Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Sebastian Alejandro Montanez

Android aplikacija za prenos glasbe v živo preko tehnologij Bluetooth in Wi-Fi Direct

diplomsko delo

visokošolski študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Aleš Jaklič

Ljubljana, 2019

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Zasnujte in izdelajte Android aplikacijo za prenos glasbe v živo med dvema mobilnima napravama. Aplikacija naj omogoča prenos preko Bluetooth in Wi-Fi Direct povezav ter sledenje geograskega položaja naprave preko sistema GPS. Z uporabo Google Maps API naj aplikacija tudi vodi zgodovino opravljenih poti naprave.

Quisiera agradecer a todos aquellos que me acompañaron durante estos años de estudio. En especial a mis padres y hermana, por el apoyo incondicional presente en cada momento. También a mis compañeros de estudio, los cuales estuvieron presentes siempre que necesite. A mis amigos, en especial a German que me ayudo con consejos durante la programación de este programa y a lo largo de mis estudios. Y muy especialmente quiero agradecer a Eva, mi novia, por estar a mi lado, apoyarme y acompañarme en los momentos felices y no tan felices de toda esta etapa.

Na koncu se najlepše zahvaljujem tudi svojemu mentorju za vse nasvete in pomoč pri izdelavi diplomske naloge.

Kazalo

Povzetek

Abstract

1 Uvod **1**

2 Funkcionalnost Music Library in Media Player **5**

2.1 Music Library 5

2.2 Media Player 7

3 Funkcionalnost Bluetooth Connection **13**

3.1 Dovoljenja 14

3.2 Vzpostavitve povezave 15

3.3 Predvajanje zvoka 21

4 Funkcionalnost WiFi Connection **23**

4.1 Dovoljenja 24

4.2 Vzpostavitev povezave 24

4.3 Predvajanje zvoka 28

5 Funkcionalnost Trace **29**

5.1 Dovoljenja 29

5.2 Pridobivanje storitve Google Maps 30

5.3 Sledenje naprave 33

5.4 Funkcionalnost SEE ON MAP 36

6 Funkcionalnost History **39**

7 Sklep **45**

Literatura **49**

Seznam uporabljenih kratic

kratica	angleško	slovensko
UI	user interface	uporabniški vmesnik
MAC	Media Access Control	nadzor dostopa do medijev
RFCOMM	Radio Frequency Communication	radiofrekvenčna komunikacija
PCM	Pulse-Code Modulation	pulzno kodna modulacija
API	Application Programming Interface	vmesnik uporabniškega programa
GPS	Global Positioning System	globalni sistem pozicioniranja
SDK	Software Development Kit	komplet za razvoj programske opreme
UUID	Universally Unique Identifier	univerzalno enolični identifikator
SDP	Service Discovery Protocol	protokol za odkrivanje storitev
URI	Uniform Resource Identifier	enolični identifikator vira

Povzetek

Naslov: Android aplikacija za prenos glasbe v živo preko tehnologij Bluetooth in Wi-Fi Direct

Avtor: Sebastian Alejandro Montanez

Za svojo diplomsko nalogo sem želel ustvariti aplikacijo za mobilne naprave z operacijskim sistemom Android, ki bi omogočala deljenje glasbe v živo med dvema napravama.
V ta namen sem moral naprej ugotoviti, kako poiskati in predvajati vse datoteke tipa ≫.mp3≪, ki se nahajajo v napravi. Nato sem omogočil povezavo med dvema napravama preko uporabe tehnologij Bluetooth in Wi-Fi Direct ter našel način, da se v obeh napravah istočasno predvaja ista skladba. Svoji aplikaciji sem dodal tudi možnost sledenja napravi preko uporabe Google Maps API, ki uporabniku omogoča vpogled v poti in razdalje, ki jih je naredil s svojo napravo.

Ključne besede: glasba, deljenje, Bluetooth, Wi-Fi Direct, sledenje.

Abstract

Title: Android Application for Music Streaming using Bluetooth and Wi-Fi Direct Technologies

Author: Sebastian Alejandro Montanez

The idea for this thesis is the development of a mobile application for Android devices that lets people share live music between two devices.
As a first step, I had to solve how to find and play all the audio files located in the device. The next step was to connect two devices using Bluetooth and Wi-Fi Direct technologies and find a way to be able to listen the same song on both devices at the same time. Lastly, I worked on getting the geolocation from the device, so I could have a record of the travelled paths and distances.

Keywords: music, share, Bluetooth, Wi-Fi Direct, geolocation.

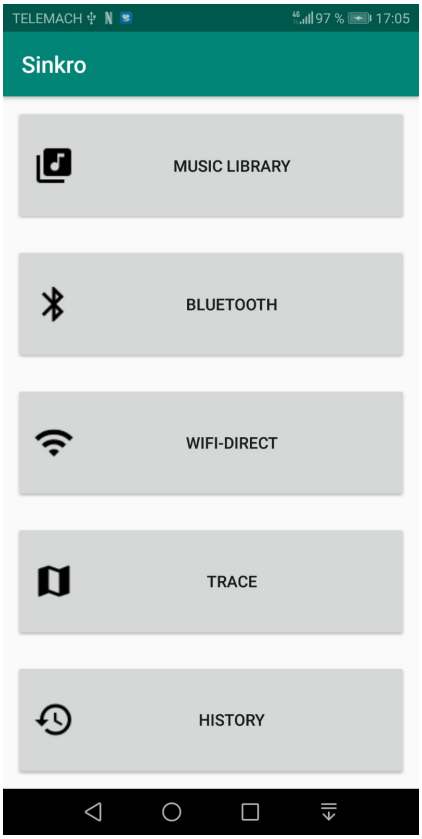
Poglavje 1

Uvod

V današnjem hitrem ritmu življenja je pomembno, da si znamo vzeti čas zase, za svoje telo in za druženje. Veliko ljudi se zato odpravi na tek, mnogi med njimi tudi v družbi.

Ideja te aplikacije je nastala z željo, da bi dvema osebama, ki se skupaj odpravita na tek, omogočila, da si med tekom med seboj delita glasbo in tako istočasno poslušata isto skladbo ter lahko na ta način tudi vzdržujeta isti ritem teka. Čeprav je aplikacija namenjena predvsem tekačem, pa se lahko uporabi tudi v drugih življenjskih situacijah, uporabljajo jo lahko na primer prijatelji, ki želijo skupaj poslušati glasbo, vendar z njo ne želijo ali ne smejo motiti drugih navzočih.

Tako je nastal »Sinkro«, aplikacija za mobilne naprave z operacijskih sistemom Andriod, ki omogoča možnost deljenja glasbe v živo med dvema Android napravama preko tehnologije Bluetooth in WiFi-Direct. Aplikacija ob tem ponuja tudi možnost sledenja napravi, tako da lahko uporabnik vidi in preverja pot ter razdaljo, ki jo je pretekel. Ker aplikacija shranjuje podatke o razdaljah, pa uporabniku omogoča tudi pregled vseh razdalj, ki jih je naredil posameznega dne. Dodatni funkciji aplikacije sem dodal z namenom, da bi bila aplikacija uporabna tudi v primeru, če uporabnik ne želi deliti glasbe, temveč želi le beležiti pot in razdaljo med tekom.



Slika 1.1: Začetni Activity

Kot prikazuje 1.1, aplikacija ponuja različne funkcije uporabe:

- »Music Library«: omogoča predvajanje vseh datotek tipa ».mp3«, ki so shranjene v napravi.
- »Bluetooth«: omogoča deljenje glasbe, ki jo trenutno predvajamo, z drugo napravo.
- »WiFi-Direct«: od funkcije »Bluetooth« se razlikuje v tem, da se naprave med seboj povežejo preko tehnologije WiFi-Direct.
- »Trace«: omogoča sledenje napravi, s čimer lahko pregledamo pot, ki jo je naredil uporabnik, ter izračunamo njeno razdaljo.

- »History«: omogoča shranjevanje in beleženje vseh razdalj, ki jih je uporabnik naredil posameznega dne, s čimer lahko spremljamo njegovo vsakodnevno fizično aktivnost.

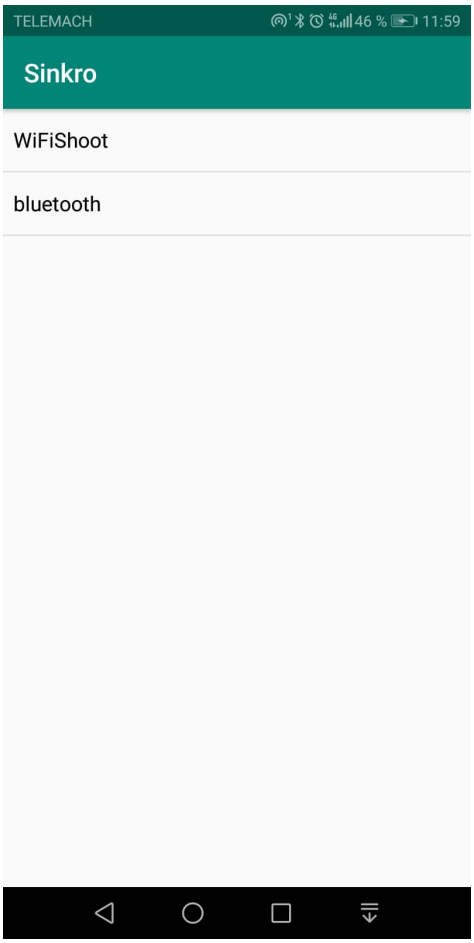
Poglavje 2

Funkcionalnost Music Library in Media Player

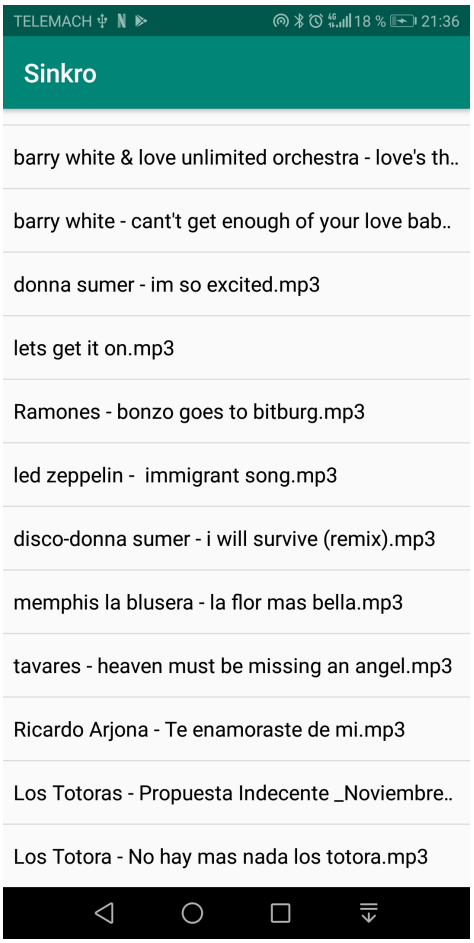
2.1 Music Library

2.1.1 Dovoljenja

Aktivnosti »Music Library« in »Media Player« sta osredotočeni na predvajanje glasbene vsebine, ki obstaja v napravi. Za to je naprej potrebno dovoljenje uporabnika za dostop do notranjega pomnilnika.
Po tem, ko je dovoljenje podano, sem pridobil mape, ki vsebujejo datoteke tipa ".mp3", za kar sem uporabil funkcijo "FolderWithMusic", ki sem jo pridobil na spletni strani StackOverflow [1]. V »Music Library« se nato preko uporabe orodja ListView prikažejo vse mape, ki vsebujejo datoteke ».mp3«. S klikom na posamezno mapo aktivnost »Music Library« znova kliče sama sebe, a tokrat v orodju ListView naloži vse datoteke tipa ».mp3«, ki jih mapa vsebuje. Ko uporabnik izbere določeno pesem, se odpre aktivnost, imenovana »Media Player«.



