Geolokacijske usluge u razvoju mobilnih aplikacija za Android platformu

Vratarić, Marina

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni preddiplomski studij računarstva

GEOLOKACIJSKE USLUGE U RAZVOJU MOBILNIH APLIKACIJA ZA ANDROID PLATFORMU

Završni rad

Marina Vratarić

Osijek, 2018.

SADRŽAJ

1. UVOD

Geolokacija se koristi za identifikaciju lokacije nekog uređaja, a njen oblik uključuje geografske koordinate (geografsku širinu i dužinu) koje omogućuju utvrđivanje adrese, najčešće pomoću GPS-a, ali i korištenjem nekih drugih tehnologija poput IP adrese, radio-frekvencije, MAC adrese, WiFi ili mobilne mreže, Bluetooth veze i slično. Geografska lokacija sastoji se od koordinata geografske širine i dužine pri čemu geokodiranje pretvara te koordinate u adresu i obratno. Adresa sadržava ime ulice i kućni broj, grad, poštanski broj i državu te kao takva predstavlja korisnu informaciju koja je ljudima razumljiva, za razliku od koordinata. Geolokacija se koristi u mnogim aplikacijama postajući vrlo popularan i skoro neizostavan dio svakodnevice, a može pomoći olakšati različite aktivnosti. Koristi se kao navigacijsko sredstvo za precizno kretanje do zadanog odredišta, Google koristi usluge temeljene na lokacijama za pohranu mjesta koja se posjete, služi u pronalasku traženog mjesta ili za prikaz vremenske prognoze na trenutnoj lokaciji bez dodatnog pretraživanja. Vrlo bitna stvar, u slučaju da se uređaj izgubi, je ta da se pomoću lokacije uređaj može pronaći. Tvrtke koje se bave online plaćanjem i kupovinom koriste geolokacijske usluge da bi otkrile i spriječile zlouporabu. Geo-ograda je virtualni ograničeni prostor koji se temelji na stvarnom geografskom prostoru. Pomoću toga se može znati je li korisnik unutar označenog prostora koristeći korisnikovu trenutnu lokaciju. Ovisno o tome jesu li *push* notifikacije postavljene, korisnik može dobiti obavijest za ulazak, odnosno izlazak iz nadgledanog prostora.

Cilj završnog rada je opisati mogućnosti primjene geolokacijskih usluga prilikom razvoja aplikacija za Android platformu. Osim toga, cilj je izraditi Android mobilnu aplikaciju koja korisniku omogućava korištenje geo-ograde kojom će moći pratiti i nadgledati označeno područje. Uz to, korisniku je dana mogućnost upisa željenog radijusa za označavanje područja geo-ograde te pretraživanje mjesta sa informacijom o lokaciji. Potrebno je da lokacija sadržava dva zapisa, jedan zapis u koordinatama geografske širine i dužine te zapis adrese.

Drugo poglavlje rada odnosi se na teorijski dio u kojem se detaljno obrađuju pojmovi poput Android platforme, gradivnih jedinica, određivanje lokacije, pretvaranje koordinata u adresu (i obratno), geo-ograde i karte. Opisane su različite usluge koje su učestale pri korištenju razvoja programske podrške kao i mogućnost njihove uporabe prilikom korištenja. Treće poglavlje prikazuje praktičnu primjenu izrade Android mobilne aplikacije uz prikaz bitnih segmenata koda kao i opis korištenih

alata i tehnologija. U posljednjem poglavlju dan je zaključak ovog završnog rada s osvrtom na ostvareno te smjernicama za moguće nadogradnje.

1.1. Zadatak završnog rada

Cilj završnog rada jest opisati mogućnost primjene geolokacijskih usluga prilikom razvoja aplikacija za Android platformu. U teorijskom je dijelu rada potrebno opisati različite geolokacijske usluge koje se često koriste prilikom razvoja programske podrške, kao i različite mogućnosti njihove uporabe te dobre prakse prilikom njihova korištenja. U praktičnom je dijelu rada nužno ostvariti programsko rješenje za Android platformu koje se oslanja na geolokacijske usluge opisane u teorijskom dijelu.

2. GEOLOKACIJSKE USLUGE

Geolokacija služi za identifikaciju lokacije nekog uređaja, a njen oblik uključuje geografske koordinate (geografsku širinu i dužinu) koje omogućuju utvrđivanje adrese. Korištenje geolokacije omogućava identificirati korisnikovu trenutnu poziciju, detektirati kretanje, služi za ciljano slanje različitih vrsta podataka korisnicima koji su u blizini određenog mjesta, za provjeru lokacije javnog prijevoza, zatim se koristi za pretragu mjesta u blizini trenutne korisnikove lokacije i za brojne druge primjene. Aplikacije poput kamere, vremenske prognoze, Google karte, aplikacije za pretragu mjesta kao što je Yelp, Letgo aplikacija, zatim alarmi zasnovani na lokaciji, društvene mreže poput Facebook-a, „Health & Fitness" aplikacije, neke igrice poput Pokemon Go, aplikacije za putovanja kao što je Booking.com Hoteli samo su neki od brojnih primjera aplikacija koje koriste geolokacijske usluge.

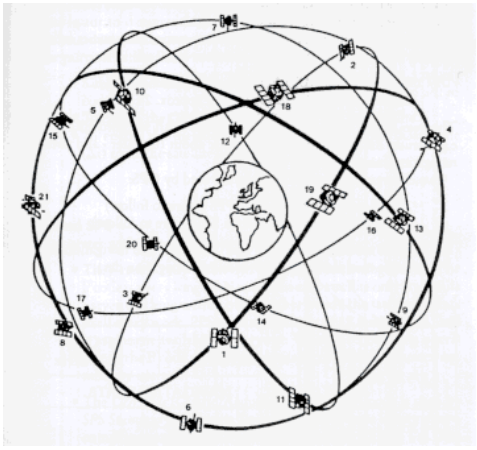
Prema [1], Android koristi tri pružatelja lokacije, a posljednji (hibridni pružatelj) predstavlja njihovu kombinaciju:

- GPS pružatelj
- mrežni pružatelj
- pasivni pružatelj
- hibridni pružatelj

GPS pružatelj lokacije dohvaća lokaciju koristeći satelite te koristi GPS čip koji je ugrađen u uređaju. Ako se nalazi na mjestima gdje GPS uređaj neće primiti dovoljno signala od satelita, tada ne daje najbolje rezultate zbog smetnji. Nedostatak je velika potrošnja baterije, a ovisno o signalu i vezi, točnost može biti i do nekoliko metara. Mrežni pružatelj lokacije utvrđuje lokaciju temeljenu na WiFi pristupnim točkama i repetitoru. Zahtijeva vrlo malu potrošnju baterije uređaja te daje dobre rezultate preciznosti lokacije. Podaci se mogu dobiti putem nekih signala kao što je Bluetooth, WiFi ili pomoću IP adrese. Pasivni pružatelj lokacije koristi pružatelje drugih aplikacija za stvaranje podataka o lokaciji. Ne zahtijeva veliku potrošnju baterije, ali ova tehnologija nije toliko točna kao prethodno navedene. Iako je GPS puno točniji i precizniji od drugih tehnologija, katkada nije dostupan.

Prema [2], postoji i hibridni (engl. *fused*) pružatelj koji kombinira tehnologije poput GPS-a i WiFi-a mijenjajući ih automatski kada je to potrebno. Daje najbolje rezultate preciznosti s gotovo nikakvom dodatnom potrošnjom kapaciteta baterije.

Sustavi pozicioniranja se mogu podijeliti na lokalne i globalne. Prema [3], satelitski navigacijski sustavi sa globalnom pokrivenošću (engl. *Global Navigation Satellite System, GNSS*) su ruski GLONASS, kineski BDS, europski Galileo i najkorišteniji S.A.D.-ov GPS. Globalni sustav pozicioniranja (GPS) potpuno je funkcionalni navigacijski sustav koji daje rezultate geolokacije sa odmakom od samo nekoliko metara, a sastoji se od 32 satelita postavljenih u šest različitih orbitalnih ravnina. Iako daje vrlo precizne rezultate, oko visokih zgrada zna biti ometan, jer se signal GPS uređaja može podijeliti na više signala, umjesto na samo jedan. Drveća, tuneli, planine, zgrade, područja ispod zemlje i slično mogu spriječiti GPS signale da dođu do prijemnika. Tri glavne komponente su GPS sateliti, GPS prijemnici i softver koji dekodira signale i izračunava geografsku poziciju korisnika. Sateliti se nalaze u orbiti na visini od oko 20 000 kilometara iznad zemlje, a na slici 2.1. prikazan je položaj satelita.



Slika 2.1. GPS sateliti, [3]

GLONASS, za razliku od GPS-a, ima 24 satelita koji dijele tri orbite. Jedina prednost GLONASS-a nad GPS-om je preciznost. Ne pokriva tako veliki prostor, ali kada se koristi zajedno s GPS-om, povećava se i preciznost i pokrivenost. Razlikuje se od GPS-a po načinu kontrole i po strukturi signala.

Galileo je namijenjen da omogući horizontalna i vertikalna mjerenja lokacije sa preciznosti unutar jednog metra i s boljim sustavom za pozicioniranje na višim geografskim širinama i dužinama.

Lokalni sustavi pozicioniranja dohvaćaju signale na mjestima na kojima globalni sustavi ne mogu, poput unutrašnjosti zgrade. Takva mjesta moraju biti obuhvaćena mrežom. Prema [4], lokalni sustavi koriste ograničene mreže poput Bluetooth-a, WPS-a (engl. *WiFi-based Positioning Systems*), UWB-a (engl. *Ultra-wideband*) i radio-frekvencijske identifikacije (RFID).

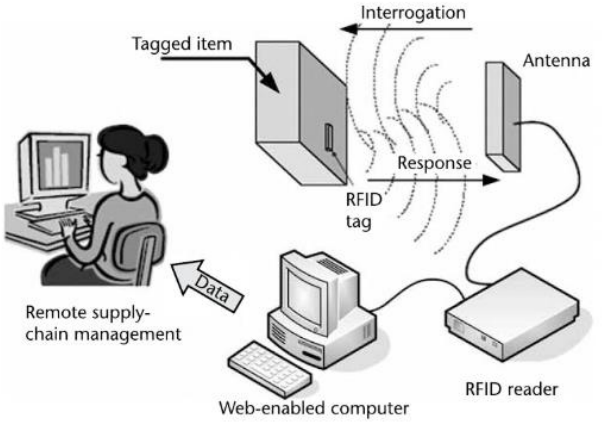
Bluetooth tehnologija se najčešće koristi za bežičnu komunikaciju kratkog dosega te ne daje preciznu i točnu, već približnu lokaciju koristeći Bluetooth odašiljače. Odašiljači su mali uređaji koji nemaju ugrađenu lokacijsku inteligenciju. Oni šalju Bluetooth signale, a uređaji koji se nalaze u blizini ih primaju, vidljivo na slici 2.2, izrađenoj prema [4]. Koristi kratke valne duljine radioprijenosa u pojasu od 2400 do 2480 Mhz od uređaja stvarajući mrežu (PAN) sa visokom razinom sigurnosti.

Slika 2.2. Princip rada Bluetooth tehnologije s odašiljačem, [4]

WPS se bazira na WLAN (engl. *Wireless Local Area Network*) infrastrukturi. WPS koristi snagu signala pomoću WLAN odašiljanja od/do WLAN pristupne točke radi utvrđivanja lokacije korisnika, a koristi se kada je GPS neadekvatan. Preciznost određivanja lokacije varira između 8 i 15 metara. Ova tehnologija omogućava praćenje svih uređaja koji imaju WiFi.

Prema [5], UWB tehnologija je radio tehnologija malog dometa koja pruža točnost manju od 30 centimetara, a koristi se za praćenje, pozicioniranje i komunikaciju. Bazira se na odašiljanju jako kratkih impulsa i koristi tehnike koje uzrokuju širenje radio energije sa vrlo malom spektralnom gustoćom snage. Pojasna širina daje visoku propusnost podataka za komunikaciju. Niska frekvencija UWB impulsa omogućava signalima prolazak kroz prepreke poput zida i objekata.

Prema [6], RFID (engl. *Radio-frequency identification*) tehnologija je bazirana na bežičnoj komunikaciji koristeći radio valove za razmjenu podataka. Sastoji se od tri glavne komponente, a to su transponder, čitač i antena. Transponder se sastoji od mikročipa i antene koji se koriste za prijenos podataka u RFID sustavu. Čitač može biti u mogućnosti i čitati i pisati podatke transponderu pretvarajući radio valove u korisne podatke. Čitač prenosi signale koji su integrirani s malim čipom koje antena prima. Čip je aktiviran samo kada ga čitač skenira, a ako je aktiviran šalje jedinstveni identifikacijski broj kojeg čitač dalje prosljeđuje programima. Osnovni koncept funkcioniranja RFID tehnologije i komponente prikazane su na slici 2.3.



Slika 2.3. Osnovne komponente RFID sustava, [6]

Prema [7], udaljenost između dvije lokacije računa se prema jednadžbi (2-1), odnosno (2-2), gdje je: *d* - udaljenost, φ-geografska širina, λ - geografska dužina, *R*- radijus Zemlje.

$$a = \sin^2(\frac{\Delta\varphi}{2}) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2(\frac{\Delta\lambda}{2}) \tag{2-1}$$

$$d = 2 \cdot R \cdot \operatorname{atan2}(\sqrt{a}, \sqrt{(1-a)}) \tag{2-2}$$

2.1. Android platforma

Prema [8], Android je operacijski sustav otvorenog koda čiji je softver baziran na Linux jezgri. Platforma je otvorena i besplatna te je dizajnirana prvenstveno za mobilne uređaje. Većina aplikacija koje rade na Android platformi pisana je u Java programskom jeziku. Aplikacije mogu biti pisane i u Kotlinu koji je postao službeni jezik Androida te u C++ programskom jeziku. Za pohranu podataka upotrebljava se *SQLite* pisan u C programskom jeziku. Takvo spremanje u bazu podataka idealno je

za ponavljajuće i strukturirane podatke. *SQLite* je ugrađen u skoro svaki Android uređaj. Njegovo korištenje ne zahtjeva podešavanje niti administraciju baze podataka.

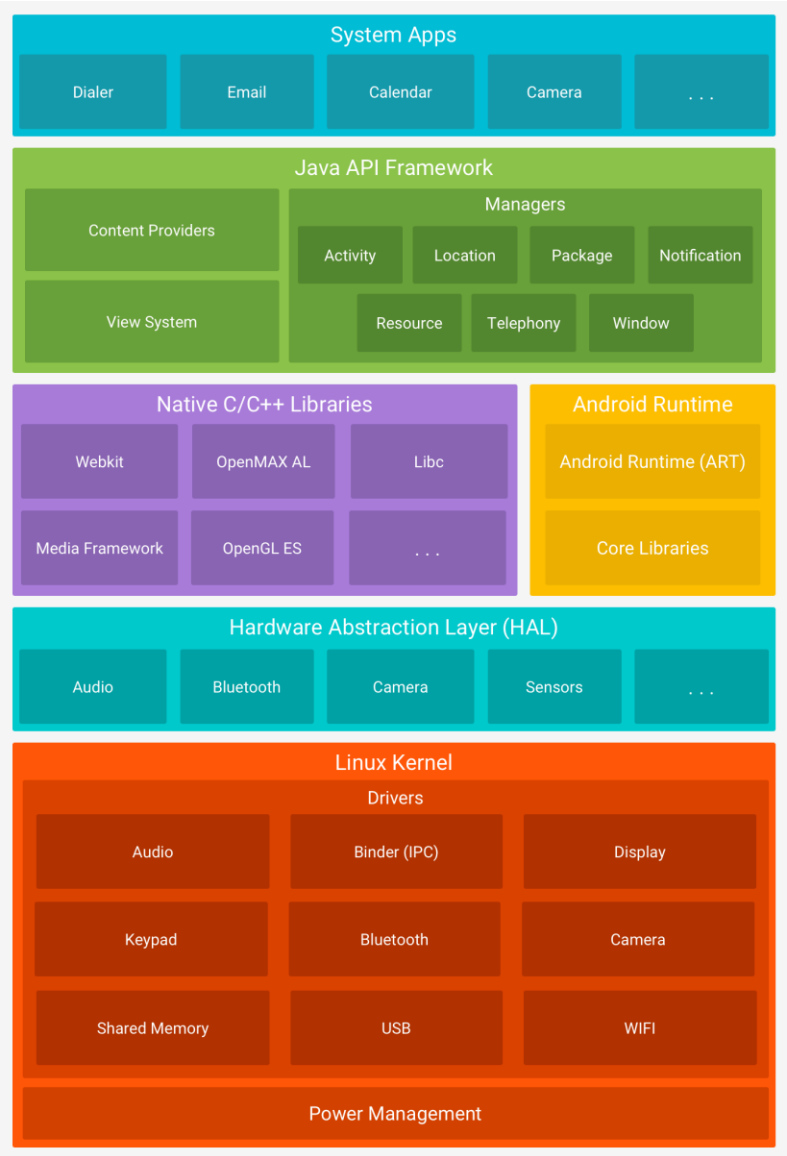
Prema [9], Android platforma podržava tri kategorije senzora, a to su senzori pokreta, okoliša i položaja. Neki od tih senzora zasnovani su na stvarnom hardveru poput žiroskopa, senzora za tlak i vlažnosti, dok su drugi bazirani na osnovu softvera kao što su senzori za vektor rotacije ili gravitacije. Senzori pokreta mjere ubrzanje sile i rotacijske sile uz pomoć tri osi. Za takvu metodu se koriste akcelerometri, gravitacijski senzori, žiroskopi i senzori rotacijskih vektora. S druge strane, senzori okoliša mjere razne parametre iz okoliša kao što su temperatura zraka, tlak, vlažnost zraka i slično. Za dobivanje podataka ovakvog tipa senzora potrebni su barometri, fotometri i termometri. Senzori položaja mjere fizičku poziciju, odnosno lokaciju uređaja uz pomoć orijentacijskih senzora i magnetometra.

