xFFFF0000), fontSize = 16.dp)
    }
}
```

*Figur 2. En knapp med röd text i XML(övre) och i Jetpack Compose(nedre)*

Fördelar med XML är att det är väl känt och stabilt. Det finns mycket information om hur saker och ting skall göras i det. XML gör det också lätt att återanvända layouter dvs. du kan använda samma fil i olika delar av din applikation för att skapa till exempel likadana knappar och textrutor. När man använder sig av XML så hålls layouter i sina egna filer och koden man kör i sina egna filer. Nackdelar med XML är att du hamnar byta mellan språk när du utvecklar din applikation. Först skriver du funktioner i Java eller Kotlin och sedan byter du till XML för att till exempel ändra hur en knapp ser ut. XML kräver också mycket mera rader av kod för att göra samma sak som Compose vilket kan göra XML-filer väldigt stora och komplexa. Fördelar med att använda Compose-funktioner är enkelhet, effektivitet och flexibilitet. Compose-funktionerna låter dig se ändringar i din applikations utseende utan att behöva starta om den varje gång du vill se en ändring. Nackdelarna med Compose är att det är relativt nytt. Version 1.0 släpptes ut 28 juli 2021 så det finns ännu inte lika mycket information om det som hur man gör saker i XML (Bellini, 2021). Det kommer hela tiden mycket förbättringar till Compose och applikationer byter sakta över från traditionella XML-filer till Compose-funktioner.

## 2.3 Kotlin vs Java

För att Java har funnits länge och varit ett av språken som man har använt mest för att

utveckla Android applikationer så är det stabilt, väl testat och det finns mycket information om hur saker skall göras. Java är plattformsoberoende vilket innebär att applikationer som utvecklas i Java kan köras på många olika plattformar utan att behöva skriva om koden, att inte behöva skriva om kod när du överför ett projekt från en plattform till en annan är ett stort plus då man inte behöver lägga tid på att skriva om något. Kotlin är ett relativt nytt språk som har börjat användas mera hela tiden och många för över sin Java kod till Kotlin kod. Syntaxen i Kotlin är mera koncis än i Java vilket innebär att man får samma sak gjort i Kotlin med mindre kod än i Java. (se figur 3) Rekommenderade programmeringsspråket för Android applikationer ändrades 7 maj 2019 från Java till Kotlin (Lardinois, 2019).

```java
Context context = getApplicationContext();
CharSequence text = "Hello World!";
int duration = Toast.LENGTH_SHORT;
Toast toast = Toast.makeText(context, text, duration);
toast.show();
```

```kotlin
val context = applicationContext
val text = "Hello World"
val duration = Toast.LENGTH_SHORT
val toast = Toast.makeText(context, text, duration)
toast.show()
```

*Figur 3. Toast exempel i Java(övre) och Kotlin(nedre)*

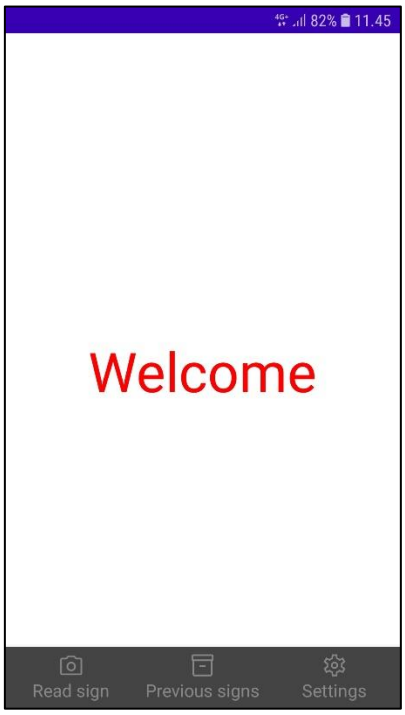
