BACHELOR'S THESIS 2022

Intelligent klockapplikation

Robin Kollenman

ISSN 1651-2197
LU-CS/HBG-EX: 2022-04

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

Intelligent Klockapplikation

Mobill Scandinavia AB

LUNDS UNIVERSITET
Campus Helsingborg

LTH Ingenjörshögskolan vid Campus Helsingborg
Institutionen för datavetenskap

Examensarbete:
Robin Kollenman

© Copyright Robin Kollenman

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

Tryckt i Sverige
Lunds universitet
Lund 2022

Sammanfattning

Examensarbetet har gått ut på att utveckla en klockapplikation åt parkeringsföretaget Mobill. I dagsläget har Mobill en mobilapp som gör det möjligt för användare att påbörja parkeringsperioder i parkeringszoner, där Mobill har ett samarbete med markägaren (till exempel kommunen). Dock är det en manuell process eftersom användaren måste ta upp mobiltelefonen för att påbörja eller avsluta en parkeringsperiod. Detta ville Mobill ändra på genom att automatisera parkeringsproceduren. För att realisera detta så utvecklades en klockapplikation som kör på Androids operativsystem Wear-OS. Denna klockapplikation kommunicerar i sin tur med Mobills mobilapp, och kan ses som ett komplement till det befintliga systemet, med huvuduppgiften att förenkla parkeringsproceduren för bilister från start till slut. Vidare skulle klockapplikationen göra denna procedur snabbare, utan krav på att användaren måste använda den befintliga mobilappen för att parkera respektive avsluta en parkeringsperiod. Med andra ord implementerades en intelligent klockapplikation, som kan detektera om användaren befinner sig i närheten av en parkeringszon, samt om användaren står still och att klockan är ansluten till ett fordon via Bluetooth. Om dessa villkor är uppfyllda kan klockapplikationen fråga användaren om en parkeringsperiod skall påbörjas inom given parkeringszon. Med ett enda knapptryck kan sedan perioden påbörjas. På samma sätt detekterar klockapplikationen om användaren kör ut från en parkering-zon där det fortfarande finns en aktiv period. Om hastigheten är större än noll, samt att en anslutning via Bluetooth är upprättad mellan fordonet och klockan, kan användaren få en förfrågan på klockskärmen om parkeringsperioden skall avslutas. Med ett enda knapptryck kan perioden därefter avlutas. Användaren sparar tid när parkeringsproceduren automatiseras. I dagsläget erbjuder inga konkurrenter inom parkeringsbranschen samma typ av lösning, vilket gör examensarbetets resultat unikt.

Nyckelord:

Mobill

Android Wear OS

WearableListenerService

MessageClient API

Java-Threads

Activity

Service

FusedLocationProviderClient

Abstract

The degree project has involved developing a watch application for the parking company Mobill. At present, Mobill has a mobile app that makes it possible for users to start parking periods in parking zones, where Mobill has a collaboration with the landowner (for example the municipality). However, it is a manual process because the user must pick up the mobile phone to start or end a parking period. Mobill wanted to change this by automating the parking procedure. To realize this, a watch application was developed that runs on Android's operating system Wear-OS. This watch application in turn communicates with Mobile's mobile app and can be seen as a complement to the existing system, with the main task of simplifying the parking procedure for motorists from start to finish. Furthermore, the clock application would make this procedure faster, without requiring the user to use the existing mobile app to park or end a parking period, respectively. In other words, an intelligent clock application was implemented, which can detect if the user is in the vicinity of a parking zone, as well as if the user is standing still and that the clock is connected to a vehicle via Bluetooth. If these conditions are met, the clock application can ask the user if a parking period should be started within a given parking zone. With a single push of a button, the period can then begin. Likewise, the clock application detects if the user is driving out of a parking zone where there is still an active period. If the speed is greater than zero, and a connection via Bluetooth is established between the vehicle and the watch, the user receives a notification on the watch screen inquiring about if the parking period shall end. With a single push of a button, the period can then be ended. The user saves time when the parking procedure is automated. At present, no competitors in the parking industry offer the same type of solution, which makes the results of the thesis unique.

Keywords:

Mobill

Android Wear OS

WearableListenerService

MessageClient API

Java-Threads

Activity

Service

FusedLocationProviderClient

Innehållsförteckning

1. **Introduktion** ... 1

1.1 Bakgrund ... 1

1.2 Intelligent Klockapplikation ... 1

1.3 Wear OS ... 1

1.4 Funktioner ... 2

1.5 Syfte ... 2

1.6 Målformulering ... 3

1.7 Problemformulering ... 3

1.8 Motivering av examensarbetet ... 3

1.9 Avgränsningar ... 3

2. **Teknisk Bakgrund** ... 4

2.1 Wear OS versus Android OS ... 4

2.2 Utvecklingsverktyg ... 5

2.2.1 Android Studio versus Eclipse: Likheter och skillnader ... 5

2.2.2 Xamarin.Android ... 6

2.3 Mobills befintliga mobilapplikation ... 6

2.4 Mobills klockapplikation på Wear OS ... 8

3. **Metod** ... 9

3.1 Scrum-liknande projektmodell ... 9

3.2 Sprintar ... 10

3.2.1 Sprint nr 1 (Kravelicitering) ... 10

3.2.2 Sprint nr 2 (Litteraturstudier inklusive övning) ... 10

3.2.3 Sprint nr 3 (Utforskning av API:er för Bluetooth-kommunikation) ... 10

3.2.4 Sprint nr 4 (Grafisk design av klockappen samt implementation av bakgrundslogiken) ... 11

3.2.5 Sprint nr 5 (Implementation av Bluetooth-avlysning för upptäckt av fordon) ... 11

3.2.6 Sprint nr 6 (Utveckling av temporär mobilapp för verifiering av klockkompatibilitet) ... 12

3.3 Källkritik ... 12

4. **Analys** ... 14

4.1 Sprint nr 1 ... 14

4.2 Sprint nr 2 ... 14

4.3 Sprint nr 3 ... 14

4.4 Sprint nr 4 ... 17

4.5 Sprint nr 5 ... 18

4.6 Sprint nr 6 19
5. Resultat 22
5.1 Klockappen 24
5.2 Mobilappen 27
6. Slutsatser 30
6.1 Utförandet av examensarbetet 31
6.2 Framtida utvecklingsmöjligheter 32
6.2.1 Uppdatering av Mobills befintliga mobilapp 32
6.2.2 Batteritid 32
6.3 Etiska reflektioner 33
7. Referenser 34
8. Bilagor 36
Bilaga 1: Android Studio's hierarki med placering av XML- respektive Javafiler 36
Bilaga 2: Ganttschema Tidsplan 37
Bilaga 3: Kodexempel för att överföra data mellan enheter med Lågnivå Bluetooth-protokoll 38
Bilaga 4: Xamarin.Android hierarki med placering av XML- respektive CS-filer 41
Bilaga 5: UML-Diagram 42
Bilaga 6: Illustration av sprintarna 43

1. Introduktion

I detta avsnitt presenteras företaget Mobill och dess befintliga parkeringsapp i mobilen. Vidare beskrivs en ny klockapplikation, och hur dess funktionalitet skulle kunna förenkla den befintliga parkeringsproceduren när det används tillsammans med Mobills befintliga mobilapp. Dessutom presenteras målformuleringen av examensarbetet och de frågeställningar som kommer att besvaras.

1.1 Bakgrund

Mobills parkeringsapp i mobilen används i dagsläget av bilister för att betala parkeringsavgifter på parkeringsområden (även kallat parkeringszoner), där ägaren av parkeringsområdet har avtal med Mobill. Dock måste bilisten ta fram och använda mobilen för att kunna parkera sin bil då parkeringszonen måste anges, samt vilket fordon (registreringsnummer) som skall parkeras. Den befintliga parkeringsappen vet inte när bilisten är i närheten av en parkeringszon, och kan därmed inte föreslå en parkering per automatik. Mobilappen vet inte heller när bilisten lämnar en parkeringszon (p-zon) och kan därmed inte avsluta parkeringen automatiskt.

1.2 Intelligent Klockapplikation

Det skulle underlätta betydligt för bilisterna om den befintliga mobilapplikationen hade fått hjälp av en klockapplikation som detekterar när en bilist befinner sig nära en parkeringszon, samt föreslår att användaren då kan parkera där. När parkeringszonen lämnas föreslår klockapplikationen att parkeringen avslutas. Användaren av klockapplikationen skall kunna interagera med bara klockan utan att behöva ta upp mobilen (förutom när ny användare eller bil behöver registreras, eller byta parkeringszon/typ av fordon). Detta förenklar parkeringsproceduren och sparar tid för användaren.

1.3 Wear OS

I dag finns en stor mängd smarta klockor som kommer från många olika företag. Det kan handla om klockor som säljs under varumärken som normalt sett tillhör modeindustrin, den ursprungliga klockindustrin (tex Casio) eller hemelektronikbranschen (tex Samsung). Dessa har olika skärmstorlekar och hårdvara, likt mobiltelefoner och drivs av ett gemensamt operativsystem, vilket är Wear OS som är motsvarigheten till Android och tillhandahålls av Google. Undantaget är Apple Smartwatch, som använder Apples eget operativsystem, dock är nyare iOS-telefoner också kompatibla med Wear OS. Apparna för Wear OS utvecklas i plattformen Android Studio, som även används för utveckling av vanliga Android-appar. Dessutom kan det finnas en möjlighet att porta appar till Apple smartwatch operativsystem i efterhand, dock skulle det vara ett separat projekt i sig för framtida utvecklare (underlättar om programmeringsspråket är samma, så som Java).

1.4 Funktioner

Klockappen kommer att användas som ett komplement till den befintliga mobilappen. Den skall kommunicera med mobilappen, och detta kan möjligtvis ske via ett Bluetooth-protokoll. Klockappen skall kunna hämta information gällande en parkeringszons timme/dygnstaxa från mobilappen när en ny parkeringsperiod skall påbörjas. Dessutom måste klockappen få information från mobilappen gällande vilket användarkonto samt fordon som skall kopplas till den nya parkeringsperioden som skall påbörjas. Klockappen kommer då använda det befintliga användarkontot samt det fordon som är förvalt i mobilappen. Om användaren vill registrera ett nytt användarkonto, lägga till ett nytt fordon eller välja ett annat befintligt fordon som förstahandsval krävs det att mobilappen används. Detsamma gäller om användaren av någon anledning vill välja en annan parkeringzon som inte ligger på samma plats.
Klockappen kommer alltid vara uppdaterad kring vilket användarkonto samt fordon som används när den föreslår att en parkeringsperiod skall påbörjas. Vidare skall klockappen ha ett användargränssnitt, vilket tar bort behovet att använda mobiltelefonen för att utföra enkla val, såsom att påbörja eller avsluta en parkeringsperiod.
Klockappen skall via GPS-positionering veta när användaren kör in i en parkeringszon och meddela användaren med en notifikation, där klockappen föreslår att en parkeringsperiod kan påbörjas på denna parkeringszon. Vissa parkeringszoner kanske har information om det specifika antalet p-platser inom zonen, samt räknare för antalet fordon som står parkerade inom zonen. Om just den parkeringszonen är full kan klockappen meddela användaren det med en notifikation. Om användaren i stället vill välja en annan parkeringszon manuellt kan det göras i mobilappen. Det notifikationsfönster som kommer upp på klockskärmen, när det föreslås att en parkeringsperiod kan påbörjas, innehåller en start-knapp plus taxan för den specifika parkeringszonen. När användaren trycker på start-knappen påbörjas parkeringsperioden. Om användaren inte vill parkera enligt vad som föreslås av klockappen kan notifikationsfönstret stängas genom att trycka på avbryt-knappen.

För att klockappen skall kunna veta när användaren kör in i en parkeringszon behöver den ha tillåtelse att nyttja mobiltelefonens inbyggda GPS-positionering. Givetvis kan detta nekas av användaren om man bara vill använda mobilapplikationen. När parkeringszonen lämnas kan klockan föreslå att parkeringsperioden avslutas genom en notifiering. Det notifikationsfönster som skulle visas upp i sådana fall innehåller en stopp-knapp, eller en avbryt-knapp om användaren vill avsluta parkeringsperioden senare manuellt.

1.5 Syfte

Syftet med examensarbetet är att förenkla parkeringsproceduren för bilister från start till slut, samt att göra denna procedur snabbare, utan krav på att användaren måste använda den befintliga mobilappen för att parkera respektive avsluta en parkeringsperiod. Givetvis kommer den befintliga mobilappen fortfarande att kunna användas separat med samma funktionalitet som innan. Skillnaden blir att klockappen kommer anropa befintlig funktionalitet i mobilappen för att kunna påbörja respektive avsluta en parkeringsperiod. Detta kommer ske i bakgrunden och alltså inget användaren kommer märka i mobilappens användargränssnitt, utöver att en pågående parkeringsperiod kan visas på mobilappen, oavsett om parkeringsperioden påbörjades genom klockappen eller genom mobilappens användargränssnitt.

1.6 Målformulering

Examensarbetet går ut på att utveckla och testa en prototyp av en klockapplikation som kan bidra till att automatisera parkeringsproceduren där användaren i princip bara behöver trycka en gång på sin smarta klocka för att starta och avsluta sin parkering. Klockan skall genom att ta reda på användarens GPS-koordinater visa den rätta parkeringszonen med dygnstaxa och annan information kopplat till just den zonen, innan användaren trycker på skärmen för att starta parkeringen. Användaren skall inte behöva ta fram mobiltelefonen för att påbörja eller avsluta parkering.

1.7 Problemformulering

De frågeställningar som kommer att besvaras i detta examensarbete är följande:

1. Hur skall den nya klockapplikationen kommunicera med mobilappen?
2. Hur skall klockappen notifiera användaren och föreslå att starta parkering?
3. Hur skall parkering avslutas?
4. Hur skall klockapplikationens användargränssnitt vara utformat med tanke på klockskärmens ringa storlek?
5. Hur vet klockapplikationen att användaren sitter i en bil som rör sig och stannar i parkeringszonen?
6. Hur vet klockapplikationen att användaren sätter sig i bilen och kör ut ur parkeringszonen?

1.8 Motivering av examensarbetet

Kunskapen som kommer förvärvas inom detta examensarbete ger konkurrenskraftiga fördelar i en värld där företag letar efter individer med bred kompetensprofil. Företagen letar efter personer att anställa som följer med den snabba förändringstakten och som inte är rädda att lära sig ny teknik. Marknaden för smarta klockor är relativt ung och outforskad, samt har framtidspotential med tanke på armbandsurets långa historia. För Mobills del får de ett försprång gentemot konkurrenterna då ingen av dessa har en klockapplikation i dagsläget.
För samhället i övrigt kommer klockapplikationen att förenkla och göra hela proceduren från påbörjad till avslutad parkeringsperiod snabbare. Därmed kan alla användare spara tid, som i stället kan användas för annat i vardagen.

