SVEUČILIŠTE U SPLITU  
FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Android aplikacija za prikaz snimljenih GPS koordinata tijekom izvođenja fizičkih aktivnosti

Ante Lovrić

Split, Kolovoz 2019.

SVEUČILIŠTE U SPLITU

FAKULTET ELEKTROTEHNIKE, STROJARSTVA I

BRODOGRADNJE

Preddiplomski sveučilišni studij:**Računarstvo**

Smjer/Usmjerenje:**/**

Oznaka programa:120

Akademska godina:2018./2019.

Ime i prezime:**Ante Lovrić**

Broj indeksa: 206-2016

**ZADATAK ZAVRŠNOG RADA**

Naslov: **Android aplikacija za prikaz snimljenih GPS koordinata tijekom izvođenja fizičkih aktivnosti**

Zadatak:Kreirati Android aplikaciju namijenjenu dinamičkom praćenju lokacije korisnika. Aplikacijom osigurati mogućnost prikazivanja odgovarajućih parametara, tijekom zabilježenih slijednih vremenskih trenutaka, pogodnih za dokumentiranje fizičkih aktivnosti korisnika. Objasniti pristup u realizaciji funkcionalnosti aplikacije, opisati se Google Maps loalizacijski servis, postupak prikupljanja i obrade podataka o lokaciji, stvaranje i korištenje baze podataka, te opisati kako dinamički stvoriti i prikazati sadržaj na zaslonu uređaja.

Datum obrane:23.09.2019.

Mentor:



prof. dr. sc. Mirjana Bonković

SVEUČILIŠTE U SPLITU

FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

Ime i prezime: **ANTE LOVRIĆ**

Broj indeksa: 206-2016

**IZJAVA O AUTORSTVU RADA I O JAVNOJ OBJAVI RADA**

Izjavljujem da sam diplomski rad / završni rad pod nazivom

**Android aplikacija za prikaz snimljenih GPS koordinata tijekom izvođenja fizičkih aktivnosti**

izradila/o samostalno te sam suglasna/suglasan o javnoj objavi rada u elektroničkom obliku.

Svi dijelovi rada, nalazi ili ideje koje su u radu citirane ili se temelje na drugim izvorima, bilo da su u pitanju knjige, znanstveni ili stručni članci, Internet stranice, zakoni i sl. u radu su jasno označeni kao takvi te adekvatno navedeni u popisu literature.

U Splitu, 16.9.19.



Potpis studenta

**SADRŽAJ**

[1 Uvod 3](#_Toc19564775)

[2 Povijest androida i izbor programskog jezika 4](#_Toc19564776)

[2.1 Kotlin 4](#_Toc19564777)

[2.2 Java 5](#_Toc19564778)

[3 Korištene tehnologije 6](#_Toc19564779)

[3.1 Android Studio 6](#_Toc19564780)

[3.2 Git 8](#_Toc19564781)

[4 Android aplikacija 9](#_Toc19564782)

[4.1 Koncept 9](#_Toc19564783)

[4.2 Podaci o aktivnostima 9](#_Toc19564784)

[4.3 Baza podataka 11](#_Toc19564785)

[4.3.1 Biblioteka 11](#_Toc19564786)

[4.3.2 Definiranje i kreiranje tablica 11](#_Toc19564787)

[4.3.3 Unos podataka 13](#_Toc19564788)

[4.3.4 Dohvaćanje podataka 13](#_Toc19564789)

[4.4 Android aktivnosti 14](#_Toc19564790)

[4.4.1 Aktivnost 14](#_Toc19564791)

[4.4.2 Životni ciklus aktivnosti 15](#_Toc19564792)

[4.4.3 Prijelazi između aktivnosti 16](#_Toc19564793)

[4.5 Aktivnost za praćenje lokacije 16](#_Toc19564794)

[4.5.1 Dizajn i XML dokumenti 16](#_Toc19564795)

[4.5.2 onCreate – Postavljanje layout-a 18](#_Toc19564796)

[4.5.3 onCreate – Postavljanje lokacijskog servisa 19](#_Toc19564797)

[4.5.4 onCreate – Postavljanje referenci na „View“-ove i „onClickListener“ 19](#_Toc19564798)

[4.5.5 Sekvenca za pokretanje lokacijskog servisa 20](#_Toc19564799)

[4.5.6 Praćenje proteklog vremena na ekranu 23](#_Toc19564800)

[4.5.7 Povratak u glavnu aktivnost 23](#_Toc19564801)

[4.5.8 „RouteDisplay“ aktivnost 24](#_Toc19564802)

[4.6 Glavna aktivnost 24](#_Toc19564803)

[4.6.1 Dizajn i XML dokumenti 24](#_Toc19564804)

[4.6.2 Dinamičko dodavanje elemenata na ekran 27](#_Toc19564805)

[4.6.3 „onCreate“ metoda 28](#_Toc19564806)

[4.6.4 Pokretanje i dohvat rezultata aktivnosti za praćenje lokacije 28](#_Toc19564807)

[4.6.5 „processActivityData“ metoda 29](#_Toc19564808)

[4.6.6 „ActivityCardCreator“ klasa 29](#_Toc19564809)

[4.7 Funkcionalnost aplikacije 32](#_Toc19564810)

[5 Gotova rješenja i biblioteke 35](#_Toc19564811)

[5.1 MPAndroidChart biblioteka 35](#_Toc19564812)

[5.2 FloatingActionButton biblioteka 35](#_Toc19564813)

[5.3 Vodič za pokretanje Google Maps lokacijskog servisa 35](#_Toc19564814)

[6 Zaključak 36](#_Toc19564815)

[Literatura 37](#_Toc19564816)

[Popis oznaka i kratica 38](#_Toc19564817)

[Sažetak i ključne riječi 39](#_Toc19564818)

# Uvod

U ovom završnom radu je opisan postupak izrade Android aplikacije za praćenje lokacije korisnika u stvarnom vremenu, načina dohvaćanja i spremanja podataka o lokaciji i kretanju korisnika, te načina obrade i prikaza dohvaćenih podataka na ekranu uređaja, napisane u Java programskom jeziku.

U poglavlju „Povijest androida i izbor programskog jezika“ mogu se pronaći osnovne informacije o Android OS-u, kako se aplikacije na Androidu pokreću, te koji se jezici mogu koristiti za razvoj aplikacija na Android platformi.

Poglavlje „Korištene tehnologije“ sadržava informacije o alatima „Git“ i „Android Studio“ korištenim pri izradi praktičnog dijela završnog rada i korištene funkcionalnosti tih alata.

U poglavlju „Android aplikacija“ je objašnjen svaki pojedini dio praktičnog dijela završnog rada, od baze podataka koja se koristi u aplikaciji do svake android aktivnosti aplikacije. Objašnjenja su popraćena isječcima koda, a na početku svakog podpoglavlja se nalaze osnovne informacije o načinu na koji funkcionira Android potrebne za razumijevanje isječka koda koji slijedi.

U poglavlju „Gotova rješenja i biblioteke“ priložene su biblioteke i gotova rješenja koja su omogućila jednostavnu implementaciju funkcionalnosti aplikacije poput crtanja grafova i animiranih gumbova.

Na kraju dokumenta mogu se pronaći literatura i popis oznaka i kratica korištenih u ovom dokumentu.

# Povijest androida i izbor programskog jezika

Android je besplatni operacijski sustav napravljen na Linux jezgri za mobilne uređaje s velikim ekranima, te raznim mogućnostima kao što je određivanje lokacije uređaja, tipkanja na dodir, spajanja na Internet i slično.

Ovaj operacijski sustav je besplatan za korištenje, te je postao najpopularniji operacijski sustav s preko 2 milijarde korisnika u svijetu od 2007. godine kada je predstavljen javnosti.

Budući da je napisan na Linux jezgri, za izradu aplikacija se može birati između velikog broja programskih jezika kao C, C++, Java, Visual Basic, Xamarin, Kotlin itd. Svaki od tih programskih jezika ima svoje dobre i loše strane. Npr. C i C++ zbog brzine trenutno rastu na popularnosti u izradi igara za Android. Od tih jezika, dva su službeno podržana od strane Google-a kao tvrtke koja je razvila Android platformu, a to su Java i Kotlin.

## Kotlin

Kotlin je programski jezik koji je razvila tvrtka JetBrains čiji je najpoznatiji proizvod IntelliJ razvojno okruženje.

Krajem 2017. godine je podržan od strane Google-a koji je izjavio da će neke mogućnosti prvo isporučivati za Kotlin, pa tek nakon za Javu. Sredinom 2019. godine je određen kao preferirani jezik za razvoj Android aplikacija.

Kotlin se koristi jer:

1. Napravljen je s ciljem smanjenja količine potrebnog koda za izradu neke funkcionalnosti. Time se i smanjuje količina koda u kojoj može nastati greška i količina koda koja se mora testirati.
2. Sam provjerava neke greške koje je potrebno ručno provjeravati u drugim programskim jezicima.
3. Podržava biblioteke napisane za Java i JavaScript jezike. S time je omogućeno pisanje aplikacija za JVM, Android i Internet preglednik.
4. Podržavaju ga sve razvojne okoline koje podržavaju Java jezik.

## Java

Java programski jezik je do sredine 2019. godine jedini službeno podržani jezik od strane Google-a za izradu Android aplikacija. Kao objektno orijentirani i najpopularniji programski jezik, postao je službenim jezikom i u njemu je pisan Android OS.

Prednosti Jave:

1. Java kod se ne mora prilagođavati ovisno o platformi na kojoj se pokreće, već to radi JVM. Java kod se pretvara u Java „byte code“ te se pokreće pomoću JVM-a.
2. Postoji velik broj ljudi stručnih u Java programskom jeziku.
3. Velika podrška od strane zajednice, uz veliku količinu biblioteka i alata besplatnih za upotrebu (Open Source).
4. Budući da nekakvo virtualno računalo pokreće Java kod, moguće je upravljati i ograničiti resurse, i tako osigurati računalo od štete koju bi mogla napraviti aplikacija neodgovornim korištenjem resursa.

Za izradu ove aplikacije koristiti će se Java programski jezik.