Slika 2.1: Mape, ki vsebujejo glasbo



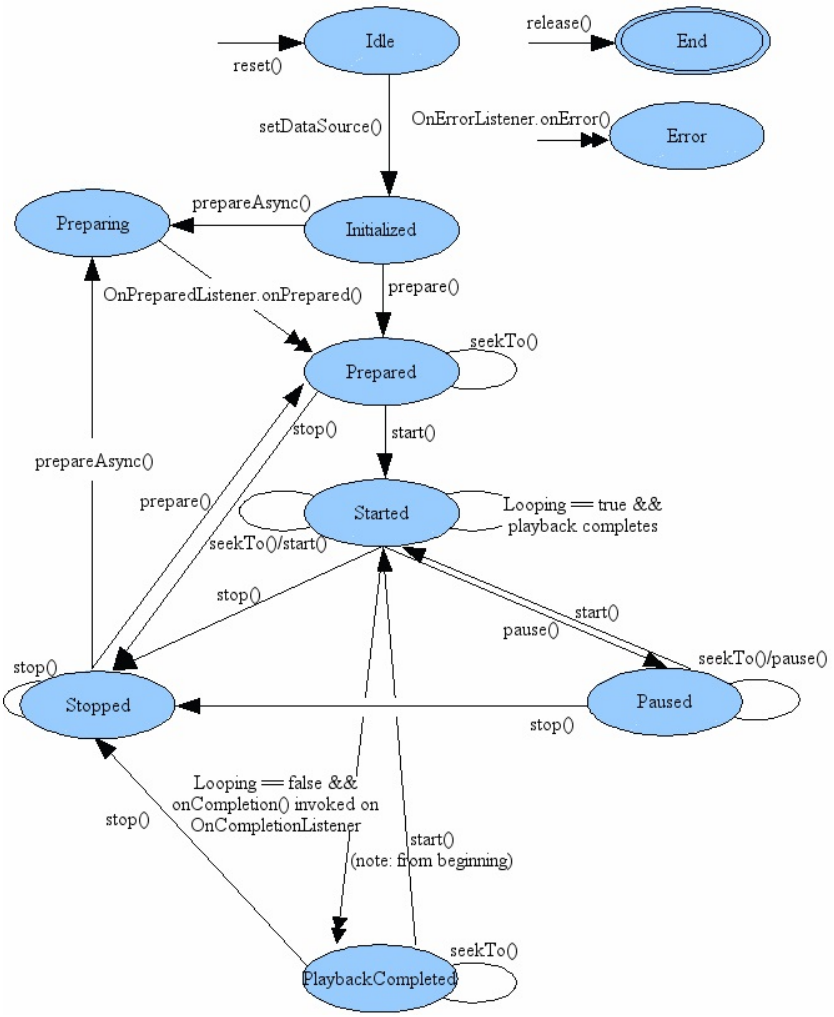
Slika 2.2: Skladbe v posamezni mapi

2.2 Media Player

Ko se odpre aktivnost »Media Player«, le ta iz aktivnosti »Music Library« pridobi pot, kjer se nahaja glasbena datoteka, ki jo želi predvajati uporabnik. Za predvajanje izbrane skladbe se v aktivnosti »Media Player« ustvari nova spremenljivka tipa MediaPlayer.
Multimedijsko ogrodje Android vključuje podporo za predvajanje različnih skupnih vrst medijev, tako lahko preprosto integriramo zvok, video in slike v svoje aplikacije. Predvajamo lahko zvok ali video iz multimedijskih datotek, shranjenih v virih aplikacije, iz samostojnih datotek v datotečnem sistemu ali iz podatkovnega toka, ki prihaja iz omrežne povezave, vse z API-ji MediaPlayer [2].

MediaPlayer class se lahko uporablja za nadzor predvajanja avdio / video datotek in podatkovnega toka (stream) [3].

Slika 2.3 prikazuje življenjski cikel in stanje objekta MediaPlayer, ki ga poganjajo podprte operacije nadzora predvajanja. Krogi predstavljajo stanja, v katerih se lahko nahaja objekt MediaPlayer. Puščice predstavljajo operacije nadzora predvajanja, ki poganjajo prehod stanja v objektu. Obstajata dve vrsti puščic. Puščice z eno glavo predstavljajo sinhronske klice metod, medtem ko puščice z dvojno glavo predstavljajo asinhronske klice metod [3].



Slika 2.3: Življenjski cikel objekta MediaPlayer [3]

V aplikaciji Sinkro si stanja v spremenljivki tipa MediaPlayer sledijo v naslednjih korakih:

- Najprej se inicializira spremenljivka tipa MediaPlayer, v kateri se naloži datoteka, ki jo želi uporabnik predvajati.

```
mediaPlayer = new MediaPlayer();
mediaPlayer.setDataSource(is.getFD());
```

Slika 2.4: Kreiranje ter inicializacija spremenljivke tipa MediaPlayer

- Določi se tip stream-a.

```
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
```

Slika 2.5: Določitev tipa stream-a

- Spremenljivka tipa MediaPlayer ostane v stanju pripravljenosti

```
mediaPlayer.prepare();
```

Slika 2.6: Spremenljivka »mediaPlayer« v stanju pripravljenosti

- Ko se uporabnik odloči predvajati glasbo in po pritisku na gumb »Play/Pause«, se stanje spremenljivke tipa MediaPlayer spremeni v »Started«.

```
mediaPlayer.start();
```

Slika 2.7: Začetek predvajanje spremenljivke »mediaPlayer«

- Ob ponovnem pritisku na gumb »Play/Pause«, se stanje spremenljivke tipa MediaPlayer spremeni v stanje pavze.

```
mediaPlayer.pause();
```

Slika 2.8: Spremenljivka »mediaPlayer« v stanju pavze

- Ko se skladba, ki se trenutno predvaja, konča, spremenljivka tipa MediaPlayer kliče funkcijo ».OnCompletionListener«. Ta sprosti spremenljivko tipa MediaPlayer in ponovi zgoraj opisani proces za naslednjo skladbo.

```
//When song finish, play next song.
mediaPlayer.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
    @Override
    public void onCompletion(MediaPlayer mp) {
        mediaPlayer.release();
        mediaPlayer = null;
        PlayNextSong(Globals.getGlobalSongsList(), songsFilePath, textSongName, play, positionBar);
    }
});
```

Slika 2.9: Spremenljivka »mediaPlayer« v stanju pavze

Poleg spremenljivke tipa MediaPlayer se v aktivnosti »Media Player« uporabljata še dve zelo pomembni orodji, to sta: Handler in Runnable. Obe se

uporabljata pri seek bar-u, ki uporabniku omogoča izbiro, kateri del pesmi želi poslušati.

Vmesnik Runnable mora biti izveden v kateremkoli razredu, ki naj bi bil izvršen v niti. Edina metoda, ki jo mora razred definirati, je metoda »run()«, ki ne dobi nobenega argumenta. Runnable prav tako zagotavlja sredstva za razred, ki je aktiven, ne da bi potrebovali uporabiti razširitev tipa »Thread«. Vmesnik Runnable se večinoma uporablja v primerih, ko je potrebna zgolj uporaba metode run () in ni potrebna nobena druga »Thread« metoda [4].

Ker je »Runnable« vrsta niti, to pomeni, da se izvaja vzporedno z glavno nitjo aplikacije, zato lahko sledimo, v katerem trenutku predvajanja pesmi smo, ne da bi s tem motili normalno delovanje aplikacije. Ko se predvajanje pesmi konča ali se začne predvajati druga skladba, se ta Runnable uniči, ustvari pa se nov Runnable, ki ponovno prikazuje trenutek predvajanja nove skladbe. Zaradi navedenega in ker Runnable ne vsebuje nobenega pogoja za prekinitev delovanja niti, sem se odločil za uporabo vmesnika Runnable.

Ko želimo sledenje predvajanja posamezne skladbe prikazati v glavni niti programa, pa moramo pri tem uporabiti novo pomembno orodje, imenovano »Handler«.

Orodje »Handler« omogoča obdelavo in procesiranje sporočil, zagotavlja mehanizem za pošiljanje (v smislu povezave) med »threads« oziroma nitmi in tako lahko pošilja sporočila iz naše sekundarne niti v UI Thread oziroma glavno nit [5].

V aktivnosti »Media Player« sem od orodja »Handler« uporabljal samo metodo ».postDelayed(...)«, ki omogoča, da je ta Runnable dodan v čakalno vrsto sporočil, ki se bodo zagnala po preteku določenega časa [6]. To pomeni, da bo po preteku določenega časa metoda prebrala sporočilo iz čakalne vrste in ga poslala v glavno nit. Z uporabo te metode sem lahko definiral, koliko časa preteče med enim in drugim sporočilom v milisekundah. Ker sem želel pokazati trajanje pesmi v sekundah, sem čas med sporočil definiral na 1000 milisekund, kar je enako eni sekundi. Na ta način sem v glavni niti lahko

prikazal, v kateri sekundi skladbe se trenutno predvaja glasba.