1.9 Avgränsningar

Denna klockapp kommer att utvecklas för Googles eget operativsystem Wear OS, vilket används av en stor grupp av smarta klockor med olika skärmstorlekar och hårvaruförutsättnignar, och är i princip motsvarigheten till Android för mobiltelefoner. Apple Smartwatches har ett eget operativsystem och kommer inte stödjas i detta skede.

2. Teknisk Bakgrund

I detta avsnitt beskrivs hur operativsystemet Wear OS förhåller sig till Android OS, samt skillnaden mellan utvecklingsverktygen Android Studio samt Eclipse. Även Xamarins utvecklingsmiljö tas upp kortfattat samt hur Mobills befintliga mobilapplikation ser ut.

2.1 Wear OS versus Android OS

En tydlig skillnad mellan en smart klocka och en mobiltelefon är skärmstorleken. En logisk slutsats är att navigationen bland applikationer i mobiltelefonen inte kan ske på samma sätt i den smarta klockan. Vad som krävs är en anpassning av användargränssnittet på den smarta klockans operativsystem, Wear OS, till den betydligt mindre skärmen. Figur 1 visar exempelvis hur en huvudmeny kan se ut på Wear OS. Till sitt förfogande har användaren även, beroende på klockmodell, tillgång till sidknappar som kan agera som antingen bakåt- respektive hemknapp. Hänsyn måste även tas till det mindre batteriet i den smarta klockan, och därmed brukar Wear OS-applikationer ha en mörkare bakgrund jämfört med vanliga Android-applikationer [12].

Wear OS är en utvidgning av operativsystemet Android, vilket innebär det att befintliga bibliotek och API:s som finns i Android OS också är tillgängligt i Wear OS. Med andra ord kommer samma bibliotek att fungera både i en telefon- eller klockapp. En Android-applikation, och därmed även en en Wear OS applikation, implementeras genom att man har ett eller flera packet av Java-klasser inne i en traditionell workspace-katalog. Vidare finns bibliotek tillgängliga i form av .jar-filer som kan utnyttjas av klasserna.

Användargränssnittet i Wear OS-appar definieras av XML-filer. Dessa definierar alltså vilken grafisk design klockappen skall ha. Detta realiseras genom statisk kod, där till exempel färgerna i appen definieras med traditionsenlig HEX-kod. Dock kan användargränssnittet göras dynamiskt, till exempel att ett resultat som har beräknats i klockappen sedan dyker upp på klockskärmen. Detta sker efter att en anropad funktion i en Java-klass har slutexekverats. För att användarvyn ska kunna uppdateras på detta sätt efter ett funktionsanrop, behöver Java-klasserna innehålla variabler som pekar på de berörda grafiska attributen i XML-filerna. Då kan ett funktionsanrop i en Java-fil be det berörda grafiska gränssnittet att ändra sitt utseende baserat på vad som skall visas för användaren. Detta genomsyras i hela applikationen och är i princip hur det går till när bakgrundsoperationerna behöver uppdatera den grafiska vyn för att förmedla informationen till användarna. Då kan det verifieras om klock-appen uppfyller den önskade funktionaliteten. De Java-klasser som implementerar användarvyn kallas i Android-termer för Activity-klasser.

Figur 1: Visar hur en huvudmeny kan se ut på Wear OS. Notera sidknapparna som kan agera som antingen bakåt- eller hemknapp.
Bildkälla: www.suunto.com

2.2 Utvecklingsverktyg

I detta avsnitt beskrivs vilka utvecklingsverktyg som används för att kunna implementera klockappen respektive lägga in funktionalitet i mobilappen. Android Studio används för att utveckla klockappen, som är skriven i Java. Den befintliga mobilappen är skriven i C# med verktyget Xamarin.Android.

2.2.1 Android Studio versus Eclipse: Likheter och skillnader

I princip är det samma struktur/hierarki på Android Studio som på utvecklingsplattformen Eclipse, med ett workspace som innehåller Java-filer plus Java-biblioteken i .jar-filerna. Dock som det nämndes i föregående avsnitt innehåller applikationer utvecklade för Wear – respektive Android OS även XML-filer som representerar applikationens grafiska attribut [19]. Dessa filer finns i en katalog som heter ”resources”. Vidare är filerna fördelade i flera lager inne i denna katalog. I katalogen ”values” finns XML-filer som innehåller strängar, där varje fil kan representera strängar på ett specifikt språk. Vidare i katalogen ”layout” finns XML-filerna som representerar applikationens grafiska huvud- respektive undersidor som visas på klockskärmen, beroende på vilka funktioner som exekveras på applikationens huvud tråd.

Bilaga 1 illustrerar hur hierarkin ser ut i Android Studio.

2.2.2 Xamarin.Android

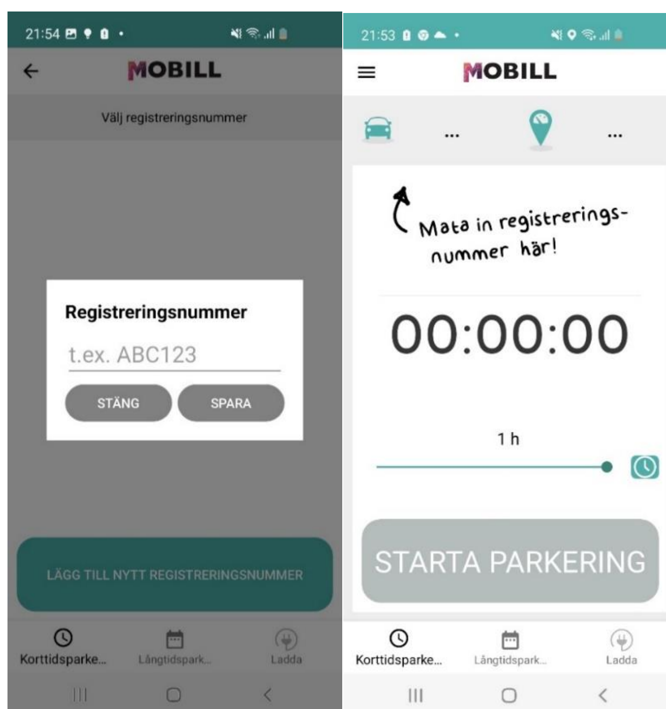
Xamarin.Android är den kompletta är utvecklingsplattformen för utvecklingsspråket C#, vilket är Microsofts egna objektorienterade programmeringsspråk [11]. I folkmun benämns C# som Microsofts motsvarighet till Oracles Java. Båda två är objektorienterade språk och härstammar från C/C++. De har båda garbage-collection, vilket underlättar för programmeraren. Syntaxen är princip likadan som i Java.

Den befintliga mobilappen är implementerat med just Xamarin-plattformen.
Även denna utvecklingsplattform använder sig av kataloger som innehåller XML-filer. Strukturen liknar den i Android Studio och beskriver användargränssnittet i form av statisk kod. I detta examensarbete kommer fokus ligga på Android Studio som utvecklingsplattform, dock med överblick över Xamarins utvecklingsmiljö, ifall funktionalitet behöver läggas till i Mobills befintliga mobil-applikation för att realisera klock-applikationen.

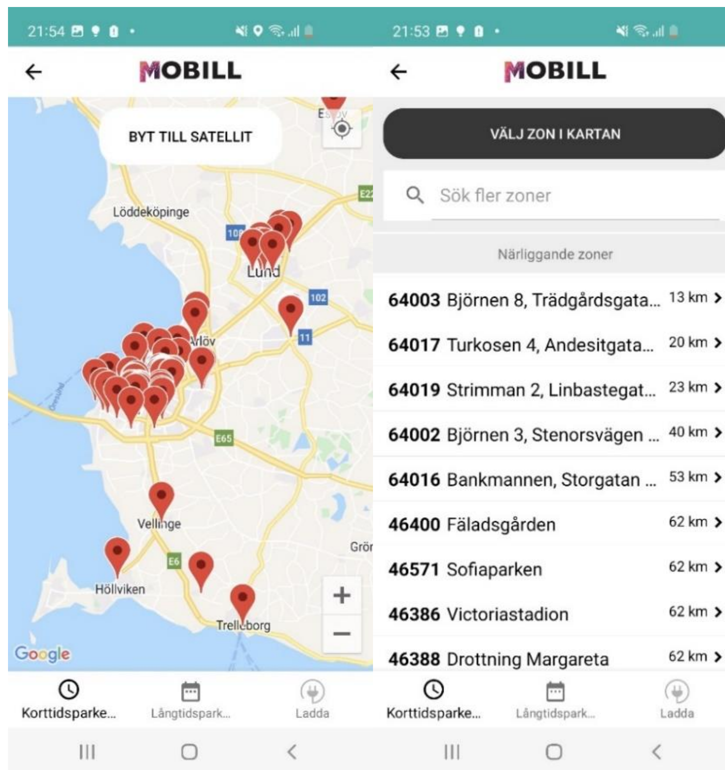
Under Bilaga 4 går det att se hur hierarkin ser ut i Xamarin.Android. Det kan jämföras med Bilaga 1 för Android Studio för att upptäcka likheterna.

2.3 Mobills befintliga mobilapplikation

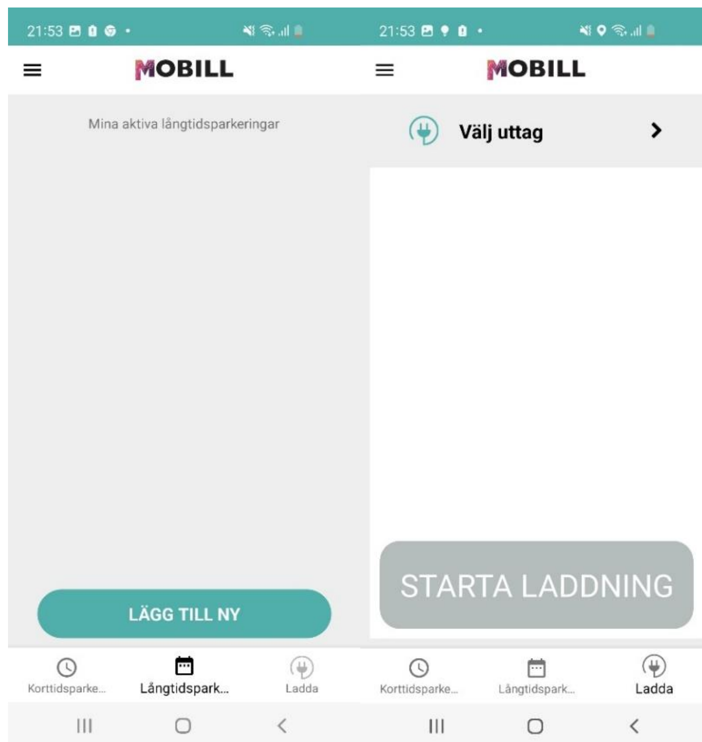
Mobill har i dagsläget en mobilapp som möjliggör parkering när användaren interagerar med den. Figur 2 visar startsidan i mobilappen. Här kan man välja antingen kort- eller långtidsparkering, samt välja eluttag eller ange registreringsnummer. Figur 3 visar hur man kan välja parkeringszoner. Det kan antingen vara från en lista eller i en kart vy. Figur 4 visar långtidsparkering samt eluttag.



Figur 2: Visar startsidan i mobilappen.



Figur 3: Visar hur man kan välja p-zoner.



Figur 4: Visar långtidsparkering samt eluttag.

2.4 Mobills klockapplikation på Wear OS

Vad det gäller den klockapp som skall utvecklas för Mobill kommer examensarbetaren att undersöka vilka API:er som kan användas för att få Mobills befintliga mobilapp att kommunicera med den nyutvecklade klockappen. Det handlar om att den nyutvecklade klockappen först skall kunna skicka en förfrågan till mobilappen via Bluetooth för att få reda på vilka perkeringszoner som kan finnas i närheten. Om mobilappen svarar tillbaka att det inte finns några parkeringsgszoner i närheten kommer klockappen att skicka samma förfrågan igen. Detta kommer upprepas tills klockappen tar emot ett svar från mobilappen att det finns parkeringszoner i närheten. Svaret levereras i from av en lista. Klockappen kommer därefter att jämföra klockans koordinater med koordinaterna i denna lista över parkeringszoner. Detta kommer att upprepas periodiskt. Om koordinaterna i listan matchar kommer klockappen undersöka om bilisten står stilla.

Om bilisten står stilla och koordinaterna matchar, samt att klockan är ansluten till bilens Bluetooth-system, kommer klockappen att fråga bilisten om hen vill parkera här. Bilisten kan antingen godkänna och påbörja parkering eller avböja. Om klockappens koordinaterna skiljer sig för mycket från koordinaterna i listan kan det vara ett tecken på att bilisten har lämnat området där parkeringszonerna finns. Då måste klockappen återgå till första steget och fråga igen om det finns parkeringszoner i närheten där bilisten befinner sig. Om bilisten påbörjar en parkering kommer det att skickas ett funktionsanrop till mobilappen att parkera på denna parkeringszon under en specifik tidsperiod. Därefter börjar klockappen räkna ner tiden och visar för bilisten hur lång tid hen har kvar på sin pågående parkeringssession. Om bilisten under den pågående parkeringsperioden sätter sig i bilen och kör i väg kommer klockappen att upptäcka detta när hastigheten beräknas, samt att klockan är ansluten till bilens Bluetooth-system. Då kommer klockappen fråga om bilisten vill avsluta parkeringensperioden i förtid. Bilisten kan återigen godkänna eller avböja. Om parkeringsperioden avslutas påbörjas hela proceduren igen och klockappen kommer att fråga mobilappen om det finns parkeringszoner i näheten.

3. Metod

I detta kapitel beskrivs projektmodellen som tillämpades under examensarbetets gång, samt vad som gjordes under varje utvecklingscykel (sprint). Totalt var det 6 sprintar. Bilaga 6 illustrerar sprintarna.

3.1 Scrum-liknande projektmodell

Examensarbetets tidsplan finns under Bilaga 2, i form av ett Gantschema. Examensarbetets upplägg av arbetet utgick ifrån en Scrum-liknande projektmodell [1]. Anledningen till att den Agila metodiken valdes över vattenfallsmetoden är att det är lättare att bygga vidare på produkten efter kontinuerlig feedback från kunden. Samtidigt är det även lättare att backa tillbaka och korrigera problem. Vad det gäller examensarbetsrapporten utvecklades även den under examensarbetet. På så sätt blir det lättare att felsöka, korrigera fel, samt även lägga till nya detaljer kring nya krav eller funktionaliteter, som kan tillkomma löpande när man har frekvent kontakt med kunden.