Prema [10], Android platforma se sastoji od šest osnovnih komponenti, a to su:

(1) sustavne aplikacije (engl. *System Apps*)
(2) Java API okvir (engl. *Java API Framework*)
(3) izvorna C/C++ biblioteka
(4) Android izvršna okolina (engl. *Android Runtime*)
(5) sloj apstrakcije hardvera (engl. *Hardware Abstraction Layer*, HAL)
(6) Linux jezgra

Na slici 2.4. prikazani su slojevi Android platforme, koji se mogu podijeliti na:

(1) Sustavne aplikacije poput kalendara, kamere, email-a, SMS poruka, kontakta, Internet pretraživanja i slično su aplikacije koje su preinstalirane na uređaju i korisnici nemaju pristup toj sistematskoj particiji. Za razliku od korisničkih aplikacija, sustavne aplikacije korisnici ne mogu deinstalirati.
(2) Java API (engl. *Application Programming Interface*) okvir omogućava programerima jednostavno pisanje aplikacije za Android uređaje. Android operacijski sustav je dostupan kroz API-je. Oni stvaraju gradivne jedinice koje su potrebne za kreiranje aplikacije i pisani su u Java programskom jeziku.



Slika 2.4. Android platforma, [10]

Java API okvir sastoji se od:

- pogleda (engl. *View System*) koji sadržava gradivne jedinice kao što su *Button, EditText, Image Panes, List/Grid View, CheckBox, RadioButton* i drugo
- pružatelja sadržaja (engl. *Content Providers*) koji omogućava pristup informacijama i dijeljenje istih
- rukovatelja resursima (engl. *Resource Manager*) za pristup datotekama koje nisu dio projekta (*layouti* i slično)

- rukovatelja obavijestima (engl. *Notification Manager*) koji omogućava prikaz obavijesti na status baru
- rukovatelja aktivitima (engl. *Activity Manager*) koji upravlja životnim ciklusom i navigacijom aplikacije

(3) Izvorna C/C++ biblioteka je biblioteka koja dopušta korištenje C/C++ koda sa Androidom. Ako aplikacija zahtijeva C ili C++ kod, može se koristiti Android NDK (engl. *Native Development Kit*) za pristup nekim izvornim bibliotekama direktno iz koda.
(4) Android izvršna okolina (ART) pokreće brojne virtualne strojeve na nisko-memorijskim uređajima. Za uređaje koji podržavaju verziju Androida 5.0 ili noviju, svaka aplikacija pokreće svoj proces sa svojom instancom ART-a, dok je za starije verzije korišten Dalvik kao ART.
(5) Sloj apstrakcije hardvera (HAL) pruža sučelje koje izlaže hardverske sposobnosti uređaja na višu razinu API okvira (audio, Bluetooth, kamera, senzori i drugo).
(6) Linux jezgra temelj je Android platforme koja se sastoji od pogonskih programa poput WiFi-a, USB-a, kamere te od upravljanja energijom (engl. *Power Management*).

2.2. Gradivne jedinice

Prema [11], neke od gradivnih jedinica Android aplikacije su:

(1) *Activity*
(2) *Views*
(3) *Services*
(4) *Content Providers*
(5) *Sqlite DataBase*
(6) *Intents*

(1) *Activity* predstavlja svaki zaslon u Android aplikaciji, a sastoji se od pogleda (engl. *View*). *Activity* pruža prozor, koji najčešće popunjava cijeli zaslon, u kojem aplikacija oblikuje svoje korisničko sučelje. Općenito, jedan *Activity* implementira jedan zaslon u aplikaciji. Svaki puta kada se stanje *Activity*-a promijeni, poziva se jedna od metoda životnog ciklusa (*onCreate(), onStart(), onResume(), onPause(), onStop(), onRestart(), onDestroj()*). Ako postoji više *Activity*-a koji se pokreću jedan za drugim, potrebno ih je povezati metodom *startActivity()* ili

startAcitivityForResult() te proslijediti *Intent* koji ih pokreće. Metoda *startAcitvity()* prikazana je u izlistanju koda 1.

```
Intent intent = new Intent(this, SecondActivity.class);
startActivity(intent);
```

Izlistanje koda 1. Pokretanje novog *Activity*-a

Ukoliko se žele koristiti vrijednosti i podaci iz jednog *Activity*-a u drugi koji se prosljeđuje, koristi se metoda *startAcitivityForResult()* koja je prikazana u izlistanju koda 2.

```
onActivityForResult(int requestCode, int resultCode, Intent data)
```

Izlistanje koda 2. Prosljeđivanje podataka između *Activity*-a

(2) *View* je osnovna gradivna jedinica za dijelove korisničkog sučelja. Omogućava prikaz ili unos teksta, slike i drugog sadržaja. Predstavlja male programe (engl. *widget*) koji se pojavljuju na zaslonu omogućavajući interakciju s korisnikom. Svaka gradivna jedinica se mora identificirati i povezati pomoću unikatnog ID-a, a za jednostavan primjer korišten je *EditText* čiji je ID naziva *my_editText* definiran u *layout*-u koji je prikazan u izlistanju koda 3

```
<EditText
        android:id="@+id/my_editText"
        android:hint="Enter some number"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
```

Izlistanje koda 3. Stvaranje *EditText*-a

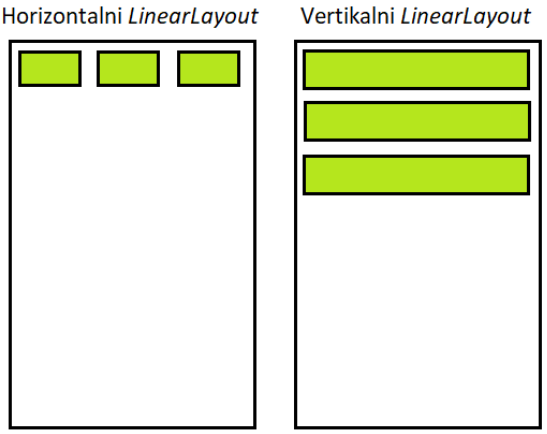
i u *onCreate()* metodi koja je prikazana u izlistanju koda 4.

```
EditText et = findViewById(R.id. my_editText);
```

Izlistanje koda 4. Povezivanje *EditText*-a pomoću unikatnog ID-a

Svaki *View* je oblika četverokuta i ima lokaciju definiranu koordinatama te dvjema dimenzijama predstavljenima kao širina i visina. Hijerarhija se grana od glavne klase *View* na osnovne podklase *ImageView, TextView* i *ViewGroup.*

- *ImageView* se sastoji od *ImageButton*-a, *QuickContactBadge*-a, *ZoomButton*-a i drugih elemenata.
- *TextView* ima elemente poput *Button*-a, *EditText*-a, *CheckBox*-a, *RadioButton*-a, *DigitalClock*-a.
- *ViewGroup* sadržava *layout*-e kao što su *LinearLayout*, *RelativeLayout*, *ConstraintLayout*, *FrameLayout, GridLayout, CoordinatorLayout. Layout* organizira elemente na zaslonu za stvaranje pogleda (engl. *View*). *LinearLayout* smješta sve elemente u jedan smjer, horizontalni ili vertikalni, vidljivo na slici 2.5. izrađenoj prema [11].



Slika 2.5. Horizontalni i vertikalni smjer *layout*-a, [11]

U *RelativeLayout*-u pozicija svakog pogleda može biti određena relevantno sa susjednim elementima ili s roditeljem. *FrameLayout* omogućava stavljanje jednog elementa preko drugog.

Tri su osnovna atributa elementa:

- veličina (engl. *size*)
- gravitacija (engl. *gravity*)
- podstava/margina (engl. *padding/margin*)

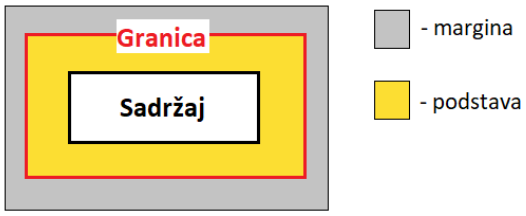
Svakom elementu se može dodijeliti veličina na tri različita načina, odnosno prema veličini roditelja (engl. *match_parent*), prema veličini sadržaja (engl. *wrap_content*) i određenim brojem. Atribut *match_parent* podrazumijeva da element uzima istu visinu i širinu od roditelja, *wrap_content* označava da se granica elementa obavija oko svog sadržaja, dok se broju, koji je mjeren u dp (engl. *Density-Idenpendent Pixel*), pridodaje željena vrijednost.

Atribut gravitacije služi za pozicioniranje elementa tako da se postroji u smjerovima gore, dolje, lijevo ili desno. Ako je element smješten u gorenjem lijevom kutu, tada se dodaju atributi kao na izlistanju koda 5:

```
android:layout_alignParentTop = „true”
android:layout_alignParentBottom = „false”
android:layout_alignParentLeft= „true”
android:layout_alignParentRight = „false”
```

Izlistanje koda 5. Dodavanje atributa za smještanje elementa u gornji lijevi kut

stavljajući vrijednost *true* za *alignParentTop* i *alignParentLeft,* odnosno gornji i lijevi dio *layout*-a. Margina je prostor između okvira i roditeljskog *layout*-a, a podstava se nalazi između sadržaja i granice što je vidljivo na slici 2.6. izrađenoj prema [11].



Slika 2.6. Granica između margine i podstave, [11]

(3) *Services* su komponente koje izvode automatsku obradu bez vidljivog korisničkog sučelja. Osmišljene su tako da neprimjetno rade dugoročne radnje u pozadini. Primjeri usluga su rukovanje mrežnim transakcijama, puštanje glazbe, email servis i drugo. Postoje tri tipa servisa:

- Prednji plan (engl. *Foreground*) - plan u kojem se izvode radnje vidljive korisniku, a one nastavljaju raditi iako korisnik ništa ne radi s aplikacijom, servis prikazuje obavijest (audio aplikacije)
- Pozadinski plan (engl. *Background*) - plan koji obavlja radnje koje korisnik ne može direktno vidjeti
- Granica (engl. *Bound*) - servis je povezan kada se aplikacijska komponenta poveže s njim

(4) *ContentProviders* omogućava dijeljenje podataka kroz aplikaciju (audio, video, slike, kontakti i drugo). Može pomoći aplikaciji pri rukovanju pristupanjem podacima koje je sama pohranila ili koje su pohranile druge aplikacije.

(5) *Sqlite DataBase* se koristi u uređajima s vrlo malom memorijom za dohvaćanje i spremanje podataka. Brzo, učinkovito i sigurno pohranjivanje i dohvaćanje podataka je ključno za uređaje sa ograničenim kapacitetom memorije.

(6) *Intent* je objekt koji predstavlja apstraktan opis povezivanja komponenti Android aplikacija. Služi za pokretanje *Activity*-a, zatim za pokretanje servisa poput email-a, web preglednika i slično te za slanje podataka. Može se koristiti metodom *startActivity* za pokretanje novog *Activity*-a, zatim *broadcastIntent* metodom za slanje *Intent*-a *BroadcastReceiver*-u (npr. u slučaju da je baterija uređaja slaba, druge aplikacije će vidjeti poruku putem *BroadcastReceiver*-a) ili za komunikaciju sa pozadinskim servisima pomoću *startService(Intent)* metode. Omogućava interakciju s komponentama iste ili s komponentama druge aplikacije te na taj način pruža mogućnost pristupa ostalim aplikacijama. Intent može biti:

- eksplicitni
- implicitni

Eksplicitni *Intent* komunicira između dva *Activity*-a unutar iste aplikacije, dok implicitni komunicira između različitih aplikacija.

2.3. Određivanje lokacije na Android platformi

Prema [12], servisi za lokaciju na Android platformi pružaju pristup sredstvima koji se mogu koristiti za određivanje lokacije nekog uređaja. Android koristi razne metode za određivanje informacija o lokaciji. Sredstva koja se koriste nazivaju se pružatelji lokacije (engl. *location providers*), a svaki od njih ima svoje prednosti i mane. Pružatelji lokacije imaju jedinstvene karakteristike koje se koriste u različitim situacijama. Tri dostupna pružatelja lokacije na Android platformi su:

- GPS pružatelj lokacije (engl. *Global Positioning System Location Provider*)
- Mrežni pružatelj lokacije (engl. *Network Location Provider*)
- Pasivni pružatelj lokacije (engl. *Passive Location Provider*)

Aplikacija može ustvrditi kojeg će pružatelja lokacije koristiti na dva načina: eksplicitnim registriranjem svakog pružatelja lokacije sa *LocationManager*-om ili specificiranjem atributa u

objektu *Criteria* i predavanjem tog objekta *LocationManager*-u. Korištenje tog objekta je korisno jer omogućava korisnicima da prilagode izvor podataka lokacije za vrijeme izvođenja.

GPS pružatelj lokacije koristi satelite i vrijeme za utvrđivanje trenutne lokacije uređaja nastojeći stvoriti najtočniji podatak lokacije. No, može koristiti više baterije nego drugi pružatelji lokacije. To može predstavljati veliki problem s obzirom na duljinu vremena koju aplikacija treba za primanje i procesiranje podataka lokacije. Također, ponekad može proći puno vremena za stjecanje podataka za lokaciju. Uz to, mala je vjerojatnost za dobivanje podataka o lokaciji unutar zgrade jer je potrebna direktna linija do neba.

Mrežni pružatelj lokacije koristi dva podatkovna izvora za dobivanje podataka, a to je lokacija Wi-Fi mreže i lokacija repetitora. Vrijeme za prvo dobivanje lokacije je kraće nego kod GPS pružatelja, ali mrežni pružatelj daje puno manju točnost. Uz to, koristi i manje baterije jer omogućava korisnicima gašenje GPS i Wi-Fi radija.

Pasivni pružatelj lokacije omogućava aplikaciji primanje informacija o lokaciji bez eksplicitnog zahtijeva o lokaciji od *LocationManager*-a. Pruža lokaciju kada druga aplikacija zahtijeva ažuriranje lokacije koristeći GPS ili mrežnog pružatelja. Zahtijeva korištenje precizne lokacije (*android.permission.ACCESS_FINE_LOCATION*) tako da se podaci iz GPS i mrežnog pružatelja mogu primiti. No, ako niti jedna druga aplikacija ne ažurira lokaciju, pasivni pružatelj neće primiti niti jedan podatak.