Poglavje 3

Funkcionalnost Bluetooth Connection

Aktivnost »Bluetooth connections« omogoča povezavo med dvema mobilnima napravama preko tehnologije Bluetooth. Za uporabo te aktivnosti je treba najprej upoštevati dva zelo pomembna pogoja, in sicer, da je bluetooth v napravi aktiviran in da so podana vsa dovoljenja, ki jih aktivnost potrebuje.

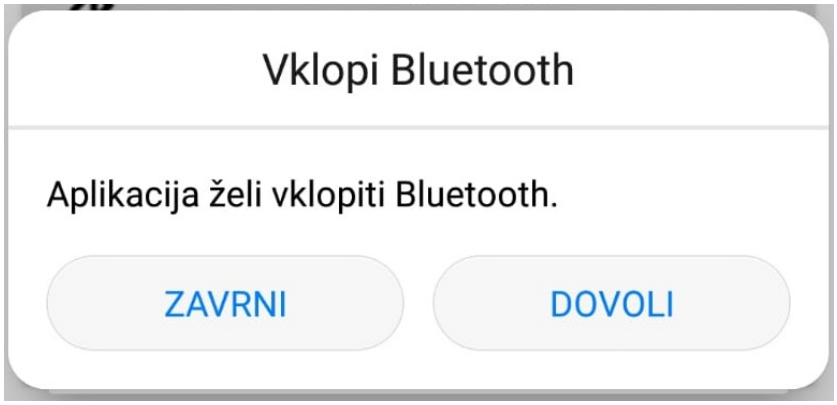
Prvi pogoj je povsem pričakovan, saj se povezava med napravama ne more vzpostaviti, dokler v napravi ni aktiviran Bluetooth. Aplikacija sama vpraša za aktivacijo Bluetooth-a, od odločitve uporabnika pa je odvisno, ali ga aktivira ali ne. Dokler pa Bluetooth v napravi ni aktiviran, aplikacija ne more izvajati aktivnosti »Bluetooth connection«.

Za aktivacijo Bluetooth-a v napravi sem uporabil naslednjo kodo, ki sem jo dobil na spletni strani Android Developers [7].

```
if (!bluetoothAdapter.isEnabled()) {
    Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
}
```

Slika 3.1: Intent za aktiviranje Bluetooth-a v napravi

Ta koda omogoča, da se uporabniku prikaže okno, v katerim prosi za priklop Bluetooth-a v napravi, kot prikazuje Slika 3.2.



Slika 3.2: Intent za aktiviranje Bluetooth-a v napravi

3.1 Dovoljenja

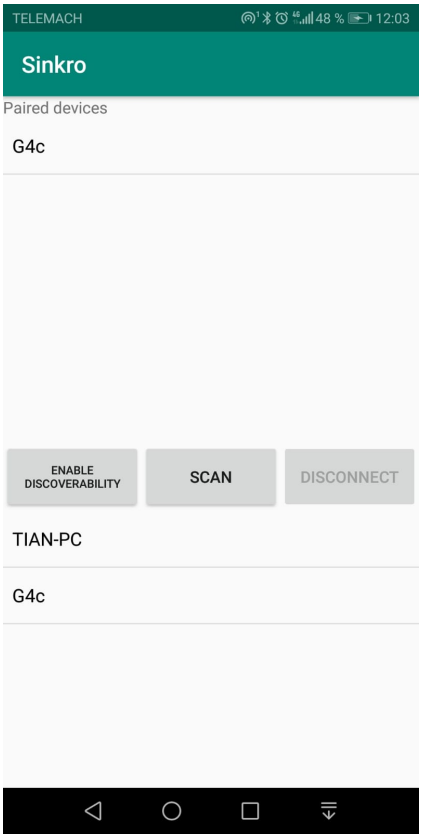
Drugi pogoj za uporabo aktivnosti »Bluetooth connections« so dovoljenja. Za pravo delovanje aktivnosti so potrebna tri dovoljenja, in sicer:

1. BLUETOOTH: To dovoljenje je potrebno za vzpostavitev kakršnekoli komunikacije prek povezave Bluetooth, na primer zahteve po povezavi, sprejema povezave in prenosa podatkov [7].

2. LOKACIJA: To dovoljenje je potrebno, ker je Bluetooth iskanje lahko uporabljeno za pridobivanje informacije o trenutni lokaciji uporabnika. Ta informacija lahko izvira iz naprave uporabnika ali iz Bluetooth oddajnikov, ki se uporabljajo na lokacijah, kot so na primer trgovine ali javni objekti [7].

3. BLUETOOTH_ADMIN: To dovoljenje je potrebno, ker je v aplikaciji treba spremeniti Bluetooth nastavitve naprave, med katerimi je tudi vidnost naprave. Da se napravi lahko najdeta in vzpostavita medsebojno

povezavo, morata namreč biti vidni druga drugi. Aplikacija mora zato omogočiti vidnost naprave [7].



Slika 3.3: Bluetooth Connections Activity

Ko so vsa dovoljenja odobrena, se prikaže aktivnost »Bluetooth Connections«. Slika 3.3 prikazuje primer te aktivnosti.

3.2 Vzpostavitve povezave

Prvi korak, ki je bil potreben pri programiranju te aktivnosti, je bil ustvariti spremenljivko tipa »BluetoothAdapter«, ki predstavlja vmesnik Bluetooth-a v napravi. BluetoothAdapter omogoča opravljanje osnovnih nalog Bluetooth-a, na primer odkrivanje naprav, poizvedovanje po seznamu seznanjenih na-

prav, inicializiranje na primer »BluetoothDevice« z »MAC« naslovom, ustvarjanje vmesnika »BluetoothServerSocket« za sprejemanje zahtev za povezavo z drugimi napravami in zagon skeniranja Bluetooth naprav [8].

Bluetooth adapter je bil v aplikaciji prvič uporabljen pri preverjanju, ali je Bluetooth v napravi aktiviran. Pri tem sem uporabil metodo »isEnabled()«, ki preveri, če je Bluetooth v napravi aktiviran ali ne.

Po tem, ko je bila spremenljivka tipa »BluetoothAdapter« ustvarjena, je bilo treba preveriti, katere naprave so bile predhodno že povezane s to napravo, za kar sem uporabil funkcijo ».getBondedDevices()«. Ta funkcija vrne seznam objektov tipa »BluetoothDevice«, ki predstavljajo naprave, ki so bile v preteklosti že povezane [7].

Z uporabo objekta tipa »BluetoothDevice« sem dobil informacije o teh napravah. V mojem primeru sem potreboval samo dva podatka, in sicer:

- Ime naprave
- MAC naslov

Ime naprave sem potreboval zato, da bi ga dodal v nek seznam in ga preko uporabe orodja »ListView« prikazal uporabniku, da bi se ta lahko odločil, s katero napravo si želi vzpostaviti povezavo.
MAC naslov pa sem potreboval, ker je za vzpostavitev povezave z drugo napravo preko Bluetooth-a potrebno poznati njen MAC naslov [7].

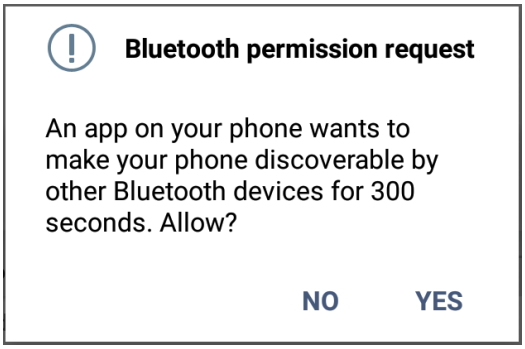
Aplikacija nudi možnost aktiviranja vidnosti naprave, kar pomeni, da s pritiskom na gumb »Enable Discoverability«, kot prikazuje Slika 3.3, naprava postane vidna za druge naprave, s tem pa je omogočena vzpostavitev medsebojne povezave. Vidnost sem aktiviral z uporabo naslednje kode [7]:

Z vrstico »discoverableIntent.putExtra(...)« sem definiral, koliko časa bo naprava vidna, v mojem primeru je to 5 minut ali 300 sekund.

```
Intent discoverableIntent =
        new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
startActivity(discoverableIntent);
```

Slika 3.4: Koda za aktiviranje vidnosti Bluetooth-a

Ob uporabi te kode se prikaže okno, ki uporabnika vpraša za odobritev aktiviranja vidnosti naprave, kot prikazuje Slika 3.5.



Slika 3.5: Dovoljenje za aktivacijo vidnosti naprave

Kot kaže Slika 3.3, aktivnost »Bluetooth Connections« vsebuje tudi gumb »Scan«, ki sproži iskanje po Bluetooth napravah, ki se nahajajo v bližini. Za aktiviranje tega iskanja sem uporabil funkcijo »startDiscovery()«. Postopek iskanja je asinhron in vrne logično vrednost, ki kaže, ali se je iskanje začelo ali ne. Postopek po napravah vključuje skeniranje, ki traja 12 sekund, temu sledi skeniranje vsake naprave, ki jo najde, da dobi njeno ime [7]. V svojem primeru sem vsako najdeno napravo dodal v nek »ArrayAdapter«, ki sem ga kasneje uporabil za prikaz vseh najdenih naprav z uporabo orodja »ListView«.

Za vzpostavitev povezave med dvema napravama sem moral implementirati tako strežniške kot tudi odjemalske mehanizme, saj mora ena naprava najprej ustvariti strežniški vtič, druga pa mora začeti povezavo z MAC na-

slovom strežniške naprave. Vsaka strežniška naprava in odjemalska naprava dobita zahtevani BluetoothSocket na različne načine. Strežnik prejme informacije o vtiču, ko je sprejeta dohodna povezava. Odjemalec pa poda informacije o vtiču, ko odpre kanal RFCOMM strežniku. Strežnik in odjemalec sta med seboj povezana, če imata na isti kanal RFCOMM priključeno povezavo Bluetooth. Na tej točki lahko vsaka naprava pridobi vhodne in izhodne tokove in prenos podatkov se lahko začne [7].