Scrum består av sprintar. Med andra ord kallas det för utvecklingscykler, och varje cykel pågår under en bestämd tid. I detta examensarbete varade en sprint i två veckor. Under dessa implementerades funktioner i klockappen som sedan visades för kunden (Mobill) när en sprint var slut. Vilka funktioner som implementerades under respektive sprint bestämdes efter samråd med Mobill inför varje sprintcykel. Det hölls möte med företaget varannan vecka, och det markerade slutet på en sprint. Under mötet visades den funktionalitet av klockappen som var implementerad i form av prototyper, och företaget återgav feedback kring huruvida produkten var på rätt spår, eller om funktionalitet behövdes läggas till eller tas bort. Vilka funktioner som implementerades under respektive sprint berodde på vilken prioritet företaget satte på dessa, och hur snabbt företaget behövde en prototyp att titta på.

Anledningen till att varje sprint skulle ta två veckor var framför allt att det skulle finnas gott om tid att utföra arbetet men samtidigt så skulle varje uppgift inte dra ut på tiden och riskera att förhala hela examensarbetet.

3.2 Sprintar

Nedan beskrivs vad som gjordes under varje sprint under examensarbetets gång.

3.2.1 Sprint nr 1 (Kravelicitering)

Den först sprinten bestod av arbetet att samla in information kring vilka krav som finns på hur klockappen skall se ut och fungera. Metoden som användes för kravelicitering var intervjuer (se avsnitt 4.1 för tillvägagångssätt) med företagets anställda. Back- samt FrontEnd-utvecklare intervjuades för att kunna få inputs kring vilken funktionalitet de olika användargränssnitten förväntades ha.
För att få en överblick, samt kunna spåra och validera kraven i efterhand, framställdes det ett kravspecifikationsdokument.

3.2.2 Sprint nr 2 (Litteraturstudier inklusive övning)

Under den andra sprinten genomfördes den första uppgiften, vilket var att lära sig hur Wear OS-applikationer för smarta klockor utvecklas. Syftet var att bekanta sig med Android Studios som utvecklingsplattform. Detta gjordes genom att utveckla en enkel applikation för den smarta klockan med ett enkelt användargränssnitt (se Figur 5). Målet i detta skede var att kunna förstå hur en användare kan interagera med klockan genom att trycka på virtuella knappar på skärmen, och på så sätt få klockan att byta tillstånd efter ett knapptryck. Först befinner sig klockan i ett tillstånd där den väntar på att användaren skall förse den med information i form av inmatningstext. Sedan kan klockappen använda denna text som inparameter för att beräkna ett resultat. Resultatet visas på klockskärmen för användaren. I detta skede valdes det att implementera en enkel valutakonverterare för att se dynamiken mellan inmatade uppgifter och visning av resultat.

3.2.3 Sprint nr 3 (Utforskning av API:er för Bluetooth-kommunikation)

De tredje spriten bestod av att börja implementera klockappen enligt givna krav. Det undersöktes vilka API:er som kan användas för att skicka meddelanden från klockappen till mobilappen och vice versa med Bluetooth som underliggande protokoll. Vidare gjordes det undersökningar hur klockappen periodvis skulle kunna utföra beräkningar i bakgrunden, för att sedan informera användaren om hen befinner sig i närheten av en parkeringszon som ingår i Mobills utbud, givet att fordonet som användaren kör står still samt att fordonets Bluetooth-system är ihopkopplat med klockan. Dessa bakgrundsberäkningar skall kunna utföras även om användaren exempelvis använder en annan applikation i klockan under tiden, eller att klockan inte används alls under denna tid. Med andra ord fick CPU:n inte övergå till viloläge. När användaren informeras om en parkeringszon i närheten presenteras detta i form av en dialogruta på klockskärmen.

Vidare lyckades klockappen skicka meddelanden till Mobills mobilapp. Syftet var att bekräfta i den tidiga prototypen huruvida det verkligen var möjligt att skicka meddelanden mellan klock- samt mobilappen via Bluetooth-gränssnittet. Meddelandena som skickades i detta skede var i princip bara en kort text, som sedan visades upp på mobiltelefonens skärm.

3.2.4 Sprint nr 4 (Grafisk design av klockappen samt implementation av bakgrundslogiken)

Den fjärde sprinten bestod av att anpassa användarvyn för att få klockappen att efterlikna den befintliga mobilappen när det gäller bakgrundsfärger samt företagets egen logo (Mobill). Därefter fortsatte implementeringen av givna krav, i detta fall hur bakgrundsoperationerna skall ske när klockappen har en lista till sitt förfogande över parkerings-zoner i närheten. Det skapades en s.k. mockuplista över parkerings-zoner i närheten. Det är ingen riktig lista som mobilappen har skickat, utan det är en lista som har genererats av klockappen. Därefter jämför klockappen sina egna koordinater med mockuplistan för att se om det finns en matchning mellan koordinater, eller om det bara är en liten avvikelse. Utöver det implementerades det även en metod som beräknar hastigheten för att se om fordonet som används av användaren står stilla eller ej.

Figur 5: Visar användarvyn av den enkla klockapp vilket implementerades som första uppgift (valutakonverterare). Appen prövkördes i Android Studio’s egna emulator.

3.2.5 Sprint nr 5 (Implementation av Bluetooth-avlysning för upptäckt av fordon)

Under denna sprint implementerades en Bluetooth-avlyssnare i klockappen, vilket skall kunna detektera när användaren sätter sig i bilen, samt även kunna särskilja mellan Bluetooth-enheter, såsom ett bilsystem eller en smart mobiltelefon. Vidare implementerades det en separat mobilapp för att bekräfta att gränssnittet WearableListenerService samt API:et MessageClient kan användas på både klockan samt mobiltelefonen för att skicka meddelanden mellan dessa.

Det framställdes även ett UML- samt Flödesdiagram som finns bifogat som Bilaga 5 respektive 6. Online-programvaran app.creately.com användes för detta ändamål [20].

3.2.6 Sprint nr 6 (Utveckling av temporär mobilapp för verifiering av klockkompatibilitet)

I tidigare sprintar lyckades klockappen skicka meddelanden till den befintliga mobilappen, som sedan visades upp på mobilskärmen i from av en textsträng. Dock som det även nämndes kraschade sedan mobilappen för att den inte kunde initiera klassen som lyssnar efter meddelanden från klockappen. Värt att notera i detta fall är att mobilappen lyckades anropa en metod för att visa upp textsträngen, och denna metod befann sig inne i klassen som inte kunde initieras. Efter omfattande felsökning tillsammans med andra utvecklare på företaget drogs slutsatsen att allt var implementerat på ett korrekt sätt. De andra utvecklarna framförde också att Microsoft inte längre stöder Xamarin.Android plattformen i Visual Studio sedan 2020. Under de senaste två åren har bara små uppdateringar släppts av Microsoft för att åtgärda mindre buggar, men att detta kommer upphöra mycket snart. Examensarbetaren tillsammans med utvecklarna misstänkte att biblioteken i Xamarin.Android inte har uppdaterats och ej är kompatibla med de nya gränssnitten samt API:erna. För att bekräfta att så var fallet utvecklades en egen mobilapp i Android Studio med samma gränssnitt samt API:er för att skicka/ta emot meddelanden mellan klock- respektive mobilappen. Det visade sig att det fungerar felfritt och inga krascher förekommer. Med andra ord initieras klasserna som de skall. Detta bekräftar examensarbetarens samt de andra utvecklarnas misstankar om att Xamarin.Android plattformen är för utdaterad för att stödja de senaste API:erna, och detta beror helt och hållet på Microsoft.

3.3 Källkritik

Referens **nr 1** är en del av kurslitteraturen som användes under projektkursen MAMF40. Kompendiets författare har djupgående erfarenheter av Kanban samt Scrum metodiken, och har noga övervägt för- och nackdelar. Med tanke på författarens arbetserfarenhet och teoretiska kunskaper inom området är källan pålitlig. Referens **nr 2, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19 respektive 21** utgår ifrån den offentliga sidan för Android-utvecklare (developer.android.com) som tillhandahålls av Google. Sidorna uppdateras kontinuerligt med ny information från Google. Det kan handla om nya API:er eller objekttyper som ersätter de äldre versionerna. När källorna lades till i referenslistan (avsnitt 7) noterades även när respektive sida senast var uppdaterad. Med tanke på att Google har skapat både Wear- respektive Android OS klassificeras dessa källor som pålitliga. Referens **nr 3** utgår ifrån en hemsida som lär sina användare nya kreativa färdigheter, i form av kurser som antingen finns i film eller skriftligt format. Det har inte provats några kurser från denna utgivare under examensarbetes gång, men baserat på informationen på hemsidan är kurserna populära i många världsdelar. Vidare nämns artikelförfattare samt publiceringsdatum. Hemsidan är uppdaterad och har bra betyg från användare. Den klassificeras som pålitlig.

Referens **nr 11** är Microsofts officiella dokumentation för Xamarin.Androids utvecklingsmiljö. Microsoft upphörde med sitt stöd för denna utvecklingsmiljö 2020, och kommer ersätta den med .NET plattformen. Dock släpps det fortfarande minde buggfixar. Hemsidan som är angiven i referenslistan (avsnitt 7) är fortfarande uppdaterad då årtalet 2022 nämns längst ner. Eftersom Microsoft tillhandahåller dokumentationen på hemsidan, samt är skaparen bakom Xamarin och C#, är den klassificerad som pålitlig. Referens **nr 20** är ett grafiskt verktyg online som utvecklare kan använda för att ta fram UML- respektive Flödesdiagram.

Det erbjuds även mer funktionalitet där utvecklare kan skapa sekvensdiagram eller designa lågnivåprototyper av programvaror. Tjänsterna på hemsidan är dock inte gratis och efter en prova-på period i 5 dagar krävs ett abonnemang för att fortsätta använda verktyget. UML-diagrammet som skapades med verktyget under examensarbetes gång finns tillgängligt i rapporten som bilaga 5. Med tanke på att bilagan demonstrerar klassernas relationer med varandra i klockappen på ett lättöverskådligt sätt är hemsidan att betrakta som pålitlig.

4. Analys

Nedan beskrivs det vilka beslut som har tagits under respektive sprint, när det gäller vilka API:er samt klasser som skall användas för att realisera klockappens användargränssnitt och funktionalitet.

4.1 Sprint nr 1 (Kravelicitering)

Intervjuerna som genomfördes med företagets anställda hade till syfte att få fram den funktionalitet Mobill förväntar sig skall finnas i klockappen. Intervjuerna genomfördes på Mobills kontor i form av gruppmöten och var strukturerade. Totalt var det tre gruppmöten, där examensarbetaren ställde frågor kring vilka grafiska attribut samt vilken funktionalitet som företaget förväntade sig att klockappen skulle innehålla. De anställda bestod av en projektledare samt två systemutvecklare, vilka fick skriva ner på papper de krav som gäller klockappens utseende samt funktionalitet. Klockappen skulle efterlikna den befintliga mobilappen när det gäller bakgrundsfärgerna och företagets logo. Vidare skulle klockappen kommunicera med mobilappen via Bluetooth och inte med Mobills databas (databas oövver parkerings-zoner). Dessutom skulle mobilappens befintliga funktioner nyttjas för att påbörja respektive avsluta parkeringsperioder, när den väl tog emot meddelanden från klockappen. Dessa krav sammanställdes sedan i ett separat kravspecifikationsdokument. Detta dokument ger en bra överblick på vilka krav som finns, samt underlättar validering av krav i efterhand samt spårning av krav.

4.2 Sprint nr 2 (Litteraturstudier inklusive övning)

För att lära sig hur Wear OS-applikationer för smarta klockor utvecklas implementerades först en enkel applikation för den smarta klockan med ett enkelt användargränssnitt (se Figur 5). Som det har beskrivits tidigare implementeras användarvyer i Android med hjälp av XML-filer, där varje komponent (i Figur 5 är det Text-vy, Knapp-vy, samt List-vy) har sina egna attribut för att beskriva utseendet. Med utseende menas exempelvis storlek och färg. Man kan ha referenser till komponenterna i Java filerna, och därmed dynamiskt kunna uppdatera användarvyn baserat på vilka krav som finns på vad användaren skall se på skärmen när hen interagerar med applikationen. Komponenterna kan jämföras med JavaFX-biblioteket som användes på examensarbetarens utbildning.

4.3 Sprint nr 3 (Utforskning av API:er för Bluetooth-kommunikation)

Det finns ett API som heter **MessageClient** [5]. Detta API, som kan utväxla information mellan enheter via Bluetooth, fungerar bara om det är samma applikation. Det vill säga att den applikation som skickar ett meddelande är den enda som kan läsa det. För att det skall kunna betraktas som samma applikation krävs det att applikationen har samma **versionskod**, samma **paketnamn** där källkoden finns, samt certifikat som är signerade med samma **nyckel** [3]. Om dessa tre villkor är uppfyllda när flera enheter, därav exempelvis flera smarta klockor med Wear OS, kör denna applikation kan dessa enheter utbyta

information med varandra genom **MessageClient** API. Enheter som är involverade i informationsutbyte genom detta API kallas i detta sammanhang för noder. I början av examensarbetet resonerades det kring huruvida man i stället skulle implementera ett låg-nivå Bluetooth-protokoll som skickar en vektor av bytes. Under Bilaga 3 kan man se ett kodexempel på hur det kan implementeras [2]. Dock använder sig **MessageClient** API av Bluetooth som underliggande protokoll och skickar meddelanden i form av bytevektorer. Om det manuellt skulle implementeras ett lågnivå-Bluetooth protokoll, med trådar som kör utanför huvud tråden i användarvyn, finns det risk att batteritiden i klockan kraftig försämras om trådarna körs konstant samtidigt. Dessutom är **MessageClient** väl beprövat och det finns gott om praktiska exempel på Googles officiella sida hur klassen kan implementeras [4].