# Korištene tehnologije

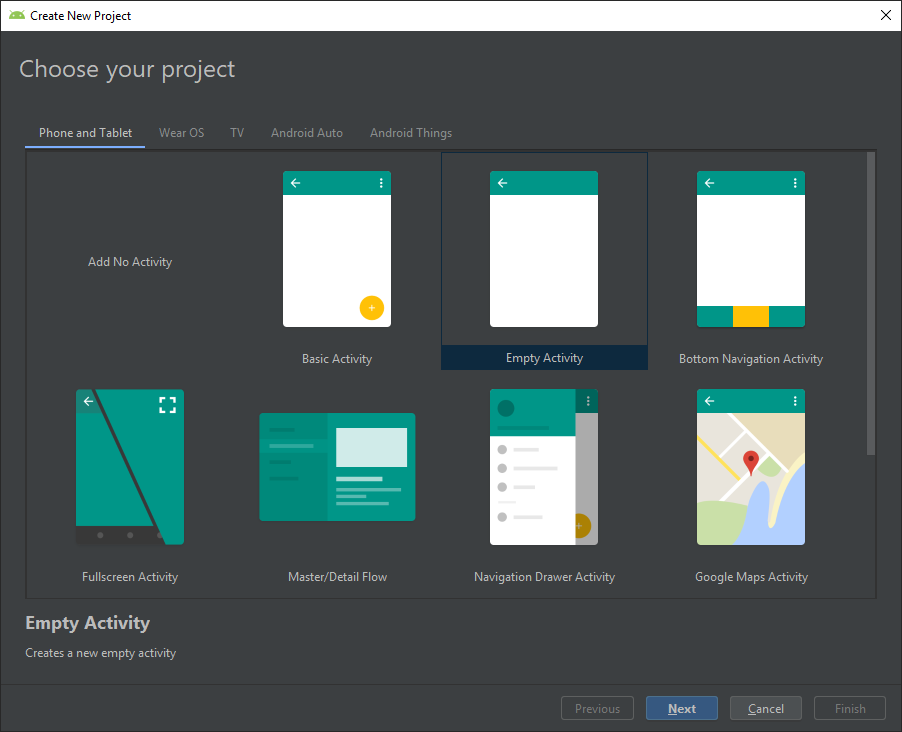
Alati i aplikacije koje su se koristile u razvoju aplikacije:

## Android Studio

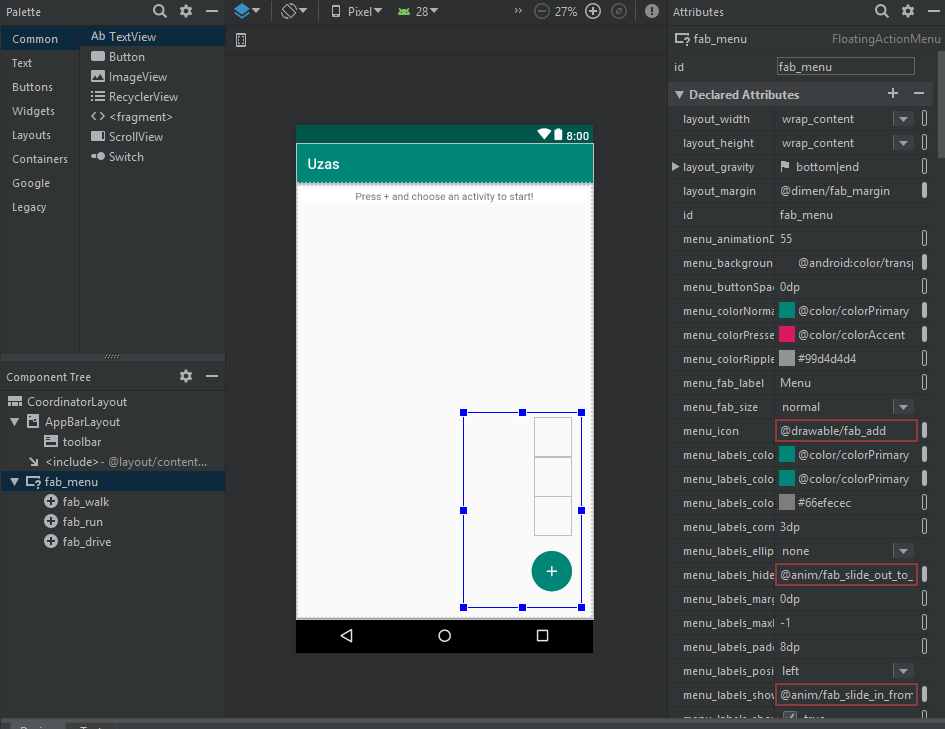
Android Studio je razvojno okruženje koju je razvio Google u svrhu lakšeg razvoja aplikacija za Android s podrškom za programske jezike Java i Kotlin.

Tekst u nastavku daje pregled nekih od mogućnosti Android Studio-a:

1. Izrada novog prikaza u aplikaciji ili pokretanje novog projekta s predloška prikaza (*Slika 3.1*) gdje ovisno o tome što želimo da aplikacija radi ili što da prikazuje možemo imati gotovi kod za osnovnu aktivnost.

***Slika 3.1 Prikaz izbornika za kreiranje novog projekta*

1. Pregled projektnih datoteka i uređivanje koda se podrazumijevaju, uz opcije automatskog oblikovanja koda, i preoblikovanja koda, automatskog preimenovanja varijabli/klasa itd.
2. Mogućnost odabira bilo kojeg elementa na ekranu, te uređivanje prikaza, dodavanje i brisanje stavki s virtualnog ekrana, te uređivanje parametara prikaza aplikacije (*Slika 3.2*). Ovaj dio će biti dalje pojašnjen u nastavku dokumenta.

***Slika 3.2 Prikaz sučelja za uređivanje izgleda aplikacije*

1. Mogućnost automatskog preuzimanja biblioteka s interneta i njihova instalacija; sve što je potrebno je dodati vezu na lokaciju gdje se nalazi projekt. Npr. *Isječak koda 3.1* prikazuje da za dodati biblioteku sa github repozitorija sve što treba dodati je „implementation 'com.github…'“. Dalje će Android Studio sve sam napraviti i biblioteka je spremna za korištenje.

dependencies {  
 implementation fileTree(**dir**: **'libs'**, **include**: [**'\*.jar'**])  
 implementation **'androidx.appcompat:appcompat:1.0.0'** testImplementation **'junit:junit:4.12'** androidTestImplementation **'androidx.test:runner:1.1.0'** androidTestImplementation **'androidx.test.espresso:espresso-core:3.1.0'** implementation **'com.github.PhilJay:MPAndroidChart:v3.1.0'** implementation **'androidx.constraintlayout:constraintlayout:1.1.3'** implementation **'androidx.cardview:cardview:1.0.0'** implementation **'com.github.clans:fab:1.6.2'** implementation **'com.google.android.material:material:1.0.0'** implementation **'com.google.android.gms:play-services-maps:16.0.0'** implementation **'com.google.android.gms:play-services-location:16.0.0'** implementation **'com.karumi:dexter:5.0.0'**}

Isječak koda 3.1 Korištenje biblioteka

1. Postoji mogućnost kreiranja i pokretanja virtualnog uređaja u sklopu Android Studio aplikacije čime je moguće direktno testiranje aplikacije, ili spajanje vlastitog mobilnog uređaja na računalo uz prethodno postavljanje razvojnih mogućnosti na tom uređaju, čime se također omogućuje testiranje aplikacije te uz sve to čitanje zapisnika o tome što se sve dešava u uređaju. Korisnik ne treba brinuti o tome kako se aplikacija prenosi na mobilni uređaj i instalira.
2. Na posljetku, postoji mogućnost dodavanja mnoštva dodataka kao npr. Git integriran u Android Studio.

## Git

Ako postoji jedan alat koji je apsolutno potreban u razvoju bilo kakve aplikacije, onda je to git. Git je alat za upravljanje verzijama dokumenata. Način na koji radi nije orijentiran na pamćenje kompletne verzije dokumenata nego samo određene promjene poput linije u dokumentu koja je uklonjena, dodana ili promijenjena.

Razlozi zašto je neizostavan su kako slijedi:

1. Omogućava istovremeno uređivanje dokumenata između više ljudi, te spajanje verzija na najlakši mogući način (ako nema sukoba među njihovim verzijama). Ako postoje sukobi među verzijama, najvjerojatnije se mogu lako razriješiti.
2. Bilo kakve promjene koje se naprave na projektu i koliko god davno bile napravljene su reverzibilne
3. Moguće je segmentiranje funkcionalnosti na jedinične „commit“-ove i raspoznavanje što se kada napravilo i koje su promjene točno napravljene za neku funkcionalnost.
4. Podaci se istovremeno nalaze na više lokacija, pa se pri gubitku podataka na jednom računalu gube samo promjene koje nisu spremljene na repozitorij. Zato se ujedno i preporučuje često isporučivanje promjena na repozitorij.

# Android aplikacija

U ovom poglavlju će biti opisani svi dijelovi aplikacije i zašto su takvi kakvi jesu, počevši od koncepta pa do korisničkog sučelja i koda. Sve će biti objašnjeno „bottom-up“ pristupom, tako da čitatelj može uvidjeti kako svaka pojedina komponenta aplikacije radi i kako se svaka komponenta uklapa u cjelinu.

## Koncept

Cilj je napraviti aplikaciju koja će pratiti korisnikovu lokaciju kroz vrijeme u periodu koji korisnik određuje, spremati određene podatke i prikazati ih na ekranu u uređenom obliku. Te podatke treba trajno pohraniti na uređaju tako da se mogu ponovno prikazati pri sljedećem pokretanju aplikacije.

Za izradu su potrebne sljedeće vještine:

* Programski jezik Java
* Osnove objektno orijentiranog programiranja
* Osnove baza podataka
* Znanje o funkcioniranju Android aplikacija i razne Android biblioteke
* Korištenje Google Maps API-ja za dohvaćanje i prikaz trenutne lokacije korisnika.

## Podaci o aktivnostima

Podaci koje Google Maps lokacijski servis pruža su trenutna geografska širina (engl. Latitude) i dužina (engl. Longitude). Trenutak u kojem se dohvati lokacija se može odrediti pozivom metode „System.CurrentTimeMillis()“ koja vraća trenutno vrijeme i datum kroz vremensku oznaku (engl. Timestamp) u milisekundama. Ti podaci su osnova ove aplikacije i iz njih se mogu izračunati sve potrebne informacije da se korisniku prikaže statistika za neku aktivnost.

Korisnik treba imati opciju biranja vrste aktivnosti koju će izvoditi – šetnja, trčanje ili vožnja, pa i tu informaciju želimo pamtiti i prikazati uz statistiku.

*Isječak koda 4.1* prikazuje klasu „TimePlace“ u koju se spremaju osnovni podaci o trenutnoj lokaciji korisnika.

