PT&SCHE

Introduction of part-time and short cycle studies in Serbia

Funded by the Erasmus+ Programme of the European Union

This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

UNIVERZITET METROPOLITAN BEOGRAD

KI205 - JAVA 8: PROGRAMIRANJE U JAVI NA ANDROID PLATFORMI

Primena lokacijskih servisa

Lekcija 08

PRIRUČNIK ZA STUDENTE

KI205 - JAVA 8: PROGRAMIRANJE U JAVI NA ANDROID PLATFORMI

Lekcija 08

PRIMENA LOKACIJSKIH SERVISA

UVOD

U narednoj lekciji cilj će biti savladavanje rada sa lokacijskim servisima u Android aplikacijama.

U narednoj lekciji cilj će biti savladavanje rada sa lokacijskim servisima u Android aplikacijama. Lokacijski servisi ili skraćeno LBS (eng, Location Based Services), omogućavaju Android aplikacijama da prate određenu lokaciju i na taj način ponude mnoštvo dodatnih servisa, kao što su lociranje određenih sadržaja u okruženju, pružanje saveta oko planiranja putanje kretanja, navigacija itd. Osnovni deo LBS aplikacije predstavljaju mape koje su vizuelna reprezentacija trenutne lokacije.

Na početku rada sa LBS, akcenat će biti na korišćenju *Google Maps* servisa u Android aplikaciji. Takođe, biće naučeno kako se koristi klasa *Geocoder* za određivanje adrese geografske lokacije.

Lekcija će, zaokružiti izlaganje, kreiranjem *konkretne GoogleMaps* aplikacije.

Lekcija će staviti poseban akcenat na sledeće teme:

- Prikazivanje Google mapa u Android aplikaciji;
- Demonstriranje kontrola za uvećanje mapa;
- Prikazivanje mapa na različite načine;
- Dodavanje markera u mape;
- Dobijanje adrese lokacije obeležene na mapi;
- Traženje lokacije;
- Kreiranje GoogleMap aplikacije.

Dakle, Savladavanjem ove lekcije studenti će ovladati standardnim mehanizmima rukovanja Google mapama. Studenti će moći da postavljaju markere na odgovarajuće lokacije i da preuzimaju informacije o lokaciji. Zatim, studenti će moći da menjaju način prikazivanja mape, da postavljaju kontrole na mapu, poput zuma, kompasa i slično. Posebno, akcenat će biti na savladavanju načina pribavljanja API šifre neophodne za funkcionisanje Android aplikacije koja radi sa Google mapama. Na kraju, studenti će razumeti i koristiti važan koncept rada sa Google mapama - geokodiranje.

Poglavlje 1

Prikazivanje mapa

UPOTREBA GOOGLE MAPA

Google Maps je standardna, ugrađena, Android aplikacija.

Primena lokacijskih servisa u Android operativnom sistemu podrazumeva primenu i operacije nad ugrađenim Google mapama koje su sastavni deo paketa programa svakog mobilnog uređaja pokrenutog Android operativnim sistemom.

Google Maps je jedna od brojnih ugrađenih aplikacija Android platforme. Aplikacija je veoma jednostavna za korišćenje i, takođe, nudi brojne funkcionalnosti koje je moguće iskoristiti u vlastitim Android aplikacijama. Ova deo kursa ima poseban zadatak da se posveti temama kao što su:

- Promena načina prikaza Google mapa;
- Primena Google mapa za određivanje geografske širine i dužine praćene lokacije;
- Izvršavanje geokodiranja (prevođenje geografske širine i dužine u adresu);
- Dodavanje markera na Google mape.

Za početak rada sa *Google* mapama, neophodno je u *Android Studio IDE* razvojnom okruženju kreirati nov projekat koji će biti nazvan *MapeDemo na Google API nivou koji odgovara verziji* operativnog sistema Android 6.0. Nakon toga, neophodno je obezbediti dodatnu datoteku pod nazivom *maps.jar* u folderu *Google APIs* . Da bi bilo moguće koristiti Google Maps servise, u vlastitoj Android aplikaciji, neophodno je proveriti dostupnost Google API interfejsa prilikom kreiranja aplikacije. Pošto *Google Maps* nije deo standardnih Android SDK alata, neophodno ga je pronaći i obezbediti među *Google API* interfejsima.

DOBIJANJE MAPS API ŠIFRE – MD5 I SHA1 OPIS

MD5 ili SHA1 opis je neophodan za dobijanje Google Maps šifre.

Nakon identifikovanja datoteke za čuvanje ključa, *debug.keystore*, neophodno je dobiti MD5 ili SHA1 opis pomoću *Keytool.exe* aplikacije integrisane u JDK instalaciju.

Za pribavljanje opisa neophodno je startovati aplikaciju *cmd.exe* i otkucati sledeći kod:

Keytool.exe –list –alias androiddebugkey –keystore “C:\Users\Vladimir Milicevic\.android\ debug.keystore” –storepass android –keypass android -v

Komanda je koristila sledeće argumente:

- list - detalji o navedenom skladištu ključeva;
- alias – drugi naziv za skladište ključeva;
- keystrore – lokacija skadišta ključeva;
- storepass – šifra za skladište ključeva;
- keypass – šifra ključa u skladištu ključeva.