Om nu flera noder (smarta klockor) är anslutna till en mobiltelefon kan det behövas en unik identifierare för mobilen. Den smarta klockan med applikationen installerad behöver veta var meddelanden genom **MessageClient** skall skickas någonstans. **CapabilityClient**-klassen är ett bra verktyg för att sortera bort noder som inte erbjuder en viss funktionalitet. Man ger mobiltelefonen, som har applikationen installerad, ett specifikt funktionsnamn inga andra noder har. Det kan till exempel vara namnet ”Mobill_mobile”. I mobilapplikationen lägger man till en XML-fil i en befintlig katalog som heter ”values”. Inne i den filen sätter man det specifika namnet. Den smarta klockan kan sedan känna igen vilken enhet den skall skicka sina meddelanden till [4].

Visserligen krävs det ändå trådar som körs i bakgrunden för att klock- respektive mobilappen skall kunna lyssna efter meddelanden skickade från respektive berörd enhet, men det finns ett API som hanterar detta i bakgrunden, nämligen **WearableServiceListener** [10]. Denna klass är en utvidgning av komponenten **Service** [9]. Service körs i bakgrunden som en separat tråd, och när ett meddelande tas emot, vars innehåll uppfyller en villkorssats inne i Service, kommer en metod i komponenten att anropas [8]. Metoden som utförs är en bestämd operation. Det kan innebära att huvudaktiviteten, som är huvud tråden med användarvyn, anropas för att uppdatera användarvyn med nya data som erhölls i meddelandepaketet i **Service** [8].

Utöver att kunna avlyssna i bakgrunden måste klockappen även periodiskt skicka förfrågningar till mobilappen. Här handlar det om att veta vilka parkeringszoner som finns i närheten av användaren. För att återigen undvika trådar som körs konstant och drar på batteriet finns ett API som kallas för **WorkManager** [6]**.** Den tillhandahåller periodiska förfrågningar till mobilappen, vilket sker genom klassen **Worker** [7]. Denna klass har en metod som anropas till exempel var 15:e minut, och kan då genomföra operationer som innebär att MessageClient används för att skicka en förfrågan till mobilappen. Det kortaste intervallet för att anropa metoden i Worker är just satt till 15 minuter. Anledningen är att spara på batteritiden. Klockappen kommer att använda sig av samma intervall.

För att slutprodukten skall motsvara företagets förväntningar och att det levereras enligt plan, har klockappens samtliga attribut samt funktioner, men även företagets krav på utvecklingsprocessen samt leverans, sammanställts i en separat kravspecifikation.

API:et **WorkManager** fungerade väl på klockan, och den skickade en förfrågan till Mobills app på mobilen var 15:e minut. Dock för att inte behöva vänta upp till 15 minuter varje gång, och för att se om mobiltelefonen tog emot meddelanden från klockan via **MessageClient** API, ersattes WorkManager temporärt med en klass som implementerar **Service**. Syftet var att mer frekvent kunna skicka meddelanden till mobilappen och bekräfta att MessageClient API:et verkligen fungerar när både klock-samt mobilappen har samma paketnamn, signatur såväl som versionskod. Klassen som implementerar Service har dessutom en tråd i sig, då den även implementerar gränssnittet Runnable (vilket instantierar en tråd och anropar metoden void run()). Det är i tråden bakgrundsberäkningarna kan göras när användaren inte aktivt interagerar med klockappen. På så sätt kan en lämplig parkeringszon i närheten presenteras för användaren, även om exempelvis klockskärmen är av, eller att en annan klockapp körs under tiden. Denna presentation sker i form av en dialogruta, som är en implementation av en **Activity**-klass.

Klockappen lyckades som sagt att kommunicera med Mobills mobilapp via **MessageClient** API, vilket använder Bluetooth-gränssnittet i bakgrunden för att skicka meddelanden mellan klock- samt mobilappen. I detta skede bestod meddelanden bara av en kort sträng med innehållet ”Hello World”, i syfte att bekräfta API:ets funktionalitet. Strängen visades på mobilappen i form av en AlertDialog-ruta efter att meddelandet togs emot. För att kunna lyssna efter data från klockappen implemterades **WearableServiceListener** i mobilappen**. Dock** slutade mobilappen att exekvera efter dialogrutan. Då kom det ett java.lang.RutimeException med felmeddelandet att klassen som implementerar **WearableServiceListener** i mobilappen inte existerar (java.lang.ClassnotFoundsException), fast det så inte var fallet. Felsökning pågick i drygt en vecka för att hitta orsaken. Koden innehöll inga fel efter observation, men många bibliotek i den befintliga mobilappen har inte uppdaterats på ett par år, då inga utvecklare har varit sysselsatta med den.

För att lösa problemet gällande risken att CPU: n övergår till viloläge när bakgrundsberäkningar görs, infördes klassens PowerManagers s.k Wake Locks [13]. Wake Locks gör det möjligt för klockappen att styra värdenhetens strömtillstånd. För att förhindra viloläge anropas Wake Locks metod acquire();, och sedan anropas release(); hos samma klass, när CPU: n inte längre måste vara låst i ett vaket tillstånd.

Figur 6 samt Figur 7 visar hur prototypen såg ut som presenterades för Mobill i slutet av sprinten.

MOBILL

Västra Hamnen

Västra Hamnen

Figur 6 (t.v) Visar hur UI ser ut när användaren startar klockappen. Observera att klockskärmen är rund, fast här är bakgrundsfärgen vit.
Figur 7 (t.h) visar vad som syns på klockskärmen när en p-zon har hittats i närheten, samtidigt som bilen står stilla. P-zonen här är en Mockup.

4.4 Sprint nr 4 (Grafisk design av klockappen samt implementation av bakgrundslogiken)

Det skapades en SVG-fil för Mobills logo, vilket är en högupplöst grafisk vektor, och sattes in i klockappens XML-fil. SVG-filen beskriver hur klockappens bakgrund skall se ut. I detta fall är det huvudsidans bakgrund. Mockuplistan var en ArrayList <ListData> av objekttypen ListData, vilket är en egen implementerad klass med en p-zons koordinater samt namn som attribut. Syftet med ListData var att lätt extrahera en p-zons namn samt koordinaterna när det behövdes. Vidare implementerar ListData typen Parcelable, med anledningen att när klockans koordinater matchar somliga (eller samtliga) angivna p-zoner i mockuplistan skall de berörda p-zonerna visas upp i användarvyn i form av en lista. För att kunna skickas över till användarvyn behöver data paketeras enligt gränssnittet **Parcelable** [14]. Listan som visas för användaren nyttjar klassen **WearableRecyclerView** [15]. Denna klass gör det möjligt att anpassa användarvyn efter klockskärmens form samt storlek. I exempelvis runda klockskärmar kan listan som visas ha en böjd layout (se Figur 6), medan i vanliga kvadratiska klockskärmar visas listan på ett konventionellt sätt, dvs rak vertikal lista.

För att klockan skulle kunna hämta sina egna koordinater i klockappen, för att sedan jämföra de med befintliga p-zoner, användes den färdigimplementerade klassen **FusedLocationProviderClient** [16], vilket är en utvidgning av GoogleApi klassen. Med andra ord hanterar **FusedLocationProviderClient** anslutningen mellan klockappen och Google Play Services.

Figur 8 visar prototypen som visades upp i slutet av sprinten. Textrutan längst ner i figuren visar klockans befintliga GPS-koordinater. När klockappen har erhållet p-zonerna som finns i närheten av mobilappen, försöker klockappen hitta sitt befintliga läge, för att på sådant sätt bestämma om klockans koordinater matchar en p-zon. Om det villkoret uppfylls kommer

klockappen beräkna användarens hastighet i from av att efterfråga klockans koordinater igen, för att se om klockan har flyttat på sig.

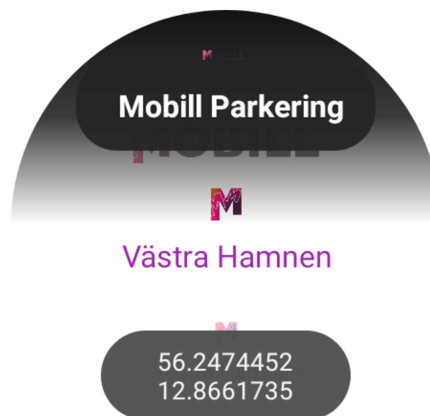
4.5 Sprint nr 5 (Implementation av Bluetooth-avlysning för upptäckt av fordon)

Den förimplementerade klassen **BluetoothAdapter** [17] användes för att få tillgång till klockans interna Bluetoothchip. Därefter kunde man genom klassens fördefinierade metod **.getBondedDevices**() hämta alla enheter klockan redan har parats ihop med. Denna metod returnerar ett objekt av typen **BluetoothDevice** [18]. Chansen är stor att den berörda bilen användaren kommer parkera med redan är ihop-parad med klockan, alternativt är det bara att para ihop klockan med bilen om antingen klockan eller bilen är nyinköpt. För att undvika att manuellt behöva para ihop klockan med bilen varje gång användaren sätter sig i bilen kan metoden **.connectgat**(); anropas. Vad som händer är att klockan kommer ansluta sig till enheter den redan har parats ihop med tidigare automatiskt, givet att dessa är i närheten. Naturligtvis kan det läggas till en if-sats om bara en enhet skall anslutas, i detta fall bilen. Då måste kockappen känna till bilens MAC-adress, vilket den får när bilen paras ihop med klockan första gången.

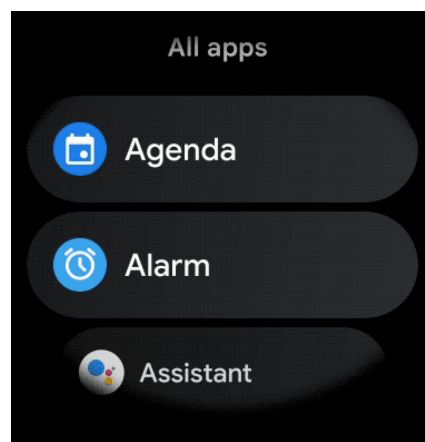
Vidare bekräftades funktionaliteten i **MessageClient API** samt i gränssnittet **WearableListenerService** när det gäller att skicka meddelanden mellan klockan samt mobilen. För detta ändamål implementerades en separat mobilapp som sedan skulle skicka samt ta emot meddelanden från klockappen. Samma metodik användes på klockappen för att skicka respektive ta emot meddelanden från mobilappen. Först skickar klockan en förfrågan till mobilappen för att få en lista över parkeringszoner. Men eftersom mobilappen ska lyssna efter tre olika kommandon från klockappen, måste de kunna urskiljas. Innan meddelandet (vilket är ett sträng-objekt) skickas från klockappen konverteras det till en bytevektor och transporteras sedan vidare över Bluetoothkanalen till mobilappen. Mobilappen tar emot bytevektorn och konverterar det tillbaka till en sträng. Lösningen här var att urskilja de tre olika kommandon från klockan genom att observera storleken på bytevektorn. Om det inkommer en bytevektor utan innehåll antar mobilappen att den skall skicka tillbaka en lista över parkeringszoner. Eftersom listan måste skickas som en sträng packades zonerna in i denna sträng enligt följande mönster: namn,koordinater-namn,koordinater. Med andra ord upprepar sig mönstret med namnet på en p-zon först, åtföljt av dess koordinater. P-zonerna separeras med hjälp av ett bindestreck (-). Om det däremot inkommer en vektor med innehåll antar mobilappen att den skall parkera användaren på en p-zon under en specifik tidsperiod.

När bytevektorn omvandlas tillbaka till en sträng finns informationen där vilken p-zon det rör sig om, samt hur lång tid det skall parkeras. Till exempel kan det se ut enligt följande: ”Stortorget,30”, vilket motsvarar p-zon Stortorget med en parkeringsperiod på 30 minuter. Både klock- samt mobilappen har fördefinierade metoder i klassen String, vilket kan utföra en split av an sträng med anropet .split();, med syftet att dela en sträng för att kunna extrahera data från den. Som exempel kan återigen strängen ”Stortorget,30” nämnas. Genom att anropa String.split(,) returneras en Sträng-vektor där plats 0 har innehållet ”Stortorget”, samt plats 1 har ”30”. Nu har en sträng blivit två separata strängar i en vektor, vilka kan nyttjas som olika attribut för att i efterhand kunna anropa dem var för sig. Sista kommandot klockappen skickar handlar om att stoppa parkeringen. En lösning kan vara att strängen som skickas bara innehåller ordet ”Stop”. Mobilappen kan då upptäcka

detta med hjälp av en if-sats, och kommer då anropa respektive metod för att stoppa parkeringen.



Figur 8: Visar hur klockappen börjar leta efter parkeringzoner när användaren startar appen.



Figur 9: Visar hur man kan skapa listor optimerade för klockskärmar. I detta fall är det en rund skärm.
Bildkälla: Create lists on Wear OS

4.6 Sprint nr 6 (Utveckling av temporär mobilapp för verifiering av klockkompatibilitet)

Som det beskrevs under i avsnitt 3.2.6 kraschade Mobills befintliga mobilapp när ett meddelande togs emot från mobilappen. Dock visades meddelandet upp i from av en textsträng på mobilens skärm innan kraschen ägde rum. Debug-terminalen visade att felet var av typen ClassNotFoundException. Detta var anmärkningsvärt med tanke på att dialogrutan av typen AlertDialog, som visar upp textsträngen på mobilskärmen, anropas först inne i klassen som inte kunde initieras (”ClassNotFoundException”). Efter omfattande felsökning tillsammans med utvecklarna på företaget kom vi fram att koden som hade utvecklats i examensarbetet var korrekt. Felet ClassNotFoundException är vanligast när ett klassnamn, vare sig det är C# eller Java, inte överensstämmer med motsvarande klassnamn

inne i Android-manifestet (en XML-fil). Klassen som inte kunde initieras är en utvidgning av gränssnittet WearableListenerService, som i sin tur utvidgar komponenten Service. Android-manifestets uppgift är att se till att samtliga komponenter (i detta examensarbete klasser som utvidgar Activity- eller Service komponenter) initieras när en app startas upp. Det kan vara antingen en klock- eller mobilapp. Det blev bekräftat att samtliga klassnamn i C# överensstämmer med motsvarande klassnamn i Android-manifestet. Vidare blev slutsatsen att Xamarin.Android med dess tillhörande bibliotek är utdaterat och ej längre kompatibelt med Androids senaste API:er. Microsoft upphörde med sitt stöd för Xamarin.Android för två år sedan. För att styrka denna slutsats implementerades en mobilapp i Android Studios med gränsnsittet **WearableListenerService** för att avlyssna meddelanden, samt **MessageClient** API för att skicka meddelanden tillbaka. Med andra ord används exakt samma gränssnitt samt API i den nya mobilappen som i den befintliga Mobills befintliga utdaterade app. Resultatet blev lyckat, och den nya mobilappen kunde visa upp meddelanden skickade från klockappen på sin skärm i form av textsträngar. Alla komponenter initierades och inga krascher förekom.