Unutar *AndroidManifest.xml* datoteke definira se preciznost podataka lokacije koja će biti zahtijevana. Ta dopuštenja su *android.permission.ACCESS_FINE_LOCATION* i *android.permission.ACCESS_COARSE_LOCATION* i ona definiraju razinu točnosti koja će biti omogućena aplikaciji od lokacijskog servisa. Kako *ACCESS_FINE_LOCATION* omogućava precizniju lokaciju, može se koristiti bez specificiranja *ACCESS_COARSE_LOCATION* koji daje manje precizne podatke, ali obratno ne vrijedi. Te se dozvole definiraju u manifest datoteci kao u izlistanju koda 6.

```
<uses-permission android:name = ”android.permission.ACCESS_FINE_LOCATION” />
```

Izlistanje koda 6. Definiranje visoke preciznosti podataka lokacije

Prema [13], podrška za servise bazirane na lokaciji na Android platformi sastoji se od *Location Manager* servisa*, Geocoding* servisa*, Google Play API* servisa i nekoliko drugih servisa koji su

uključeni u platformi. Međutim, puno bolji izbor predstavlja *Google Play API* lokacijski servis jer automatski odabire kojeg će pružatelja primijeniti temeljeno na preciznosti dohvaćanja lokacije, potrošnji baterije i slično. Puno je brži te se mogu koristiti dodatne napredne značajke poput geo-ograda. Kako bi se koristio Android *Location Manager* servis, mora se osigurati *Location Provider* i *location listener*. Tehnologije koje *Location Provider* upotrebljava za dohvaćanje su GPS, WiFi te Cell-ID. *LocationManager* koristi Android lokacijske API-je koje se uvoze u Java datoteku kao u izlistanju koda 7.

```
import android.location.LocationListener;
```

Izlistanje koda 7. Umetanje lokacijskog API-ja

Lokacija je zahtijevana od *LocationManager*-a pozivanjem metode *requestLocationUpdates()*. U izlistanju koda 8 prikazano je pozivanje *LocationManager*-a s jednostavnim *location listener*-om.

```
LocationManager locationManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);
locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,0,0,location
Listerner);
```

Izlistanje koda 8. Povezivanje *LocationManager*-a s *location listener*-om

Prvi parametar koji se poziva je vrsta pružatelja lokacije, zatim minimalni vremenski interval između obavijesti i minimalna promijena u udaljenosti između obavijesti. Zadnji parametar je *LocationListener* koji prima poziv za povrat od ažuriranja lokacije.

U *Geocoding* servisu, metoda *getFromLocationName()* omogućava naziv lokacije (*locationName*) , specificira broj maksimalnih rezultata (*maxResults*) te geografske koordinate, širinu i dužinu. Primjer korištenja je dan u izlistanju koda 9.

```
Public List<Address > getFromLocationName(String locationName, int maxResult, double
lowerLeftLatitude, double lowerLeftLongitude, double upperRightLatitude, double
upperRightLongitude)
```

Izlistanje koda 9. Primjer korištenja metode *getFromLocationName()*

Za korištenje lokacije u aplikaciji, može se koristiti i postojeći Google Play servis koji koristi Google API lokacijske servise. Za omogućavanje API-a, u *build.gradle* datoteci aplikacijskog modula dodaje se primjer koda kao u izlistanju koda 10.

```
dependencies {
        compile 'com.google.android.gms:play-services-maps:15.0.1'
        compile 'com.google.android.gms:play-services location:15.0.1'
            }
```

Izlistanje koda 10. Dodavanje biblioteka za korištenje Google API lokacijskih servisa

U *AndroidManifest.xml* datoteci, unutar <*application*> elementa dodaje se dio koda koji je vidljiv u izlistanju koda 11.

```
<meta-data
    android:name = ”com.google.android.gms.version”
    android:value = ”@integer/google_play_services_version” />
```

Izlistanje koda 11. Dodavanje verzije Google Play servisa

U Java datoteku uvodi se primjer koda prikazan u izlistanju koda 12.

```
import com.google.android.gms.location.LocationListener;
```

Izlistanje koda 12. Umetanje klase *LocationListener*

Njihovim korištenjem aplikacija može zatražiti zadnju poznatu lokaciju korisnikovog uređaja, koja je najčešće ekvivalentna trenutnoj korisnikovoj lokaciji.

2.4. Prevođenje iz koordinata u adresu

Prema [14], geokodiranje je proces pronalaska geoografskih koordinata (geografske širine i dužine) od dane adrese ili lokacije, dok je obrnuto geokodiranje proces suprotan navedenome, odnosno prevođenje koordinata u adresu ili lokaciju. Za geokodiranje, odnosno obrnuto geokodiranje koristi se ugrađena klasa *Geocoder* koja se može umetnuti na način prikazan u izlistanju koda 13, a primjer korištenja iste klase dan je u izlistanju koda 14.

```
import android.location.Geocoder;
```

Izlistanje koda 13. Umetanje klase *Geocoder*

```
Geocoder geokoder = new Geocoder (this);
    if(geokoder.isPresent()) {
        List <Address> lista = geokoder.getFromLocationName("Ulica Kneza Trpimira 2B, 31000,
        Osijek, Hrvatska", 1);
                Address adresa = lista.get(0);
                double širina = adresa.getLatitude();
                double dužina = adresa.getLongitude();
                textView.setText(String.valueOf(širina) + ",  " + String.valueOf(dužina));
}
```

Izlistanje koda 14. Primjer stvaranja klase *Geocoder* za ispis koordinata

Ukoliko koordinate odnosno adresa nije pronađena, kao rezultat se vraća lista rezultata koja će biti prazna. Kod obrnutog geokodiranja radi se o listi objekata *Address* klase koji sadržava informacije o lokaciji. Detaljan opis lokacije u obrnutom gekodiranju može varirati, tako da može sadržavati samo informacije o gradu i poštanskom broju pa sve do zapisa potpune fizičke adrese. Nakon što se obavi provjera je li *Geocoder* u sustavu, moguće je predati mu lokaciju i zatražiti dohvaćanje adresa, a taj je zahtjev i čekanje na odgovor potrebno obaviti na drugoj niti. Metoda *getFromLocationName()* vraća polje adresa koje opisuju lokaciju uz postavljanje broja maksimalnog rezultata (1), a taj broj rezultata treba biti što manji. Pozivanjem metode *lista.get(0)* uzima se prvi elementi iz liste i sprema ga se u adresu. Korištenjem metode *valueOf()* pretvara se vrijednost varijabli *širina* i *dužina*, koje su tipa *double,* u *string* kako bi se koordinate mogle ispisati. Na taj se način dobije prikaz koordinata adrese „Ulica Kneza Trpimira 2B, 31000, Osijek, Hrvatska".

U izlistanju koda 15 prikazan je primjer obrnutog geokodiranja.

```
    Geocoder geokoder = new Geocoder (this);
    if(geokoder.isPresent()) {
        List <Address> lista = geokoder.getFromLocation(45.556814, 18.695803, 1);
        Address adresa = lista.get(0);
        StringBuffer sbuffer = new StringBuffer();
        sbuffer.append(”Adresa: ” + adresa.getAddressLine(0));
        String string = sbuffer.toString();
        textView.setText(string);
}
```

Izlistanje koda 15. Primjer obrnutog geokodiranja

Umjesto adrese, zapisane su koordinate (45.556814, 18.695803) u metodi *getFromLocation()* koja pretvara te koordinate u adresu. Stvoren je objekt sbuffer koji prikazuje adresu pomoću metode *getAddressLine*(). Ta metoda vraća adresu koju je numerirao dani indeks ili vraća null ako niti jedna takva linija ne postoji. Za prikaz se varijabla sbuffer mora pretvoriti u string te se sprema u novu varijablu kako bi se na kraju mogla koristiti za ispis adrese pomoću *textView*-a na zaslonu.

2.5. Geo-ograde

Geo-ograda je virtualna granica definiranog područja koja, kada se postavi, može dati obavijesti o ulasku ili izlasku iz te označene zone. Poznavanje trenutne korisnikove lokacije omogućava detektiranje kretanja korisnika s obzirom na postavljenu geo-ogradu, koja je najčešće oblika kruga. Geo-ograda može biti aktivirana statički, dinamički i kombinirano. Statička aktivacija se zasniva na korisnikovoj lokaciji glede određenog područja (na primjer, svakom korisniku stiže poruka kada uđe u trgovinu). Primjer dinamičkog aktiviranja je slanje obavijesti kada se oslobodi parkirno mjesto u blizini korisnikove lokacije. Kombinirani pristup se zasniva na korisnikovoj lokaciji o drugim korisnicima.

Prema [15], za kreiranje geo-ograde koristi se *GeofencingApi* sučelje koji uključuje klase poput:

(1) *Geofence,*
(2) *GeofencingRequest*
(3) *GeofenceApi ,*
(4) *GeofenceStatusCodes*

(5) *GeofencingEvent*

(1) Prema [16], geo-ograda se stvara korištenjem klase *Geofence.Builder* kojoj se dodaju neke od sljedećih metoda:

- *build()* – kreira objekt geo-ograde
- *setCircularRegion()* – postavlja područje definirano geografskom širinom, dužinom i polumjerom
- *setExpirationDuration()* – postavlja vrijeme trajanja geo-ograde
- *setLoiteringDelay()* – postavlja kašnjenje između ulaska i boravljenja u geo-ogradi
- *setNotificationResponsiveness()* – postavlja najučinkovitiju obavijest odziva geo-ograde
- *setRequestId()* – postavlja unikatni ID geo-ograde
- *setTransitionTypes()* – postavlja prijelaznu fazu

Primjer kreiranja geo-ograde prikazan je u izlistanju koda 16:

```
Geofence geo-ograda = new Geofence.Builder()
                .setRequestId (REQ_ID)
                .setCircularRegion (LATITUDE, LONGITUDE, RADIUS)
                .setTransitionTypes (Geofence.GEOFENCE_TRANSITION_ENTER |
                                     Geofence.GEOFENCE_TRANSITION_EXIT)
                .setExpirationDuration (GEO_DURATION)
                .build();
```

Izlistanje koda 16. Stvaranje geo-ograde korištenjem klase *Geofence.Builder*

Postoje tri tipa prijelaza geo-ograde:

- *GEOFENCE_TRANSITION_ENTER* – označava da je korisnik ušao u područje geo-ograde
- *GEOFENCE_TRANSITION_EXIT* – označava da je korisnik izašao iz područja geo-ograde
- *GEOFENCE_TRANSITION_DWELL* – označava da je korisnik ušao i proveo neko vrijeme u području geo-ograde

(2) Klasa *GeofencingRequest* dohvaća geo-ograde koje se trebaju pratiti, a stvaranjem objekta može se dodati geo-ograda korištenjem metode *addGeofence()*. Za aktiviranje obavijesti, kada se geo-ograda doda, koristi se *setInitialTrigger()* metoda kao u izlistanju koda 17.

```
GeofencingRequest zahtjev = new GeofencingRequest.Builder()
                .addGeofence ( geo-ograda)
                .setInitialTrigger (GeofencingRequest.INITIAL_TRIGGER_ENTER)
                .build();
```

Izlistanje koda 17. Stvaranje objekta za dodavanje geo-ograde i aktivaciju obavijesti

(3) Klasa *GeofencingApi* se koristi za dodavanje i brisanje geo-ograde, a kako bi radila ovisi o *GoogleApiClient*-u. Polazna je točka za sve interakcije s Google API-em. Za dodavanje geo-ograde dan je primjer u izlistanju koda 18.

```
PendingResult <Status> addGeofences (googleApiClient, geofencingRequest, pendingIntent)
```

Izlistanje koda 18. Primjer dodavanja geo-ograde

Primjer brisanja geo-ograde prikazan je u izlistanju koda 19.

```
PendingResult <Status> removeGeofences (googleApiClient, pendingIntent)
```

Izlistanje koda 19. Primjer brisanja geo-ograde

Za brisanje, umjesto *pendingIntent*-a, može se koristiti *requestId*.

(4) *GeofenceStatusCodes* je klasa koja se referira na moguće greške prilikom kreiranja geo-ograde. Obavijesti o tome mogu se zapisati na način kao u izlistanju koda 20.

```
GeofenceStatusCodes.GEOFENCE_NOT_AVAILABLE;
GeofenceStatusCodes.GEOFENCE_TOO_MANY_GEOFENCES;
GeofenceStatusCodes.GEOFENCE_TOO_MANY_PENDING_INTENTS;
```

Izlistanje koda 20. Definiranje mogućih grešaka koristeći klasu *GeofenceStatusCodes*

GEOFENCE_NOT_AVAILABLE javlja grešku ako usluga nije dostupna, najčešće u slučaju kada korisnik isključi lokaciju. *GEOFENCE_TOO_MANY_GEOFENCES* javlja grešku kada se registrira više od 100 geo-ograda. *GEOFENCE_TOO_MANY_PENDING_INTENTS* javlja grešku u slučaju da postoji više od 5 različitih *PendingIntent*-a.

(5) Klasa *GeofencingEvent* predstavlja događaj od *GeofencingApi*-a, a događa se kada je geo-ograda zabilježena i kada su uvjeti aktivirani i ispunjeni. Taj događaj može biti stvoren kada se dogodi

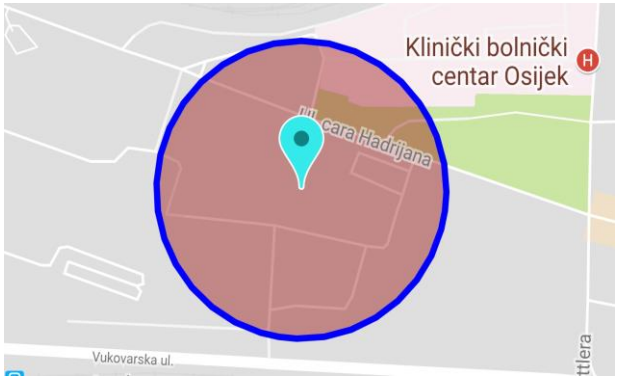
prijelaz (iz geo-ograde ili u geo-ogradu) ili kada se dogodi greška nakon što su geo-ograde registrirane i promatrane. Primjer takvog događaja je pomicanje koordinata unutar geo-ograde.

Osim kreiranja, mora se nacrtati oblik geo-ograde, a najčešći primjer je oblik kruga. Definiranje opcija kruga prikazano je primjerom u izlistanju koda 21.

```
CircleOptions krug = new CircleOptions()
        .center (geofenceMarker.getPostition())
        .strokeColor (Color.BLUE)
        .strokeWidht (15)
        .fillColor (Color.argb(100,150,0,0))
        .radius (200)
```

Izlistanje koda 21. Definiranje opcija kruga geo-ograde

U središte geo-ograde stavljena je oznaka koristeći metodu *center()*, a rub kruga debljine 15 definiran je metodom *strokeWidht()* i plavom bojom u metodi *strokeColor()*. Crvena boja unutar kruga definirana je pomoću metode *Color.argb()* pri čemu prvi broj (100) predstavlja prozirnost boje, odnosno vidljivost područja, drugi broj predstavlja jakost crvene, zatim plave i zelene boje. Krug geo-ograde je radijusa 200 metara definiran metodom *radius()* i izgleda kao na slici 2.7.



Slika 2.7. Geo-ograda

Geo-ograda se može koristiti u društvenim mrežama (Facebook, Snapchat...), u marketingu (ciljano reklamiranje korisnicima koji su u blizini neke trgovine i slično), za zabavu (ako se organizira nekakav događaj, informacije o njemu će biti proširene u tom mjestu), za praćenje ljudi (dobivanje obavijesti kada se netko pojavi na određenom mjestu, neke tvrtke nadgledaju svoje zaposlenike npr. oni koji rade izvan ranog mjesta na terenu, roditelji koji prate kretanje djece u krugu škole), u

prometu (korisnici dobivaju obavijest ako se dogodila nesreća ili gužva za izbjegavanje tog područja), za sigurnost (postavljanjem geo-ograde npr. oko kuće korisnik može biti obaviješten kada netko dolazi) i za brojne druge primjene.