**public class** TimePlace **implements** Serializable {  
 **private** LatLng **place**;  
 **private long time**;  
  
 **public** TimePlace(LatLng place, **long** time) {  
 **this**.**place** = place;  
 **this**.**time** = time;  
 }  
  
 **public** LatLng getPlace() {  
 **return place**;  
 }  
  
 **public double** getLatitude() {  
 **return place**.**latitude**;  
 }  
  
 **public double** getLongitude() {  
 **return place**.**longitude**;  
 }  
  
 **public long** getTime() {  
 **return time**;  
 }  
}

*Isječak koda 4.1 Klasa za pohranu informacija o lokaciji*

Klasa TimePlace sama za sebe nema mnogo značaja i može sadržavati samo jednu lokaciju od mnogih u aktivnosti, tako da sljedeća klasa imena „ActivityData“ (*Isječak koda 4.2*) sadržava sve potrebne informacije o jednoj aktivnosti.

**public class** ActivityData {  
 **private int activityId**;  
 **private** String **activityType**;  
 **private** ArrayList<TimePlace> **timePlaces**;  
  
 **public** ActivityData(**int** id,  
 String activityType,  
 ArrayList<TimePlace> timePlaces) {  
 **this**.**activityId** = id;  
 **this**.**activityType** = activityType;  
 **this**.**timePlaces** = timePlaces;  
 }  
  
 **public int** getActivityId() {  
 **return activityId**;  
 }  
  
 **public** String getActivityType() {  
 **return activityType**;  
 }  
  
 **public** ArrayList<TimePlace> getTimePlaces() {  
 **return this**.**timePlaces**;  
 }  
}

Isječak koda 4.2 Klasa za pohranu informacija o aktivnosti

Ova klasa sadržava sve pomoćne informacije koje će se spremati u bazu podataka uz niz informacija o lokacijama korisnika kroz vrijeme koje trajanja aktivnosti.

## Baza podataka

Baza podataka se uvodi jer omogućava spremanje i čitanje svih novih i starih podataka o aktivnostima.

### Biblioteka

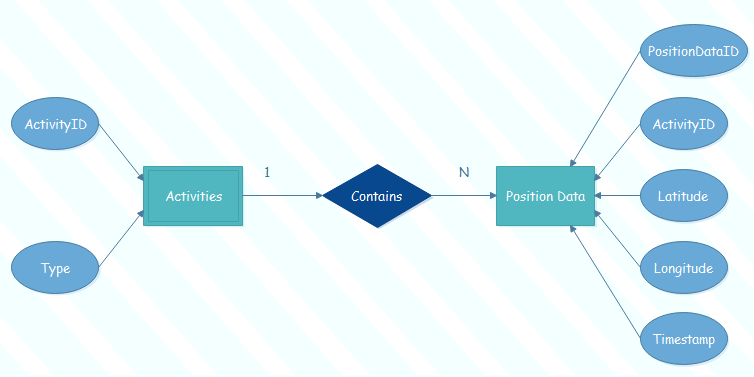
SQLite je standardna biblioteka za kreiranje relacijskih baza podataka pomoću SQL jezika na Androidu. Korištenje SQLite zahtjeva kreiranje pomoćne (engl. Helper) klase koja će naslijediti baznu klasu „SQLiteOpenHelper“ (*Isječak koda 4.3*) koja sadrži sve potrebne funkcionalnosti za upravljanje relacijskom bazom podataka.

**public class** DatabaseHelper **extends** SQLiteOpenHelper

*Isječak koda 4.3 Deklaracija pomoćne klase*

### Definiranje i kreiranje tablica

*Slika 4.1* prikazuje ER dijagram baze podataka korištene u aplikaciji.

*Slika 4.1 ER dijagram baze podataka*

U pomoćnoj klasi su definirane vrijednosti imena stupaca u tablicama (*Isječak koda 4.4*) kako bi se izbjegla greška pri implementaciji pretrage podataka u tablici i unosa podataka u tablicu.

**private static final** String ***DATABASE\_NAME*** = **"ActivitiesData.db"**;  
**private static final** String ***TABLE\_NAME\_ACTIVITIES*** = **"Activities"**;  
**private static final** String ***ACTIVITY\_ID*** = **"ActivityID"**;  
**private static final** String ***ACTIVITY\_TYPE*** = **"Type"**;  
  
**private static final** String ***TABLE\_NAME\_POSITION\_DATA*** = **"PositionData"**;  
**private static final** String ***POSITION\_DATA\_ID*** = **"PositionDataID"**;  
**private static final** String ***LATITUDE*** = **"Latitude"**;  
**private static final** String ***LONGITUDE*** = **"Longitude"**;  
**private static final** String ***TIMESTAMP*** = **"Timestamp"**;

Isječak koda 4.4 Definicija imena stupaca u tablicama

Pri inicijalizaciji pomoćne klase (pozivu konstruktora) potrebno je pozvati konstruktor SQLiteOpenHelper klase kojim specificiramo ime baze podataka i verziju. Ako je verzija baze podataka novija od prošlog puta kad se pokretala aplikacija, pokreće se metoda „onUpgrade“ koju također trebamo implementirati (*Isječak koda 4.5*), u kojoj bi trebali izbrisati postojeće tablice koje su neispravne zbog novije verzije istih.

DatabaseHelper(@Nullable Context context) {  
 **super**(context, ***DATABASE\_NAME***, **null**, 3);  
}  
  
@Override  
**public void** onUpgrade(SQLiteDatabase db, **int** oldVersion, **int** newVersion) {  
 db.execSQL(**"DROP TABLE IF EXISTS "** + ***TABLE\_NAME\_ACTIVITIES***);  
 db.execSQL(**"DROP TABLE IF EXISTS "** + ***TABLE\_NAME\_POSITION\_DATA***);  
  
 onCreate(db);  
}

*Isječak koda 4.5 Konstruktor pomoćne klase i onUpgrade metoda*

Metoda „onCreate“ (Isječak koda 4.6) se poziva pri paljenju aplikacije i služi za kreiranje tablica unutar baze podataka.

@Override  
**public void** onCreate(SQLiteDatabase db) {  
 db.execSQL(**"CREATE TABLE IF NOT EXISTS "** + ***TABLE\_NAME\_ACTIVITIES*** +  
 **" ("** +  
 ***ACTIVITY\_ID*** + **" INTEGER PRIMARY KEY AUTOINCREMENT, "** +  
 ***ACTIVITY\_TYPE*** + **" TEXT);"**);  
  
 db.execSQL(**"CREATE TABLE IF NOT EXISTS "** + ***TABLE\_NAME\_POSITION\_DATA*** +  
 **" ("** +  
 ***POSITION\_DATA\_ID*** + **" INTEGER PRIMARY KEY AUTOINCREMENT, "** +  
 ***LATITUDE*** + **" REAL NOT NULL, "** +  
 ***LONGITUDE*** + **" REAL NOT NULL, "** +  
 ***TIMESTAMP*** + **" LONG NOT NULL, "** +  
 ***ACTIVITY\_ID*** + **" INTEGER NOT NULL, "** +  
 **"FOREIGN KEY("** + ***ACTIVITY\_ID*** + **") REFERENCES "** +  
 ***TABLE\_NAME\_ACTIVITIES*** + **"("** + ***ACTIVITY\_ID*** + **"));"**);  
}

Isječak koda 4.6 Kreiranje tablica u bazi podataka

### Unos podataka

Nakon svake aktivnosti u bazu se unose novi podaci, što se ostvaruje pomoću SQL naredbi za umetanje (engl. Insert) redaka u tablicu.

Na sljedećem isječku (*Isječak koda 4.7*) su prikazane metode pomoćne klase koje vrše prijenos podataka iz argumenata metoda u oblik pogodan za bazu podataka.

**public int** insertActivity(String activityType) {  
 SQLiteDatabase db = **this**.getWritableDatabase();  
 db.execSQL(**"INSERT INTO "** + ***TABLE\_NAME\_ACTIVITIES*** + **" ("** +  
 ***ACTIVITY\_TYPE*** + **") VALUES ("** +  
 **"'"** + activityType + **"');"**);  
  
 Cursor cursor = db.rawQuery(**"SELECT last\_insert\_rowid()"**, **null**);  
 cursor.moveToFirst();  
 **return** cursor.getInt(0);  
}  
  
**void** insertPositionData(**int** activityId, TimePlace data) {  
 SQLiteDatabase db = **this**.getWritableDatabase();  
  
 db.execSQL(**"INSERT INTO "** + ***TABLE\_NAME\_POSITION\_DATA*** + **" ("** +  
 ***ACTIVITY\_ID*** + **", "** + ***LATITUDE*** + **", "** + ***LONGITUDE*** + **", "** +  
 ***TIMESTAMP*** + **") VALUES ("** +  
 activityId + **", "** +  
 data.getLatitude() + **", "** +  
 data.getLongitude() + **", "** +  
 data.getTime() + **");"**);  
}

*Isječak koda 4.7 Metode za unos u bazu podataka*

Metoda „insertActivity“ kreira novi unos u tablicu aktivnosti i dohvaća automatski generiranu identifikacijsku oznaku unešene aktivnosti. Da bi se podaci o aktivnosti pravilno unijeli u bazu podataka, metoda „insertActivity“ se treba pozvati prva, a zatim se identifikacijska oznaka koja je rezultat te metode treba prosljeđivati u svaki poziv metode „insertPositionData“. Metoda „insertPositionData“ u tablicu „PositionData“ unosi jedan redak s vrijednostima iz parametara s kojima je pozvana.

### Dohvaćanje podataka

Prilikom pokretanja aplikacije, trebaju se učitati svi podaci koji već postoje u bazi podataka. SQL upit za dohvat iz baze podataka vraća redove podataka kojima se može pristupiti uz pomoć klase „Cursor“ koja omogućava šetanje po redovima i dohvat vrijednosti iz određenog stupca.

Vrijednosti iz tablica omogućavaju obnovu originalnih podataka u obliku klase „ActivityData“, iz kojih će se izvući sve informacije te prikazati korisniku.

Sljedeći isječak koda (*Isječak koda 4.8*) prikazuje jednu od metoda za dohvat podataka iz baze podataka. Metoda „getAllActivities“ se poziva izvana i vraća sve podatke spremljene u bazu podataka u obliku liste objekata tipa „ActivityData“.