Vidare kunde den nya mobilappen behandla informationen som den erhöll, och skicka tillbaka svar till klockappen i form av exempelvis en lista över p-zoner i närheten, eller information om att en parkeringsperiod har påbörjats eller avslutats manuellt i mobilappen. I det fallet när klockan tar emot ett meddelande från mobilappen att en parkeringsperiod har påbörjats manuellt i telefonen, måste klockan sluta leta efter p-zoner och sätta en aktiv parkeringsperiod på motsvarande p-zon, med samma återståendetid som syns i mobilappen. Om klockappen tar emot ett meddelande från mobilen att parkeringsperioden har avslutats manuellt börjar klockan att leta efter p-zoner igen och följa samma automatiska procedur med att undersöka om användaren befinner sig i närheten av en p-zon, för att sedan beroende på om användaren står still och är ansluten till bilens Bluetooth-system, fråga om användaren vill påbörja en parkeringsperiod i denna p-zon.

I motsatt riktning fungerar det på ett likadant sätt, det vill säga att när klockan skickar ett meddelande till mobilen att påbörja en parkeringsperiod i berörd p-zon, sätts det en aktiv parkeringsperiod på motsvarande p-zon i mobilappen. Om så är fallet kan inte en ny parkeringsperiod startas på någon annan p-zon manuellt, eftersom det redan finns en aktiv parkeringsperiod. Om klockappen meddelar mobilen att avsluta befintlig parkeringsperiod i förtid (detta kan ske om användaren sätter sig i bilen och kör iväg från sin parkeringsplats, eftersom klockappen kan detektera om den är ansluten till bilens Bluetooth-system samt om användaren har en hastighet) avslutas den i mobilappen och en ny lista över p-zoner i närheten skickas till klockappen, som då kan börja jämföra användarens koordinater igen med p-zonerna i listan, för att sedan komma med förslag till användaren om samtliga villkor har uppfyllts (hastighet samt Bluetooth-anslutning).

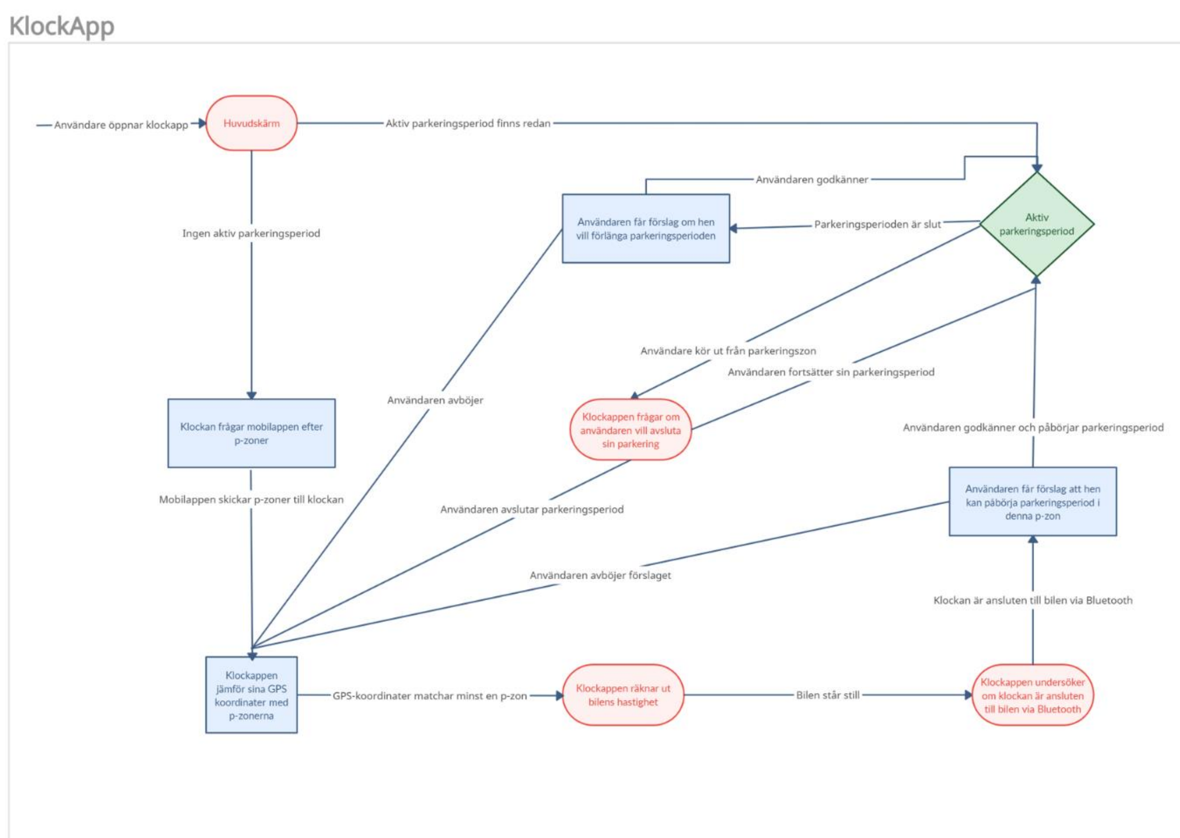
Utöver **WearableListenerService** samt **MessagClient** API har den nya mobilappen en klass som utvidgar komponenten Service. Den används för att just skicka en lista över p-zoner i närheten till klockappen. Detta skall kunna ske utan krav att användaren aktivt använder mobilappen, och därför är den implementerad som en Service (för att starta en bakgrundstråd). Vidare har den nya mobilappen olika användarvyer som kan visa både listan över p-zoner i närheten samt vilken p-zon som har en aktiv parkeringsperiod, med motsvarande återstående tid av parkeringsperioden, som räknas ner kontinuerligt i användarvyn. Om exampelvis användaren startar en ny parkeringsperiod manuellt från

telefonen, givet att det är tillåtet, skickas det ett meddelande till klockappen via MessageClient API från klassen som just implementerar användarvyn. Detsamma gäller om användaren stoppar parkeringsperioden manuellt i mobilen. Med andra ord förekommer MessaClient API och dess metoder i klasser av varierande typer, och med det kan slutsatsen dras att detta API är väldigt flexibelt när det gäller var det kan nyttjas. Dock finns det en begränsning, och det är att en apps huvudtråd får inte användas för att skicka meddelanden med MessageClient API, utan en separat tråd måste startas i bakgrunden för detta ändamål. Detta kan lätt relaiseras, och kan göras på likannde sätt som i klockappen, nämligen att låta berörde klasser i mobilappen implementera gränssnittet **Runnable** och dess metoder.

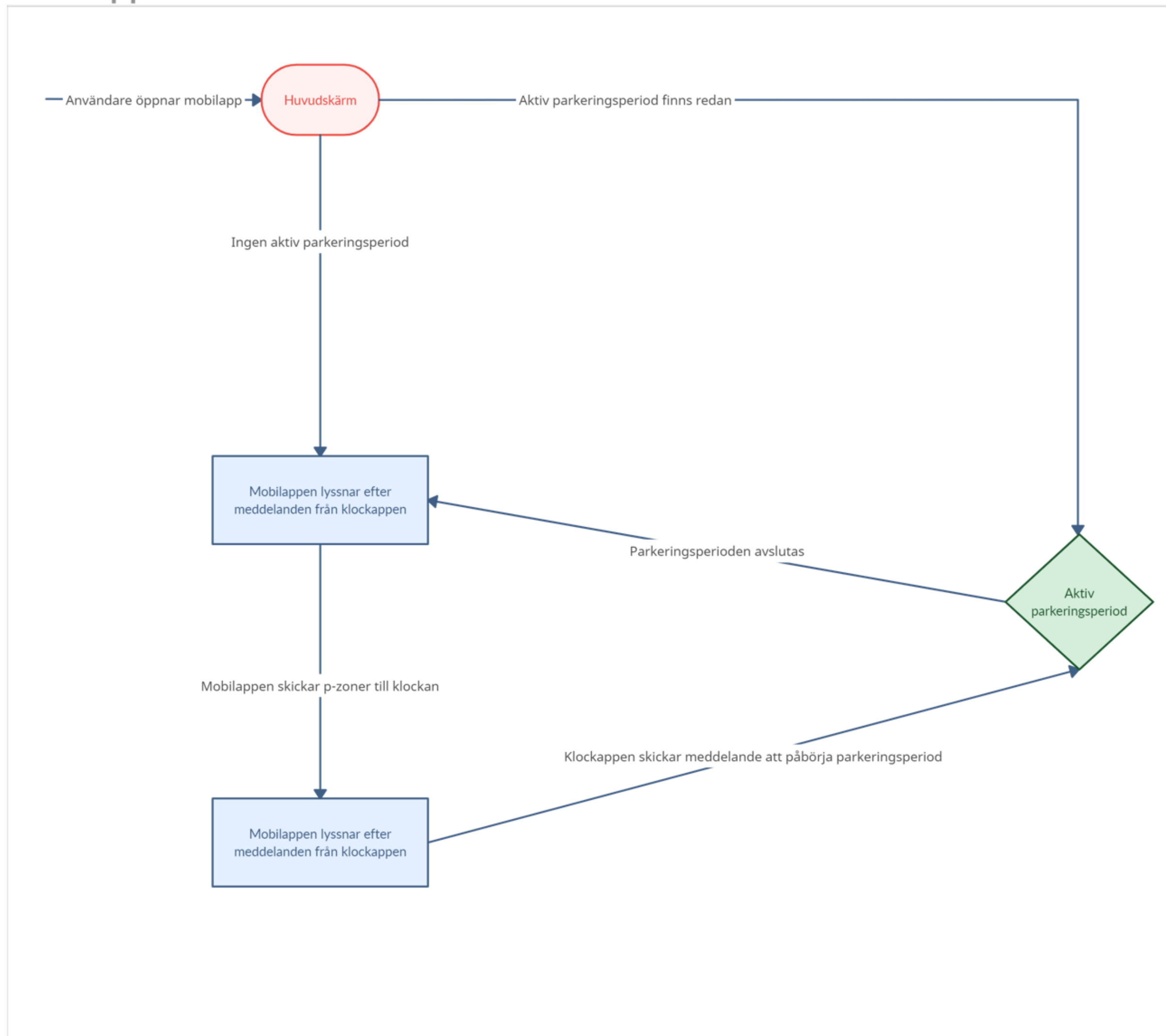
5. Resultat

I detta kapitel beskrivs den slutliga versionen av klockappen samt mobilappen. Detta i form av figurer som visar det grafiska gränssnittet, samt flödes- respektive UML-diagram i kombination med text för att beskriva funktionaliteten.

Se flödesdiagrammen Figur 10 respektive Figur 11 för att se i vilken ordning klock- samt mobilappen exekveras och när metoder anropas beroende på om villkoren har uppfyllts för att anropa respektive metod. Båda apparna ”binds samman” med hjälp av gränssnittet som lyssnar efter meddelanden. Detta gränssnitt illustreras på vänster sida i respektive flödesdiagram. I klockappen återges det som ”Klockan frågar mobilappen efter p-zoner” medan i mobilappen beskrivs det som ”Mobilappen lyssnar efter meddelanden från klockappen”.



Figur 10: Visar ett flödesdiagram av klockappsexekveringen och ger en bra överblick av i vilken ordning programmet exekveras och när respektive metoder anropas beroende på vilka villkor som har uppfyllts. Pilarna visar i vilken riktning exekveringen går.



Figur 11: Visar ett flödesdiagram av mobilappsexekveringen och ger en bra överblick av i vilken ordning programmet exekveras och när respektive metoder anropas beroende på vilka villkor som har uppfyllts. Pilarna visar i vilken riktning exekveringen går.

Bilaga 5 visar vilket **förhållande** klasserna i klockappen har med varandra. Detta i form av ett UML-diagram.

5.1 Klockappen

Figur 12 visar hur ikonen för klockappen ser ut på klockans hemskärm. Figur 13 visar hur användargränssnittet ser ut när klockappen startas. Användaren presenteras då med klockappens huvudskärm och klassen **MainActivity**.java, vilket visar aktiva parkeringsperioder med tjock färg. Alla parkeringsperioder som avslutas sparas i bakgrunden för att användaren skall kunna välja samma p-zon igen i framtiden om hen så önskar. De avslutade parkeringsperioderna dimmas ut i användargränsnsittet enligt Figur 13. Om inga befintliga parkeringsperioder finns kommer klockappen automatiskt att börja leta efter parkeringszoner i närheten av användaren. Då anropas klassen **LookingforParkingThread**.java, som startar en tråd för att hämta in parkeringszoner inom området användaren befinner sig i. Detta görs genom att skicka en förfrågan till mobilappen om vilka parkeringzoner som finns i närområdet. **MessageClient** API används för att kommunicera med mobilen och utväxla information i form av strängobjekt.

Inne i mobilappen lyssnar gränssnittet **WearableListenerService** efter meddelanden från klockappen. Den klass som implementerar WearableListenerService i mobilappen har **if-satser** i sig. Varje if-sats jämför meddelandets innehåll, som alltid är en sträng. Om nu strängmeddelandet som skickas från klockappen är tom, initieras en ny klass i mobilappen som heter **SendingActivity**.java. Denna klass hittar parkeringszoner i närområdet där användaren befinner sig. I detta fall handlar det om parkeringszoner som finns i den stadsdel användaren befinner sig i. Parkeringszonerna skickas sedan tillbaka till klockappen med hjälp av **MessageClient** API. Klockappen använder också **WearableListenerService** för att ta emot meddelanden från mobilappen. När klockappen har erhållit parkeringszonerna som finns i närheten fortsätter tråden med att var 5:e minut anropa metoden **getCurrentLocation**, som tillhandahålls av **fusedLocationClient** API, för att bestämma klockans position. Om klockans position matchar (eller avviker med mindre än en tusendel i latitud respektive longitud) en p-zon anropas funktionen **calculateSpeed**, vilket är en privat metod inne i **LookingforParkingThread**.java. Om klockans koordinater är samma när **getCurrentLocation** anropas igen antar klockan att användaren står still alldeles intill p-zonen. Förutsatt att klockan är ansluten till bilen via Bluetooth anropas klassen **DialogList**.java, som visar vilken p-zon som finns precis bredvid användarens position. Eftersom det ibland kan finnas flera p-zoner bredvid varandra presenteras en vy med en lista för användaren. Om det bara finns en p-zon bredvid användaren visas bara denna i listan. Om det däremot finns flera visas dessa i listan. Se Figur 14.