# 3 Applikationens struktur och funktionalitet

Detta kapitel omfattar applikationens struktur och funktionalitet

## 3.1 Applikationsflöde

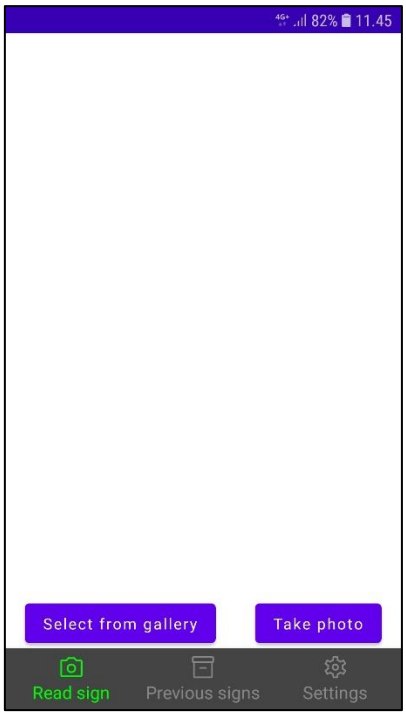
När applikationen är nerladdad och startad så presenteras användaren med huvudvyn. I framtiden kunde denna vy fråga om användaren vill logga in, skapa ett konto eller fortsätta använda applikationen utan ett konto. Med hjälp av ett konto kunde bilder man tidigare tagit sparas i en molntjänst för att i framtiden kunna se dem. Också

inställningarna kunde sparas där för att lätt få samma inställningar tillbaka om man byter telefon. (se figur 4)




*Figur 4. Huvudvyn*

Härifrån kan man navigera med hjälp av navigeringsbalken nere på skärmen till olika vyer och mellan dem. I Read sign vyn kan man välja mellan att ta en bild av en parkeringsskylt med hjälp av kameran eller välja en bild från sitt galleri som man tagit tidigare. Implementering av möjligheten för att välja en bild från användarens galleri funderades på en stund om detta är nödvändigt men det finns säkert olika scenarion var denna funktionalitet skulle uppskattas. (se figur 5)




*Figur 5. Read sign vyn*

Om det är första gången man öppnar applikationen så kommer applikationen att fråga lov att använda kameran om man trycker på Take photo knappen eller fråga lov att läsa galleriet om man väljer Select from gallery. Om man inte ger lov så kommer inte funktionen man försökte använda fungera. När du tagit en bild med kameran eller valt en från galleriet så presenteras du med en vy som frågar om man är nöjd med bilden var du kan antingen välja att ta en ny bild eller skicka i väg bilden till servern som skall känna igen vilka märken finns i bilden. (se figur 6)


*Figur 6. Vyn efter att man valt en bild som man vill skicka*

När användaren skickat bilden till servern och servern skickat information tillbaka så visar applikationen informationen servern hittat till användaren. För tillfället visar applikationen om igenkänningen av bilden lyckades, vilka skyltar den hittade i bilden och berättar åt användaren om och hur länge man får parkera. (se figur 7) I framtiden kunde denna vy också fråga om alla skyltar i bilden hittades och om de kändes igen korrekt. Sedan kunde användaren trycka på en Send information knapp som skulle öppna en liten ruta var man kan välja om skyltarna kändes igen korrekt eller inte och om den inte kände igen korrekt, vilka gick fel. Denna knapp skulle inte vara obligatorisk att trycka på men om man vill hjälpa att utveckla applikationen kunde denna funktion användas. Sedan skulle informationen skicks till servern för att ge den mera data för att känna igen skyltar som är mera sällsynta och förbättra igenkänningen allmänt också. I Previous signs eller Settings vyerna finns det för tillfället ingenting i.