**public** ArrayList<ActivityData> getAllActivities() {  
 SQLiteDatabase db = **this**.getReadableDatabase();  
 ArrayList<ActivityData> storedData = **new** ArrayList<>();  
  
 Cursor cursor = db.rawQuery(**"SELECT \* FROM "** + ***TABLE\_NAME\_ACTIVITIES***,  
 **null**);  
 **if** (cursor.moveToFirst()) {  
 **while** (!cursor.isAfterLast()) {  
 **int** id = cursor.getInt(cursor.getColumnIndex(***ACTIVITY\_ID***));  
 String activityType = cursor.getString(cursor.getColumnIndex(***ACTIVITY\_TYPE***));  
  
 storedData.add(**new** ActivityData(id, activityType, getActivityPositions(id)));  
 cursor.moveToNext();  
 }  
 }  
 **return** storedData;  
}

*Isječak koda 4.8 Metoda za dohvat podataka iz baze*

## Android aktivnosti

Prije razrade sljedećih poglavlja u kojima će biti objašnjeni logički dijelovi aplikacije, potrebno je razumjeti pojam aktivnosti u Androidu i životni ciklus aktivnosti u Androidu.

### Aktivnost

Aktivnost (engl. Activity) je dio aplikacije koji predstavlja jedan od više mogućih prikaza na ekranu preko kojeg korisnik može vršiti interakciju s aplikacijom.

Aktivnost je određena klasom koja nasljeđuje klasu „Activity“. Svaka klasa koja implementira aktivnost mora imati implementaciju metode „onCreate“ gdje se aktivnost inicijalizira. Inicijalizacija podrazumijeva postavljanje svih početnih varijabli i elemenata u klasi i na ekranu u odgovarajuću vrijednost.

Metoda „setContentView“ omogućava postavljanje ekrana koji prikazuje elemente definirane unutar XML dokumenta i njihov raspored.

Metoda „findViewById“ omogućava dohvaćanje reference objekta nekog elementa sa ekrana pomoću koje se može upravljati tim objektom i mijenjati njegova svojstva (npr. Za objekt tipa „TextView“ moguće je mijenjati text, boju, raspored, veličinu, tip slova itd.). Da bi se mogao dohvatiti neki element na ovaj način, potrebno mu je u XML dokumentu odrediti identifikacijsku oznaku.

### Životni ciklus aktivnosti

Kako su aktivnosti implementirane klasama, postoje točno određene metode koje se pozivaju pod točno određenim uvjetima i redoslijedom. Npr. Metoda „onCreate“ se poziva prilikom inicijalizacije aktivnosti, a metoda „onStop“ prilikom gašenja aktivnosti. Sve metode koje su dio životnog ciklusa aktivnosti se mogu implementirati i prilagoditi po potrebi.

Sljedeća slika (Slika 4.2) prikazuje dijagram životnog ciklusa Android aktivnosti.



Slika 4.2 Životni ciklus aktivnosti

### Prijelazi između aktivnosti

Ukoliko je u aplikaciji potrebno više aktivnosti, prijelazi između njih su omogućeni korištenjem metoda „startActivity()“ i „startActivityForResult()“ koje primaju objekt tipa „Intent“.

„Intent“ objekt sadrži apstraktni opis radnje koju želimo izvršiti (prijelaz u drugu aktivnost), i u njega je moguće pakirati razne tipove podataka koji se potom raspakiraju u klasi pokrenute aktivnosti (u slučaju ove aplikacije to su lokacijski podaci jedne aktivnosti, odnosno objekt tipa „ActivityData“).

Ako je aktivnost pokrenuta metodom „startActivityForResult()“, od nje se očekuje nekakva povratna informacija. Ta informacija se postavlja preko objekta tipa „Intent“ i jedne od predefiniranih vrijednosti poput „RESULT\_OK“ ili „RESULT\_CANCELED“ prosljeđenih u metodu „setResult“, a na prethodnu aktivnost se vraća pozivom na „finish()“. Početna aktivnost treba imati implementiranu metodu „onActivityResult()“ gdje provjerava odakle je rezultat i koji su parametri postavljeni za rezultat i poduzima odgovarajuće akcije.

## Aktivnost za praćenje lokacije

### Dizajn i XML dokumenti

U ovom podpoglavlju je kratka digresija od aktivnosti, tako da se razjasni način kreiranja elemenata koji se žele prikazati na ekranu, i neka njihova svojstva. Ovo poglavlje usmjereno je na „statičko“ kreiranje elemenata na ekranu, a „dinamičko“ će biti objašnjeno u jednom od podpoglavlja za glavnu aktivnost.

Izgled dijela aplikacije za praćenje lokacije se dijeli na „Google Maps“ dio i traku koja će prikazivati vrijeme trajanja aktivnosti i prijeđenu udaljenost za to vrijeme. Na slici (*Slika 4.3*) vidi se „dizajn“ rasporeda elemenata u aktivnosti za praćenje lokacije. Gumb koji se koristi za početak i prestanak praćenja lokacije se ne vidi jer je mu je atribut vidljivosti postavljen u „INVISIBLE“ dok se ne primi prva lokacija korisnika.

*Slika 4.3 Raspored elemenata u aktivnosti za praćenje lokacije*

Sljedeći isječak (*Isječak koda 4.9*) prikazuje kako u XML dokumentu izgleda kod za raspored elemenata koji odgovara prethodno prikazanoj slici (*Slika 4.3*).

*<?***xml version="1.0" encoding="utf-8"***?>*<**androidx.coordinatorlayout.widget.CoordinatorLayout  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:keepScreenOn="true"  
 xmlns:fab="http://schemas.android.com/apk/res-auto"  
 tools:context=".LocationTracker"**>  
 <**LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:layout\_gravity="bottom"  
 android:orientation="vertical"**>  
 <**fragment  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:map="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:hardwareAccelerated="true"  
 android:id="@+id/map"  
 android:name="com.google.android.gms.maps.SupportMapFragment"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:layout\_weight="9"  
 tools:context=".LocationTracker"** />  
 <**LinearLayout...**>  
 </**LinearLayout**>  
  
 <**com.github.clans.fab.FloatingActionButton...>**  
</**androidx.coordinatorlayout.widget.CoordinatorLayout**>

*Isječak koda 4.9 Primjer XML dokumenta za raspored elemenata*

Neki elementi ovog XML dokumenta se ne prikazuju, sklopljeni su da isječak ne bi bio pre-dug. Ako se želi vidjeti cijeli kod, može se naći cijeli projekt na githubu. Bitno je naglasiti da se elementi u XML-u dodaju u obliku stabla. To stablo se sastoji od dva tipa elemenata: „ViewGroup“ i „View“.

U stablu elemenata, element tipa „ViewGroup“ može sadržavati više elemenata tipa „View“ ili „ViewGroup“, dok je element tipa „View“ samostalan.

„ViewGroup“ elementi su najčešće „Layout“ elementi kao „LinearLayout“, „RelativeLayout“, „ConstraintLayout“, „CoordinatorLayout“ itd. Oni definiraju postavke koje određuju na koji način će se njihova djeca prikazivati na ekranu.

„View“ element može biti bilo kakav element kao npr. „TextView“, „EditView“, „FloatingActionButton“, „Button“, „ImageView“ itd.

Svaki element mora imati postavljene atribute visine i širine elementa, čije su najčešće vrijednosti „match\_parent“ ili „wrap\_content“, a ako se elementu želi pristupiti iz koda, potrebno mu je dodati id (npr. "@+id/map"). Svaki element može biti interaktivan i uzrokovati neku akciju ako mu se doda „onClickListener“ metoda koja se poziva kada korisnik pritisne taj element.

### onCreate – Postavljanje layout-a

„onCreate“ metoda se poziva pri paljenju aktivnosti. U sljedećem isječku (*Isječak koda 4.10*) je prikazan isječak koda iz te metode koji pozivanjem metode „setContentView“ postavlja uređaj da prikazuje elemente i raspored elemenata (engl. Layout) definirane u XML dokumentu s nazivom „activity\_location\_tracker“ koji se nalazi u direktoriju „layout“.

@Override  
**protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_location\_tracker***);

*Isječak koda 4.10 Postavljanje layout-a*

### onCreate – Postavljanje lokacijskog servisa

Sljedeći isječak (*Isječak koda 4.11*) prikazuje postavljanje lokacijskog servisa tako da se svi funkcijski pozivi vezani za lokacije preusmjeravaju unutar klase „LocationTracker“ prema „override“-anim metodama: „onMapReady“, „onConnected“, „onConnectionSuspended“ i „onConnectionFailed“. Ovo je omogućeno kroz objekt mGoogleApiClient kojem se prosljeđuje instanca trenutne klase (LocationTracker) gdje su implementirane te metode.

**if** (requestLocationPermission()) {  
 SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()  
 .findFragmentById(R.id.***map***);  
 mapFragment.getMapAsync(**this**);  
  
 **mGoogleApiClient** = **new** GoogleApiClient.Builder(**this**)  
 .addConnectionCallbacks(**this**)  
 .addOnConnectionFailedListener(**this**)  
 .addApi(LocationServices.***API***)  
 .build();  
  
 **mLocationManager** = (LocationManager) **this**.getSystemService(Context.***LOCATION\_SERVICE***);  
  
 **if** (!checkIsLocationEnabled()) {  
 returnToMainActivity();  
 }  
} **else** {  
 returnToMainActivity();  
}

*Isječak koda 4.11 Postavljanje Google Maps Lokacijskog servisa*

Metoda „requestLocationPermission“ služi za provjeru da li aplikacija ima dozvolu pristupa lokaciji mobilnog uređaja, i ako nema, zatraži ju. Ako korisnik odbije dozvolu, aplikacija vraća korisnika na glavnu aktivnost.

Metoda „checkIsLocationEnabled“ služi za provjeru da li je uključena opcija uređaja za određivanje položaja. Ako nije, korisniku se šalje obavijest o tome, te se aplikacija vraća na glavnu aktivnost.

### onCreate – Postavljanje referenci na „View“-ove i „onClickListener“

Posljednji dio metode „onCreate“ je prikazan na sljedećem isječku (*Isječak koda 4.12*). Taj dio koda pohranjuje referencu prikazanih elemenata u varijable klase „LocationTracker“, a za gumb u donjem desnom kutu postavlja „onClickListener“ – metodu koja će se izvršiti kada korisnik pritisne gumb. Prvi pritisak na gumb uzrokuje da se aktivnost počne pratiti. Tada gumb mijenja oblik, a ponovnim pritiskom se zaustavlja praćenje lokacije, aplikacija se vraća na glavni prikaz i kreira se novi ispis na ekranu.