Za kreiranje strežniškega vtiča mora strežnik, poleg MAC naslova, definirati tudi ime aplikacije in UUID številko. Ker je ime moje aplikacije »Sinkro«, sem isto ime uporabil tudi v tem primeru.

```
private static String APP_NAME = "Sinkro";
```

Slika 3.6: Kreiranje ter inicializacija spremenljivke »APP_NAME«

To ime se uporablja za identifikacijo te storitve in ga sistem avtomatsko doda v podatkovno bazo protokola za odkrivanje storitev (SDP) v napravi. Za definiranje UUID številke pa sem uporabil spletni generator, ki sem ga našel na spletnem naslovu: *https://www.uuidgenerator.net/*. UUID se prav tako vnese v podatkovno bazo SDP in je osnova za pogodbo o povezavi z odjemalsko napravo. To pomeni, da odjemalec, ko poskuša vzpostaviti povezavo s to napravo, nosi UUID, ki enolično identificira storitev, s katero se želi povezati. Ti UUID-ji se morajo ujemati, da bo povezava sprejeta [7].

UUID je standardiziran 128-bitni format od ID niza, ki se uporablja za enolično identifikacijo informacij. Bistveno pri UUID je to, da je dovolj velik, da lahko izberemo poljuben naključni ID, ki se ne bo ujemal z nobenim drugim ID-jem. V mojem primeru se uporablja za edinstveno prepoznavanje Bluetooth storitve te aplikacije [7].

Po pridobitvi UUID preko spleta sem ga v kodi inicializiral na slednji

način:

```
private static UUID MY_UUID = java.util.UUID.fromString("34940103-baa1-4b79-b1a5-27da6e11e7a1");
```

Slika 3.7: Kreiranje ter inicializacija spremenljivke »MY_UUID«

Kot zadnji korak za kreiranje strežniškega vtiča sem uporabil naslednjo vrstico v kodi:

```
bcServerSocket = bcBluetoothAdapter.listenUsingRfcommWithServiceRecord(APP_NAME,MY_UUID);
```

Slika 3.8: Inicializacija spremenljivke »bcServerSocket«

Na strani odjemalca sem nato kreiral odjemalski vtič z naslednjim ukazom:

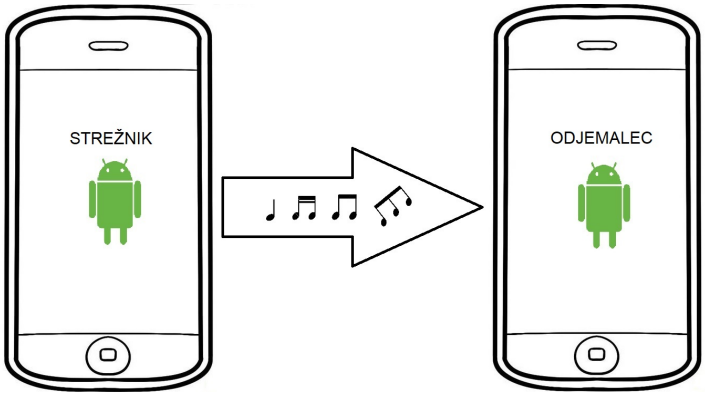
```
bcClientSocket = device.createRfcommSocketToServiceRecord(MY_UUID);
```

Slika 3.9: Inicializacija spremenljivke »bcClientSocket«

V primeru uporabe Bluetooth-a v tej aplikaciji bo smer pošiljanja podatkov vedno od strežnika k odjemalcu, kar pomeni, da bo glasbo, ki jo predvaja strežniška naprava, lahko poslušala tudi odjemalska naprava in ne obratno. Navedeno prikazuje Slika 3.10.

Diagram na Sliki 3.11 prikazuje potek povezave med napravama. Ko se aktivnost »BluetoothConnections« zažene v napravi, najprej sproži možnost povezave, v kateri je naprava v vlogi strežnika, in čaka na njeno vzpostavitev. Od tedaj dalje obstajata dve možnosti:

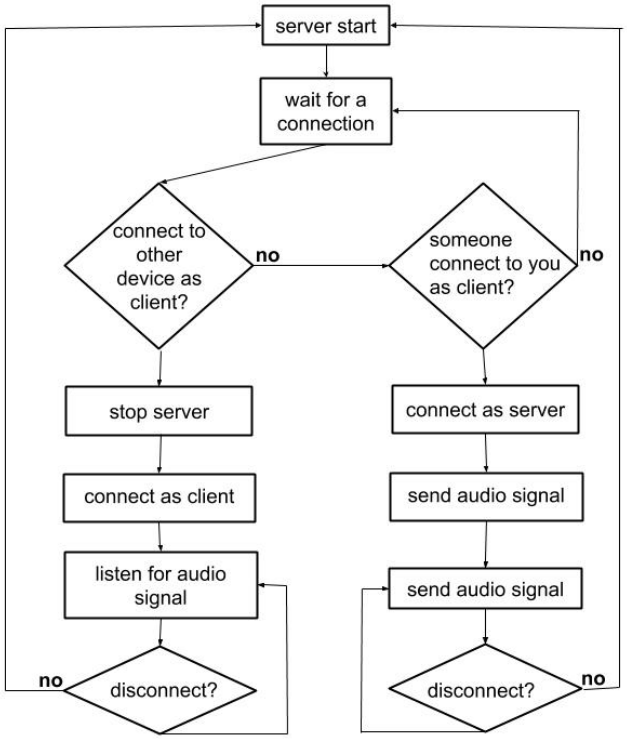
- Naprava se priklopi kot strežnik: V tem primeru se komunikacija med napravama vzpostavi tako, da je naprava v vlogi strežnika, kar pomeni,



Slika 3.10: Smer podatkovnega toka v Bluetooth povezavi

da bo podatkovni tok izviral iz te naprave. Kot je vidno na Sliki 3.10, smer podatkovnega toka v tej aktivnosti aplikacije vedno poteka od strežnika k odjemalcu, zaradi česar naprava, takoj ko se priklopi kot strežnik, začne pošiljati podatke. Ko pride do prekinitve povezave, naprava avtomatsko sproži možnost nove povezave, v kateri je naprava znova v vlogi strežnika.

- Naprava se priključi kot odjemalec: V tem primeru naprava prekine možnost novih povezav in se poveže z drugo napravo v vlogi odjemalca. Kot prikazuje Slika 3.10, bo ta naprava predvajala podatkovni tok, ki bo izviral iz druge naprave, s katero je povezana. Ko se naprava poveže, začne predvajati vhodni podatkovni tok, dokler ne pride do prekinitve povezave. Ob prekinitvi pa naprava avtomatsko sproži možnost nove povezave z napravo, ki je v bližini in bi se želela povezati kot odjemalec.



Slika 3.11: Diagram poteka

3.3 Predvajanje zvoka

Za predvajanje dohodnega podatkovnega toka, ki prihaja v obliki toka bajtov, sem uporabljal orodje »AudioTrack«.

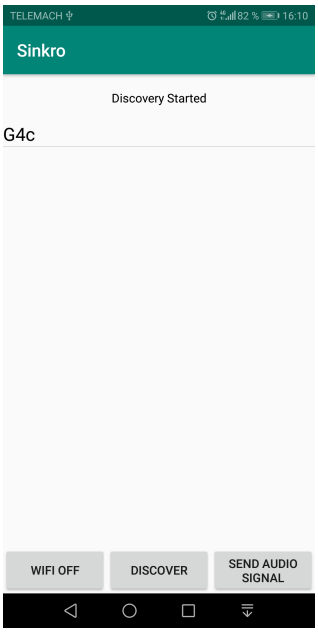
Razred AudioTrack upravlja in predvaja en audio vir za aplikacije Java. Omogoča prenos audio medpomnilnika odjemalcu za predvajanje zvoka [9].

Način, s katerim sem uporabljal orodje AudioTrack, se imenuje »stream«. Z njim sem omogočil, da aplikacija lahko zapiše neprekinjen tok podatkov v AudioTrack-u z uporabo metode »write()« [8]. Ko se naprava poveže z drugo napravo kot odjemalec, kreira objekt tipa AudioTrack in ga takoj začne predvajati z uporabo funkcije ».play()«. Ko začne prihajati tok podatkov in

je količina podatkov zadostna, AudioTrack predvaja del pesmi, ki je naložen.

Poglavje 4

Funkcionalnost WiFi Connection



Slika 4.1: WiFi Connections Activity

Poleg možnosti povezave preko tehnologije Bluetooth, aplikacija nudi tudi možnost povezave med napravami za prenos glasbe v živo preko uporabe tehnologije WiFi-Direct.

4.1 Dovoljenja

Pri programiranju aktivnosti »WiFi Connection« sem moral najprej zagotoviti potrebna dovoljenja, ki jih za svoje delovanje potrebuje ta aktivnost, in sicer:

- ACCESS_COARSE_LOCATION: Do verzije Androida 6 je lahko vsaka aplikacija preko uporabe Wi-Fi dobila lokacijo naprave brez dovoljenja uporabnika. Od verzije Android 6 dalje pa je Google poskušal to težavo rešiti tako, da je uporabnikom sporočil, da aplikacija z dostopom do lokalnega omrežja pridobi tudi podatek o lokaciji naprave [10]. To dovoljenje je edino od naštetih, ki ga odobri uporabnik.

- CHANGE_WIFI_STATE: To dovoljenje aplikaciji omogoča spremembo stanja povezave Wi-Fi [11].

- ACCESS_WIFI_STATE: To dovoljenje aplikaciji omogoča dostop do informacij o omrežjih Wi-Fi [11].

- INTERNET: To dovoljenje je potrebno zato, ker povezava Wi-Fi Direct ali Wi-Fi P2P uporablja standardne Java vtiče, čeprav Wi-Fi P2P ne zahteva povezave z internetom [12].

Šele, ko so vsa zgoraj navedena dovoljenja podana, se lahko odpre aktivnost »WiFi Connection«.

4.2 Vzpostavitev povezave

Prvi korak pri programiranju te aktivnosti je bil kreiranje treh pomembnih objektov, to so »WifiManager«, »WifiP2pManager« in »WifiP2pManager .Channel«.
Razred »WifiManager« aplikaciji ponuja primarni API za upravljanje vseh vidikov povezave Wi-FI. Ukvarja se z:

- Seznamom konfiguriranih omrežij, ki ga je možno pogledati in posodobiti ter spremeniti atribute posameznih vnosov.
- Trenutno aktivnim omrežjem Wi-Fi, če ta obstaja. Povezavo je možno vzpostaviti ali prekiniti, prav tako je mogoče pridobiti dinamične informacije o stanju omrežja.
- Rezultati skeniranja dostopne točke, ki vsebuje dovolj informacij za odločanje o dostopni točki, s katero se povezuje.
- Definira imena različnih Intent akcij, ki omogočajo spremembe v stanju Wi-Fi [13].