2.6. Karte

Karta predstavlja vizualni prikaz područja zemljine površine koja omogućava korisniku jednostavniji i primjereniji prikaz područja te bolji uvid u lokaciju, za razliku od koordinata. U aplikaciju se mogu dodati karte koristeći Google Play SDK (engl. *Software Development Kit*) usluge koje automatski upravljaju skidanjem podataka, prikazivanjem karte i pristupom Google kartama. Na kartu se mogu dodati oznake, odnosno ikone koje prikazuju odabrano mjesto. Prema [17], svakoj oznaci dodaju se neka svojstva poput:

- *position()* - vraća lokaciju oznake
- *title()* - služi za prikaz naslova
- *snippet()* - prikazuje podnaslov oznake
- *alpha()* - svojstvo zamućenosti oznake (što je broj veći, oznaka postaje prozirnija)
- *rotation()* - predstavlja rotaciju oznake u stupnjevima
- *draggable()* - daje mogućnost povlačenja oznake
- *icon()* - definira izgled ikone

Primjer kreiranja oznake i opcija iste prikazan je u izlistanju koda 22.

```
MarkerOptions oznaka = new MarkerOptions()
        .position(new LatLng(45.556814, 18.695803) )
        .title(”FERIT”)
        .snippet(”Fakultet elektrotehnike, računarstva i inf. tehnologija”)
        .rotation(40)
        .alpha(1)
        .icon(BitmapDescriptorFactory.defautMarker(BitmapDescriptorFactory.HUE_ORANG))
        .draggable(true);
```

Izlistanje koda 22. Stvaranje i definiranje oznake na karti

Oznaci, koja se nalazi na lokaciji koordinata (45.556814, 18.695803), dodan je naslov „FERIT” sa podnaslovom „Fakultet elektrotehnike, računarstva i inf. tehnologija”. Rotacija oznake je 40 stupnjeva, a vidljivost je neprozirna sa ikonom narančaste boje. Omogućeno je svojstvo pomicanja oznake. Na slici 2.8. prikazana je kreirana oznaka.

Slika 2.8. Oznaka

Da bi se na oznaci dobile prikazane opcije naslova i podnaslova, potrebno je uključiti *GoogleMap.OnMarkerClickListener* koje se poziva kada je oznaka dotaknuta korištenjem *onMarkerClick()* metode kao u izlistanju koda 23.

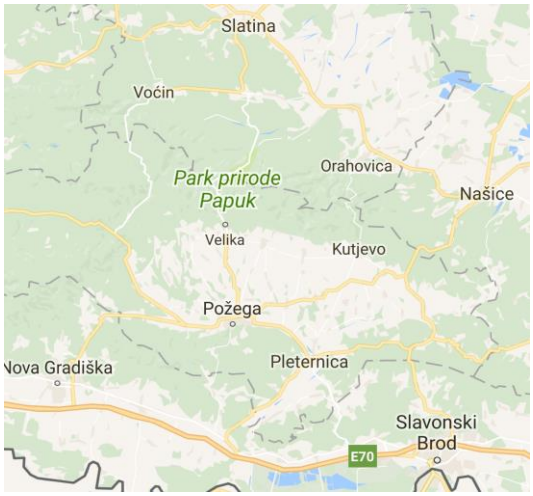
```
public boolean onMarkerClick(Marker marker){
    return false;}
```

Izlistanje koda 23. Metoda koja se poziva kada korisnik dotakne oznaku na karti

Osim oznaka, uz cestovni prikaz (Sl. 2.9.) mogu se mijenjati različiti pogledi na karti, poput satelitskog (Sl. 2.12.), terenskog (Sl. 2.10.) ili hibridnog prikaza (Sl. 2.11.) kao što je prikazano u izlistanju koda 24.

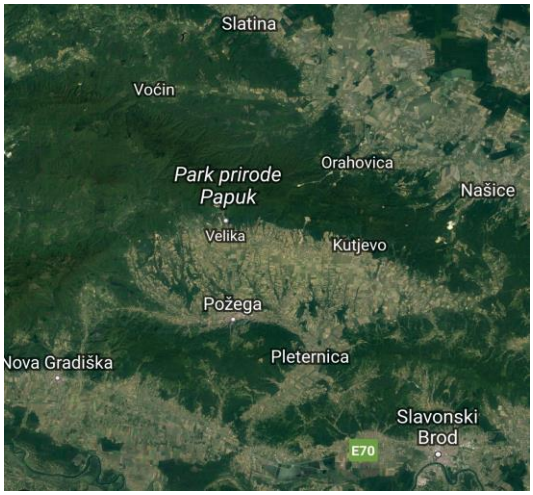
```
GoogleMap map;
map.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
```

Izlistanje koda 24. Primjer dodavanja satelitskog prikaza na karti



Slika 2.9. Cestovni prikaz

Slika 2.10. Terenski prikaz



Slika 2.11. Hibridni prikaz

Slika 2.12. Satelitski prikaz

Hibridni prikaz je spoj cestovnog i satelitskog prikaza, dok terenski prikazuje reljef.

Karta se može zumirati na različite razine, a što je razina veća, to se više detalja može vidjeti na karti. Prema [18], kada se razina smanji, dobiva se veća slika svijeta s manje detalja, a cijeli prikaz svijeta je vidljiv na razini zumiranja 0. Na razini oko 5 vidljivi su kontinenti, na razini 10 vidljivi su gradovi, zatim razina 15 prikazuje ulice i najdetaljniji prikaz (prikaz zgrada) je na razini 20.

API karte omogućavaju kretanje kamere tako da se kamera pomakne od trenutnih do novih atributa čije se trajanje može definirati. Kako bi se to napravilo, koristi se klasa *CameraUpdate* kojom se kamera pomakne trenutno ili se pomiče polako. Za trenutno pomicanje koristi se primjer kao u izlistanju koda 25,

```
map.moveCamera(CameraUpdate);
```

Izlistanje koda 25. Trenutno pomicanje kamere do novih atributa

dok za polagano pomicanje kamere prema novim atributima poziva kao u izlistanju koda 26, pri čemu *cameraUpdate* opisuje razinu zumiranja i lokaciju gdje će kamera biti pomaknuta, *duration* predstavlja vrijeme pomicanja u milisekundama te *callback* koji definira metode *onCancel*() i *onFinished*().

```
map.animateCamera(cameraUpdate, duration, callback);
```

Izlistanje koda 26. Polagano pomicanje kamere do novih atributa

Opcije koje klasa *CameraUpdate* pruža su:

(1) mijenjanje razine zumiranja
(2) mijenjanje pozicije kamere
(3) pomicanje

(1) Mijenjanje razine zumiranja se postiže pozivanjem *CameraUpdateFactory.zoomIn()* ili *CameraUpdateFactory.zoomOut()* metode za približavanje, odnosno udaljavanje od novih atributa. *CameraUpdateFactory.zoomTo(float)* mijenja razinu zumiranja na danu vrijednost. *CameraUpdateFactory.zoomBy(float)* povećava ili smanjuje razinu zumiranja za danu vrijednost. Ako se pozove *zoomTo(15)*, razina zumiranja će biti 15, a ako se pozove *zoomBy(-5)*, tada će razina biti 10.
(2) *CameraUpdateFactory.newLatLng(LatLng)* pomiče kameru do zadane pozicije, odnosno zadanih koordinata, dok *CameraUpdateFactory.newLatLngZoom(LatLng, zoom)* uz pomicanje do zadane lokacije, postavlja i razinu zumiranja.
(3) *CameraUpdateFactory.scrollBy(float x, float y)* pomiče kameru i premješta centar pogleda prema zadanim vrijednostima x i y.

Karte su predstavljene dvjema klasama, *MapFragment* i *GoogleMap* klasom. *MapFragment* je komponenta karte i najjednostavniji način za postavljanje karte u aplikaciju. Može se dodati u *layout* datoteku kao što je prikazano u izlistanju koda 27,

```
<fragment
    android:name=”com.google.android.gms.maps.MapFragment”
    android:id=”@+id/idMap”
    android:layout_width=”match_parent”
    android:layout_height=”match_parent” />
```

Izlistanje koda 27. Dodavanje *MapFragment* komponente u *layout* datoteku

te u Java datoteku pomoću metode *getMapAsync()* za postavljanje poziva za povrat prikazano izlistanjem koda 28.

```
MapFragment fragment = (MapFragment) getMapFragment().findFragmentById(R.id.map)
                    fragment.getMapAsync(this);
```

Izlistanje koda 28. Postavljanje poziva za povrat metodom *getMapAsync()*

GoogleMap se dobiva korištenjem *OnMapReadyCallback* sučelja koje se poziva kada je karta spremna za korištenje. Za upravljanje *GoogleMap* objektom, poziva se *onMapReady()* metoda, a sam objekt predstavlja kartu.

Prema [19], dodavanje novih oznaka na kartu može se postići korištenjem *OnMapClickListener*-a koji se postavlja pozivanjem *GoogleMap.setOnMapClickListener(OnMapClickListener)*. Kada korisnik dotakne kartu, prima se *onMapClick(LatLng)* događaj koji upućuje na dotaknutu lokaciju. Za razliku od te metode, *onMapLongClick(LatLng)* metoda koja zahtijeva dugi pritisak na zaslonu kako bi se postigao isti rezultat. Za takav se slučaj koristi *OnMapLongClickListener* koji se postavlja pozivanjem *GoogleMap.setOnMapLongClickListener(OnMapLongClickListener)*. Tada će se registrirati samo dugi dodir na kartu. Ukoliko se korisniku želi onemogućiti kreiranje nove oznake dodirom na kartu, poziva se *setClickable(false)* metoda. Ta metoda ne utječe na druge kontrole na karti, a prikazana je u izlistanju koda 29.

```
MapFragment fragment;
fragment.getView().setClickable(false);
```

Izlistanje koda 29. Onemogućavanje opcije kreiranja novih oznaka na karti

Kako bi se karte mogle koristiti, najbolja opcija je korištenje Google-ovog servisa Maps za kojeg je potrebno registrirati projekt i nabaviti Google API ključ. Generiranjem ključa omogućava se pristup kartama. Kako bi se dodao generirani API ključ u aplikaciju, u AndroidManifest.xml datoteku unutar <application> elementa stavlja se dio kao u izlistanju koda 30.

```
<meta-data
        android:name = ”com.google.android.geo.API_KEY”
        android:value = ”API_KLJUČ” />
```

Izlistanje koda 30. Dodavanje generiranog API ključ u aplikaciju

Dok API ključ služi za identifikaciju korisnika i praćenje, druge alternative poput enkripcije javnog ključa (engl. *Public Key Encryption*) ili HoK tokena (engl. *Holder-of-Key Tokens*) obavljaju bolji posao pružajući više sigurnosti. Prema [20], vodeći pružatelji servisa za karte, uz Google Maps API, su i Mapbox API, OpenStreet maps, Yahoo maps, Bing maps, Leaflets, Amazon map, MapQuest, Apple maps. U tablici 2.1. prikazane su razlike nekih navedenih pružatelja servisa za karte.

Tablica 2.1. Razlike servisa za karte

	Google Maps	**Bing Maps**	**MapQuest**	**OpenStreet map**
Razina zumiranja	22	19-22	17	19
Tip karte	Cestovni, terenski, satelitski, hibridni	Terenski, satelitski, hibridni, 3D, prometni	Satelitski	Terenski, satelitski
Ažuriranje slike karte	Dnevno	Mjesečno	-	Uživo
Stupnjevi kretanja	Vertikalno, horizontalno, dubinski, 360 panoramski, 3D način rada, rotacijski	Vertikalno, horizontalno, dubinski, 360 panoramski, 3D način rada	Vertikalno, horizontalno, dubinski	Vertikalno, horizontalno, dubinski
Lokacija	Poštanski broj, naziv ulice, naselje, grad, koordinate	Poštanski broj, naziv ulice, naselje, grad, županija, koordinate	Poštanski broj, naziv ulice, naselje, država	Poštanski broj, naziv ulice, naselje, predgrađe, regija, grad, država, koordinate
Dijeljenje karte	Da	Da, preko Email-a, Facebook-a, Twitter-a	Ne	Da

Karte se mogu koristiti za pretraživanje mjesta, računanje udaljenosti između dvije lokacije, za označavanje područja pomoću oznaka, lociranje trenutne pozicije, za navigaciju kretanja, zatim za vizualizaciju područja te za druge brojne primjene.

3. PROGRAMSKO RJEŠENJE ZA GEOLOKACIJSKE USLUGE

Cilj ovog poglavlja je izrada Android mobilne aplikacije koja će korisniku omogućiti stvaranje geo-ograde, unos željenog radijusa za geo-ogradu te pretraživanje mjesta zajedno sa informacijom o lokaciji u obliku adrese i geografskih koordinata. Svrha aplikacije je obavijestiti korisnika kada uđe u područje kreirane geo-ograde. Osim osnovnih, tu su i dodatne mogućnosti poput opcije spremanja geo-ograde, odabira različitih tipova karte, brisanja oznaka i geo-ograde te zumiranje trenutne korisnikove lokacije.

3.1. Specifikacija zahtjeva i korisničkih slučajeva

Sve mogućnosti korištenja aplikacije objašnjene su u nastavku. U tablici 3.1. nalazi se popis svih zahtjeva i korisničkih slučajeva. Ulaskom u mobilnu aplikaciju korisnik ima nekoliko mogućnosti korištenja aplikacije koje su objašnjene u korisničkim slučajevima ispod.

Tablica 3.1. Zahtjevi i korisnički slučajevi

Specifikacija zahtjeva	Korisnički slučaj
Korisnik može upisati i pretražiti željeno mjesto na karti	1
Korisnik može upisati radijus geo-ograde	2
Korisnik može stvoriti oznaku za geo-ogradu	3
Korisnik može dodirom na oznaku vidjeti informacije o lokaciji	4
Korisnik može stvoriti geo-ogradu	5
Korisnik može obrisati geo-ogradu	6
Korisnik može obrisati sve oznake na karti	7
Korisnik može zumirati svoju trenutnu lokaciju	8
Korisnik može odabrati tip karte	9
Korisnik može spremiti geo-ogradu	10

U korisničkom slučaju 1 korisnik može upisati željeno mjesto koje će se, pritiskom na gumb '*traži'*, prikazati na karti. Za pretragu, uvjet je da korisnik ima uključen WiFi ili mobilne podatke.

- Prvi scenarij: Korisnik je upisao točan i postojeći naziv lokacije te se ista pojavljuje i zumira na karti nakon dodira na gumb '*traži'*.
- Drugi scenarij: Korisnik je upisao lokaciju koja ne postoji ili joj je naziv pogrešan. Tada se, nakon pritiska na gumb *'traži'*, prikazuje upozorenje za upis ispravne i dostupne lokacije.

- Treći scenarij: Ako korisnik nije upisao naziv, odnosno polje je ostavio prazno, a dodirnuo je gumb '*traži'*, prikazat će se poruka da lokaciju nije upisana.

U korisničkom slučaju 2 korisnik može upisati željeni radijus za geo-ogradu. Uvjet je da radijus bude između 30 i 3000 metara.

- Prvi scenarij: Korisnik upiše broj između 30 i 3000 metara, uključujući i granice, te pritiskom na gumb '*OK'* unesena se vrijednost sprema i koristi za stvaranje geo-ograde.
- Drugi scenarij: Korisnik upiše broj manji od 30 ili veći od 3000 te ga potvrdi pritiskom na gumb '*OK'*. Tada se prikazuje poruka za ponovni upis radijusa, a broj se ne sprema i ne može se koristiti za stvaranje geo-ograde.
- Treći scenarij: Ako korisnik ne upiše broj, odnosno polje ostavi prazno te pritisne gumb '*OK',* prikazuje se poruka za upis radijusa. Kako broj nije upisan, odnosno nije spremljen, geo-ograda se neće moći kreirati.

U korisničkom slučaju 3 korisnik može stvoriti oznaku koja predstavlja središte oko koje se može kreirati geo-ograda. Oznaka se stvara tako da korisnik pritisne mjesto na karti u trajanju od oko dvije sekunde. Nadovezujući se na korisnički slučaj 1, oznaka za geo-ogradu može se stvoriti i upisom traženog mjesta.

U korisničkom slučaju 4, ukoliko korisnik dotakne oznaku, prikazuju se informacije o lokaciji u obliku geografske širine i dužine te u obliku adrese. Dodirom na bilo koje drugo mjesto na karti, informacija se prestaje prikazivati.

U korisničkom slučaju 5, odabirom na 'Stvori geo-ogradu' u izborniku, korisnik može kreirati geo-ogradu. Uvjet je uključena lokacija i upisani radijus koji je spomenut i objašnjen u korisničkom slučaju 2. Nakon ispunjenih uvjeta, geo-ograda će biti prikazana na karti te će se pojaviti poruka da je kreirana.

U korisničkom slučaju 6 korisnik može odabrati opciju brisanja geo-ograde u izborniku pod nazivom 'Obriši geo-ogradu'. Zatim se geo-ograda uklanja sa karte te se pojavljuje obavijest da je obrisana.