*Figur 7. Informationen som hittades i bilden*

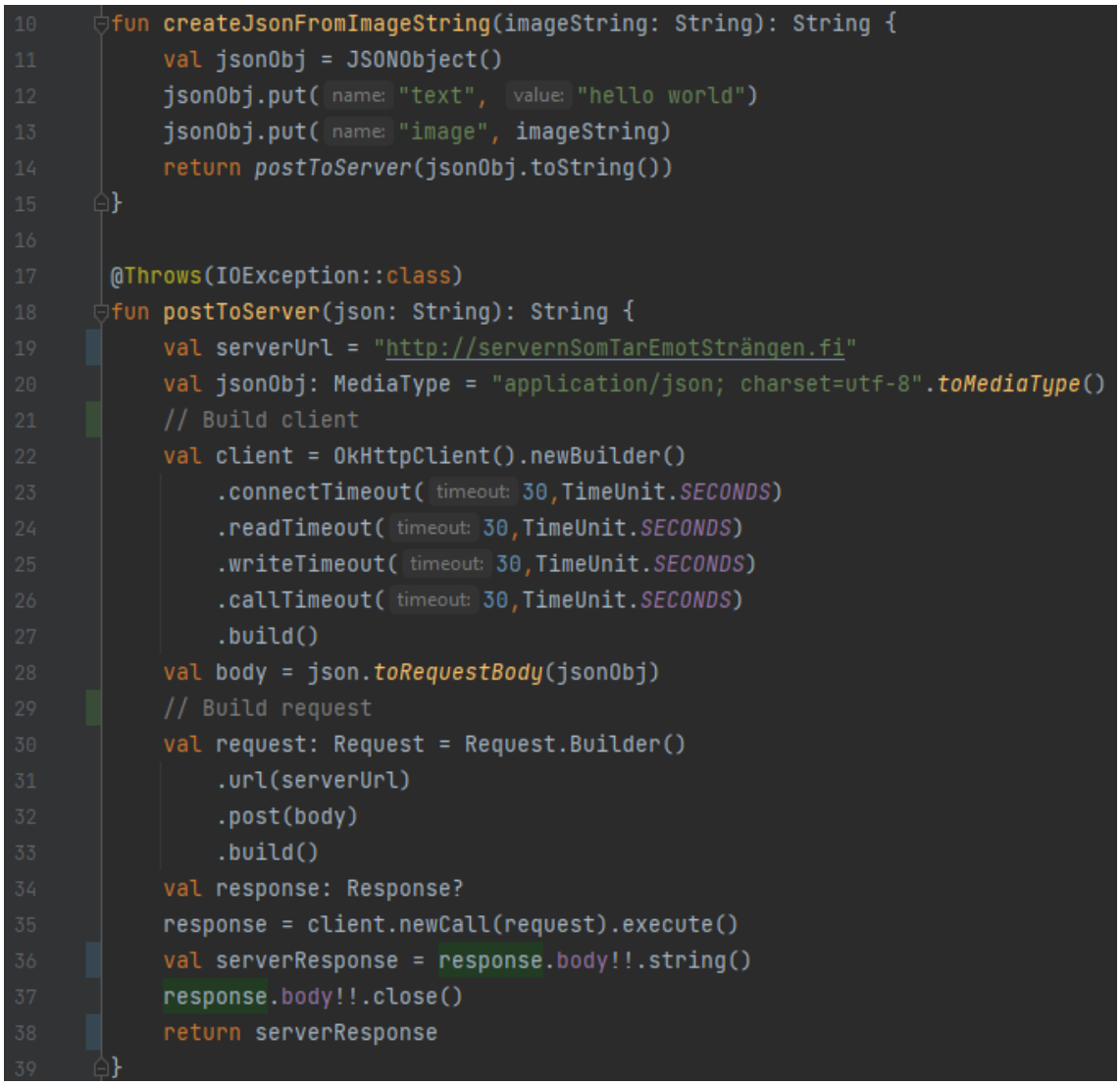
## 3.2 Vad händer med bilden?

Bilden som användaren tar sparas inte på telefonens minne utan bara i temporärt minne så att användarens galleri inte skulle fyllas med bilder som hen inte gör något med. När användaren är nöjd med den tagna bilden och tryckt på att skicka i väg bilden så skickas bildens URI till funktionen imageToBase64 var bilden komprimeras för att göra upp-laddningen samt igenkänningen av bilden snabbare. Efter komprimeringen konverteras bilden till en byte array och sedan till en sträng av Base64-format. (se figur 8)

```
fun imageToBase64(context: Context, uri: Uri): JSONObject {
    val input = context.contentResolver.openInputStream(uri)
    val baos = ByteArrayOutputStream()
    BitmapFactory.decodeStream(input,  outPadding: null,  opts: null)?.compress(Bitmap.CompressFormat.JPEG,  quality: 50, baos)
    input?.close()
    baos.close()
    val imageBytes = baos.toByteArray()
    val imageBase64 = Base64.encodeToString(imageBytes, Base64.NO_WRAP)
    val returnedJsonString = createJsonFromImageString(imageBase64)
    return JSONTokener(returnedJsonString).nextValue() as JSONObject
}
```

*Figur 8. Komprimering och konvertering av bilden*

Sedan läggs strängen i ett JSON-objekt och skickas till servern via en POST-metod. (se figur 9)



```
fun createJsonFromImageString(imageString: String): String {
    val jsonObj = JSONObject()
    jsonObj.put( name: "text",  value: "hello world")
    jsonObj.put( name: "image", imageString)
    return postToServer(jsonObj.toString())
}

@Throws(IOException::class)
fun postToServer(json: String): String {
    val serverUrl = "http://servernSomTarEmotSträngen.fi"
    val jsonObj: MediaType = "application/json; charset=utf-8".toMediaType()
    // Build client
    val client = OkHttpClient().newBuilder()
        .connectTimeout( timeout: 30,TimeUnit.SECONDS)
        .readTimeout( timeout: 30,TimeUnit.SECONDS)
        .writeTimeout( timeout: 30,TimeUnit.SECONDS)
        .callTimeout( timeout: 30,TimeUnit.SECONDS)
        .build()
    val body = json.toRequestBody(jsonObj)
    // Build request
    val request: Request = Request.Builder()
        .url(serverUrl)
        .post(body)
        .build()
    val response: Response?
    response = client.newCall(request).execute()
    val serverResponse = response.body!!.string()