Razred »WifiP2pManager« aplikaciji ponudi API za upravljanje povezave Wi-Fi peer-to-peer. Aplikaciji omogoči odkrivanje razpoložljivih vrstnikov (peers), vzpostavi povezavo in naredi poizvedbo po seznamu vrstnikov [14].

Objekt tipa »WifiP2pManager.Channel« pa aplikacijo poveže z ogrodjem Wifi P2P [15].

V naslednjem koraku sem moral kreirati razred »BroadcastReceiver«. Ta aplikaciji omogoča sledenje vsem spremembam v sistemu »Wi-Fi P2P state« [12]. Od navedenega razreda sem uporabil nekaj konstant, in sicer:

- WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION: označi, ali je Wi-Fi p2p aktiviran ali ne.
- WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION: označi, ali se je zgodila kakšna sprememba v seznamu vrstnikov.
- WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION: označi, ali se je stanje povezave spremenilo.
- WifiP2pManager. WIFI_P2P_THIS_DEVICE_CHANGED_ACTION: označi spremembe v nastavitvah naprave [12].

Po kreiranju razreda »BroadcastReceiver« sem se vrnil v glavni razred aktivnosti »WifiConnection«, kjer so prikazani gumbi »WIFI ON/OFF«, »DISCOVER« in »SEND AUDIO SIGNAL«, kot prikazuje Slika 4.1.

S pritiskom na gumb »WIFI ON/OFF« je možen vklop in izklop tehnologije Wi-Fi v napravi. Za to sem uporabil metodo ».setWifiEnable(true/fase)« iz objekta »WifiManager«.

Gumb »DISCOVER« omogoča, da je naprava vidna drugim napravam in da lahko najde druge naprave.

Za iskanje vrstnikov, ki so na voljo, sem uporabil funkcijo »discoverPeers« iz objekta »WifiP2pManager«. Ta funkcija sproži postopek odkrivanja, ki vključuje skeniranje razpoložljivih vrstnikov Wi-Fi z namenom vzpostavitve povezave. Klic funkcije takoj obvesti aplikacijo o uspehu ali neuspehu pri odkrivanju drugih naprav, aplikacija pa obvesti uporabnika, če se je odkrivanje uspešno začelo ali če je prišlo do kakšne napake. Postopek odkrivanja drugih naprav ostane aktiven, dokler se ne vzpostavi kakšna povezava ali dokler se ne oblikuje skupina p2p [16].

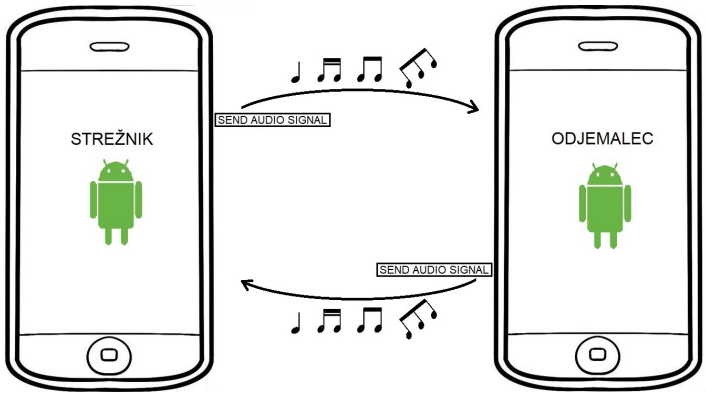
Opisano iskanje se razlikuje od iskanja v primeru Bluetooth povezave, saj se pri povezavi Wi-Fi iskanje naprav prekine avtomatsko, medtem ko mora v primeru Bluetooth povezave iskanje izrecno (ročno) prekiniti programer.

Za prikaz naprav, ki so na voljo za vzpostavitev povezave, sem uporabil vmesnik WifiP2pManager.PeerListListener, ki zagotavlja informacije o vrstnikih, ki so bili odkriti v omrežju Wi-Fi P2P [12]. Z uporabo tega vmesnika in z informacijo o napravah, ki sem jo dobil, sem izpolnil »listView-a« v aktivnosti »WiFi Connections«.

Orodje »listView-a« vsebuje seznam razpoložljivih naprav za vzpostavitev povezave. S pritiskom na ime naprave se sproži poskus vzpostavitve pove-

zave z izbrano napravo, za kar sem uporabil objekt tipa »WifiP2pConfig«, v katerega sem kopiral podatke o napravi preko uporabe objekta »WifiP2pDevice«. Nato sem klical funkcijo »connect()«.

Gumb »SEND AUDIO SIGNAL« omogoča pošiljanje podatkovnega toka od ene naprave k drugi. Razlika med tehnologijo Bluetooth in Wi-Fi Direct je v tem, da ko se naprave povežejo preko Bluetooth-a, je ena od njih povezana kot strežnik, druga pa kot odjemalec, podatkovni tok pa bo vedno potoval v smeri od strežnika k odjemalcu, kot kaže Slika 3.10. Pri povezavi Wi-Fi Direct je ena naprava prav tako vedno povezana kot strežnik, druga pa kot odjemalec, vendar pa lahko podatkovni tok potuje od ene naprave do druge v obeh smereh, smer pa je odvisna od tega, kdo pritisne gumb »SEND AUDIO SIGNAL«, kot prikazuje Slika 4.2.



Slika 4.2: Smer podatkovnega toka v Wi-Fi Direct povezavi

Ko se napravi povežeta z uporabo tehnologije Wi-Fi Direct, ustvarita skupino, v katero se lahko povežejo tudi druge naprave. V skupini mora biti vedno ena naprava lastnica skupine. Pri določanju, katera naprava naj bo lastnica skupine, Wi-Fi Direct preveri možnosti upravljanja, uporabniškega vmesnika in storitev vsake naprave in s temi informacijami izbere napravo, ki lahko učinkovito upravlja strežniške odgovornosti [12].

Ena izmed razlik med povezavama preko tehnologije Bluetooth in Wi-Fi

Direct je v določitvi, katera naprava je povezana kot strežnik in katera kot odjemalec. Pri povezavi Bluetooth to določi program, saj naprava, ki se želi povezati z drugo napravo, avtomatično deluje kot odjemalec. Pri povezavi Wi-Fi Direct pa odločitev o tem, kdo je strežnik ali lastnik skupine ter kdo odjamalec, sprejmeta napravi.

4.3 Predvajanje zvoka

Ko se napravi povežeta med seboj, vsaka ustvari objekt tipa »AudioTrack«, ki bo omogočal predvajanje zvoka ter bo v stanju čakanja na vhodni podatkovni tok. Enako kot pri povezavi Bluetooth, objekt »AudioTrack« predvaja zvok, ko pridobi določeno količino bajtov, v nasprotnem primeru pa ne predvaja ničesar.

V napravi, v kateri uporabnik pritisne na gumb “SEND AUDIO SIGNAL”, se začne izvajati funkcija, ki spremeni zvočno datoteko, ki se trenutno predvaja, v tok byte-ov ter jih začne pošiljati drugi napravi. Naprava, ki sprejme podatkovni tok, naloži vhodne bajte v svoj objekt tipa »AudioTrack« ter, ko je njihova količina zadostna, predvaja zvok.

Poglavje 5

Funkcionalnost Trace

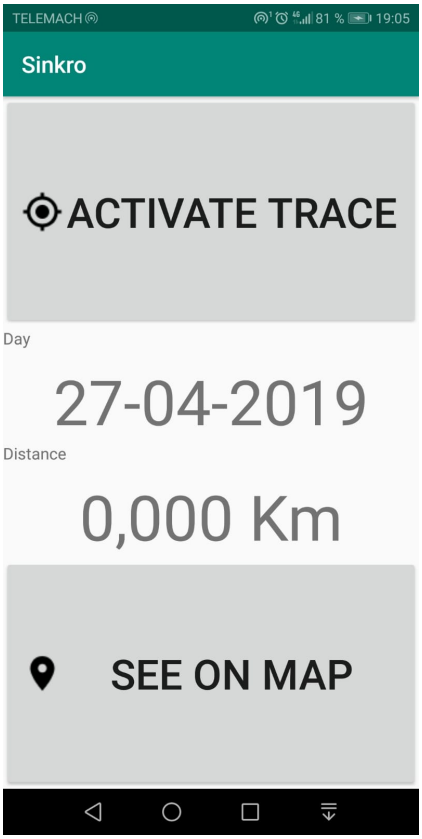
Namen aktivnosti »Trace« je sledenje napravi preko sistema GPS. Uporabniku omogoča spremljanje, kakšno pot in razdaljo je naredil oziroma pretekel.

5.1 Dovoljenja

Za svoje delovanje aktivnost »Trace« potrebuje dve dovoljenji do dostopa geolokacije naprave, in sicer:

- ACCESS_COARSE_LOCATION: To dovoljenje aplikaciji omogoča dostop do približne lokacije naprave [11].
- ACCESS_FINE_LOCATION: To dovoljenje aplikaciji omogoča dostop do natančne lokacije naprave [11].

Ko sta dovoljenji podani, lahko dostopamo do aktivnosti, kot prikazuje Slika 5.1.