**timeTrackerTextView** = findViewById(R.id.***timePassedTextView***);  
**distanceTrackerTextView** = findViewById(R.id.***distancePassedTextView***);  
**finishButton** = findViewById(R.id.***fab\_finish***);  
**finishButton**.setOnClickListener(**new** View.OnClickListener() {  
 **public void** onClick(View v) {  
 **if** (**startPressed**) {  
 returnToMainActivity();  
 } **else** {  
 **startPressed** = **true**;  
 **finishButton**.setImageDrawable(getResources()  
 .getDrawable(android.R.drawable.***checkbox\_on\_background***));  
  
 **mStartTime** = System.*currentTimeMillis*();  
 **timerHandler** = **new** Handler();  
 **timerHandler**.postDelayed(**timerRunnable**, 0);  
  
 **if** (**mMap** != **null**) {  
 **mMap**.addMarker(**new** MarkerOptions()  
 .position(**mCurrentLocation**).title(**"Starting position"**));  
 }  
 **mPrevTimePlace** = **new** TimePlace(**mCurrentLocation**, **mCurrentTime**);  
 }  
 }  
});

*Isječak koda 4.12 Postavljanje referenci na View-ove i onClickLitener metode*

### Sekvenca za pokretanje lokacijskog servisa

Nakon metode „onCreate“ u kojoj su namještene postavke Google API klijenta, poziva se metoda „onStart“ koja poziva klijentovu metodu „connect“. Ta metoda spaja klijenta na Google Play servis. Ako je konekcija uspješna, poziva se metoda „onConnected“, a u suprotnom se poziva metoda „onConnectionFailed“. Na sljedećem isječku (*Isječak koda 4.13*) mogu se vidjeti implementacije metoda „onStart“, „onConnected“ i „startLocationUpdates“.

@Override  
**protected void** onStart() {  
 **super**.onStart();  
 **if** (**mGoogleApiClient** != **null**) {  
 **mGoogleApiClient**.connect();  
 }  
}  
  
@Override  
**public void** onConnected(@Nullable Bundle bundle) {  
 **if** (ActivityCompat.*checkSelfPermission*(**this**,  
 android.Manifest.permission.***ACCESS\_FINE\_LOCATION***) !=  
 PackageManager.***PERMISSION\_GRANTED***) {  
 **return**;  
 }  
 startLocationUpdates();  
  
 **mLocation** = LocationServices.***FusedLocationApi***.getLastLocation(**mGoogleApiClient**);  
 **if** (**mLocation** == **null**) {  
 startLocationUpdates();  
 }  
 **if** (**mLocation** == **null**) {  
 Toast.*makeText*(**this**, **"Location not Detected"**, Toast.***LENGTH\_SHORT***).show();  
 }  
}  
  
**protected void** startLocationUpdates() {  
 **mLocationRequest** = LocationRequest.*create*()  
 .setPriority(LocationRequest.***PRIORITY\_HIGH\_ACCURACY***)  
 .setInterval(***UPDATE\_INTERVAL***)  
 .setFastestInterval(***FASTEST\_INTERVAL***);  
  
 **if** (ActivityCompat.*checkSelfPermission*(**this**,  
 android.Manifest.permission.***ACCESS\_FINE\_LOCATION***) !=  
 PackageManager.***PERMISSION\_GRANTED***) {  
 **return**;  
 }  
 LocationServices.***FusedLocationApi*** .requestLocationUpdates(**mGoogleApiClient**, **mLocationRequest**, **this**);  
}

*Isječak koda 4.13 Prikaz metoda onStart, onConnected i startLocationUpdates*

Metoda „onConnected“ implementirana je sa ciljem da pokrene praćenje lokacije, i zove pomoćnu metodu „startLocationUpdates“. Ta metoda kreira i šalje novi zahtjev za periodičnim dohvaćanjem lokacije prema lokacijskom servisu. Ujedno šalje instancu trenutne klase kao parametar za „locationListener“, što postavlja da se metoda „onLocationChanged“ poziva kada lokacijski servis pošalje neku vrijednost. U sljedećem isječku (*Isječak koda 4.14*) je prikazana metoda „onLocationChanged“.

@Override  
**public void** onLocationChanged(Location location) {  
 **mCurrentLocation** = **new** LatLng(location.getLatitude(), location.getLongitude());  
  
 SupportMapFragment mapFragment =  
 (SupportMapFragment) getSupportFragmentManager().findFragmentById(R.id.***map***);  
 mapFragment.getMapAsync(**this**);  
}

*Isječak koda 4.14 Prikaz metode onLocationChanged*

Metoda „onLocationChanged“ dohvaća vrijednost geografske širine i dužine iz lokacije poslane od strane lokacijskog servisa. Potom se postavlja zahtjev za asinkronim pozivom metode „onMapReady“ prikazane u sljedećem isječku (Isječak koda *4*.*15*).

@Override  
**public void** onMapReady(GoogleMap googleMap) {  
 **float**[] distanceResult = **new float**[1];  
 **mCurrentTime** = System.*currentTimeMillis*();  
 **mMap** = googleMap;  
 **if** (**mCurrentLocation** != **null**) {  
 **if** (**isFirstRun**) {  
 **mMap**.getUiSettings().setMapToolbarEnabled(**false**);  
 **mMap**.getUiSettings().setZoomControlsEnabled(**false**);  
 **mMap**.moveCamera(CameraUpdateFactory.*newLatLngZoom*(**mCurrentLocation**, 16f));  
  
 **if** (ActivityCompat.*checkSelfPermission*(**this**,  
 android.Manifest.permission.***ACCESS\_FINE\_LOCATION***) == PackageManager.***PERMISSION\_GRANTED***) {  
 **mMap**.setMyLocationEnabled(**true**);  
 }  
  
 **isFirstRun** = **false**;  
 **finishButton**.setVisibility(View.***VISIBLE***);  
 } **else if** (**startPressed**) {  
 Location.*distanceBetween*(**mPrevTimePlace**.getLatitude(), **mPrevTimePlace**.getLongitude(),  
 **mCurrentLocation**.**latitude**, **mCurrentLocation**.**longitude**, distanceResult);  
  
 **if** (distanceResult[0] > **mMinimalDistanceToTrack**) {  
 **if** (**mEntryId** == -1) {  
 **mEntryId** = **mDatabaseHelper**.insertActivity(**mActivityType**);  
  
 **mDatabaseHelper**.insertPositionData(**mEntryId**, **mPrevTimePlace**);  
 }

**mMap**.animateCamera(CameraUpdateFactory

.*newLatLng*(**mCurrentLocation**));  
 **mMap**.addPolyline(**new** PolylineOptions()  
 .add(**mPrevTimePlace**.getPlace(), **mCurrentLocation**)  
 .width(20)  
 .color(Color.***CYAN***)  
 .jointType(JointType.***ROUND***));  
  
 **mPrevTimePlace** = **new** TimePlace(**mCurrentLocation**, **mCurrentTime**);  
 **mDatabaseHelper**.insertPositionData(**mEntryId**, **mPrevTimePlace**);  
  
 **mTotalDistance** = **mTotalDistance** + distanceResult[0];  
 **distanceTrackerTextView**.setText(String  
 .*format*(Locale.*getDefault*(), **"Distance: %.2fm"**, **mTotalDistance**));  
 }  
 }  
 }  
}

Isječak koda 4.15 Prikaz metode onMapReady

Metoda „onMapReady“ se poziva kad je objekt Google karte ili element tipa „fragment“ na ekranu spreman za učitavanje novih postavki i slično. Ova metoda ima cilj prikazati na karti sve što se desilo s najnovijom promjenom lokacije, ali prvi korak joj je namjestiti postavke karte.

Kada se ova metoda prvi put otvori, varijabla „isFirstRun“ je postavljena u false. Tada metoda postavlja osnovne postavke na Google karti, pomiče kameru na dobivenu lokaciju (spremljenu u varijabli „mCurrentLocation“) i postavlja zoom karte. Uključuju se plava točka za korisničku trenutnu lokaciju na karti i prikazuje se gumb u donjem desnom kutu ekrana (koji je prethodno bio sakriven tako da bi izbjegli situaciju u kojoj korisnik može početi pratiti aktivnost prije nego mu je početna lokacija određena).

Objašnjeni dio metode se izvrši samo jedan put po aktivnosti, a drugi dio metode se izvršava uvijek, ali tek nakon što korisnik pritisne gumb za početak praćenja aktivnosti. Računa se udaljenost između prethodne i trenutne lokacije po geografskoj dužini i širini. Ako je udaljenost manja od neke vrijednosti koja je ovisna o tome koji se tip aktivnosti prati, ignorira se (tako da zbog nepreciznosti lokacijskog servisa ne bi imalo N lokacijskih podataka koji su uzrokovani greškom dok korisnik zapravo stoji u mjestu). Kamera se pomiče na trenutnu lokaciju i ostavlja se trag na karti od prethodne lokacije do trenutne. Na kraju se dodaje novi unos o poziciji u bazu podataka, te računa ukupna prijeđena udaljenost i ispisuje na ekran.

### Praćenje proteklog vremena na ekranu

Praćenje proteklog vremena i njegov ispis na karti je implementiran na način da postoji „timer“ koji pokreće određenu metodu svako 500ms koja računa vrijeme proteklo od prvog sistemskog zahtjeva za vremenski otisak spremljenog u varijablu „startTime“ do trenutnog sistemskog vremenskog otiska. Timer se prvi put pokreće kad se pokrene praćenje aktivnosti pritiskom na gumb u donjem desnom kutu, a svaki sljedeći put sam sebe pokreće sa kašnjenjem od 500ms. Timer se gasi prije povratka u glavnu aktivnost pozivom metode „removeCallbacks“.

### Povratak u glavnu aktivnost

Na sljedećem isječku (*Isječak koda 4.16*) je prikazana metoda „returnToMainActivity“, koja u ovisnosti o tome da li je korisnik pritisnuo gumb za početak praćenja i da li je aktivnost unesena u bazu podatka, vraća rezultat da je radnja ili otkazana ili je sve u redu i pakira identifikacijsku oznaku aktivnosti spremljene u bazu podataka u rezultat (koji se šalje u glavnu aktivnost).

**public void** returnToMainActivity() {  
 Log.*d*(***TAG***, **"Stop timer, return to main activity and send data"**);  
  
 **timerHandler**.removeCallbacks(**timerRunnable**);  
  
 **if** (!**startPressed** || **mEntryId** < 0) {  
 setResult(***RESULT\_CANCELED***);  
 finish();  
 }  
  
 Intent data = **new** Intent(**this**, MainActivity.**class**);  
 data.putExtra(**"ActivityId"**, **mEntryId**);  
  
 setResult(***RESULT\_OK***, data);  
 finish();  
}

*Isječak koda 4.16 Prikaz metode returnToMainActivity*

### „RouteDisplay“ aktivnost

Kako bi se korisniku mogao reproducirati put iz aktivnosti koja je prikazana na ekranu, kreirana je nova klasa koja sadrži neke elemente klase „LocationTracker“, pa zato neće biti detaljno objašnjena. Ova klasa uzima identifikacijsku oznaku aktivnosti, pomoću nje iz baze podataka dohvaća sve podatke o aktivnosti i iscrtava je na Google kartu, uz oznake gdje je početak a gdje kraj aktivnosti. Layout koji se koristi je isti kao i od „LocationTracker“ aktivnosti, s tim da je donja traka s vremenom trajanja i prijeđenom udaljenosti uklonjena. Povratak u glavnu aktivnost je moguć korištenjem tipke za natrag.