    response.body!!.close()
    return serverResponse
}
```

*Figur 9. Funktionerna som gör ett JSON-objekt och skickar den till servern*

Vi valde att skicka bilden till servern i Base64-format för att det är lättare att skicka en sträng än att skicka en fil. Servern tar emot JSON-objektet, läser strängen, konverterar den till en bild och känner igen trafikmärkena. Efter det skickar servern ett JSON-objekt med data den hittat tillbaka till applikationen och applikationen visar resultatet åt användaren. (se figur 10)

```
Box(modifier = Modifier.fillMaxSize()){ this: BoxScope
    if (imageSent){
        Column(
            modifier = modifier.fillMaxSize(),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center,
        ) { this: ColumnScope
            var loadingImage = true
            // Wait for image to load, then display result
            while (jsonObject.length() == 0){
                if (loadingImage){
                    loadingImage = false
                }
            }
            Text(text = "Result: " + jsonObject.getString( name: "result"), fontSize = 16.sp)
            Text(text = "Detected signs:", fontSize = 16.sp)
            Text(text = jsonObject.getString( name: "detected_signs"), fontSize = 16.sp)
            Text(text = "Can you park? " + jsonObject.getString( name: "can_you_park"), fontSize = 16.sp)
            // Empty jsonObject when it has been displayed
            jsonObject = JSONObject()
        }
```

*Figur 10. En Jetpack Compose-layout som visar till användaren vad som hittades i bilden.*

# 4 Testning och framtidsplaner

## 4.1 Testning

Applikationen har testats regelbundet under utvecklingen på en Samsung A5 2017 vars API-nivå är 27. När applikationen var klar testades den också på en OnePlus 6 vars API-nivå är 30, då märktes det att den inte fungerade rätt. Det hade att göra med att i API-nivå 28 ändrades de så att data man skickar från telefonen måsta vara krypterat. Det skickas för tillfället inga sensitiva data mellan applikationen och servern så beteen-det ändrades så att applikationen tillåter överföringen data som inte är krypterat. Funktionaliteten mellan applikationen och servern har testats alltid när det skett änd-ringar i kommunikationen eller i formatet data skickas. Testen bestod av att ta en bild med telefonen, skicka den till servern för att kännas igen och sedan läsa informationen som hittades i bilden på telefonen.