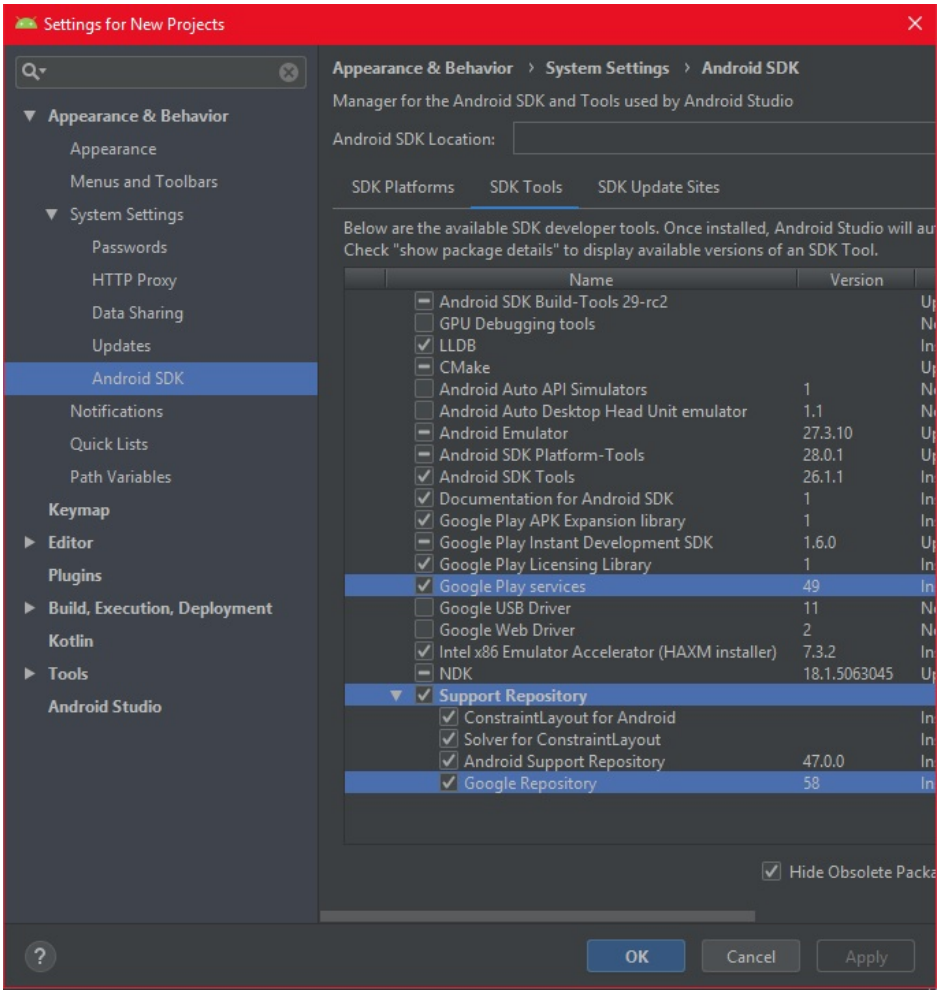


Slika 5.1: Aktivnost »Trace«

5.2 Pridobivanje storitve Google Maps

Pri programiranju te aktivnosti sem sledil korakom, ki jih svetuje spletna stran Googla za razvijalce: `https://developers.google.com/maps/documentation/android-sdk/start`.

Najprej sem moral preveriti, če imam v Android Studiu naloženo vse, kar se zahteva za uporabo Google API-jev. Kot je vidno na Sliki 5.2, sem preveril, če imam naložene Google Play storitve ter »Support repository«.



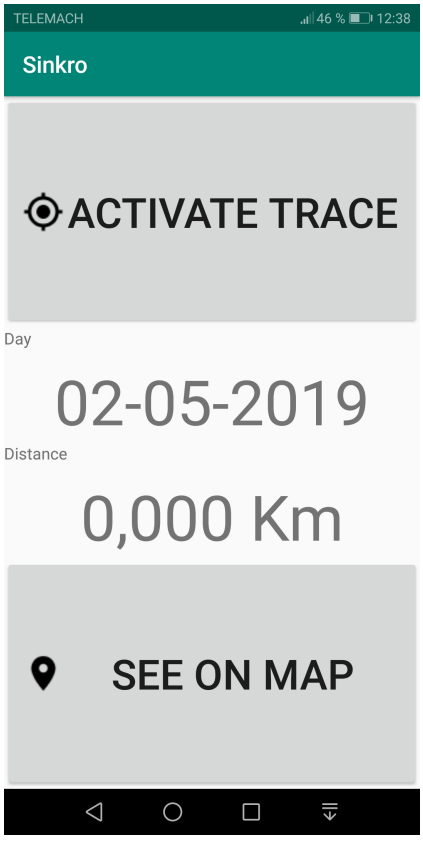
Slika 5.2: Naloženi SDK

V naslednjem koraku sem moral v svojo aplikacijo dodati Google Play storitve, kar sem naredil tako, da sem v datoteki »build.gradle« (v delu »dependencies«) dodal naslednjo vrstico:

```
dependencies {
    ...
    implementation 'com.google.android.gms:play-services-maps:16.0.0'
    ...
}
```

Slika 5.3: Naloženi SDK

Od verzije Android v1.0 dalje Google za uporabo storitve »Google Maps« zahteva, da vsak razvijalec dobi svoj API ključ, s čimer lahko nadzoruje uporabo te storitve [17]. Ključ se dobi na spletni strani: `https://console.developers.google.com`. Na navedeni spletni strani sem najprej ustvaril nov projekt, nato pa sem izbral API, ki sem ga potreboval, v mojem primeru je bil to »Maps SDK for Android«. Nato sem ga aktiviral in zahteval svoj API ključ za aplikacijo. Po tem sem v aplikaciji ustvaril novo aktivnost, tokrat tipa »Google Map Activity«, v kateri bi bila prikazana pot, ki jo je pretekel uporabnik. V datoteko tipa ».xml« navedene aktivnosti sem kopiral svoj API ključ, s tem pa je bila aplikacija pripravljena za uporabo Google Maps API.



Slika 5.4: Aktivnost Trace, sledenje ni aktivirano

5.3 Sledenje naprave

Kot prikazuje Slika 5.4, se sledenje naprave aktivira s pritiskom na gumb »ACTIVATE TRACE«. Po pritisku na gumb se v programu sproži nit, ki shrani geolokacijo naprave. Nit, ki sem jo poimenoval »PositionThread«, uporablja objekt tipa »LocationManager«, ki omogoča dostop do storitev lokacije naprave. Te storitve aplikaciji omogočajo redno posodabljanje geografske lokacije naprave [18].

V objektu tipa »LocationManager« sem uporabil funkcijo ».requestLocationUpdates(...)«, v kateri sem moral določiti štiri pomembne parametre, to so:

- Ponudnik storitve (»Provider«): Uporabil sem »GPS_PROVIDER«, ki določa lokacijo preko uporabe satelitov [18].

  Obstaja tudi »NETWORK_PROVIDER«, ki določi lokacijo naprave na podlagi celic mobilnega omrežja in dostopnih točk WiFi, vendar v mojem primeru uporaba tega izvora ni bila primerna, saj ni pravilno upoštevala poti, ki jo je naredil uporabnik. Prav tako obstaja tudi možnost, da se definirata oba navedena izvora, program pa se sam odloči, katerega bo uporabljal.

  Odloči se na podlagi tega, kateri izvor je hitreje na voljo. V moji aplikaciji takšen način uporabe izvorov ni bil ustrezen, ker je bil »NETWORK_PROVIDER« vedno hitreje na voljo kot »GPS_PROVIDER«, zato se je pot izrisala od hiše do hiše, namesto da bi potekala po dejanski poti.

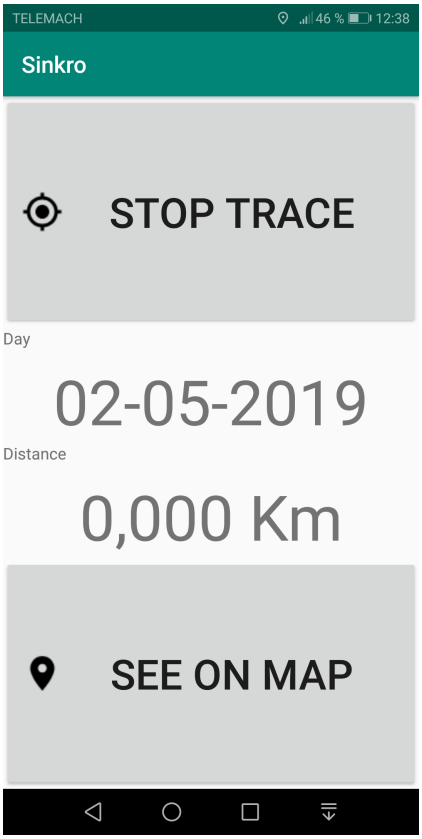
- »minTime« ali minimalni čas: Ta parameter določa minimalni časovni interval med posodobitvami lokacije v milisekundah [18]. Ko sem pri testiranju pridobivanja geolokacije parameter določil na 0 sekund, so se posodobitve izvajale ves čas, naprava pa je, čeprav se ni premikala, prikazovala več napačnih lokacij. Da sem preprečil to napako, sem v aplikaciji določil interval 5 sekund.

- »minDistance« ali minimalna razdalja: Ta parameter določa minimalno razdaljo med posodobitvami lokacij v metrih [18]. Tudi tu sem pri testiranju parameter določil na 0 metrov, pri čemer je naprava, kljub mirovanju, prikazovala več napačnih lokacij. Napako sem odpravil s tem, da sem v aplikaciji določil parameter na razdaljo 20 metrov.

- »LocationListener«: Je metoda in ne parameter, ki je klicana za vsako posodobitev lokacij [18]. V njej obstajajo štiri funkcije, od katerih sem uporabil samo eno, in sicer »onLocationChanged«. Navedena funkcija je klicana, ko se geolokacija naprave spremeni.

Nit »PositionThread« deluje na naslednji način. Ko se sledenje aktivira, kot prikazuje Slika 5.5, in se ta nit sproži, začne shranjevati vse posodobljene lokacije, kjer se naprava nahaja, in sicer tako, da metoda »LocationListener« zazna posodobitev lokacije, ta pa se shrani v seznam »locationList«. Seznam vsebuje spremenljivke tipa »LatLng«, ki je nespremenljiv razred in predstavlja par koordinat zemljepisne širine in zemljepisne dolžine shranjene v stopinjah [19]. Delovanje niti »PositionThread« se ponavlja, dokler se sledenje ne deaktivira.

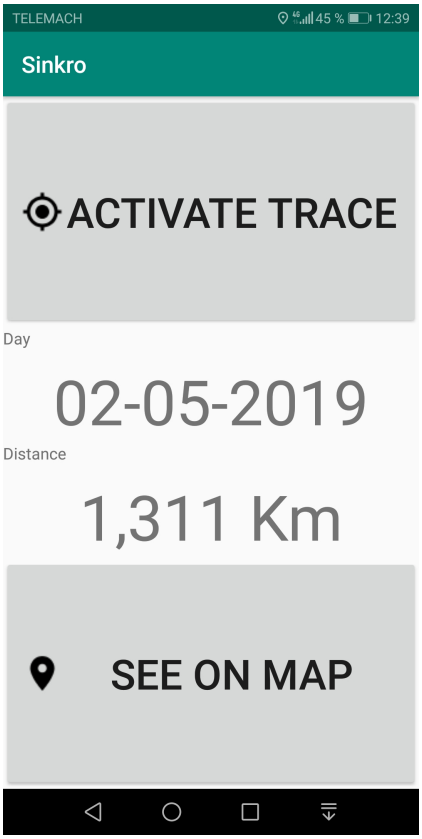
Ko se sledenje deaktivira, se kličeta še dve funciji. Prva funkcija, imenovana »DistanceCount«, izračuna razdaljo med začetno in končno točko poti, druga funkcija, imenovana »SaveDistance«, pa shrani to razdaljo. Tekač oziroma uporabnik aplikacije lahko na ta način pregleda, kdaj in kakšne razdalje je pretekel.