## Glavna aktivnost

Glavna aktivnost je središnji dio ove aplikacije. Unutar nje se svi podaci obrađuju za prikaz korisniku, ona se pokreće pri paljenju aplikacije, i iz nje se pokreće aktivnost „LocationTracker“ gdje se prati lokacija korisnika.

### Dizajn i XML dokumenti

Na slici (*Slika 4.4*) je prikazan dizajn glavne aktivnosti bez ikakvih prethodno učitanih podataka. Na ekranu se nalazi naziv aplikacije, upute i gumb za izbornik u donjem desnom kutu. Prilikom dodavanja podataka o prvoj aktivnosti, upute će se skloniti s ekrana, a dodati će se element koji sadržava podatke i grafove o korisnikovoj aktivnosti.

Slika 4.4 Raspored elemenata u glavnoj aktivnosti

XML dokument u kojem su definirani osnovni elementi koji se prikazuju na ekranu je podijeljen u dvije datoteke. U prvoj datoteci prikazanoj na sljedećem isječku (*Isječak koda 4.17*) je definiran osnovni raspored elemenata, elementi trake na vrhu ekrana i meni s gumbovima.

*<?***xml version="1.0" encoding="utf-8"***?>*<**androidx.coordinatorlayout.widget.CoordinatorLayout  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 xmlns:fab="http://schemas.android.com/apk/res-auto"  
 tools:context=".MainActivity"**>

<**com.google.android.material.appbar.AppBarLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:theme="@style/AppTheme.AppBarOverlay"**>

<**androidx.appcompat.widget.Toolbar  
 android:id="@+id/toolbar"  
 app:title="Uzas"  
 android:layout\_width="match\_parent"  
 android:layout\_height="?attr/actionBarSize"  
 android:background="?attr/colorPrimary"  
 app:popupTheme="@style/AppTheme.PopupOverlay"** />  
 </**com.google.android.material.appbar.AppBarLayout**>  
 <**include layout="@layout/content\_main"** />  
 <**com.github.clans.fab.FloatingActionMenu...**>  
</**androidx.coordinatorlayout.widget.CoordinatorLayout**>

*Isječak koda 4.17 Prikaz XML dokumenta glavne aktivnosti*

Ova datoteka poziva drugu datoteku imena „content\_main“ koja sadržava elemente koji će se ispisivati ispod gornje zelene trake i sve elemente koji će se kasnije dodati na ekran za prikaz. Poziv na drugu datoteku se izvodi dodavanjem elementa „include“, a krajnji rezultat je kao da je sadržaj pozvane datoteke napisan tamo gdje je pozvan.

Druga datoteka je prikazana na sljedećem isječku (*Isječak koda 4.18*). Ovaj dokument je postavljen tako da se na ekranu omogući „scrollanje“, odnosno omogućava to da se u aktivnost može učitati i prikazati više elemenata nego može stati na ekran koji se mogu pomicati gore – dolje. Element koji to omogućava je „NestedScrollView“. Unutar njega je potrebno definirati još element „LinearLayout“ i postaviti ga da svoju djecu prikazuje na vertikalno orijentirani način (jedan ispod drugog) tako da se elementi automatski usklađuju s obzirom na druge elemente. Za njega je postavljena identifikacijska oznaka „cardContainer“ i u njega će se dodavati svi elementi sa sadržajem za korisnika.

*<?***xml version="1.0" encoding="utf-8"***?>*<**androidx.core.widget.NestedScrollView  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:id="@+id/scrollView"  
 android:orientation="vertical"  
 app:layout\_behavior="@string/appbar\_scrolling\_view\_behavior"  
 tools:context=".MainActivity"  
 tools:showIn="@layout/activity\_main"**>  
  
 <**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:id="@+id/cardContainer"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:orientation="vertical"  
 tools:context=".MainActivity"**>  
  
 <**androidx.cardview.widget.CardView  
 xmlns:card\_view="http://schemas.android.com/apk/res-auto"  
 android:id="@+id/directionsContainer"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_margin="8dp"  
 card\_view:cardCornerRadius="4dp"**>  
  
 <**TextView  
 android:id="@+id/initialHelpTextView"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginStart="8dp"  
 android:layout\_marginEnd="8dp"  
 android:layout\_gravity="center\_horizontal"  
 android:text="Press + and choose an activity to start!"** />  
 </**androidx.cardview.widget.CardView**>  
 </**LinearLayout**>  
</**androidx.core.widget.NestedScrollView**>

*Isječak koda 4.18 Prikaz pomoćnog XML dokumenta za glavnu aktivnost*

„CardView“ je element koji grupira elemente unutar jednog kvadrata sa zaobljenim rubovima. U sklopu ove aplikacije se koristi zbog estetike, a kao zamjena se može koristiti bilo koji drugi element tipa „Layout“ odnosno „ViewGroup“.

### Dinamičko dodavanje elemenata na ekran

Da bi se elementi dinamički dodavali na ekran potrebna je referentna točka, odnosno identifikacijska oznaka elementa u kojeg će se kao dijete dodati neki element kreiran kroz kod. Taj element je označen kao „cardContainer“. U njega će se dodavati neki element tipa „CardView“ koji će također sadržavati neku svoju djecu koja će se dinamički alocirati da bi se korisniku prikazale sve potrebne informacije.

U klasi glavne aktivnosti je definirana još jedna klasa pod nazivom „ActivityCardCreator“ u kojoj je izdvojena funkcionalnost dinamičkog kreiranja elemenata za prikaz, obradu podataka o aktivnostima i punjenja tih elemenata s obrađenim podacima.

Za svaki element iz XML dokumenta postoji pripadajuća klasa implementirana za programiranje u Java programskom jeziku. Potrebno je svaki element dinamički kreirati, postaviti svakom elementu odgovarajuće parametre preko klase „LayoutParamas“, postaviti njihove vrijednosti i dodatne postavke, te ih dodati kao dijete nekog drugog elementa. Za dodavanje djece u elemente se koristi metoda „addView“.

Budući da se u projektu koristi mnogo različitih elemenata, da bi se jasnije prikazao način na koji ovo radi, sljedeći isječak (*Isječak koda 4.19*) pruža primjer kako se dinamički elementi dodaju na ekran.

**void** example(Context context)  
{  
 **float** d = getResources().getDisplayMetrics().**density**;  
 LinearLayout cardContainer = findViewById(R.id.***cardContainer***);  
 TextView activityTypeTextView = **new** TextView(context);  
   
 LinearLayout.LayoutParams textLayoutParams =   
 **new** LinearLayout.LayoutParams(  
 ViewGroup.LayoutParams.***MATCH\_PARENT***,  
 ViewGroup.LayoutParams.***WRAP\_CONTENT***);  
 textLayoutParams.**leftMargin** = (**int**) (4\*d);  
   
 activityTypeTextView.setLayoutParams(textLayoutParams);  
 activityTypeTextView.setText(**"Activity: Walk"**);  
 cardContainer.addView(**card**, 0);  
}

*Isječak koda 4.19 Primjer dinamičkog kreiranja elemenata i dodavanja na ekran*

### „onCreate“ metoda

Budući da glavna aktivnost treba učitati podatke iz baze podataka pri pokretanju aplikacije, u „onCreate“ metodi prikazanoj na isječku (*Isječak koda 4.20*) je pozvana metoda za dohvaćanje svih aktivnosti iz baze podataka koja je opisana u poglavlju 4.3.4. Svaki član liste dohvaćenih aktivnosti se pretvara u element koji se prikazuje na ekranu uz pomoć metode „processActivityData“ koja će biti opisana u jednom od sljedećih podpoglavlja.

@Override  
**protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_main***);  
  
 setViewReferences();  
  
 **mDatabaseHelper** = **new** DatabaseHelper(**this**.getApplicationContext());  
 **mCardCreator** = **new** ActivityCardCreator();  
  
 ArrayList<ActivityData> activityList = **mDatabaseHelper**.getAllActivities();  
 **for** (ActivityData it : activityList) {  
 **if** (it.getTimePlaces().size() != 0) {  
 processActivityData(it);  
 }  
 }  
}

*Isječak koda 4.20 Prikaz metode onCreate u klasi MainActivity*

Ova metoda također poziva metodu nazvanu „setViewReferences“ gdje se postavljaju reference na elemente prikazane na ekranu i „onClickListener“ metode.

### Pokretanje i dohvat rezultata aktivnosti za praćenje lokacije

Na sljedećem isječku (*Isječak koda 4.21*) su prikazane „startTracking“ metoda koja pokreće „LocationTracker“ aktivnost, „showRoute“ metoda koja pokreće „RouteDisplay“ aktivnost i „onActivityResult“ metoda koja rukuje s rezultatom primljenim od „LocationTracker“ metode. U poglavlju 4.4.3 je opisano kako se implementira prijelaz između dvije aktivnosti, a metode „getExtra“ i „putExtra“ se koriste za prosljeđivanje podataka između njih.

**public void** startTracking(String activityType) {  
 Intent intent = **new** Intent(**this**, LocationTracker.**class**);  
 intent.putExtra(**"activityType"**, activityType);  
  
 startActivityForResult(intent, ***NEW\_ACTIVITY\_REQUEST***);  
}  
  
@Override  
**protected void** onActivityResult(**int** requestCode, **int** resultCode, Intent data) {  
 **if** (requestCode == ***NEW\_ACTIVITY\_REQUEST***) {  
 **if** (resultCode == ***RESULT\_OK***) {  
 **int** receivedData =  
 (**int**) data.getSerializableExtra(**"ActivityId"**);  
 **if** (receivedData < 0) {  
 Log.*e*(**"OnActivityResult"**, **"Bad Activity Data!"**);  
 **return**;  
 }  
  
 processActivityData(**mDatabaseHelper**.getActivityById(receivedData));  
 }  
 }  
}

*Isječak koda 4.21 Prikaz metoda za pokretanje aktivnosti za praćenje i dohvaćanje rezultata*

Kada se dohvate pozicijski podaci, oni se obrađuju u svrhu kreiranja novog elementa koji se prikazuje na ekranu, pa se unose u bazu podataka.