När användaren väljer en p-zon i listan anropas **DialogActivity**.java. Figur 15 visar hur klassen presenteras för användaren. Vyn innehåller p-zonens namn och taxa. Observera att innan användaren antingen godkänner eller nekar förslaget finns det en ScrollBar i samma vy, vilket låter användaren välja hur lång tid hen vill parkera. Om användaren nekar förslaget återgår vyn till den föregående listan. Om användaren därefter trycker på ”Avbryt” längst ned kommer klockappen återgå till det första steget och fråga mobilappen efter de närmaste p-zonerna. Om användaren däremot godkänner förslaget samt anger hår lång tid hen vill parkera (kortaste tiden är 30 min och den längsta 2 timmar) skickas det ett meddelande till mobilappen genom samma **MessageClient** API. **WearableListener** i mobilappen registrerar strängmeddelandet, och om det innehåller ordet ”start” startas det en timer av objekttypen **CountDownTimer** i mobilappen, som räknar ner återstående parkeringstid. Mobilappen uppdaterar sedan sin användarvy med parkeringszonens namn samt den återstående

parkeringstiden. När en parkeringsperiod är påbörjad kommer den visas längst upp i vyn i **MainActivity**.java. Se Figur 13. I klockappen avslutas därefter den befintliga tråden och en ny tråd skapas när klassen **ExitParkingThread**.java anropas. Den skall räkna ner hur lång tid användaren har kvar på sin parkeringsperiod, men även detektera om användaren sätter sig i bilen och kör ut från p-zonen. Metoden **calculateSpeed** använd även i denna tråd för hastighetsberäkning, likt vad den gjorde i föregående tråd. Dock i stället för att reagera när användaren står still kommer den reagera när användaren kör ut ur p-zonen och är ansluten till bilen via Bluetooth. Då anropar klockappen metoden **StopParking**.java. Figur 16 visar hur klassen presenteras för användaren. Det är i princip bara en förfrågan om användaren vill avsluta sin parkering eller fortsätta sin parkeringsperiod. Om användaren godkänner skickas ett meddelande till mobilappen. Mobilappens klass **WearableListenerService** registrerar detta genom att bytevektorn den får in innehåller ordet ”stop”. Då sätts värdet **sant** i en boolesk variabel som finns i samma klass för att avsluta parkeringsperioden. Timern som räknar ner återstående parkeringstid i mobilappen har en **if-sats** som körs om värdet i den booleska variabeln är satt till sant. Då avbryts timern i förtid och mobilappen återgår till ursprungsläget, vilket är att skicka en lista över p-zoner i närheten till klockan, eller att starta en ny parkeringsperiod om det efterfrågas av klockappen. Klockappen återgår till det första steget och börjar återigen leta efter p-zoner i närheten.

I slutprodukten används inte klassen Worker eller WorkManager. I stället används Service som det beskrevs i sprint nr 3. Anledningen var som sagt att bekräfta att klockappen verkligen kollar användarens koordinater periodiskt och jämför dessa med eventuella närliggande parkeringszoner. Eftersom lösningen fungerar så pass väl sitter den kvar permanent. I framtiden finns det alltid möjlighet att ändra detta åter till Worker och WorkManager, om målet är att förlänga klockans batteritid, eftersom Worker bara kör sin tråd minst var 15:e minut.

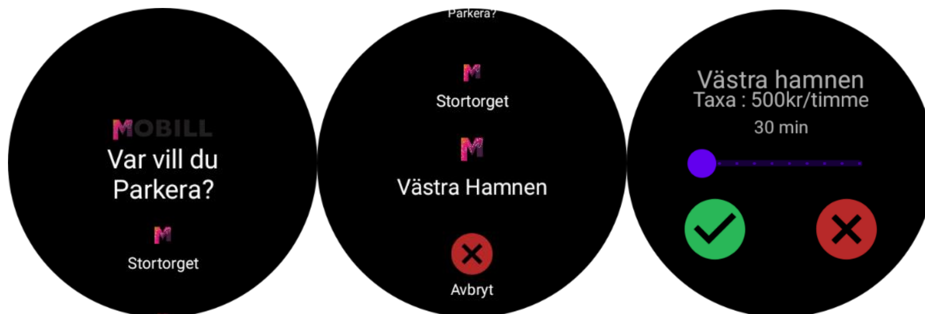
Observera för att mobilappen respektive klockappen skall kunna urskilja vilka meddelanden som skickas mellan enheterna, samt anropa rätt funktioner därefter, används in-parametern inne i **WearableListenerService**. Närmare bestämt anropas metoden **getPath** som tillhör inparametern i WerableListenerService’s metod **onMessageReceived**. Strängen som utvinns ur **getPath** jämförs sedan med respektive förprogrammerade sträng inne i **onMessageReceived**. Det kan vara ”stop”, ”start” eller kan det vara tomt. Om strängen är tom tolka mobil- samt klockappen det som att det skall skickas respektive tas emot en lista över p-zoner i närheten av användaren.

MOBILL

Västra Hamnen

Västra Hamnen

Figur 12 (t.v): Visar hur klockappens ikon ser ut på klockans hemskärm innan appen startas. Figur 13 (t.h): Visar hur klockappen ser ut när den startas.



Figur 14 (t.v och mitten): Visar listan som visas för användaren. Figur 15 (t.h): Visar den vy som visas när användaren väljer en parkeringszon från listan.

Figur 16: Visar vyn som visas för användaren om denna kör ut från p-zonen där bil är parkerad.

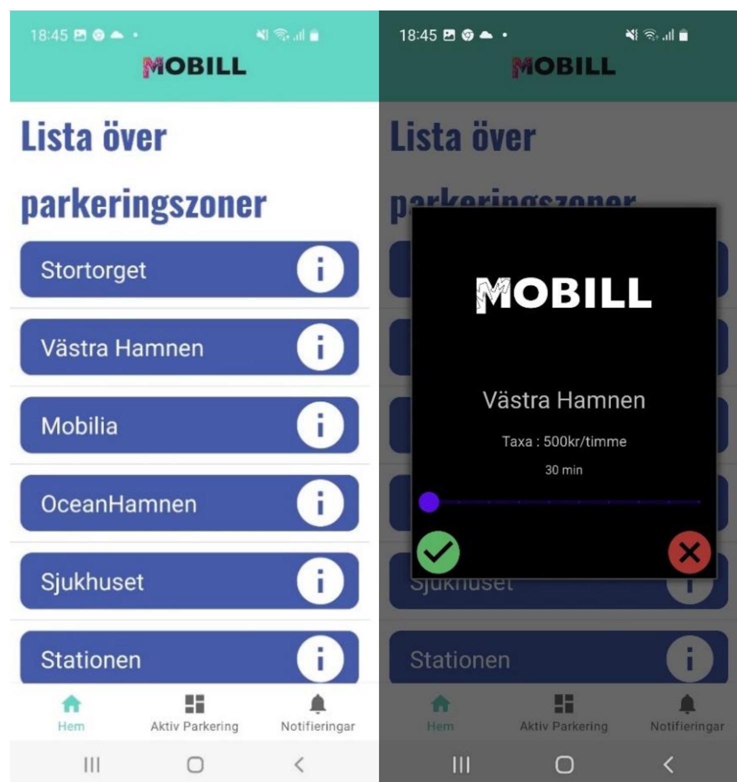
5.2 Mobilappen

Figur 17 visar hur hemskärmen ser ut på när mobilappen startas. Activity-komponenten (som är huvudklassen) **MainActivity**.java anropas Användargränssnittet är uppdelat i olika fönster som användaren kan navigera mellan med hjälp av navigationsfältet som finns längst ner på skärmen i mobilappen. Dessa fönster är av typen **Fragment**, vilket tillhör huvudklassen MainActivity.java. Vidare placeras alla Fragment som tillhör en Activity-komponent i en stack, och kan plockas fram och visas för användaren när det krävs. Fragment tar lika mycket utrymme på skärmen som Activity-komponenten som den tillhör. Den första Fragment som visas när mobilappen startas är den som innehåller listan över p-zoner i närheten, och heter **HomeFragment.**java. Först finns det inget att visa, utan användaren måste trycka på skärmen för att uppdatera vyn och få fram en lista. Vad som händer i bakgrunden är att det finns en lista över p-zoner med tillhörande koordinater, och dessa jämförs med mobilappens GPS-koordinater för att avgöra vilka som är i närheten. De som är i närheten visas upp på listan i användargränssnittet. Det efterliknar klockappens logik när den frågar mobilappen efter en lista över närliggande p-zoner som klockappen vill få skickat till sig.

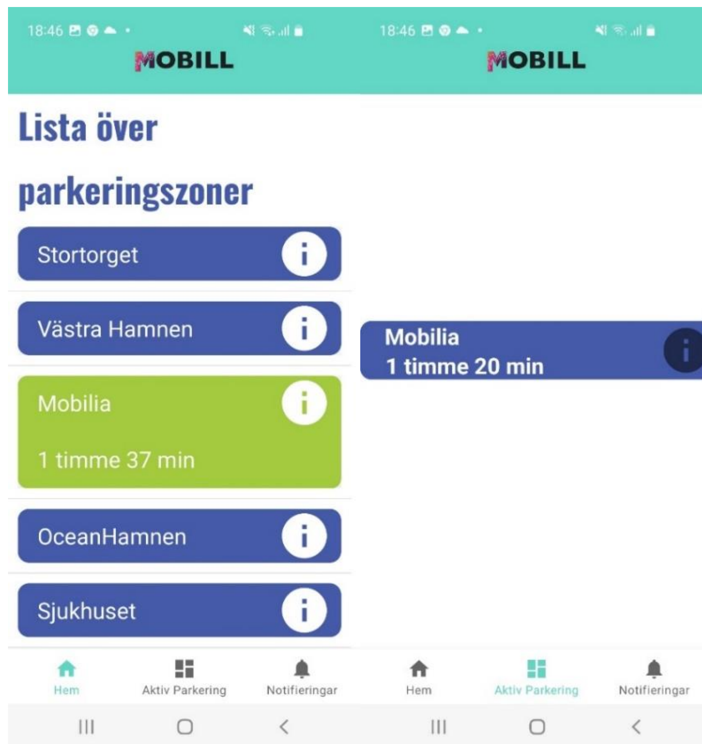
Om det inte finns en aktiv parkeringsperiod vare sig i klock- eller mobilappen kan användaren trycka på en valfri p-zon som finns på listan i användarvyn. Därefter anropas klassen **parkActivity**.java (se Figur 18). Där kan användaren välja hur lång tid hen vill parkera genom att använda slidern, för att sedan antingen trycka på ”OK” eller ”Avbryt”. Om knappen ”OK” trycks kommer mobilappen att påbörja en parkeringsperiod och skicka ett meddelande till klockan att starta samma parkeringsperiod på samma p-zon. Detta görs genom metoden **sendMessageToHandheld** (byte Array), som återfinns i MessageClient API:et. Dock måste det göras på en separat bakgrundstråd och inte huvudtråden. Lösningen är att parkingActivity.java även implementerar gränssnittet **Runnable**. När det är klart har användaren påbörjat en parkeringsperiod manuellt, och HomeFragment.java kommer uppdatera sin vy med att markera den p-zon som har en aktiv parkeringsperiod. Vidare kommer även återstående parkeringstid att synas precis under namnet på den berörda p-zonen (se Figur 19). När parkeringstiden går ut utan att användaren varken kör ut från parkeringsplatsen och får klockan att avsluta parkeringsperioden i förtid, eller avslutar den manuellt i mobilappen, avmarkeras den berörda p-zonen i användarvyn och återgår till samma läge som de andra p-zonerna i listan, det vill säga den har ingen aktiv parkeringsperiod.

Om användaren navigerar till fönstret som finns i mitten genom att trycka på navigationsfältet längst ner på mobilskärmen, dyker Fragmentet **DashboardFragment**.java upp i användarvyn. I vyns mitt återfinns en dialogruta som visar om det finns en aktiv parkeringsperiod. Om så är fallet uppdateras vyn med den berörda p-zonens namn samt återstående tid i parkeringsperioden, vilket även här räknas ner och uppdateras kontinuerligt på skärmen (se Figur 20). Om dialogrutan visar en aktiv parkeringsperiod är det möjligt för användaren att trycka på den. Då startas **stopActivity**.java och visas på skärmen. Användaren får då en förfråga om hen vill avsluta parkeringsperioden i förtid. Återigen finns en ”OK” samt ”Avbryt” knapp (se figur 21). Om parkeringsperioden avslutas skickas ett meddelande genom MessageClient API, inne i stopActivity.java, till klockappen, som då avslutar befintlig parkeringsperiod och skickar en förfrågan till mobilappen för att få en lista

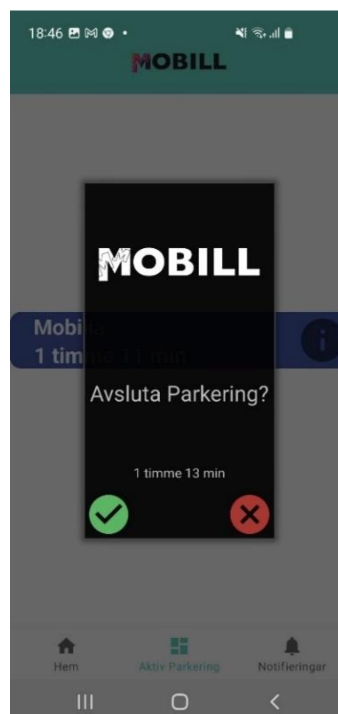
över närliggande p-zoner. Därefter kan klockappen återigen räkna ut vilken p-zon so är närmast användaren och avgöra om användaren står still samt är ansluten till bilens Bluetooth-system.



Figur 17 (t.v): Visar hur mobilappens hemskärm ser ut efter att appen startas. Figuren visar listan över närliggande p-zoner. Figur 18 (t.h): Visar fönstret som dyker upp om användaren väljer att trycka på en p-zon i listan i Figur 17.



Figur 19 (t.v): Visar en aktiv parkeringsperiod för en p-zon samt återstående parkeringstid. Figur 20 (t.h): Visar det andra fönstret om användaren trycker på ”Aktiv Parkering” i navigationsfältet längst ner i mobilappen.



Figur 21: Visar fönstret som dyker upp om användaren trycker på den blå rutan i Figur 20, förutsatt att det finns en aktiv parkeringsperiod. Användaren kan då välja att avsluta parkeringsperioden manuellt i förtid.