Slika 5.5: Aktivnost Trace, sledenje je aktivirano

Delovanje funkcije »DistanceCount« je preprosto in poteka na način, da sešteje razdalje med vsemi točkami v seznamu »locationList« in prikaže rezultat na zaslonu, kot prikazuje Slika 5.6. Razdaljo med dvema različnima geolokacijama se lahko izračuna z uporabo metode »Location.distanceBetween(...)«, ki kot parametre dobi zemljepisno širino in zemljepisno dolžino dveh geolokacij ter vrne razdaljo med njima.

Delovanje funkcije »SaveDistance« pa bom obrazložil v naslednjem poglavju.



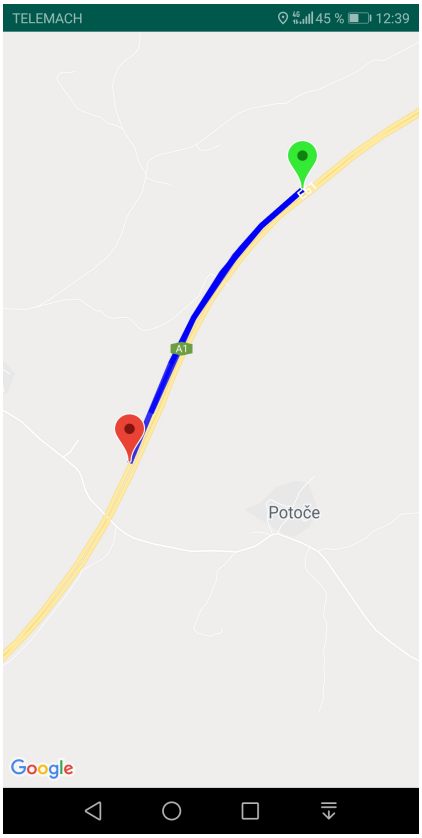
Slika 5.6: Aktivnost Trace, konec sledenja

5.4 Funkcionalnost SEE ON MAP

Aktivnost »SEE ON MAP« (poglej na zemljevidu) omogoča vizualizacijo geolokacije naprave na zemljevidu. Po pritisku na gumb »SEE ON MAP«, pri čemer moramo imeti vzpostavljeno povezavo z internetom, se nam prikaže zemljevid ter na njem kazalec lokacije, ki predstavlja trenutno geolokacijo naprave. Ko se ta aktivnost odpre, se v primeru, ko je sledenje aktivirano, iz aktivnosti »Trace« v »SEE ON MAP« prenese seznam lokacij naprave. Delovanje aktivnosti »SEE ON MAP« je zelo podobno aktivnosti »Trace«, ko je v njej aktivirano sledenje. Za sledenje napravi uporablja metodo »locationManager.requestLocationUpdates(...)« z istimi parametri kot aktivnost »Trace«, z razliko, da ob posodobitvi lokacije, nove lokacije doda v seznam,

hkrati pa jih pokaže tudi na zemljevidu.

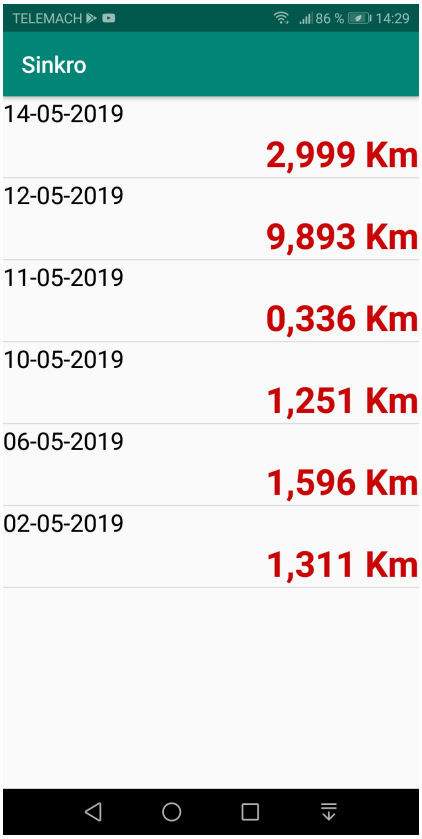
Slika 5.7 prikazuje primer izrisane poti na zemljevidu. Risanje poti sem določil tako, da se po vsaki posodobitvi lokacije naprave pot izriše znova od prve do zadnje lokacije, pri čemer je prva točka označena z zelenim označevalcem, zadnja točka pa z rdečim označevalcem.



Slika 5.7: Aktivnost »SEE ON MAP«

Poglavje 6

Funkcionalnost History



Slika 6.1: Aktivnost »History«

Zadnja aktivnost, ki jo ponuja aplikacija, je »HISTORY«. V njej se beležijo dnevne aktivnosti uporabnika, in sicer kakšno razdaljo je pretekel oziroma naredil na posamezni dan.

Za ustvarjanje podatkovne baze Android uporablja sistem SQLite. Podatkovne baze, ki so ustvarjene v posamezni aplikaciji, so dostopne zgolj tej aplikaciji, kar pomeni, da druge aplikacije ne morejo dostopati do teh podatkov [17].

Pri programiranju te aktivnosti sem uporabljal podatkovno bazo SQLite. Oblika podatkovne baze ni bila zahtevna, saj vsebuje samo eno tabelo, imenovano »distances«, ki vsebuje dva stolpa:

- DAY
- DISTANCE

Prvi korak pri kreiranju podatkovne baze je bil definiranje »contract class«. Ta razred sem ustvaril z namenom, da vsebuje konstante, ki definirajo imena za URI-je, tabele in stolpe [20]. V mojem primeru definiranje tega razreda izgleda tako:

```
public final class HistoryContract{

    private HistoryContract(){}

    public class HistoryEntry implements BaseColumns {
        public static final String TABLE_NAME = "distances";
        public static final String COLUMN_NAME_TITLE_DAY = "day";
        public static final String COLUMN_NAME_TITLE_DISTANCE = "distance";
    }
}
```

Slika 6.2: Contract class

Za tem sem moral definirati metodo, ki ustvari tabelo »distances«. Metoda, ki sem jo uporabil, je prikazana spodaj.

Naslednji korak je bil definiranje »SQLiteOpenHelper«. To je razred, ki vsebuje nabor uporabnih API-jev za upravljanje podatkovne baze. Za

```
private static final String SQL_CREATE_ENTRIES = "CREATE TABLE " +
        HistoryContract.HistoryEntry.TABLE_NAME +
        "(" + HistoryContract.HistoryEntry._ID + " INTEGER PRIMARY KEY," +
        HistoryContract.HistoryEntry.COLUMN_NAME_TITLE_DAY + " TEXT," +
        HistoryContract.HistoryEntry.COLUMN_NAME_TITLE_DISTANCE + " TEXT)";
```

Slika 6.3: Metoda za kreiranje tabele

uporabo tega razreda sem moral kreirati nov razred, imenovan »HistoryDbHelper«, ki preglasi(overide) metode »onCreate« in »onUpgrade« [20]. Metoda »onCreate« je klicana, ko je podatkovna baza kreirana. Metoda »onUpgrade« pa se uporabi, ko se izvaja kakšna posodobitev v podatkovni bazi.

Da sem omogočil dostop do podatkovne baze, sem moral inicializirati razred »HistoryDbHelper«, kar sem naredil na naslednji način:

```
History.HistoryDbHelper dbHelper;
dbHelper = new HistoryDbHelper(getApplicationContext());
```

Slika 6.4: Razred DbHelper

Podatki, ki se shranijo v podatkovni bazi, so datumi in razdalje, ki jih je uporabnik naredil na posamezni dan. Ti podatki se dobijo v aktivnosti »Trace«, ko se opcija sledenja deaktivira. Funkcija, ki omogoča shranjevanje teh podatkov, se imenuje »SaveDistance«. Prvi korak, ki sem ga moral narediti v tej funkciji, je bil inicializirati spremenljivko tipa »HistoryDbHelper«, s čimer sem lahko upravljal s podatkovno bazo. Za tem sem inicializiral tudi spremenljivko tipa »SQLiteDatabase«, preko katere sem z uporabo metode »dbHelper.getWritableDatabase« lahko kreiral ali dostopal do podatkovne baze. S tem sem lahko iz podatkovne baze bral podatke ali jih pisal vanjo [21].

Da se podatki shranijo v podatkovni bazi, jih je treba najprej preurediti v objekt tipa »ContentValues«, s čimer omogočimo, da so podatki urejeni v formatu, ki ga podatkovna baza lahko obdeluje. To sem naredil na naslednji način:

```
ContentValues values = new ContentValues();
values.put(History.HistoryContract.HistoryEntry.COLUMN_NAME_TITLE_DAY, formattedDate);
values.put(History.HistoryContract.HistoryEntry.COLUMN_NAME_TITLE_DISTANCE, distance);
```

Slika 6.5: Objekt za urejanje podatkov

Datum sem pridobil tako, da sem definiral spremenljivko tipa »Date«, v kateri sem preko uporabe metode »Calendar.getInstance().getTime()«, dobil trenutni datum. Za tem sem definiral format, v katerem bo ta datum prikazan, in sicer z uporabo metode »SimpleFormat«. Spremenljivka »formatedDate« pa je končni datum, zapisan v formatu (dd-MM-yyyy), ki se mi je zdel najbolj primeren za prikaz tega podatka. Opisane korake prikazuje spodnja koda.

```
Date date = Calendar.getInstance().getTime();
SimpleDateFormat df = new SimpleDateFormat( pattern: "dd-MM-yyyy");
String formattedDate = df.format(date);
```

Slika 6.6: Pridobivanje datuma in definiranje formata

Za vnos podatkov v podatkovno bazo sem uporabljal »insert« metodo.

```
long newRowId =
db.insert(History.HistoryContract.HistoryEntry.TABLE_NAME, nullColumnHack: null,values);
```

Slika 6.7: Uporaba insert metode

Prvi parameter te metode določi tabelo, v kateri se bo shranil podatek v podatkovni bazi. Drugi paramater definira, kaj naj aplikacija naredi v primeru, ko je objekt tipa »ContentValues« prazen. Ker se v mojem pri-

meru to nikoli ne zgodi, sem ta parameter določil na vrednost »null«. Tretji parameter pa so vrednosti, ki jih želim dodati v podatkovno bazo [20].