U korisničkom slučaju 7 korisnik može obrisati sve oznake. U izborniku se nalazi opcija za brisanje svih oznaka na karti pod nazivom 'Obriši sve oznake'. Nakon odabira opcije, pojavljuje se obavijest brisanja istih. Pri tome se brišu sve oznake, poput oznake za geo-ogradu ili oznake traženog mjesta.

U korisničkom slučaju 8 dodirom na gumb u gornjem desnom kutu na karti korisnikova lokacija će biti zumirana i prikazana. Uz to, prikazana je i oznaka korisnikove lokacije. Ako se ona dodirne, dobiva se informacija trenutne adrese sa podnaslovom 'Ovdje se nalazim', a geografske koordinate su prikazane pri vrhu aplikacije. Uvjet je uključena lokacija.

U korisničkom slučaju 9 korisnik može odabrati tip karte. U izborniku postoje opcije za odabir tipa karte kao što je cestovni, satelitski, hibridni ili terenski prikaz. Odabirom na jednu od ponuđenih opcija, izgled karte se mijenja. Ukoliko se ne odabere niti jedan tip karte, pojavit će se zadani, odnosno cestovni prikaz. Uvjet je uključen WiFi ili mobilni podaci.

U korisničkom slučaju 10 korisnik može odabrati opciju spremanja geo-ograde korištenjem gumba za uključivanje/isključivanje opcije za spremanje (engl. *Switch*) pod nazivom 'Spremi geo-ogradu'. Uvjet je uključena lokacija.

- Prvi scenarij: Korisnik ne kreira geo-ogradu i odabere opciju 'uključeno' za spremanje geo-ograde. Kada se ponovno uđe u aplikaciju, geo-ograda neće biti prikazana, a opcija će biti isključena.
- Drugi scenarij: Korisnik kreira geo-ogradu i odabere opciju 'uključeno' za spremanje geo-ograde. Ponovnim ulaskom u aplikaciju stvorena geo-ograda će biti prikazana kao i odabrana opcija 'uključeno'.
- Treći scenarij: Korisnik kreira geo-ogradu, ali ju ne spremi, odnosno odabere opciju 'isključeno'. Pri idućem ulasku u aplikaciju, geo-ograda neće biti sačuvana i prikazana.

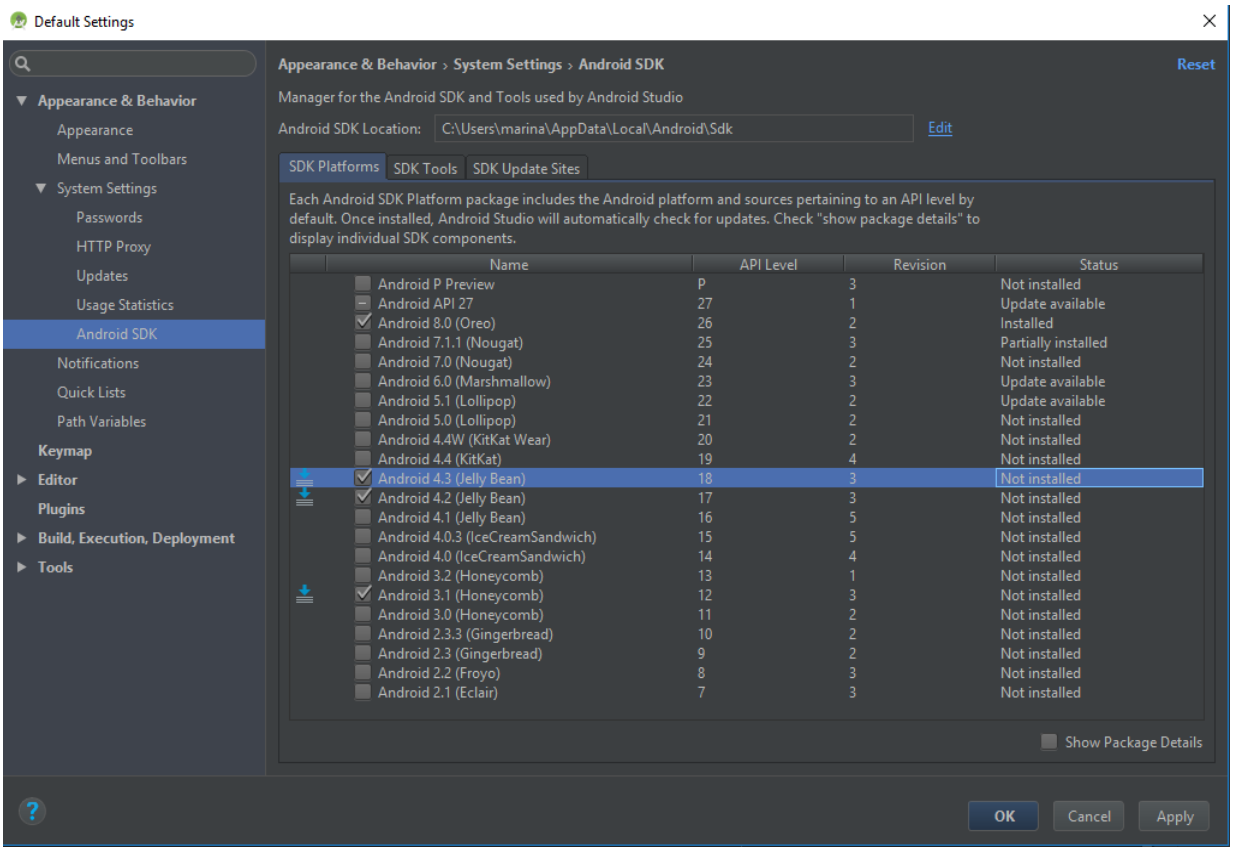
3.2. Opis korištenih alata i tehnologija

Tehnologije i alati koji su korišteni prilikom izrade ove Android mobilne aplikacije su Android Studio, Android SDK (engl. *Software Development Kit*), Android AVD (engl. *Android Virtual Devic*e), Android JDK (engl. *Java Development Kit*), Java programski jezik te XML (engl. *Extensible Markup Language*).

Prema [21], Android Studio je integrirano razvojno okruženje za Android operacijski sustav koji je osmišljen posebno za razvoj softvera za Android. Podržava programske jezike poput Jave, Kotlina i Pythona te ga je moguće koristiti na programskim sustavima Windows, Linux i MacOS. Razvojno sučelje je bazirano na IntelliJ IDEA. Android sučelje sastoji se od:

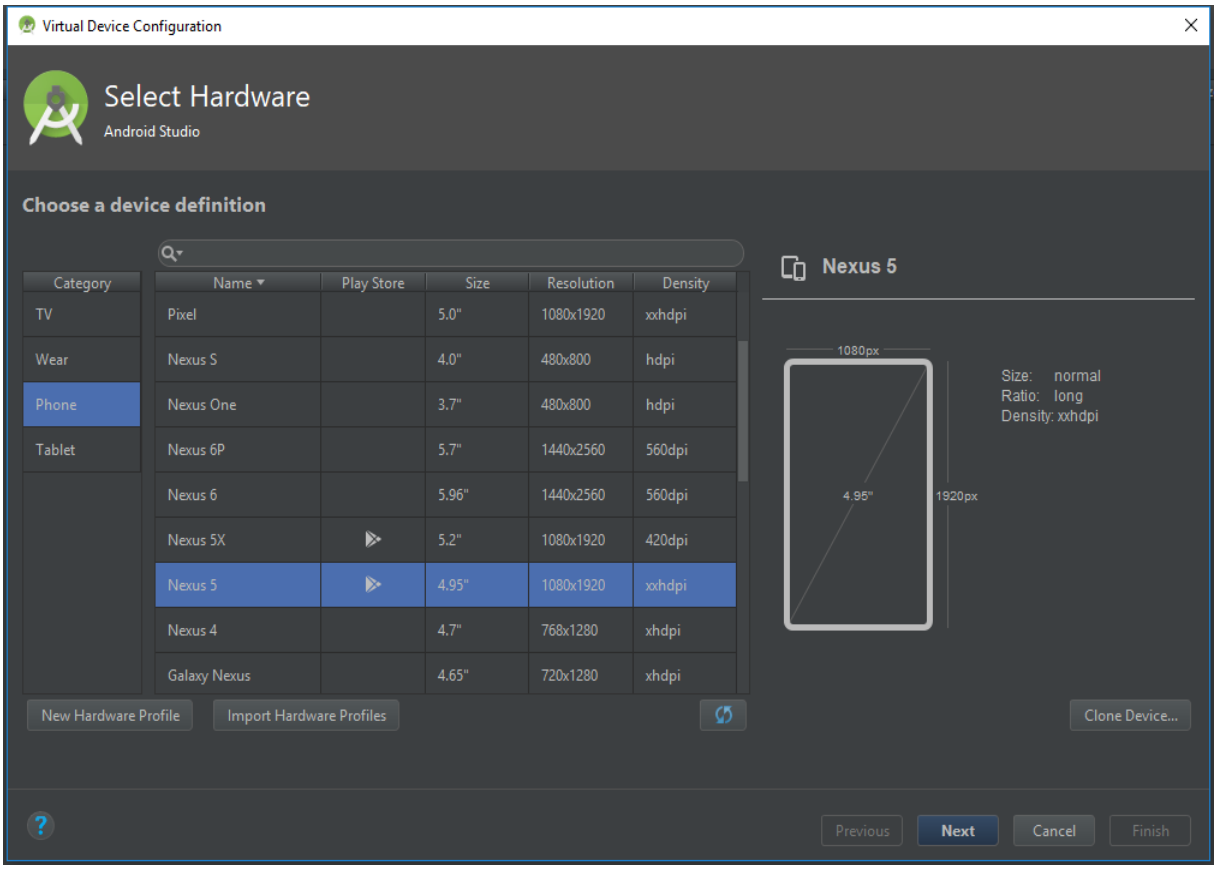
- alatne trake (engl. *toolbar*) koja sadržava širok izbor aktivnosti koje uključuju pokretanje aplikacije, ispravljanje pogrešaka i drugo
- navigacijske trake (engl. *navigation bar*) koja pomaže pri upravljanju kroz projekt sadržavajući otvorene datoteke za uređivanje
- prozora za uređivanje (engl. *editor window*) u kojem se stvara i mijenja kod
- prozora s alatima (engl. *window bar*) koji se nalazi uz rub, izvan prozora za razvojno okruženje, te omogućava proširenje ili smanjenje pojedinih alatnih prozora
- alatnih prozora (engl. *tool windows*) koji se također mogu proširiti ili smanjiti, a daju pristup određenim zadacima poput pretraživanja, upravljanja projektima i drugo
- trake stanja (engl. *status bar*) koja prikazuje poruke, upozorenja te status projekta

Uz Android Studio, koriste se i navedeni razvojni paketi kao što su Android SDK, alat koji omogućava stvaranje aplikacije te jednostavno i efikasno pisanje koda, a dostupan je za različite programske jezike, te Android JDK koji je dostupan samo za Java programski jezik. Ukoliko se želi koristiti određena verzija Androida, tada se moraju skinuti potrebne platforme i SDK alati. Oni se nalaze odabirom na *Tools>Android>SDK Manager*. Tada se otvara prozor sa popisom verzija koje su instalirane, nisu instalirane ili zahtijevaju ažuriranje, prikazano slikom 3.1. Tu se odabiru željene verzije koje je moguće jednostavno skinuti i instalirati. Android SDK uključuje primjer izvornog koda, vodič za Android OS, emulator, dokumentaciju za sučelje Android aplikacijskih programa, program za ispravljanje grešaka te potrebne knjižnice.



Slika 3.1. Android SDK Manager

Za potrebe testiranja aplikacije koristi se fizički ili virtualni uređaj. Android virtualni uređaj (AVD) oponaša izgled stvarnog uređaja koji se pokreće i radi s Android emulatorom. Pri tome se može odabrati željeni uređaj, veličina ekrana, verzije Androida i slično. Za korištenje virtualnog uređaja potrebno je skinuti zahtjevane komponente i postaviti specifikacije. Da bi se to napravilo, treba se otvoriti *Tools>Android>AVD Manager*. Odabirom na *Create Virtual Device* otvara se prozor kao na slici 3.2. Nakon što su specifikacije odabrane, odabirom na gumb *Next,* otvara se popis dostupnih verzija Androida koje će možda biti potrebno skinuti. Ponovnim odabirom na gumb *Next* dolazi se do zadnjeg koraka u kojem se upisuje naziv virtualnog uređaja te se, odabirom na gumb *Finish,* kreira novi virtualni uređaj. Tada se može pokrenuti aplikacija bez korištenja fizičkog uređaja.



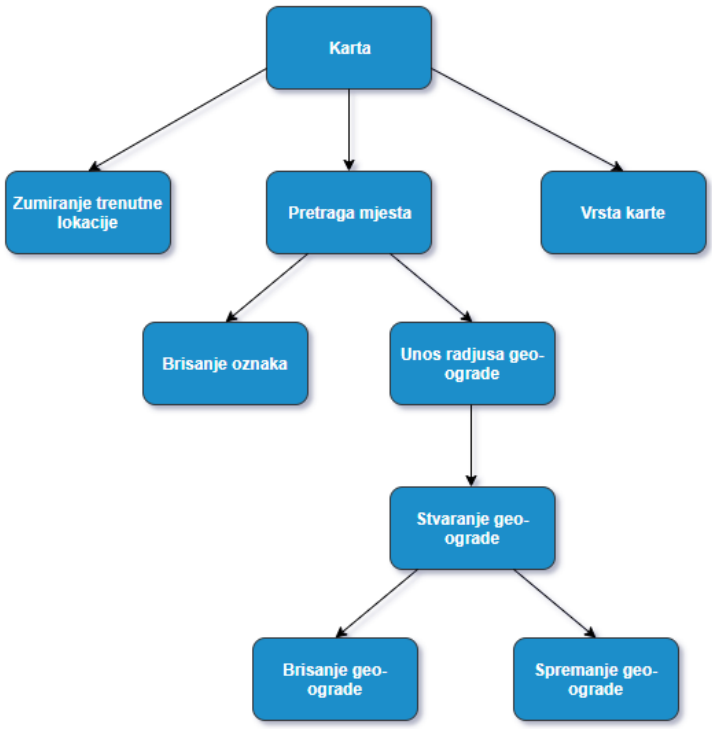
Slika 3.2. AVD Manager

Prema [22], XML (engl. *Extensible Markup Language*) je standardizirani jezik za označavanje podataka koji je čitljiv i razumljiv kako ljudima tako i računalnim programima. Format oznaka u XML-u sličan je formatu oznaka u HTML jeziku. XML jezik vrlo je raširen i koristi se za razne namjene poput razmjene i pohrane podataka, povećanje dostupnosti podataka i drugo. Dok se glavni dio koda za definiranje ponašanja i karakteristika aplikacije nalazi u Java datoteci, izgled aplikacije se definira u drugom dijelu koda, odnosno u *actvity_main.xml* datoteci, gdje XML jezik definira izgled dokumenta. Tu se definiraju gradivni elementi poput gumba, *EditTexta*, *CheckBoxa* i ostalog. U Android Studiu, unutar XML datoteke, nalaze se dvije kartice. Jedna prikazuje dizajn aplikacije, a druga njen kod. U dijelu dizajna prikazan je izgled aplikacije, a tu se mogu dodavati, uklanjati i povlačiti gradivni elementi po zaslonu. Druga kartica, u kojoj se nalazi XML kod, omogućava mijenjanje i dodavanje novih linija koda za definiranje izgleda aplikacije. Isto tako se u mapi pod nazivom *Values* nalazi XML kod u kojem se spremaju vrijednosti varijabli poput naziva aplikacije i vrijednosti boja. Uz to, postoji i *AndroidManifest.xml* datoteka koja definira ključne činjenice poput naziva aplikacije koji će biti vidljiv korisnicima, dozvole aplikacije i drugo.

Prema [23], Java je objektno-orijentirani programski jezik i jedan je od najkorištenijih programskih jezika. Velika prednost je ta što se programi pisani u Javi mogu izvoditi na svim operacijskim sustavima, za razliku od nekih drugih programskih jezika koje je potrebno prilagođavati platformi na kojoj se izvode. Da bi se programi pisani u Javi ili u drugim programskim jezicima koji su kompilirani u Java *bytecode* pokretali na bilo kojem računalu, koristi se Java virtualni stroj (engl. *Java Virtual Mashine*). Java programski jezik svrstavamo u objektno-orijentirane programske jezike jer je baziran na klasama, odnosno objektima.