6. Slutsatser

Alla examensarbetets mål som finns i avsnitt 1.6 har blivit uppfyllda. Med hjälp av klockappen har parkeringsproceduren automatiserats. Klockappen avgör vilken parkeringszon som är i närheten av användaren, och om denna står still i sitt fordon samt är ansluten till fordonets Bluetooth-system. Om dessa villkor är uppfyllda visas en förfrågan på klocksärmen om användaren vill parkera på den berörda p-zonen. Dygnstaxan visas upp tillsammans med p-zonens namn. Allt som behöver göras då är att välja hur lång tid parkeringsperioden skall pågå, och sedan trycka på OK. Det motsvarar stycket i målformuleringen att användaren bara behöver trycka en gång för att påbörja parkeringsperioden. När användaren sedan sätter sig i bilen igen och kör ut från p-zonen detekterar klockan att den är ansluten till bilens Bluetooth-system, samt att fordonet har en hastighet. Då dyker det upp en förfrågan om användaren vill avsluta sin parkeringsperiod. Återigen behövs det bara ett knapptryck på OK för att avsluta perioden, vilket ännu en gång uppfyller målet att användaren bara behöver trycka en gång för att avsluta parkeringsperioden.

Frågeställningarna i avsnitt 1.7 har också besvarats:

1. Hur skall den nya klockapplikationen kommunicera med mobilappen?

   Klockappen kommunicerar med mobilappen via Bluetooth. Anledningen är att det bara är mobilappen som har tillgång till samtliga p-zoner i Mobills utbud, Vidare är det bara mobilappen som har tillgång till en användares profil, registreringsnummer samt betalningsuppgifter. Klockappens uppgift är att se till att välja den p-zon som kan vara aktuell för användaren att parkera på baserat på var hen befinner sig.

2. Hur skall klockappen notifiera användaren och föreslå att starta parkering?

   När klockappen hittar en p-zon i närheten och givna villkor är uppfyllda kommer det upp på klockaskärmen så att användaren kan se vilken p-zon det handlar om, vilken dygnstaxa det finns samt välja hur lång tid det skall parkeras, innan man trycker på OK.

3. Hur skall parkering avslutas?

   När klockappen detekterar att den är ansluten till bilens Bluetooth-system, samt att fordonet rör på sig, kommer det upp på klockskärmen en bekräftelse på om användaren vill avsluta sin parkeringsperiod. Återstående parkeringstid är inkluderat i vyn. Användaren kan sedan trycka på OK för att avsluta parkeringen.

4. Hur skall klockapplikationens användargränssnitt vara utformat med tanke på klockskärmens ringa storlek?

   När klockappen startas laddas det upp en hemskärm med Mobills logo inkluderad. Vidare laddas det upp en lista av p-zoner användaren har parkerat på innan, om man vill göra ett snabbval och parkera på samma p-zon igen. Oavsett om det finns en historik eller ej påbörjas den automatiska proceduren direkt med att leta efter p-zoner i närheten. Eftersom klockskärmen är så pass

minimal är det enda lämpliga användargränssnittet att ha antingen en lista man kan navigera för att välja objekt, eller en helskärm som visar ett par rader text för att beskriva p-zonen. Vidare kan denna helskärm innehålla en slider som användaren kan dra fram och tillbaka för att exempelvis sätta lämplig parkeringstid, plus två knappar längst ner för att gå vidare eller tillbaka. Idén som genomsyrar hela det grafiska användargränssnittet är att det skall vara minimalistiskt med tanke på enhetens ringa skärmstorlek, utan för mycket text, knappar eller andra attribut som gör det rörigt att navigera sig igenom. Dessutom är det bara helskärmar som gäller för vad som skall visas upp på klockskärmen, dialogrutor skall definitivt undvikas då det blir oläsligt. Men funktionalitet får inte offras med denna minimalistiska design. Det leder till att varje vy i klockappen måste utformas med omsorg, för att alla delar skall vara kompatibla med varandra och ge ett bra flöde, som tillsammans utgör en helhet.

5. Hur vet klockapplikationen att användaren sitter i en bil som rör sig och stannar i parkeringszonen?

   Klockappen kommer att veta om den är ansluten till bilens Bluetooth-system genom att undersöka vilka typer av enheter den är ansluten till. Klockappen kan skilja på ett bilsystem samt en smartphone genom att varje enhet har ett attribut som beskriver vad det är för typ av enhet. Därefter kan klockappen jämföra sina egna koordinater med en p-zons för att avgöra om användaren är i närheten. Som sista steg kontrolleras klockans GPS-koordinater ett par gånger under ett tidsintervall för att avgöra om användaren har en hastighet eller står still.

6. Hur vet klockapplikationen att användaren sätter sig i bilen och kör ut ur parkeringszonen?

   Klockappen kommer att veta om den är ansluten till bilens Bluetooth-system genom att undersöka vilka typer av enheter den är ansluten till. Klockappen kan skilja på ett bilsystem samt en smartphone genom att varje enhet har ett attribut som beskriver vad det är för typ av enhet. Därefter kontrolleras klockans GPS-koordinater ett par gånger under ett tidsintervall för att avgöra om användaren har en hastighet eller står still.

6.1 Utförandet av examensarbetet

Som det beskrivs i avsnitt 3.1 tillämpades en Scrum-liknande projektmodell under examensarbetes gång. Med tanke på att omständigheter förändras och att företag ibland måste göra omprioriteringar var denna metodik att föredra framför vattenfallsmodellen. Med vattenfallsmodellen får man ingen löpande feedback från företaget samt att det är lätt att missa krav som företaget lägger till medan examensarbetet fortlöper. Därmed blir det betydligt svårare att göra ändringar eftersom inga prototyper visas upp på förhand, utan slutprodukten levereras direkt. Risken är stor att man då får börja om från början om alltför många krav har ändrats, lagts till eller tagits bort under utvecklingsperioden.

Kommunikationen med Mobill's ledning fungerade bra och ledningen delade examensarbetarens syn på produkten, vilken funktionalitet som förväntas samt tidsplanen. Vad som var mindre bra var de långa svarstiderna från utvecklarna på företaget, ibland

kunde det ta flera dagar att få svar på en fråga gällande Mobills befintliga mobilapplikation. Vidare visste utvecklarna på företaget att det fanns problem med Xamarin.Android, vilket inte framgick av diskussionerna med dem.

Eftersom Mobills befintliga mobilapp var för utdaterad för att använda sig av de senaste API:erna och svarstiderna från utvecklarna ibland var flera veckor, blev lösningen att uveckla en egen mobilapp med likande funktionalitet, och som använder samma gränssnitt och API:er för att skicka meddelanden över Bluetooth. Därmed kunde examensarbetet fortlöpa då en viktig frågeställning hade besvarats, nämligen om klockappen kan kommunicera med mobilappen och på sådant sätt styra vad mobilappen skall göra.

6.2 Framtida utvecklingsmöjligheter

I detta avsnitt beskrivs vad som kan förbättras i framtida uppdateringar av både klockappen samt Mobills egna mobilapp. Det kan handla om batteritid på klockan såväl som migration av mobilappen till en ny utvecklingsmiljö.

6.2.1 Uppdatering av Mobills befintliga mobilapp

Som det nämndes i sprint 6 i både Metod- samt Analyskapitlet stöder Microsoft inte längre Xamarins.Forms, vilket inkluderar Xamarin.Android sedan två år tillbaka. Mobill är medveten om detta, men utvecklarna har valt att inte göra mycket åt saken i dagsläget, åtminstone för att göra de befintliga biblioteken i mobilappen kompatibla med de senaste API:erna (vilket inkluderar MessageClient samt WearableListenerService). Kanske det inte går att göra relativt mycket från utvecklarnas sida utöver att migrera hela existerande workspacet till en ny utvecklingsmiljö, och kanske byta programmeringsspråk. Dock finns det mycket potential i vad som kan åstadkommas när Mobill har en klockapp vid sidan om sin mobilapp, vilket har konstaterats med detta examensarbete. Användaren sparar tid när parkeringsproceduren automatiseras, och varje minut är extra viktig när det är tidsbrist.

6.2.2 Batteritid

Som det togs upp i resultatkapitlet används komponenten Service för att köra bakgrundsprocesser. Detta tar batteri då man dessutom manuellt måste låsa CPU-resurser för att inte klocka skall gå in i viloläge. Dock är det en nödvändighet för att klockappen skall kunna utföra bakgrundsoperationer utan att användaren aktivt interagerar med den. Ett alternativ till Service, och som rekommenderas av Google själva, är att använda typen Worker för att utföra periodiska operationer i bakgrunden. Dock är det minsta tidsintervallet 15 minuter, och operationer kan inte utföras oftare än var 15:e minut. Detta kan orsaka problem när till exempel klockappen vill avgöra om användaren står still i sin bil utanför en p-zon genom att jämföra klockans GPS-koordinater under ett tidsintervall. Om användaren står still på grund av ett trafikljus kommer det inte vara fortsatt rött i 15 minuter, utan kanske bara några sekunder. I sådana fall blir det för sent att efter 15 minuter inhämta klockans GPS-koordinater igen, eftersom användaren redan kan befinna sig långt ifrån GPS-koordinaterna som inhämtades tidigare. Därmed går en möjlighet förlorad att påbörja en parkeringsperiod eftersom fordonet redan kan vara långt ifrån p-zonen som var aktuell. Kanske det kommer

tillföras en lösning av Google i framtiden som liknar typen Worker, men som kan utföra bakgrundsoperationer mer frekvent utan att äventyra enhetens batteritid.

6.3 Etiska reflektioner

Klockapplikationen som har utvecklats bidrar till att förkorta tiden för användaren att starta respektive avsluta en parkeringsperiod. Tack vare klockapplikationen är det inte nödvändigt för användaren att ta upp mobiltelefonen och välja parkeringszon manuellt. Klockapplikationen hittar i stället den parkeringszon som ligger närmast användaren automatiskt genom att frekvent jämföra klockans GPS-koordinater för att se en matchning. När användaren står still i bilen bredvid en parkeringszon tar klockapplikationen tillfället i akt och frågar användaren om hen vill påbörja en parkeringsperiod på denna parkeringszon. När användaren har en aktiv parkeringsperiod och kör ut med bilen från parkeringszonen, tar klockapplikationen tillvara även på den möjligheten och frågar användaren om parkeringen skall avslutas. Med andra ord kan en parkeringsperiod påbörjas/avslutas med ett enda klick på klockskärmen, utan krav på att behöva navigera sig fram i flera steg för att hitta rätt parkeringszon i mobilapplikationen. I längden kan den totala tidsbesparingen bli betydande, speciellt när användaren frekvent är inne i städerna för att hantera sina ärenden. Dessutom blir det mer trafiksäkert när parkeringsproceduren kan skötas från klockskärmen i stället för att behöva ta upp mobiltelefonen. Detta eftersom bilisten kommer fortsätta att uppmärksamma trafiken i omgivningen. Även risken för att parkera på fel parkeringszon minskar betydligt eftersom klockapplikationen alltid utgår efter användarens nuvarande position för att avgöra vilken parkeringszon som ligger närmast. Dock är det fortfarande möjligt för användaren att påbörja/avsluta parkeringsperioder manuellt i mobilapplikationen om detta är att föredra.

Eftersom klockappen använder sig av lokalisering är det krav i den senaste Wear-OS versionen att fråga användaren om hen tillåter att appen använder GPS-positionering [21]. Detta sker när klockappen startas första gången. Om användaren inte ger sin tillåtelse kommer inte appen att kunna användas. Dock kan tillåtelse ges i ett senare skede under fliken Inställningar i klockan. Då kan klockappen användas. Syftet med kravet är att värna om personlig integritet och att inte hamna i konflikt med General Data Protection Regulation (EU) (GDPR).

7. Referenser

[1]. Kniberg, H., Skarin, M. (2010). Kanban and Scrum - making the most of both. *C4Media, Publisher of InfoQ.com.*

[2]. Google's Officiella sida för Android-utvecklare. Transfer Bluetooth Data.
Uppdaterat:2021-10-27 UTC Länk: Bluetooth Hämtat: 05/02-22

[3]. Thornsby, J. (27/04-18). Get Wear OS and Android Talking.
Uppdaterat:2018-04-27 Länk: Exchange Information Hämtat: 06/02-22

[4]. Google's Officiella sida för Android-utvecklare. Send & Receive Messages on Wear.
Uppdaterat:2021-10-27 UTC Länk: Send & Receive Hämtat: 22/02-22

[5]. Google's Officiella sida för Android-utvecklare. MessageClient API.
Uppdaterat:2021-04-27 UTC Länk: MessageClient Hämtat: 24/02-22

[6]. Google's Officiella sida för Android-utvecklare. Schedule Tasks with WorkManager.
Uppdaterat:2021-10-27 UTC Länk: WorkManager Hämtat: 01/03-22

[7]. Google's Officiella sida för Android-utvecklare. Abstract Class Worker.
Uppdaterat:2022-01-12 UTC Länk: Worker Hämtat: 03/03-22

[8]. Google's Officiella sida för Android-utvecklare. Handle Data Layer Events on Wear.
Uppdaterat:2021-10-27 UTC Länk: Data Layer Hämtat: 15/03-22

[9]. Google's Officiella sida för Android-utvecklare. Services Overview
Uppdaterat:2021-10-27 UTC Länk: Service Hämtat: 15/03-22

[10]. Google's Officiella sida för Android-utvecklare. WearableListenerService
Uppdaterat:2021-04-27 UTC Länk: WearableListener Hämtat: 15/03-22

[11]. Microsoft's Officiella sida för .NET-utvecklare. Xamarin.Android
Uppdaterat:2022 Länk: Xamarin Hämtat: 22/03-22

[12]. Google's Officiella sida för Android-utvecklare. Wear OS vs. Mobile Development
Uppdaterat:2021-05-19 UTC Länk: Wear vs. Mobile Hämtat: 22/03-22

[13]. Google's Officiella sida för Android-utvecklare. Keep the device awake
Uppdaterat:2022-05-11 UTC Länk: PowerManager Hämtat: 22/03-22

[14]. Google's Officiella sida för Android-utvecklare. Data packaging
Uppdaterat:2022-05-11 UTC Länk: Parcelable Hämtat: 01/04-22

[15]. Google's Officiella sida för Android-utvecklare. ListView
Uppdaterat:2022-05-11 UTC Länk: RecyclerView Hämtat: 01/04-22

[16]. Google's Officiella sida för Android-utvecklare. Manage Device Location
Uppdaterat:2021-02-18 UTC Länk: GoogleApi Hämtat: 10/04-22

[17]. Google's Officiella sida för Android-utvecklare. BluetoothAdapter
Uppdaterat:2022-05-11 UTC Länk: BluetoothAdapter Hämtat: 10/04-22