### „processActivityData“ metoda

Na sljedećem isječku (Isječak koda 4.22) je prikazana metoda „processActivityData“. Ona je pozvana na par mjesta u kodu: kad se dohvate sve aktivnosti iz baze podataka i kad završi „LocationTracker“ aktivnost. Njen cilj je postaviti prozor za pomoć u nevidljivo stanje i prosljediti podatke o aktivnosti u „ActivityCardCreator“ klasu na obradu.

**private void** processActivityData(ActivityData data) {  
 **if** (**directionsContainer**.getVisibility() == View.***VISIBLE***) {  
 **directionsContainer**.setVisibility(View.***INVISIBLE***);  
 }  
  
 **mCardCreator**.createActivityCardFromData(data, **this**);  
}

Isječak koda 4.22 Prikaz metode „processActivityData“

### „ActivityCardCreator“ klasa

Da bi cijela funkcionalnost što se tiče dinamičkog kreiranja elemenata na ekranu bila izdvojena od upravljačke funkcionalnosti glavne aktivnosti, unutar klase glavne aktivnosti napravljena je podklasa koja enkapsulira svu tu funkcionalnost, uključujući statističke proračune koji se ispisuju na ekranu poput grafa brzina/vrijeme.

Budući da je u poglavlju 4.6.2 prikazano kako se dinamički dodaju i kreiraju elementi, u ovom poglavlju će biti prikazani samo dijelovi proračuna i grafova, kao i konačan rezultat. Ukupan kod možete naći na github projektu.

„createActivityData“ metoda je ulazna točka u ovu klasu. Ona kreira nove objekte za sve elemente koje će se koristiti, potom poziva 4 funkcije:

1. „processActivityData“ - pokreće obradu podataka
2. „setupViews“ – postavlja sve što elementi trebaju prikazivati korisniku
3. „setViewLayoutParams“ – postavlja vizualne detalje svakog elementa koji će se prikazivati korisniku
4. „insertActivityEntryToLayout“ – dodaje elemente u hijerarhijsku organizaciju i prikazuje ih korisniku

Na sljedećem isječku (*Isječak koda 4.23*) je prikazana metoda „processActivityData“. Ona analizira podatke iz „ActivityData“ objekta i sprema ih za prikaz korisniku. U ovoj metodi se računaju i postavljaju vrijednosti koje ce biti upisane u grafove za tempo i prijeđenu udaljenost.

**private void** processActivityData(ActivityData activity) {  
 **mActivityId** = activity.getActivityId();  
 **mActivityType** = activity.getActivityType();  
 **mDistanceEntries** = **new** ArrayList<>();  
 **mPaceEntries** = **new** ArrayList<>();  
 **mDistance** = 0;  
  
 ArrayList<TimePlace> data = activity.getTimePlaces();  
  
 **float** timeDiffSeconds, timePassed = 0;  
 **float**[] distanceResult = **new float**[1];  
 **double** prevPosLatitude = data.get(0).getLatitude();  
 **double** prevPosLongitude = data.get(0).getLongitude();  
 **long** prevPosTime = **mStartTime** = data.get(0).getTime();  
 **mDuration** = (**float**) (data.get(data.size() - 1).getTime() - **mStartTime**) / 1000;  
  
 **mDistanceEntries**.add(**new** Entry(0, 0));  
 **mPaceEntries**.add(**new** Entry(0, 0));  
  
 **for** (TimePlace it : data) {  
 timeDiffSeconds = (**float**) (it.getTime() - prevPosTime) / 1000;  
 **if** (timeDiffSeconds == 0)  
 **continue**;  
  
 Location.*distanceBetween*(prevPosLatitude, prevPosLongitude, it.getLatitude(),  
 it.getLongitude(), distanceResult);  
 **mDistance** = **mDistance** + distanceResult[0];  
 timePassed = timePassed + timeDiffSeconds;  
  
 **mDistanceEntries**.add(**new** Entry(timePassed, **mDistance**));  
 **mDistanceEntries**.add(**new** Entry(timePassed, **mDistance**));  
  
 **mPaceEntries**.add(**new** Entry(timePassed, (timeDiffSeconds / 60) / (distanceResult[0] / 1000)));  
  
 prevPosLatitude = it.getLatitude();  
 prevPosLongitude = it.getLongitude();  
 prevPosTime = it.getTime();  
 }  
}

*Isječak koda 4.23 Prikaz metode „processActivityData“*

Nešto što prije nije bilo spomenuto je da pri kreiranju elemenata, iako identifikacijska oznaka služi kao nešto pomoću čega možemo pristupiti određenom elementu, ona ne mora biti jedinstvena. Treba biti jedinstvena samo unutar dijela stabla elemenata nad kojim se vrši pretraga. Stoga je moguće dinamički kreiranim elementima postaviti bilo kakve vrijednosti za identifikacijsku oznaku.

Zato se identifikacijska oznaka svakog kreiranog „cardLayout“ elementa postavlja u vrijednost identifikacijske oznake aktivnosti u bazi podataka. Uz to se dodaje „onClickListener“ na gumb čija je uloga da prikaže tu rutu.

Budući da je gumb dijete od „cardLayout“ elementa, od gumba možemo navigirati do njega i dohvatiti njegovu identifikacijsku oznako kao što je prikazano na isječku (Isječak koda 4.24). Tu identifikacijsku oznaku šaljemo u aktivnost „RouteDisplay“.

**showRouteButton**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 showRoute(v);  
 }  
});

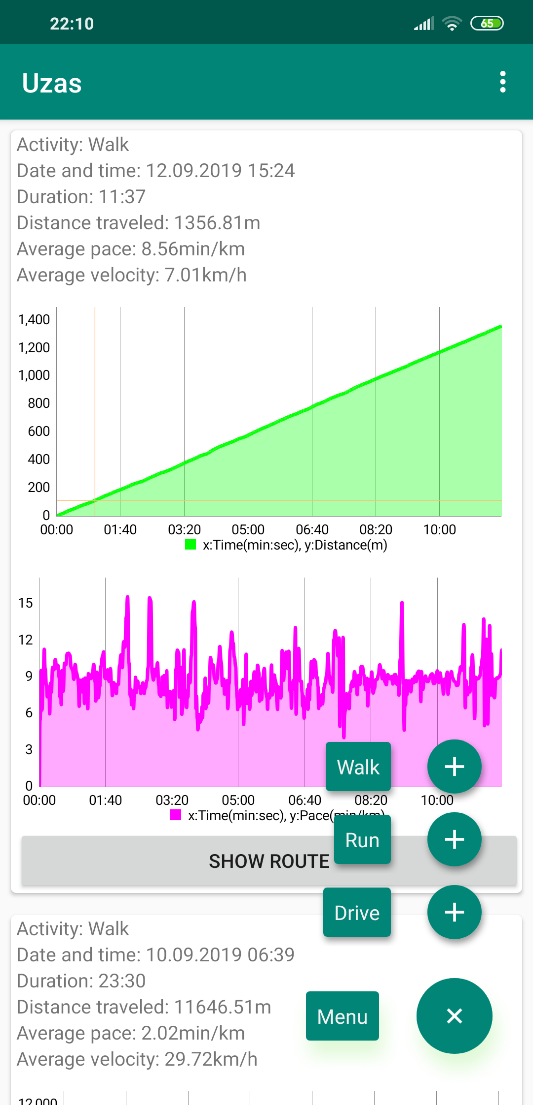
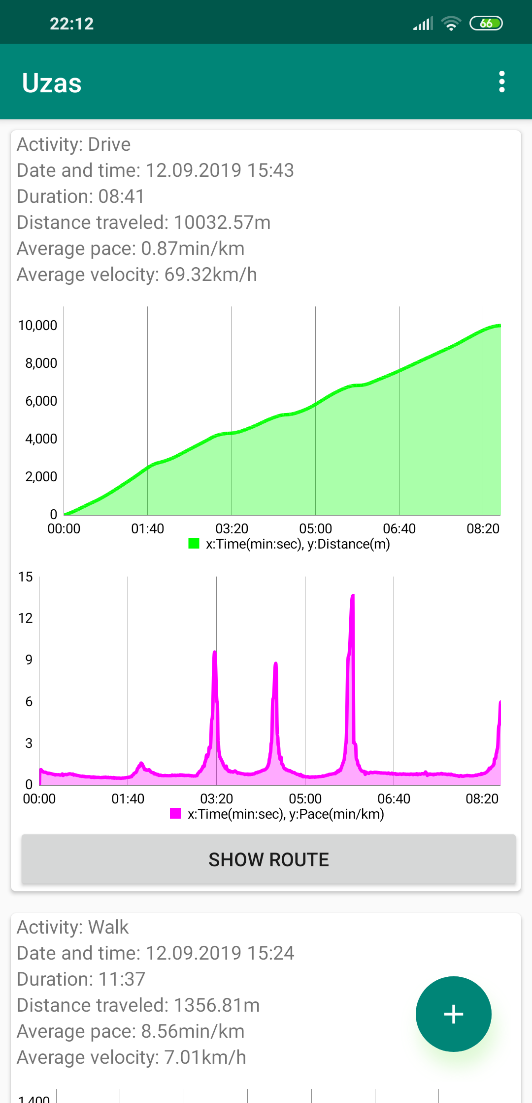
**public void** showRoute(View button) {  
 **int** activityId = ((View) button.getParent()).getId();  
  
 Intent intent = **new** Intent(**this**, RouteDisplay.**class**);  
 intent.putExtra(**"activityId"**, activityId);  
 startActivityForResult(intent, ***SHOW\_ROUTE\_REQUEST***);  
}