3.3. Bitni segmenti programskog koda

Radi jednostavnosti prikaza, na slici 3.3. dan je blok dijagram koji prikazuje elemente aplikacije.



Slika 3.3. Elementi programskog rješenja

Na karti se može zumirati trenutna korisnikova lokacija, pretražiti mjesto unosom u polje ili dodavanjem oznake na kartu te se može mijenjati pogled karte, odnosno vrsta. Nakon definirane lokacije, njena oznaka se može obrisati. Upisom radijusa kreiranoj oznaci se može stvoriti geo-ograda, a potom ju je moguće obrisati ili spremiti.

3.3.1. Unos radijusa za geo-ogradu

Kako bi korisnik mogao upisati željeni radijus geo-ograde, stvoren je jedan *EditText* naziva 'mEdit'. U tu varijablu spremaju se vrijednosti koristeći metodu *getText().toString()*. Na slici 3.4. prikazana je metoda *conditionsInput()* koja definira uvjete unosa. Metodom *getText.length()* se provjerava je li argument unesen. Za svaki korisnički slučaj ispisuje se poruka prikazana slikom 3.4. U metodi *getRadius()* prikazano je definiranje gumba kao i metoda *conditionsInput()* koja se poziva nakon pritiska na gumb vidljivo na slici 3.5.

```
private EditText mEdit;
public void conditionsInput() {
    mEdit = (EditText) findViewById(R.id.etRadijus);
    if (mEdit.getText().length() < 1) {
        Toast.makeText(getApplicationContext(), text: "Niste upisali radijus!", Toast.LENGTH_LONG).show();
        return;
    }
    if (Float.parseFloat(mEdit.getText().toString()) > 3000 || Float.parseFloat(mEdit.getText().toString()) < 30) {
        Toast.makeText(getApplicationContext(), text: "Upišite broj između 30 i 3000 metara", Toast.LENGTH_LONG).show();
    } else {
        geofenceRadius = Float.valueOf(String.valueOf(mEdit.getText()));
        Toast.makeText(getApplicationContext(), text: "Upisali ste radijus od " + mEdit.getText() + "m", Toast.LENGTH_LONG).show();
    }
}
```

Slika 3.4. Uvjeti pri unosu argumenta za radijus geo-ograde

```
private Button mButton;
public void getRadius() {
    mButton = (Button) findViewById(R.id.btnOK);
    mButton.setOnClickListener(
            new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    conditionsInput();
                }
            }
    );
}
```

Slika 3.5. Definiranje gumba

3.3.2. Stvaranje geo-ograde i njene oznake

Na slici 3.6. prikazana je metoda *createGeofence* koja uzima dva parametra, koordinate i radijus. Koristeći klasu *Geofence.Builder* kreira se geo-ograda, postavljajući područje željenog radijusa i koordinata pomoću *setCircularRegion()* metode, zatim trajanje geo-ograde u miliseknudama pomoću *setExpirationDuration()* te prijelaze koristeći *setTransitionTypes()*.

```
private static final long GEO_DURATION = 120 * 120 * 1000;
private static final String GEOFENCE_REQ_ID = "ID1";

private Geofence createGeofence(LatLng latLng, float radius) {
    return new Geofence.Builder()
            .setRequestId(GEOFENCE_REQ_ID)
            .setCircularRegion(latLng.latitude, latLng.longitude, radius)
            .setExpirationDuration(GEO_DURATION)
            .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER
                    | Geofence.GEOFENCE_TRANSITION_EXIT)
            .build();
}
```

Slika 3.6. Stvaranje geo-ograde

U metodi *drawGeofence()*, koja je prikazana slikom 3.7., definira se krug geo-ograde pomoću klase *CircleOptions*. Opcije koje su definirane su središte kruga u kojem se nalazi oznaka za geo-ogradu, boja ruba kruga pomoću metode *strokeColor()* te širina ruba koristeći *strokeWidth()*. Također je unutrašnjost kruga ispunjen bojom koristeći *fillColor()* metodu, pri čemu prvi parametar *alpha* predstavlja prozirnost boje.

```
private void drawGeofence() {
    if (geoFenceLimits != null)
        geoFenceLimits.remove();
    CircleOptions circleOptions = new CircleOptions()
            .center(geoFenceMarker.getPosition())
            .strokeColor(0xff888888)
            .strokeWidth(7)
            .fillColor(Color.argb( alpha: 100,  red: 190,  green: 90,  blue: 100))
            .radius(geofenceRadius);
    geoFenceLimits = map.addCircle(circleOptions);
}
```

Slika 3.7. Definiranje kruga geo-ograde

Kako bi se geo-ograda kreirala, potrebno je stvoriti oznaku koja predstavlja središte kruga. Na slici 3.8., metoda *markerForGeofence()* prikazuje stvaranje oznake te definiranje njenih opcija koristeći klasu *MarkerOptions*. Oznaka ima naslov koji sadržava geografske koordinate te podnaslov koji prikazuje adresu. Naslov je spremljen u varijablu 'title' koja sadržava širinu i dužinu, a podnaslov koristi metodu *getCompleteAddressString()* za pretvaranje koordinata u adresu.

```
private Marker geoFenceMarker;

private void markerForGeofence(LatLng latLng) {
    String title = latLng.latitude + ", " + latLng.longitude;
    MarkerOptions markerOptions = new MarkerOptions()
            .position(latLng)
            .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_CYAN))
            .title(title)
            .snippet((getCompleteAddressString(latLng.latitude, latLng.longitude)));
    if (map != null) {
        if (geoFenceMarker != null)
            geoFenceMarker.remove();
        geoFenceMarker = map.addMarker(markerOptions);
    }
}
```

Slika 3.8. Definiranje oznake geo-ograde

3.3.3. Brisanje geo-ograde

Kako bi se geo-ograda obrisala, koristi se klasa *LocationServices* koja poziva metodu *removeGeofences()*, a ona prima dva parametra, *googleApiClient* i *createGeofencePendingIntent()*. Ukoliko je status uspješan, uklanja se crtež geo-ograde pomoću metode *removeGeofenceDraw()*. Kada je obrisana, ispisuje se poruka na ekran, vidljivo na slici 3.9.

```
private GoogleApiClient googleApiClient;
private void clearGeofence() {
    LocationServices.GeofencingApi
            .removeGeofences(googleApiClient, createGeofencePendingIntent())
            .setResultCallback((status) → {
                if (status.isSuccess()) {
                    removeGeofenceDraw();
                    Toast.makeText(getApplicationContext(), text: "Geo-ograda je obrisana", Toast.LENGTH_LONG).show();
                }
            });
}
```

Slika 3.9. Brisanje geo-ograde

Na slici 3.10. prikazana je metoda *removeGeofenceDraw()* za uklanjanje crteža geo-ograde. Oznaka geo-ograde provjerava se za *null* vrijednost, odnosno provjerava se postoji li objekt, te ako postoji, uklanja se. Isto se provjerava i za krug spremljen u varijablu *geoFenceLimits*.

```
private void removeGeofenceDraw() {
    if (geoFenceMarker != null)
        geoFenceMarker.remove();
    if (geoFenceLimits != null)
        geoFenceLimits.remove();
}
```

Slika 3.10. Brisanje crteža geo-ograde

3.3.4. Spremanje geo-ograde i oznake

U metodi *Switch()* koja je prikazana na slici 3.11., pomoću *SharedPreferences* klase sprema se stanje *Switch* kontrole. U *if* izrazu provjerava se je li *Switch* kontrola uključena. Ako je, stvara se objekt 'editor' koji sprema *boolean* vrijednost, a pomoću metode *editor.apply()* spremaju se sve promjene.

```
public void Switch(){
    switchSave.setOnCheckedChangeListener((buttonView, isChecked) -> {
            if(switchSave.isChecked()){
                SharedPreferences.Editor editor = getSharedPreferences( name: "stanje", MODE_PRIVATE).edit();
                editor.putBoolean("radiobutton", true);
                editor.apply();

                Toast.makeText(getApplicationContext(), text: "Uključeno", Toast.LENGTH_LONG).show();
            }else {
                SharedPreferences.Editor editor = getSharedPreferences( name: "stanje", MODE_PRIVATE).edit();
                editor.putBoolean("radiobutton", false);
                editor.apply();
                geofenceRadius = 0;

                Toast.makeText(getApplicationContext(), text: "Isključeno", Toast.LENGTH_LONG).show();
            }
    });
}
```

Slika 3.11. Spremanje stanja *Switch* kontrole

Na slici 3.12. prikazano je učitavanje stvorene geo-ograde i stanja *Switch* kontrole čija se spremljena vrijednost dohvaća metodom *getBoolean()*. Ako je kontrola uključena, dohvaća se spremljena vrijednost radijusa i pomoću te vrijednosti se stvara geo-ograda.

```
@Override
public void onConnected(@Nullable Bundle bundle) {
    SharedPreferences sharedPreferences = getSharedPreferences( name: "stanje",MODE_PRIVATE);
    switchSave.setChecked(sharedPreferences.getBoolean( key: "radiobutton", defValue: true));

    if(switchSave.isChecked()) {
        SharedPreferences sharedPreference = getPreferences(Context.MODE_PRIVATE);
        float radius = sharedPreference.getFloat( key: "ograda", geofenceRadius);
        geofenceRadius = radius;
        startGeofence();
    }
}
```

Slika 3.12. Učitavanje spremljene geo-ograde i stanja *Switch* kontrole

Za spremanje podataka koristi se klasa *SharedPreferences.Editor* kojom je stvoren objekt 'editor'. Pomoću njega se postavljaju vrijednosti geografske širine i dužine koristeći *putLong()* i *putFloat()* metode. Metodom *apply()* se sve promjene zapisuju i spremaju u memoriju, kao što je prikazano slikom 3.13.

```
private void saveGeofence() {
    SharedPreferences sharedPref = getPreferences(Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = sharedPref.edit();

    editor.putLong(KEY_GEOFENCE_LAT, Double.doubleToRawLongBits(geoFenceMarker.getPosition().latitude));
    editor.putLong(KEY_GEOFENCE_LON, Double.doubleToRawLongBits(geoFenceMarker.getPosition().longitude));

    editor.putFloat("ograda", geofenceRadius);
    editor.apply();
}
```

Slika 3.13. Spremanje oznake geo-ograde i vrijednosti radijusa

Metoda za učitavanje oznake geo-ograde prikazana je slikom 3.14. Stvoreni objekt *sharedPref* koristi se za dohvaćanje vrijednosti koordinata pomoću *getLong()* metode. Vrijednosti se spremaju u varijable 'lat' i 'lon' koje se koriste u metodi *markerForGeofence()*.

```
private void loadGeofenceMarker() {
    SharedPreferences sharedPref = getPreferences(Context.MODE_PRIVATE);

    if (sharedPref.contains(KEY_GEOFENCE_LAT) && sharedPref.contains(KEY_GEOFENCE_LON)) {
        double lat = Double.longBitsToDouble(sharedPref.getLong(KEY_GEOFENCE_LAT,  defValue: -1));
        double lon = Double.longBitsToDouble(sharedPref.getLong(KEY_GEOFENCE_LON,  defValue: -1));
        LatLng latLng = new LatLng(lat, lon);
        markerForGeofence(latLng);
    }
}
```

Slika 3.14. Učitavanje oznake geo-ograde

3.3.5. Pretraživanje mjesta

U varijablu 'location' spremljen je uneseni argument pomoću metode *getText().toString()*. U 'addressList' sprema se polje adresa koje opisuju traženu lokaciju koristeći *getFromLocationName()* metodu. Ako 'addressList' nije prazan, u varijablu 'address' sprema se oblikovana prva linija adrese (ako postoji), grad i naziv države. Pozivanjem metode *markerForGeofence()* pojavljuje se oznaka za geo-ogradu koja omogućava njeno kreiranje na traženom mjestu, kao što je vidljivo na slici 3.15.

```
public void geoLocate(View view) throws IOException {
    EditText et = (EditText) findViewById(R.id.upišiMjesto);
    final String location = et.getText().toString();
    List<Address> addressList;

    if (!location.isEmpty()) {
        Geocoder geocoder = new Geocoder( context: this);
            addressList = geocoder.getFromLocationName(location,  maxResults: 1);
        if (!addressList.isEmpty()) {
            Address address = addressList.get(0);
            LatLng latLng = new LatLng(address.getLatitude(), address.getLongitude());

            map.addMarker(new MarkerOptions()
                    .position(latLng)
                    .title(latLng.latitude + ", " + latLng.longitude)
                    .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_VIOLET))
                    .snippet(address.getAddressLine( index: 0)));

            Toast.makeText(getApplicationContext(),  text: "Mjesto koje ste tražili: " + location, Toast.LENGTH_LONG).show();

            markerForGeofence(latLng);

            float zoom = 15f;
            CameraUpdate cameraUpdate = CameraUpdateFactory.newLatLngZoom(latLng, zoom);
            map.animateCamera(cameraUpdate);

        } else Toast.makeText(getApplicationContext(),  text: "Upišite dostupnu lokaciju!", Toast.LENGTH_LONG).show();

    }else Toast.makeText(getApplicationContext(),  text: "Niste upisali lokaciju!", Toast.LENGTH_LONG).show();
}
```

Slika 3.15. Unos i pretraživanje traženog mjesta

3.3.6. Marker trenutne lokacije

U metodi *markerLocation()*, koja je prikazna na slici 3.16., stvara se oznaka trenutne korisnikove lokacije. Koristeći klasu *MarkerOptions* postavlja se lokacija oznake, naslov i podnaslov. Za naslov je korištena metoda *getCompleteAddressString()* koja prima dva parametra, geografsku širinu ('latitude') i dužinu ('longitude') koja pretvara koordinate u adresu.

```
private Marker locationMarker;

private void markerLocation(LatLng latLng) {
    MarkerOptions markerOptions = new MarkerOptions()
            .position(latLng)
            .title(getCompleteAddressString(latLng.latitude, latLng.longitude))
            .snippet("Ovdje se nalazim");

    if (map != null) {
        if (locationMarker != null)
            locationMarker.remove();
        locationMarker = map.addMarker(markerOptions);
    }
}
```

Slika 3.16. Oznaka trenutne lokacije

Zapis koordinata trenutne korisnikove lokacije ispisan je u dva *TextView*-a pomoću metode *writeActualLocation*() kao na slici 3.17. Metoda *setText()* ispisuje geografsku širinu i dužinu metodom *getLatitude()*, odnosno *getLongitude()*.

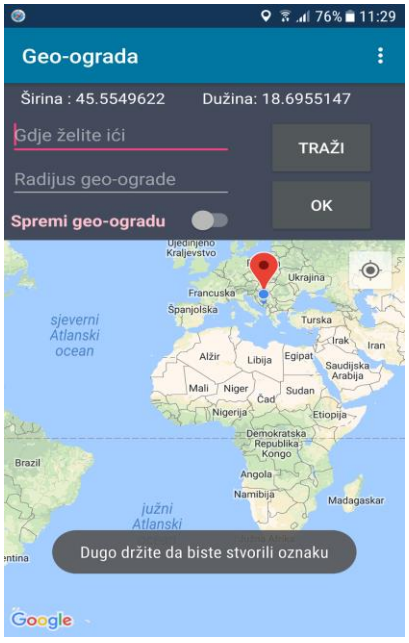
```
private void writeActualLocation(Location location) {
    textLat.setText("Širina : " + location.getLatitude());
    textLong.setText("Dužina: " + location.getLongitude());

    markerLocation(new LatLng(location.getLatitude(), location.getLongitude()));
}
```

Slika 3.17. Prikaz koordinata trenutne lokacije

3.4. Testiranje programskog rješenja

Prilikom ispitivanja funkcionalnosti aplikacije korišten je fizički uređaj Android verzije 7.0 (*Nougat*) u svrhu testiranja. Ulaskom u aplikaciju pojavljuje zaslon cestovnog tipa karte sa porukom 'Dugo držite da biste stvorili oznaku', vidljivo na slici 3.18. U gornjem dijelu zaslona nalazi se zapis geografske širine i dužine trenutne korisnikove lokacije. Na slici 3.19. prikazan je izbornik.