[18]. Google's Officiella sida för Android-utvecklare. BluetoothDevice
Uppdaterat:2022-03-17 UTC Länk: BluetoothDevice Hämtat: 10/04-22

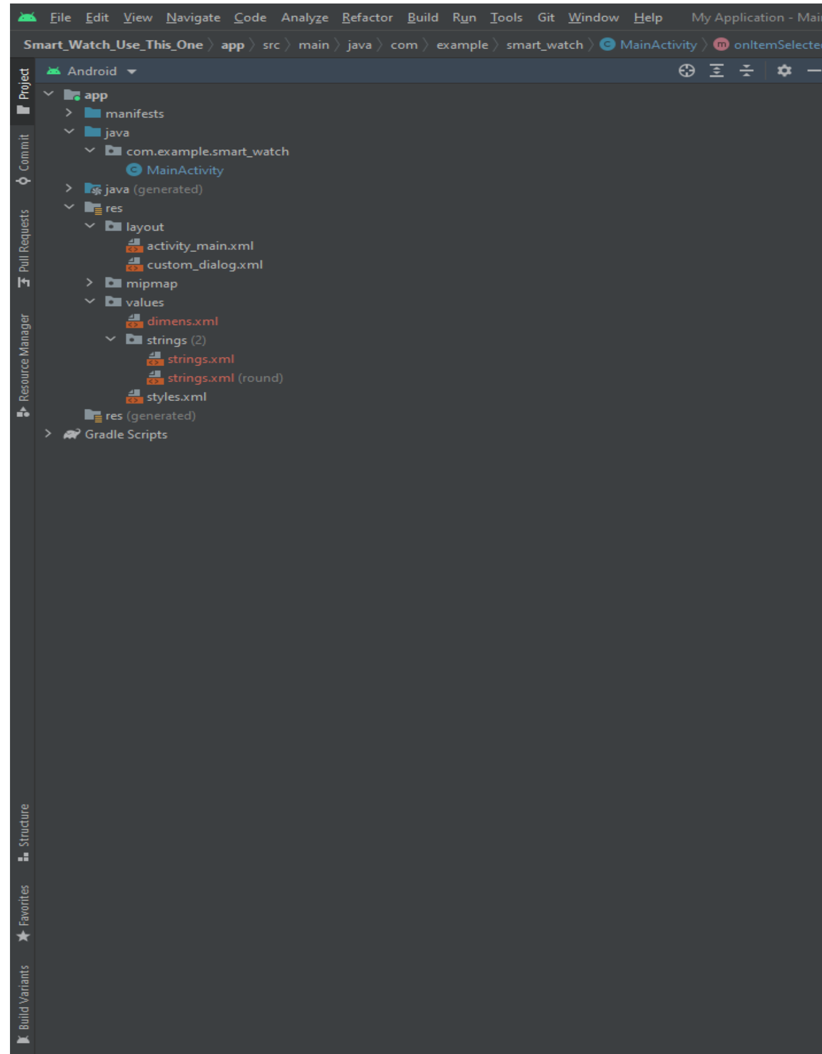
[19]. Google's Officiella sida för Android-utvecklare. Projects overview
Uppdaterat:2021-12-21 UTC Länk: Projects overview Hämtat: 05/02-22

[20]. Grafiskt verktyg för framtagning av UML-resp. Flödesdiagram. Creately
Uppdaterat:2022 Länk: app.creately.com Hämtat: 05/02-22

[21]. Google's Officiella sida för Android-utvecklare. Request location permissions
Uppdaterat:2022-05-12 UTC Länk: Location Permissions Hämtat: 10/04-22

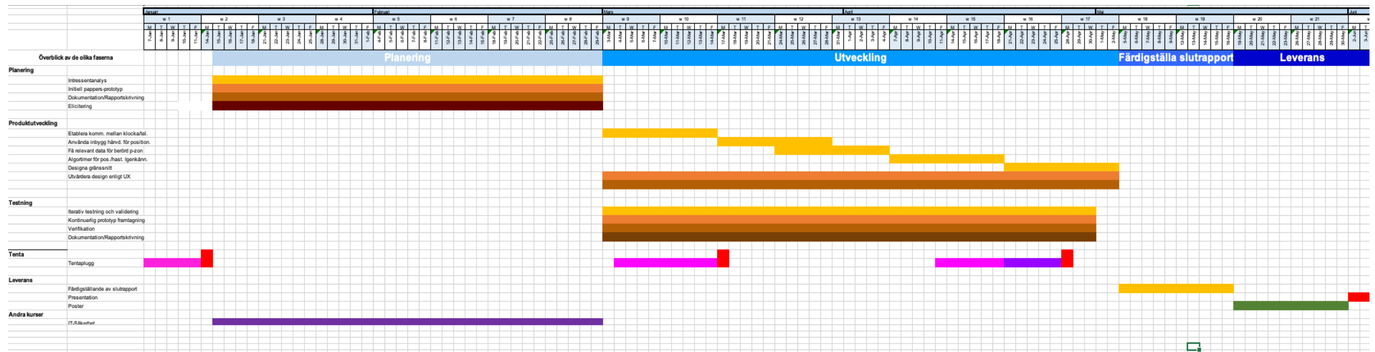
8. Bilagor

Bilaga 1: Android Studio’s hierarki med placering av XML- respektive Javafiler



Bilaga 1: Observera att de relevanta katalogerna för XML filerna är ”Layout” respektive ”Values”. Den förstnämnda innehåller den grafiska vyn varav den andra innehåller språket i applikationen, e.g.

Bilaga 2: Ganttschema Tidsplan



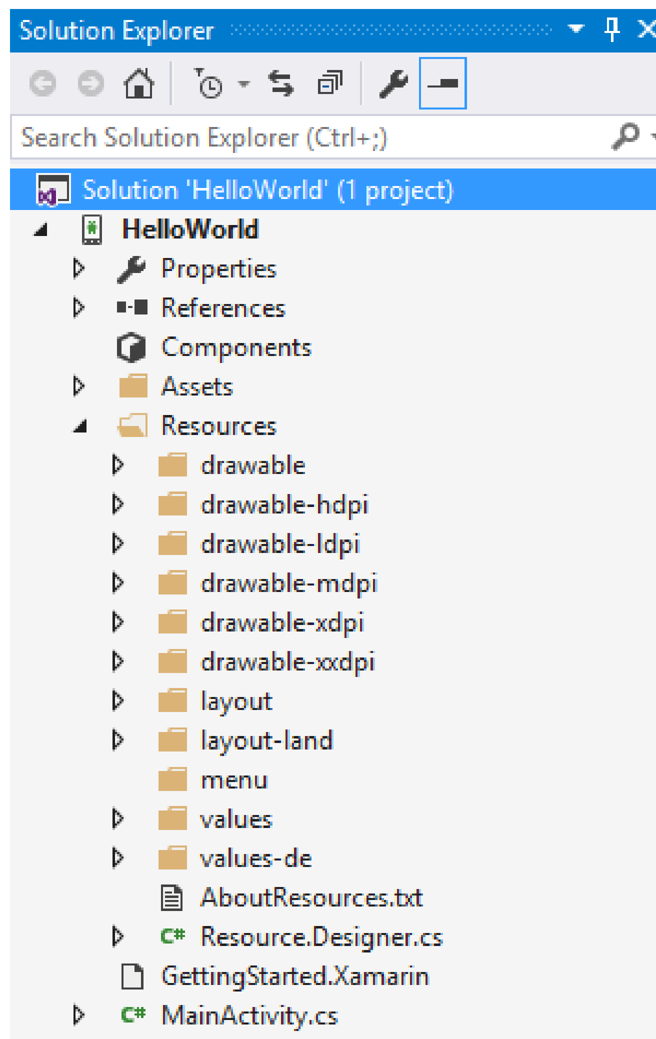
Bilaga 3: Kodexempel för att överföra data mellan enheter med Lågnivå Bluetooth-protokoll

```
public class MyBluetoothService {
  private static final String TAG = "MY_APP_DEBUG_TAG";
  private Handler handler; // handler that gets info from Bluetooth service

  // Defines several constants used when transmitting messages between the
  // service and the UI.
  private interface MessageConstants {
    public static final int MESSAGE_READ = 0;
    public static final int MESSAGE_WRITE = 1;
    public static final int MESSAGE_TOAST = 2;

    // ... (Add other message types here as needed.)
  }

  private class ConnectedThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final InputStream mmInStream;
    private final OutputStream mmOutStream;
    private byte[] mmBuffer; // mmBuffer store for the stream

    public ConnectedThread(BluetoothSocket socket) {
      mmSocket = socket;
      InputStream tmpIn = null;
      OutputStream tmpOut = null;

      // Get the input and output streams; using temp objects because
      // member streams are final.
      try {
        tmpIn = socket.getInputStream();
      } catch (IOException e) {
        Log.e(TAG, "Error occurred when creating input stream", e);
      }
      try {
        tmpOut = socket.getOutputStream();
      } catch (IOException e) {
        Log.e(TAG, "Error occurred when creating output stream", e);
      }

      mmInStream = tmpIn;
      mmOutStream = tmpOut;
    }
```

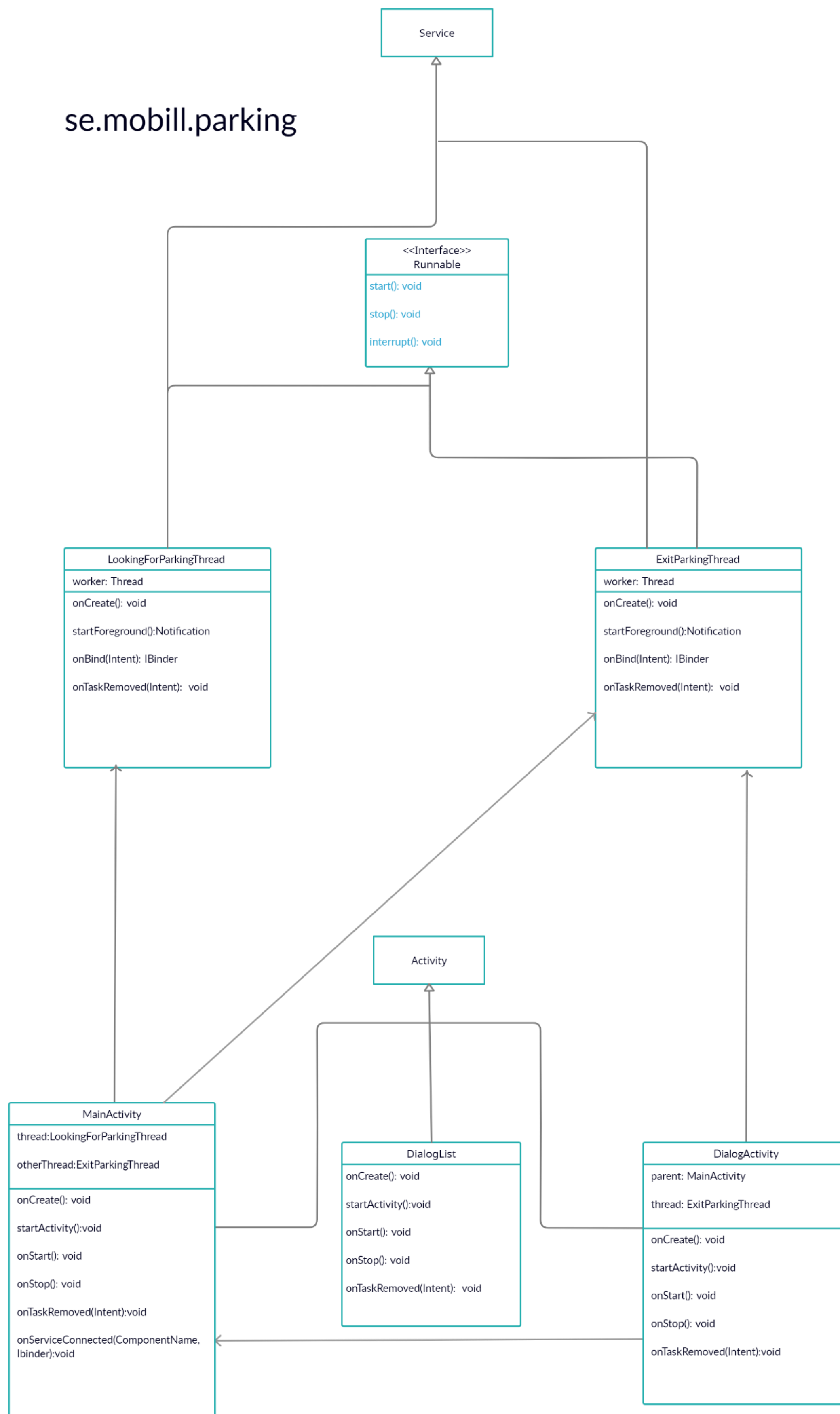
```
public void run() {
  mmBuffer = new byte[1024];
  int numBytes; // bytes returned from read()

  // Keep listening to the InputStream until an exception occurs.
  while (true) {
    try {
      // Read from the InputStream.
      numBytes = mmInStream.read(mmBuffer);
      // Send the obtained bytes to the UI activity.
      Message readMsg = handler.obtainMessage(
          MessageConstants.MESSAGE_READ, numBytes, -1,
          mmBuffer);
      readMsg.sendToTarget();
    } catch (IOException e) {
      Log.d(TAG, "Input stream was disconnected", e);
      break;
    }
  }
}

// Call this from the main activity to send data to the remote device.
public void write(byte[] bytes) {
  try {
    mmOutStream.write(bytes);

    // Share the sent message with the UI activity.
    Message writtenMsg = handler.obtainMessage(
        MessageConstants.MESSAGE_WRITE, -1, -1, mmBuffer);
    writtenMsg.sendToTarget();
  } catch (IOException e) {
    Log.e(TAG, "Error occurred when sending data", e);

    // Send a failure message back to the activity.
    Message writeErrorMsg =
        handler.obtainMessage(MessageConstants.MESSAGE_TOAST);
    Bundle bundle = new Bundle();
    bundle.putString("toast",
        "Couldn't send data to the other device");
    writeErrorMsg.setData(bundle);
    handler.sendMessage(writeErrorMsg);
  }
}

// Call this method from the main activity to shut down the connection.
public void cancel() {
```

```
        try {
            mmSocket.close();
        } catch (IOException e) {
            Log.e(TAG, "Could not close the connect socket", e);
        }
    }
  }
}
```

Bilaga 4: Xamarin.Android hierarki med placering av XML- respektive CS-filer



Bilaga 5: UML-Diagram



Bilaga 6: Illustration av sprintarna