Isječak koda 4.24 Prikaz „onClickListener“ metode gumba i metode „showRoute“

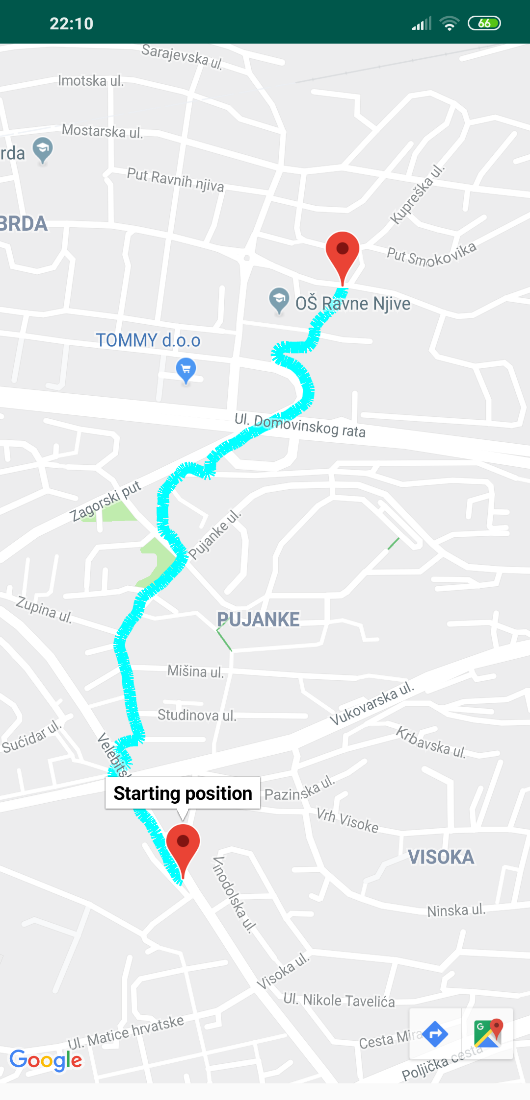
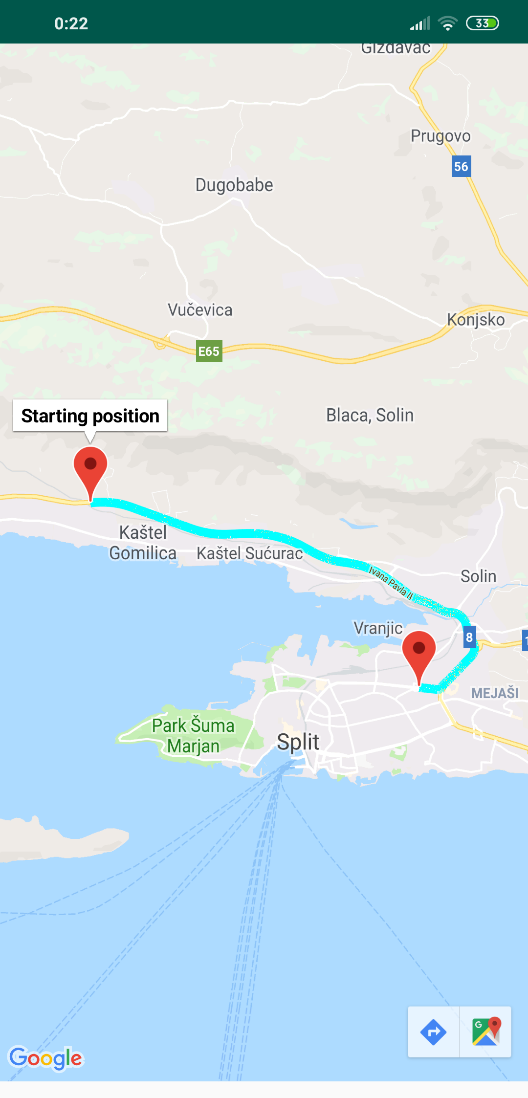
Metoda „showRoute“ se nalazi unutar klase „MainActivity“, a „onClickListener“ za gumb se postavlja unutar metode „insertActivityEntryToLayout“.

## Funkcionalnost aplikacije

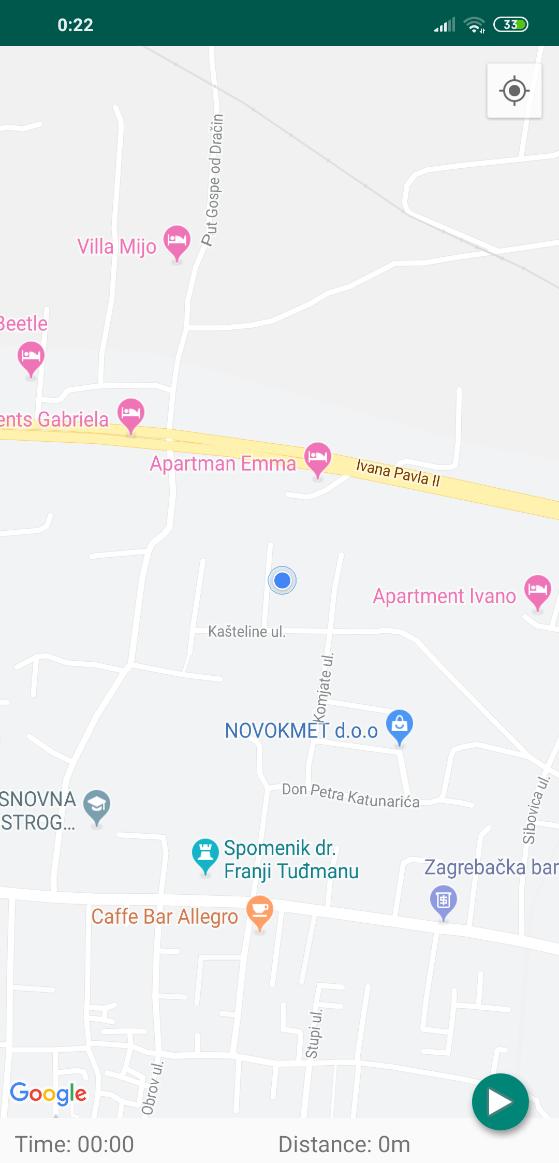
Konačni izgled aplikacije je prikazan kroz sljedećih nekoliko slika:

Slika 4.5 Dva prikaza ekrana s različitim tipovima aktivnosti i otvorenim/zatvorenim izbornikom

Slika 4.6 Prikaz rute aktivnosti za šetnju i vožnju



Slika 4.7 Prikaz ekrana pri ulasku u „LocationTracker“ aktivnost

# Gotova rješenja i biblioteke

## MPAndroidChart biblioteka

MPAndroidChart[8] je „open source“ projekt i biblioteka koja omogućava jednostavno crtanje raznih vrsta grafova i njihovo uređivanje.

## FloatingActionButton biblioteka

FloatingActionButton implementacija koja omogućava kreiranje menija u obliku gumbova implementiranih na donjem desnom rubu ekrana u glavnoj i aktivnosti za praćenje lokacije. Postoji službena biblioteka, međutim ova omogućava jednostavno kreiranje menija. Ova biblioteka se više ne razvija.[9]

## Vodič za pokretanje Google Maps lokacijskog servisa

Iako Android Studio sadrži predložak za pokretanje Google Maps lokacijskog servisa, to nije dovoljno da se omogući praćenje lokacije korisnika koristeći taj servis. Ovaj[10] vodič daje rješenje i objašnjenje za:

1. Zahtjev prema korisniku za dopuštenje pristupa lokaciji mobilnog uređaja
2. Postavljanje lokacijskog servisa da vraća trenutnu lokaciju mobilnog uređaja na Google Maps prikazu.

# Zaključak

Android je jedna od najpopularnijih mobilnih platformi, s nebrojivo mnogo dostupnih aplikacija na Google-ovoj trgovini aplikacijama. Mogućnost zaposlenja i napredovanja u industriji za razvijanje Android aplikacija nije upitna, kao ni samozapošljavanja, sve što treba imati je volje i upornosti za učiti.

Za nekog tko tek ulazi u područje razvijanja Android aplikacija, može se činiti kao da ste prevladani količinom informacija i mogućnostima tako da ne znate što ćete prije, ali nakon što ovladate osnovama uvidite da uopće nije složeno. Većina stvari je napravljena generički i da bude što jednostavnija za korištenje.

Naravno, uz neko predznanje sve to postaje još lakše, pa tako za one koji od prije već poznaju Java programski jezik, Android okolina im ne bi trebala predstavljati problem. Za one koji ne znaju, strpljenje i upornost su ključni.

Najsloženija funkcionalnost u ovom završnom radu je Google Maps API i periodičko dohvaćanje lokacije. To su stvari koje početnici u ovoj okolini ne bi znali napraviti, ali postoji odlična dokumentacija i na tisuće ljudi koji su spremni pomoći, tako da se odgovor na svako pitanje vezano uz ovu platformu može naći uz samo malo snalažljivosti. Ostatak aplikacije nije previše složen, ali pruža odličnu okolinu za učenje jednom početniku u tom okruženju. Sigurno postoji mnogo toga što se može poboljšati, ali moj cilj da naučim nešto novo, napravim nešto što ima svrhu i radi je sigurno ispunjen!

# Literatura

Sve informacije s literature su izvučene do 4.rujna.2019.

1. [https://www.android.com](https://www.android.com/)
2. [https://stackoverflow.com](https://stackoverflow.com/)
3. [https://www.quora.com](https://www.quora.com/)
4. <https://specialties.bayt.com/en/specialties/q/4553/why-do-we-use-java-for-creating-android-application/>
5. [https://kotlinlang.org](https://kotlinlang.org/)
6. <https://developer.android.com/reference/android/database/Cursor>
7. <https://developer.android.com/reference/android/app/Activity>
8. <https://github.com/PhilJay/MPAndroidChart>
9. <https://github.com/Clans/FloatingActionButton>
10. <https://demonuts.com/android-current-location-on-google-map/>
11. <https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient>

# Popis oznaka i kratica

|  |  |
| --- | --- |
| Kratica/oznaka | Značenje |
| Bottom-up | Pristup razvijanju nečega tako da se prvo definiraju jednostavni elementi, pa se od njih slažu cijeline |
| Engl. | Engleski |
| GPS | Global Positioning System |
| JVM | Java Virtual Machine |
| SQL | Structured Query Language |
| XML | Extensible Markup Language |
| Layout | Raspored elemenata koji se prikazuju na ekranu |

# Sažetak i ključne riječi

The thesis introduces an Android application designed to dynamically track a user's location. The application has the ability to display the appropriate parameters during the recorded successive time instances and is suitable for documenting the physical activities of users.

The paper minutely explains the approach that ensures the functionality of the application, describes the Google Maps localization service, the process of collecting and processing location data, creating and using a database, and explains how to dynamically create and display content on the device screen.

Key words: Android application, Location tracking, Processing location data related to physical activity, Processing GPS data, Google Maps API usage, Google location services usage

U ovom završnom radu predstavljena je Android aplikacija namijenjena dinamičnom praćenju lokacije korisnika. Aplikacija ima mogućnost prikazivanja odgovarajućih parametara tijekom zabilježenih slijednih vremenskih trenutaka i pogodna je za dokumentiranje fizičkih aktivnosti korisnika.

U radu se podrobno objašnjava pristup koji osigurava funkcionalnost aplikacije, opisuje se Google Maps loalizacijski servis, postupak prikupljanja i obrade podataka o lokaciji, stvaranje i korištenje baze podataka, te objašnjava kako dinamički stvoriti i prikazati sadržaj na zaslonu uređaja.

Ključne riječi: Android aplikacija, Praćenje lokacije, Obrada lokacijskih podataka vezanih za fizičkih aktivnosti, Obrada GPS podataka, Korištenje Google Maps API-ja, Korištenje Google lokacijskog servisa