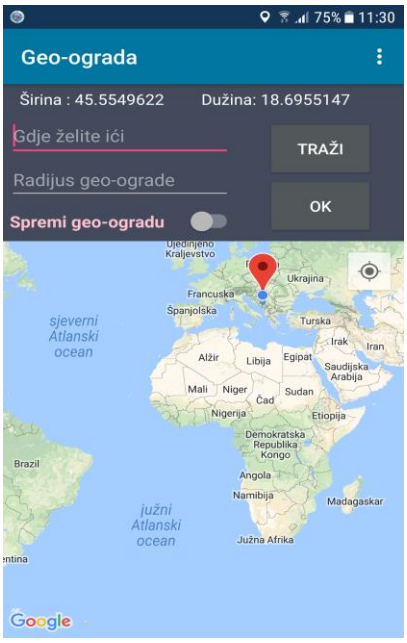


Slika 3.18. Početni zaslon

Slika 3.19. Izbornik

Odabirom u izborniku na jednu od opcija za tip karte (satelit, hibrid i teren), izgled karte se mijenja kao što je prikazano slikama 3.20., 3.21. i 3.22.

Slika 3.20. Satelitski prikaz

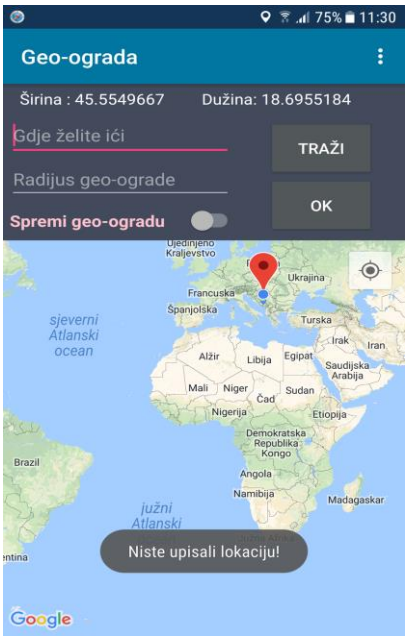


Slika 3.21. Terenski prikaz

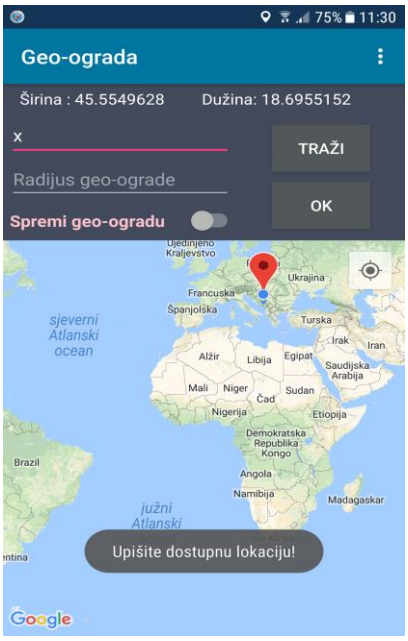
Slika 3.22. Hibridni prikaz

Pretraživanje željenog mjesta se upisuje u polju naziva 'Gdje želite ići' te pritiskom na gumb 'TRAŽI' pretraga se aktivira. Slika 3.25. prikazuje 'kampus Osijek' kao traženo mjesto koje je označeno ljubičastom oznakom. Oznaka daje informacije o adresi, odnosno o koordinatama traženog mjesta. Također je ispisana poruka traženog mjesta.

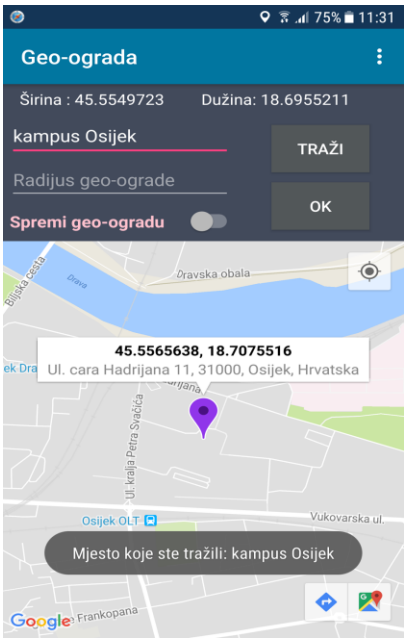
Ukoliko se adresa ne upiše, a gumb 'TRAŽI' se aktivira, pojavljuje se poruka 'Niste upisali lokaciju!' prikazano slikom 3.23., a ukoliko se upiše nepostojeća adresa, na primjer 'x' kao na slici 3.24., ispisat će se poruka 'Upišite dostupnu lokaciju!'.



Slika 3.23. Lokacija nije upisana

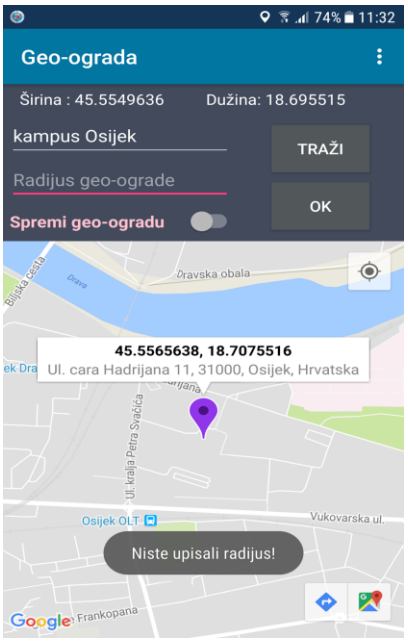


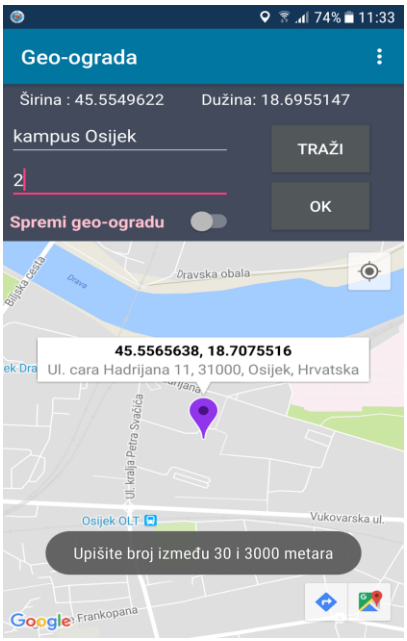
Slika 3.24. Lokacija ne postoji

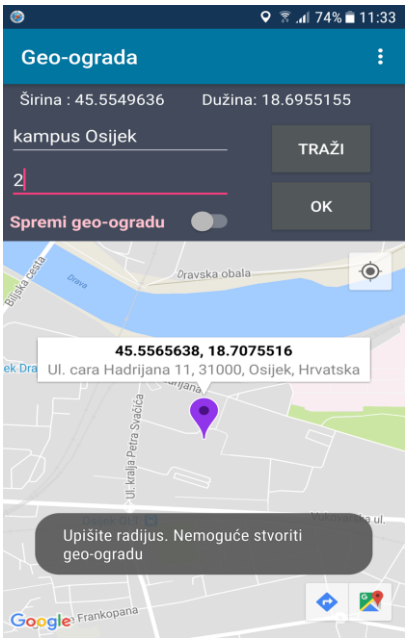


Slika 3.25. Pretraživanje mjesta

U polju 'Radijus geo-ograde' upisuje se željeni radijus. Ako se upiše, na primjer 2, te se unos potvrdi gumbom 'OK', ispisat će se poruka 'Upišite broj između 30 i 3000 metara' kao na slici 3.27. Kako uvjet nije ispunjen, odabirom u izborniku na 'Stvori geo-ogradu', geo-ograda neće biti stvorena i ispisat će se poruka 'Upišite radijus. Nemoguće stvoriti geo-ogradu' kao na slici 3.28. Ista poruka će se pojaviti ukoliko se ne upiše radijus, a pritiskom na gumb 'OK' ispisat će se poruka 'Niste upisali radijus!', kao što je vidljivo na slici 3.26.

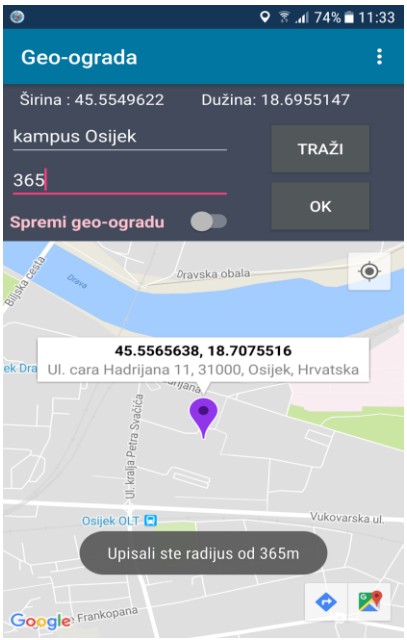


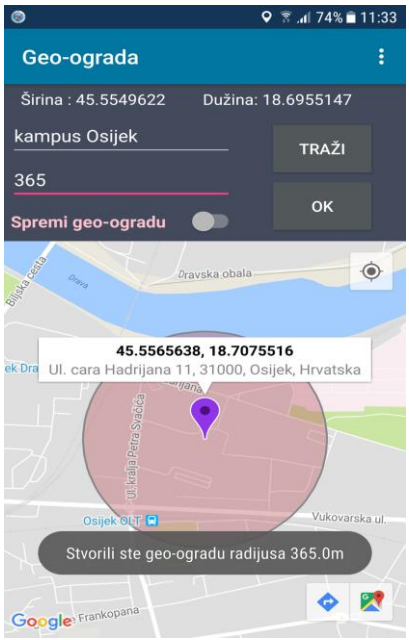




Slika 3.26. Nije upisan radijus Slika 3.27. Upis neispravnog broja Slika 3.28.Neuspješno kreiranje

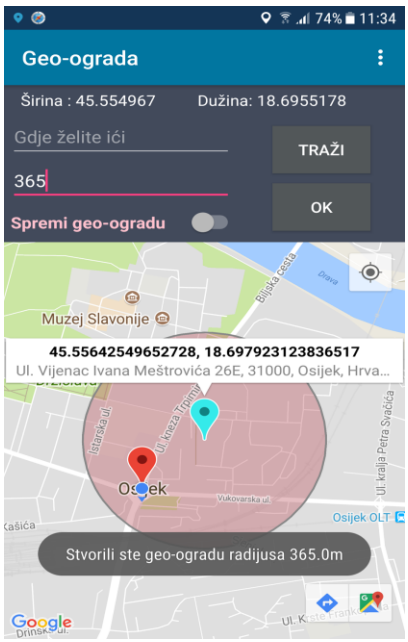
Upisom odgovarajućeg broja, na primjer 356, u polje 'Radijus geo-ograde' i pritiskom na gumb 'OK', pojavljuje se poruka 'Upisali ste radijus od 356m' prikazano na slici 3.29. Nakon toga se može stvoriti geo-ograda odabirom na izbornik>Stvori geo-ogradu. Pojavljuje se kreirana geo-ograda prikazana crvenim krugom kao na slici 3.30., te poruka 'Stvorili ste geo-ogradu radijusa 365.0m'.



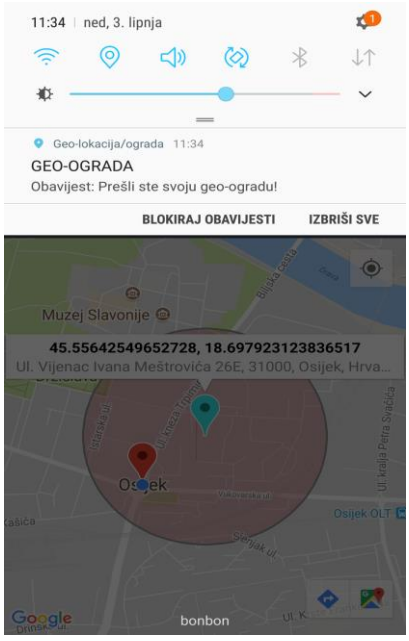


Slika 3.29. Upis radijusa Slika 3.30. Uspješno kreiranje geo-ograde

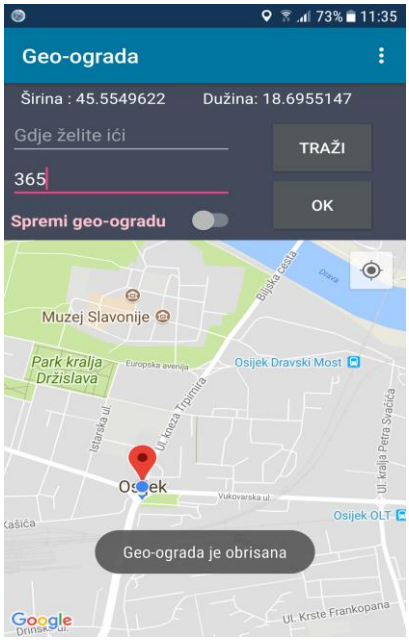
Ukoliko se korisnik nalazi u području geo-ograde (kao na slici 3.31.), korisnik dobiva obavijest koja je prikazana na slici 3.32., a glasi 'Obavijest: Prešli ste svoju geo-ogradu!'. Ako se želi obrisati stvorena geo-ograda, u izborniku se odabire opcija 'Obriši geo-ogradu'. Tada se ona uklanja sa karte i pojavljuje se poruka 'Geo-ograda je obrisana' vidljivo na slici 3.33.



Slika 3.31. Unutar geo-ograde



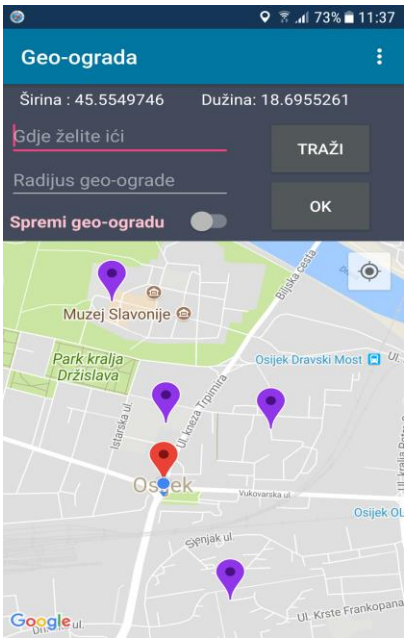
Slika 3.32. Obavijest



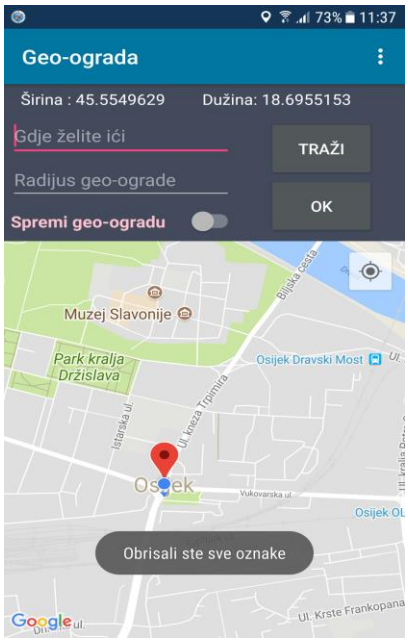
Slika 3.33. Brisanje geo-ograde

Uz mogućnost stvaranja oznake za geo-ogradu preko pretraživanja mjesta, oznaka se može dodati i dugim dodirom na kartu, čija je uputa 'Dugo držite da biste stvorili oznaku' već prikazana na slici 3.18.

Sve oznake se mogu ukloniti sa karte u opciji 'Obriši sve oznake' koja se nalazi u izborniku. Slika 3.34. prikazuje nekoliko oznaka, slika 3.35. brisanje istih sa ispisanom porukom 'Obrisali ste sve oznake'.