## 4.2 Framtidsplaner och vidareutveckling

För tillfället är applikationen ännu en prototyp men ett bevis på att det är möjligt att ut-veckla en applikation som kan känna igen trafikmärken.

Innan applikationen är redo att lanseras på Google Play så krävs det en del finslipning, felhantering och användarvänligare vyer. För tillfället så kommer det inga felmeddelande om applikationen inte kan nå servern, applikationen kraschar om det tar för länge för servern att behandla informationen man skickat till den och vyn som berättar om du får parkera kunde se grafiskt bättre ut. Det är bara några av många saker som borde utvecklas vidare för att ha en väl funktionerande applikation. Applikationen borde också testas på flera mobiltelefoner för att försäkra funktionalitet.

Vi funderade på i projektets början om vi borde känna igen bilderna lokalt på telefonen eller skicka dem till en server som har bättre hårdvara för att skära ner på tiden det tar att känna igen bilderna. Vi kom fram till att känna igen bilderna på en server så att projektet skulle tydligt delat på hälften och för att vi ville att applikationen kunde köras på långsammare telefoner också. I framtiden skulle vi villa implementera möjlighet för offline-läge så att igenkänningen skulle kunna ske på telefonen.

Andra framtida ändringar man skulle kunna implementera:

- Svenska och finska som språkalternativ
- Sparandet av tidigare bilder och resultat
- Visa åt användaren var informationen i bilden hittats
- Inbyggd trafikmärkeskatalog

För att så många som möjligt kunde använda applikationen borde den också utvecklas för iOS som är operativsystemet Apple-telefoner använder sig av.

# 5 Slutledning

I detta arbete har verktygen som använts för att förverkliga applikationen samt hur applikationen fungerar tagits fram. Med Joakim Mutka skapades en prototyp av en fullständig applikation som man kan ta bilder med som sedan skickas till en server som känner igen olika parkeringsmärken och berättar åt användaren om hen får parkera.

Det märks att det finns ett behov för en sådan applikation när man läst artiklar i olika kvällstidningar var det frågas hur man tolkar olika parkeringsskyltar. För tillfället finns

det inte en sådan applikation på marknaden. Det finns applikationer som berättar var det finns parkeringsplatser men i dem finns inte alla parkeringsplatser och användaren måste känna till parkeringsmärken när hen kommer fram.

För att utveckla applikationen vidare kräver det mycket arbete ännu. Finslipa allting samt lägga till funktioner som man förväntar sig av alla applikationer som till exempel olika språkalternativ. Applikationen borde också utvecklas för iOS så att så många som möjligt kunde utnyttja den.

Jag är riktigt nöjd med prototypen vi utvecklat och jag kommer säkert att fortsätta utveckla Android-delen på min fritid.

# Källor


Android Developers. (17 maj 2023a). *Write better Android apps faster with Kotlin. Kotlin is a modern statically typed programming language used by over 60% of professional Android developers that helps boost productivity, developer satisfaction, and code safety*. Android Developers https://developer.android.com/kotlin

Android Developers. (17 maj 2023b). *Get the official Integrated Development Environment (IDE) for Android app development*. Android Developers. https://developer.android.com/studio

Bellini, A.-C. (28 juli 2021). *Jetpack Compose is now 1.0: announcing Android's modern toolkit for building native UI*. Android Developers Blog. https://android-developers.googleblog.com/2021/07/jetpack-compose-announcement.html

Lardinois, F. (7 maj 2019). *Kotlin is now Google's preferred language for Android app development*. TechCrunch. https://techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development/

Rönkkö, P. J. (23 november 2018). *Harva tietää, mitä nämä liikennemerkit tarkoittavat?*. Iltalehti. https://www.iltalehti.fi/autouutiset/a/201710112200454429