V zadnjem koraku sem moral omogočiti branje podatkov iz podatkovne baze. To se zgodi v aktivnosti »History«. Najprej je bilo treba omogočiti dostop do podatkovne baze, kar sem naredil s kreiranjem spremenljivke tipa »SQLiteDatabase«.

```
SQLiteDatabase db = dbHelper.getWritableDatabase();
```

Slika 6.8: Omogočanje dostopa do podatkovne baze

Nato sem moral definirati, katere stolpe iz tabele želim brati (pri čemer sem potreboval vse tri stolpe, ki so v tabeli), kar sem naredil z naslednjim delom kode:

```
String[] projection = {
        BaseColumns._ID,
        HistoryContract.HistoryEntry.COLUMN_NAME_TITLE_DAY,
        HistoryContract.HistoryEntry.COLUMN_NAME_TITLE_DISTANCE
};
```

Slika 6.9: Izbira stolpov

Izhodne podatke sem uredil na podlagi stolpa »ID«, in sicer tako, da so najnovejši podatki prikazani na začetku seznama.

```
String sortOrder = HistoryContract.HistoryEntry._ID + " DESC";
```

Slika 6.10: Urejanje podatkov

Ko sem določil, katere stolpe iz tabele želim brati in v kakšnem vrstnem redu, sem moral definirati poizvedbo za te podatke. To sem naredil na naslednji način:

Poizvedba ali »query« vrne objekt tipa »Cursor«, ki omogoča naključni dostop za branje in pisanje do niza teh rezultatov [22]. Ob odpiranju te

```
Cursor cursor = db.query(
        HistoryContract.HistoryEntry.TABLE_NAME,
        projection,
         selection: null,
         selectionArgs: null,
         groupBy: null,
         having: null,
        sortOrder
);
```

Slika 6.11: Poizvedba podatkov

aktivnosti pa preko uporabe »while« zanke, program iterira z objektom tipa »Cursor« ter podatke pokaže v seznamu preko uporabe orodja »list view«.

```
while (cursor.moveToNext()){
    items.add(cursor.getString( columnIndex: 1));
    subItems.add(cursor.getString( columnIndex: 2));
}
```

Slika 6.12: »While« zanka za prikaz podatkov

Poglavje 7

Sklep

Cilj te diplomske naloge je bil ustvariti mobilno aplikacijo namenjeno predvsem osebam, ki tečejo oziroma trenirajo v skupini. Aplikacija omogoča, da si lahko dve osebi medsebojno delita glasbo in tako istočasno poslušata isto skladbo, hkrati pa uporabniku omogoča tudi beleženje vsakodnevne fizične aktivnosti.

Ustvarjanja te aplikacije sem se lotil na način, da sem si delo razdelil glede na posamezne funkcionalnosti (aktivnosti), ki jih ponuja aplikacija, to so:

- "Music Library"
- "Bluetooth"
- "Wi-Fi Direct"
- "Trace"
- "History"

Kot prvo sem programiral aktivnost »Music Library«, ki uporabniku omogoča predvajanje datotek tipa ».mp3«, ki jih vsebuje mobilna naprava.

Nadalje sem se posvetil tehnologiji Bluetooth, pri čemer sem moral najprej ugotoviti, kako deluje, kaj potrebuje, kateri tipi podatkov se lahko

pošiljajo med različnimi napravami in v kakšnem formatu. Ko sem se seznanil z vsem navedenim, sem začel s programiranjem te aktivnosti, pri čemer sem definiral delovanje niti, ki se nahajajo v aktivnosti, ter povezavo med njimi.

Naslednja aktivnost je bila Wi-Fi Direct. Pri njenem programiranju sem v večini sledil istim korakom, kot pri uporabi tehnologije Bluetooth, z nekaterimi razlikami, ki sem jih opisal v diplomski nalogi.

Pri programiranju aktivnosti "Trace" sem moral najprej raziskati uporabo Google API-jev, in sicer, kako jih pridobim ter uporabim. Ko sem pridobil potrebno API, sem moral ugotoviti, kako jo uporabiti v programu in na ta način pridobiti informacije glede razdalj in poti, ki jih je naredil uporabnik.

Zadnja aktivnost, ki jo ponuja aplikacija, je "History". Najprej sem moral presoditi, katere podatke sem želel prikazati, pri čemer sem se odločil za prikaz razdalje, ki jo je naredil uporabnik, ter datuma, ko je bila le ta narejena. Na tej podlagi sem oblikoval podatkovno bazo. Nadalje sem moral ugotoviti, kakšen sistem podatkovne baze uporablja operacijski sistem Android, in sicer je to SQLite, nato pa sem začel s programiranjem podatkovne baze.

Menim, da bi svojo aplikacijo lahko izboljšal na način, da bi za predvajanje glasbe namesto uporabe objekta tipa »Media Player« uporabil zgolj objekte tipa »AudioTrack«. Ugotovil sem namreč, da je pri starejših napravah, ki imajo manj zmogljive procesorje, kot današnje novejše mobilne naprave, sprememba iz enega v drugi tip objekta bolj zahtevna, kar posledično povzroči, da je v zvoku več motenj.

Programiranje te aplikacije je bil zame nov izziv, pri katerem pa sem se

veliko naučil. Večkrat sem se želel lotiti takšnega projekta in ustvariti svojo aplikacijo, vendar do sedaj takšne priložnosti še nisem imel. Vedno me je zanimalo programiranje mobilnih aplikacij, predvsem glede na to, česa vsega so sposobne današnje moderne mobilne naprave in kaj vse lahko ustvarjamo z njimi. Že na začetku sem se odločil za aplikacijo, ki temelji na operacijskem sistemu Android, saj je le ta za programiranje bolj »dostopen«, v prihodnje pa se želim naučiti in ustvariti kakšno aplikacijo tudi za sistem IOS.

Literatura

[1] STACK OVERFLOW. How to search for media files in folders and subfolders? Dosegljivo: https://stackoverflow.com/questions/20068606/how-to-search-for-media-files-in-folders-and-subfolders. [Dostopano 27. 5. 2019].

[2] Google Developers. MediaPlayer overview. Dosegljivo: https://developer.android.com/guide/topics/media/mediaplayer. [Dostopano 26. 5. 2019].

[3] Google Developers. MediaPlayer. Dosegljivo: https://developer.android.com/reference/android/media/MediaPlayer.html. [Dostopano 26. 5. 2019].

[4] Google Developers. Runnable. Dosegljivo: https://developer.android.com/reference/java/lang/Runnable. [Dostopano 26. 5. 2019].

[5] Digital Learning SL. Multitarea en Android con clases AsyncTask, Thread, Handler y Runnable. Dosegljivo: https://academiaandroid.com/multitarea-android-clases-asynctask-thread-handler-runnable/. [Dostopano 26. 5. 2019].

[6] Google Developers. Handler. Dosegljivo: https://developer.android.com/reference/android/os/Handler#postDelayed(java.lang.Runnable,%20long). [Dostopano 26. 5. 2019].

[7] Google Developers. Bluetooth overview. Dosegljivo: https://developer.android.com/guide/topics/connectivity/bluetooth. [Dostopano 26. 5. 2019].

[8] Google Developers. BluetoothAdapter. Dosegljivo: https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html. [Dostopano 27. 5. 2019].

[9] Google Developers. AudioTrack. Dosegljivo: https://developer.android.com/reference/android/media/AudioTrack. [Dostopano 27. 5. 2019].

[10] Help Scout. Why does the app require location permission for wifi signal when other apps don't? Dosegljivo: https://support.netanalyzer-an.techet.net/article/124-why-does-the-app-require-location-permission-for-wifi-signal-when-other-apps-dont. [Dostopano 27. 5. 2019].

[11] Google Developers. Manifest.permission. Dosegljivo: https://developer.android.com/reference/android/Manifest.permission. [Dostopano 27. 5. 2019].

[12] Google Developers. Create P2P connections with Wi-Fi. Dosegljivo: https://developer.android.com/training/connect-devices-wirelessly/wifi-direct#java. [Dostopano 27. 5. 2019].

[13] Google Developers. WifiManager. Dosegljivo: https://developer.android.com/reference/android/net/wifi/WifiManager. [Dostopano 27. 5. 2019].

[14] Google Developers. WifiP2pManager. Dosegljivo: https://developer.android.com/reference/android/net/wifi/p2p/WifiP2pManager.html. [Dostopano 27. 5. 2019].

[15] Google Developers. WifiP2pManager.Channel. Dosegljivo: https://developer.android.com/reference/android/net/wifi/p2p/WifiP2pManager.Channel.html. [Dostopano 27. 5. 2019].

[16] Google Developers. WifiP2pManager. Dosegljivo: https://developer.android.com/reference/android/net/wifi/p2p/WifiP2pManager.html#discoverPeers(android.net.wifi.p2p.WifiP2pManager.Channel,%2520android.net.wifi.p2p.WifiP2pManager.ActionListener). [Dostopano 27. 5. 2019].

[17] Wei Meng Lee. *Beginning Android application development*. Indianapolis (IN) : Wiley, cop. 2011, 2011.

[18] Google Developers. LocationManager. Dosegljivo: https://developer.android.com/reference/kotlin/android/location/LocationManager. [Dostopano 27. 5. 2019].

[19] Google Developers. LatLng. Dosegljivo: https://developers.google.com/android/reference/com/google/android/gms/maps/model/LatLng. [Dostopano 27. 5. 2019].

[20] Google Developers. Save data using SQLite. Dosegljivo: https://developer.android.com/training/data-storage/sqlite. [Dostopano 27. 5. 2019].

[21] Google Developers. SQLiteOpenHelper. Dosegljivo: https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html. [Dostopano 27. 5. 2019].

[22] Google Developers. Cursor. Dosegljivo: https://developer.android.com/reference/android/database/Cursor.html. [Dostopano 27. 5. 2019].