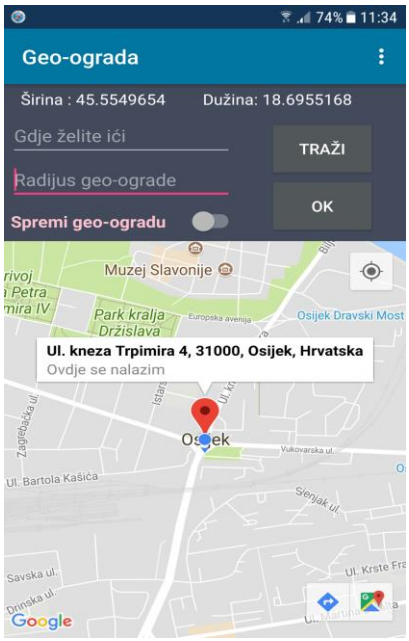


Slika 3.34. Dodavanje oznaka



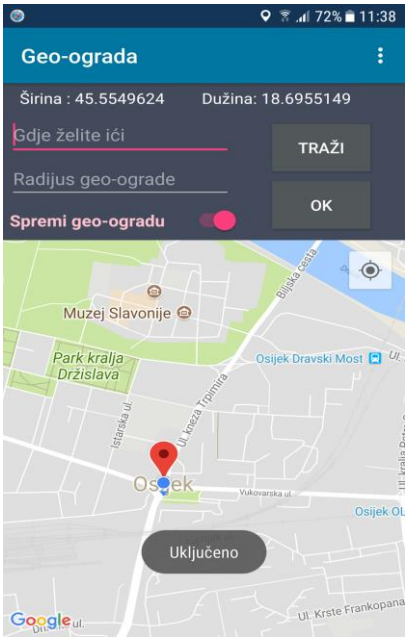
Slika 3.35. Brisanje oznaka

Kako bi se pronašla trenutna korisnikova lokacija na karti, pritiskom na gumb u gornjem desnom kutu karte kamera se pomiče do korisnikove lokacije. Dodirom na crvenu oznaku prikazuje se informacija o adresi korisnikove lokacije sa podnaslovom 'Ovdje se nalazim' kao što je prikazano na slici 3.36. Koordinate korisnikove lokacije prikazane su u gornjem dijelu pod nazivom 'Širina:' i 'Dužina:'.

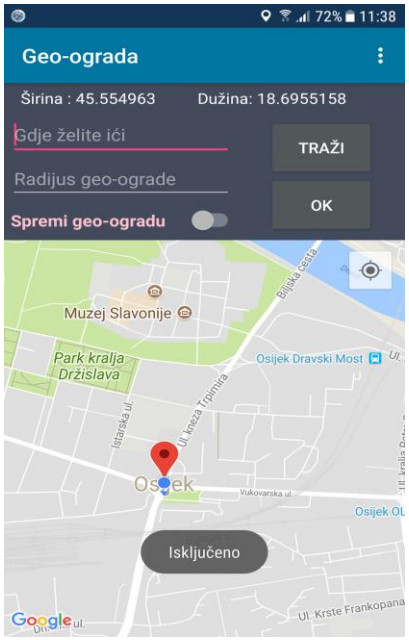


Slika 3.36. Oznaka trenutne lokacije

Ako se želi spremiti geo-ograda, dovoljno je uključiti *switch* kontrolu naziva 'Spremi geo-ogradu'. Tada se pojavljuje poruka 'Uključeno' kao na slici 3.37. Isključivanjem se onemogućuje spremanje geo-ograde te se ona neće prikazati pri idućem ulasku u aplikaciju. Ako se kontrola pomakne s desna na lijevo, ispisat će se poruka 'Isključeno'. Slika 3.38. prikazuje isključivanje *switch* kontrole s porukom.



Slika 3.37. Uključeno spremanje

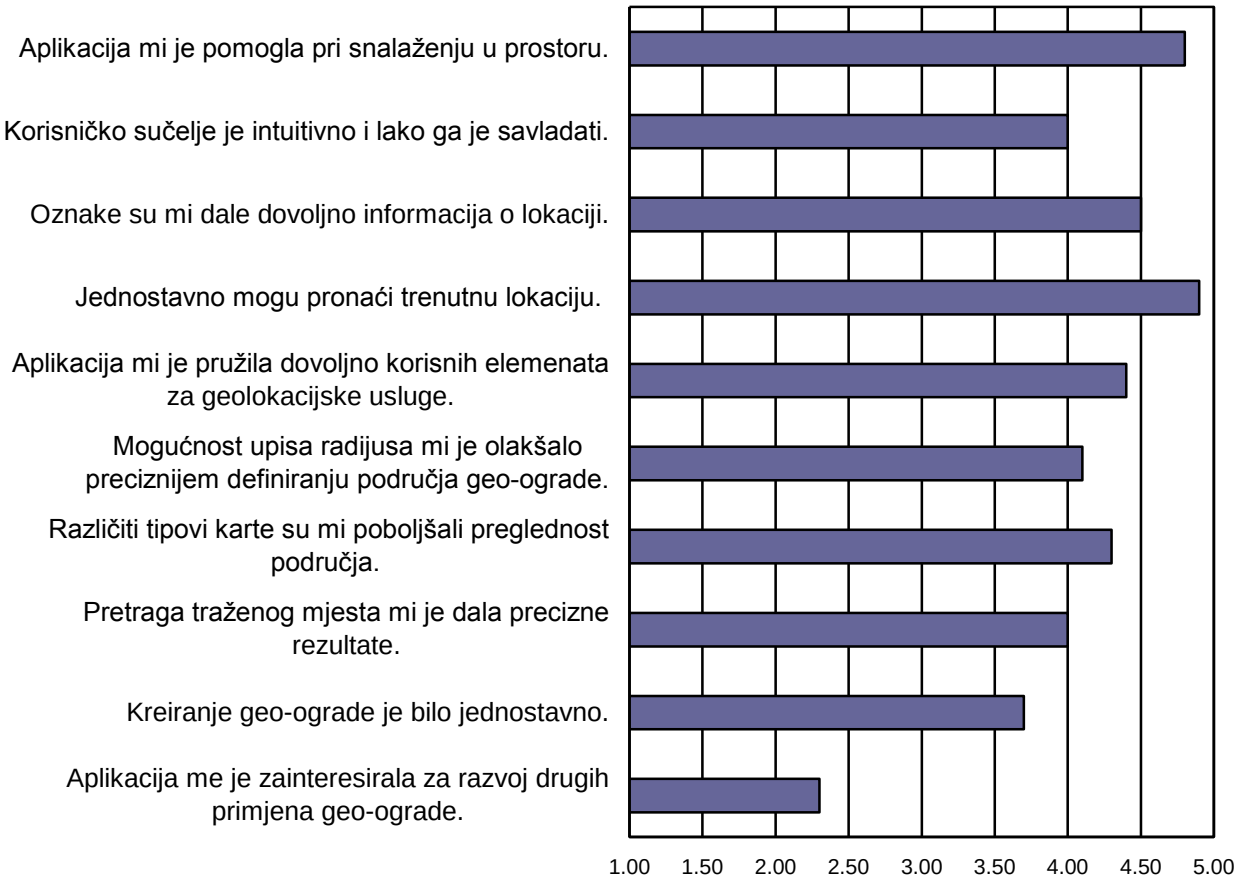


Slika 3.38. Isključeno spremanje

Testirana aplikacija nudi brojne opcije za lakše snalaženje u prostoru poput pretraživanja mjesta, ulaska u označeno područje pomoću geo-ograde, zumiranja trenutne korisnikove lokacije te prikaza četiri vrste karte za njen bolji pregled.

3.5. Korisničko testiranje

Nakon što je više korisnika testiralo aplikaciju, korisnicima je dana anketa u kojoj se provjeravalo zadovoljstvo aplikacijom. Anketa je sadržavala tvrdnje na koje je korisnik mogao dati jedan od pet mogućih odgovora, čime je izražen stupanj slaganja, odnosno neslaganja sa stavom izraženim u tvrdnji. U odgovorima, u kojima su pridruženi brojevi od jedan do pet, jedan predstavlja korisnikovo potpuno neslaganje, dok pet predstavlja korisnikovo potpuno slaganje s tvrdnjom. Na slici 3.39. prikazan je graf koji predstavlja rezultate ankete.



Slika 3.39. Rezultati ankete

Rezultati pokazuju koliko su korisnici bili uspješni u snalaženju i savladavanju pojedinih elementa aplikacije te koliko ih je aplikacija zainteresirala za dodatan razvoj, unaprjeđenje i učenje novih tehnologija. Iz toga se zaključuje da je interes za razvoj aplikacije ocijenjen kao najmanje poželjna značajka, dok je snalaženje u prostoru ocijenjeno kao najbolja značajka aplikacije. Također se zaključuje da su korisnici zadovoljni sa opcijama koje aplikacija nudi.

4. ZAKLJUČAK

Geolokacija se koristi za identifikaciju trenutne pozicije nekog uređaja, a oblik uključuje geografske koordinate za utvrđivanje adrese. Za ovaj završni rad primarno je bilo istražiti mogućnosti primjene geolokacije i servise koji omogućuju različite načine njena konzumiranja, a zatim je bilo potrebno geolokacijske usluge (elemente poput geo-ograde) ugraditi u programsko rješenje. U teorijskom dijelu su opisane različite geolokacijske usluge kao i mogućnosti njihove uporabe, a u praktičnom je dijelu ostvareno programsko rješenje za Android platformu koje se bazira na opisanim geolokacijskim uslugama. Ostvarene su opcije poput stvaranja, spremanja i brisanja geo-ograde, upisa njenog radijusa, pretrage mjesta i trenutne korisnikove lokacije, brisanje oznaka te različiti tipovi karte za bolji pregled područja. Uz to, omogućena je opcija primanja obavijesti pri ulasku u naznačeno područje geo-ograde te je oznakama dana informacija o lokaciji u obliku geografskih koordinata (širine i dužine) te u obliku adrese. Aplikacija je kreirana u Android Studiu, u kojem je programski dio koda napisan u Java programskom jeziku, a izgled same aplikacije u XML jeziku. Za dodatni razvoj i unaprjeđenje ove aplikacije može se omogućiti opcija kreiranja više geo-ograda, mogućnost pregleda svih stvorenih oznaka unutar liste, zatim različiti oblici geo-ograde koju bi korisnik kreirao dodavanjem i spajanjem točaka na kartu definirajući tako oblik ili mogućnost povezivanja s drugim korisnicima.

LITERATURA

[1] Android Location Providers – gps, network, passive – Tutorial, Nazumlidris, https://developerlife.com/2010/10/20/gps/ [posjećeno 24.05.2018.]

[2] Simple, battery-efficient location API for Android, Google Developers, https://developers.google.com/location-context/fused-location-provider/ [posjećeno 24.05.2018.]

[3] B. Hofmann-Wellenhof, H. Lichtenegger, E.Wasle, GNSS – Global Navigation Satellite Systems: GPS, GLONASS, Galileo and more, str:277,341, Beč, 2008.

[4] A. Carvalho, S. Rodriguez-Gonzalez, J. De Paz, J. Corchado, Distributed Computing and Artificial Intelligence, str.303-310, Berlin, Njemačka, 2010.

[5] Y. Huang, H. Chao, D.Deng, J. Park, Advanced Technologies, Embedded and Multimedia for Human-centric Computing, str. 350, Dordrecht, Nizozemska, 2014.

[6] H. Lehpamer, RFID Design Principles, str.55, Norwood, Massachusetts, 2012.

[7] Calculate distance, bearing and more between latitude/longitude points, Movable Type Scripts, https://www.movable-type.co.uk/scripts/latlong.html [posjećeno 02.09.2018.]

[8] Android Platform, techopedia, https://www.techopedia.com/definition/4219/android-platform [posjećeno 26.05.2018.]

[9] Sensors Overview, Developers, https://developer.android.com/guide/topics/sensors/sensors_overview [posjećeno 28.05.2018.]

[10] Platform Architecture, Developers, https://developer.android.com/guide/platform/ [posjećeno 28.05.2018.]

[11] S. Hashimi, S. Komatineni, D. MacLean, Pro Android 3, str.29, New York, 2011.

[12] G. Milette, A. Stroud, Professional Android Sensor Programming, str. 16,17, Indianapolis, Indiana, 2012.

[13] S. Helal, R. Bose, W. Li, Mobile Platforms and Development Environments, str. 65-67, 2012.

[14] I. Darwin, Android Cookbook: Problems and Solutions for Android Developers, str. 606, Sebastopol, California, 2017.

[15] Geofences on Android, Tin Megali, https://code.tutsplus.com/tutorials/how-to-work-with-geofences-on-android--cms-26639 [posjećeno 31.05.2018.]

[16] Public Methods, Google APIs for Android, https://developers.google.com/android/reference/com/google/android/gms/location/Geofence.Builder [posjećeno 01.06.2018.]

[17] Marker, Google APIs for Android, https://developers.google.com/android/reference/com/google/android/gms/maps/model/Marker [posjećeno 01.06.2018.]

[18] Zoom, Mooving the camera, Changing zoom level and setting minimum/maximum zoom, Google Maps Platform, https://developers.google.com/maps/documentation/android-sdk/views [posjećeno 01.06.2018.]

[19] R. Amal, Learning Android Google Maps, str.149, Birmingham, Ujedinjeno Kraljevstvo, 2015.

[20] Comparison of web map services, Wikipedia, https://en.wikipedia.org/wiki/Comparison_of_web_map_services [posjećeno 14.06.2018.]

[21] AMC The School of Bussiness, Android Studio:Step-by-step Training Manual: Apps Development, str:2,4

[22] E.Ray, Learning XML: Creating Self-Describing Data, str:1-2, Sebastopol, California, 2003.

[23] Java (programski jezik), Wikipedia, https://hr.wikipedia.org/wiki/Java_(programski_jezik) [posjećeno 06.06.2018.]

SAŽETAK

Cilj ovog završnog rada bio je kreiranje Android aplikacije koja će korisniku omogućiti stvaranje geo-ograde i uporabu geolokacije. Geo-ograda je virtualni ograničeni prostor na karti pomoću koje je moguće znati je li korisnik unutar tog označenog područja, a temelji se na stvarnom geografskom prostoru. Geolokacija služi za identifikaciju lokacije nekog uređaja, a njen oblik uključuje geografske koordinate (geografsku širinu i dužinu) koje omogućuju utvrđivanje adrese. Geokodiranje je proces koji pretvara adresu u koordinate, a obrnuto geokodiranje pretvara geografske koordinate u adresu koja je čitljiva i razumljiva ljudima. U teorijskom dijelu utvrđeni pojmovi i brojni primjeri pomogli su razumijevanju kreiranja aplikacije. Programsko rješenje omogućuje korisnicima korištenje raznih elementa aplikacije, poput kreiranja i brisanja geo-ograde, njenog spremanja i upisa radijusa, pretragu željenog mjesta, zumiranje trenutne korisnikove aplikacije i ostalog. Testiranje aplikacije uspješno je pokazalo njeno jednostavno korištenje nudeći brojne opcije. U realizaciji su korištene tehnologije poput Android Studia, Android SDK i JDK, Android virtualnog uređaja (AVD), Java programskog jezika te XML jezika.

Ključne riječi: Android, geolokacija, geo-ograda, Java, XML

ABSTRACT

Geolocation services in mobile application development for Android platform

The aim of this thesis was to create an Android application that will enable the user to create a geo-fence and use geolocation. Geo-fence is a limited virtual space on a map for a real-world geographic area, by which we can identify if the user is within the highlighted area. It is based on real geographical areas. Geolocation is used to identify the location of a device by means of geographical coordinates (latitude and longitude) that enable identification of an address. Geocoding is the process that converts an address into the coordinates, whereas reverse geocoding transforms the geographic coordinates into an address, which human users can read and understand easily. The theoretical part examples provided for better understanding of the applications due to well defined terms and numerous examples. The platform allows users to use various elements of applications, from creating and deleting geo-fence, its storage and making radius entries, up to searching for a particular point, zooming the current user's applications etc. Testing period successfully demonstrated application´s user friendly design by offering a number of options. Its standardizations required the implementation of technologies such as Android Studio, Android SDK and JDK, Android Virtual Device (AVD), the Java programming language and XML languages.

Key words: Android, geolocation, geo-fence, Java, XML

ŽIVOTOPIS

Marina Vratarić rođena je 29.12.1995. godine u Slavonskom Brodu. Pohađala je osnovnu školu „Ivan Mažuranić” u Sibinju od 2002. do 2010. godine. Nakon završene osnovne škole upisuje prirodoslovno-matematičku gimnaziju „Matija Mesić” u Slavonskom Brodu. Maturirala je 2014. godine te je iste godine upisala Preddiplomski sveučilišni studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.

PRILOZI

Na CD-u:

1. Završni rad „Geolokacijske usluge u razvoju mobilnih aplikacija za Android platformu.docx“
2. Završni rad „Geolokacijske usluge u razvoju mobilnih aplikacija za Android platformu.pdf“
3. Izvorni kod