Izvršavanjem komande dobiće se podaci o vrednosti MD5 opisa. Takođe, ukoliko programer koristi najnoviju verziju *Eclipe IDE* razvojnog okruženja, ovaj posao je znatno olakšan i MD5 opis može da se pročita u već pokazanom *Build* prozoru. Takođe, SHA1 opis može biti uslov za ključ (nova Google konzola).

Pribavljanje opisa prikazano je sledećom slikom.

Slika 1.1 Pribavljanje MD5 i SHA1 opisa

DOBIJANJE MAPS API ŠIFRE – DEBUG SERTIFIKAT

Za dobijanje Google Maps API šifre neophodna je registracija.

Za integrisanje *Google Maps* servisa u vlastitu Android aplikaciju, neophodno je registrovanje sa ciljem dobijanja besplatnog *Google Maps ključa (šifre).* Prijavljivanje za *Google Maps* šifru vrši se na lokaciji: *code.google.com/android/maps-api-signup.html* , nakon čega je neophodno pratiti određena uputstva i pravila korišćenja određena od strane kompanije *Google.*

Ukoliko se Android aplikacija testira na emulatoru ili Android uređaju koji su povezani sa razvojnim računarom, neophodno je locirati SDK debug sertifikat koji se nalazi u podrazumevanom folderu (C:\Users\<naziv_korisnik>\.android\). Postojanje sertifikata se proverava, u *Eclipse IDE* razvojnom okruženju, praćenjem putanje menija *Windows-Preferences-Android-Build.* U prozoru *Build* može da se uoči lokacija debug sertifikata.

Build prozor, sa debug sertifikatom, prikazan je sledećom slikom.

Slika 1.2 Debug sertifikat

DOBIJANJE MAPS API ŠIFRE – ANDROID STUDIO

Neophodno je pokazati i postupak dobijanja šifre primenom Android Studio IDE.

Budući da Google sve više favorizuje Android Studio i sukcesivno ukida podršku razvoja Android aplikacija drugim razvojnim okruženjima, neophodno je posebno pokazati kako je moguće započeti postupak dobijanja potrebne Google Maps API šifre za funkcionisanje aplikacije koja će biti u nastavku kreirana za potrebe primera.

Za dobijanje Google Maps API šifre neophodno je obezbediti SHA1 sertifikat. Pribavljanje ovog sertifikata realizuje se praćenjem sledećih koraka:

1. Pokretanje Android Studio IDE razvojnog okruženja;

2. Otvaranje relevantnog projekta;
3. Kliknuti na Gradle (na desnoj strani panela, pojaviće se Gradle Bar);
4. Kliknuti na opciju Refresh nakon čega će se pojaviti lista Gradle skripti odgovarajućeg projekta;
5. Kliknuti na naziv projekta (naziv projekta formira koren liste);
6. Kliknuti na Tasks;
7. Kliknuti na Android;
8. Duplim klikom na signingReport (pojaviće se SHA1 i MD5 u Run Bar - u).

Navedeno je moguće prikazati sledećom slikom.

Slika 1.3 Pribavljanje SHA1 ključa

Sledećim video materijalom su pokazana osnovna podešavanja Google Maps aplikacije i nabavljanje Google API ključa.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

DOBIJANJE MAPS API ŠIFRE – PRIJAVLJIVANJE

Google je obezbedio lokaciju sa koje se, primenom MD5 ili SHA1 opisa, prijavljuje i preuzima ključ.

Nakon što su obezeđeni opisi sertifikata, pristupa se stranici, pomoću web čitača, na kojoj se vrši prijavljivanje i preuzimanje API ključa. Unošenjem originalne adrese stranice code.google.com/android/maps-api-signup.htm vrši se preusmeravanje na lokaciju na kojoj je implementirana inovirana konzola za pribavljanje API ključa. Nova konzola zahteva da se ispoštuju sledeći koraci:registrovanje paketa u kojem se nalazi aplikacija koja zahteva API ključ, a zatim se unosi vrednost SHA1 opisa. Još je neophodno ključu dodeliti naziv i kliknuti na *Create* dugme (sledeća slika).

Slika 1.4 Prijavljivanje za API ključ

Nakon što su obavljeni navedeni zadaci, dobija se Google Mapsključ (sledeća slika). Ono što bi još trebalo napomenuti jeste da ovaj ključ služi za prevođenje i testiranje aplikacije na emulatoru ili mobilnom uređaju i nije validan ako se za aplikaciju pokušava kreirati APK datoteka. Kada je aplikacija gotova i spreman da se postavi na Google Play Market, ili distribuira na neki drugi način, neophodno je ponovo se prijaviti za Maps API ključ, sertifikatom za *potpisivanje* kreirane aplikacije.

Slika 1.5 Google Maps API ključ

Sledeći zadatak, koji se postavlja prek kreatora Android aplikacije, u kojoj se koriste Google mape, jeste čuvanje pribavljenog Google Maps ključa u odgovarajućoj datoteci projekta. Datoteka u kojoj će se nalaziti dobijeni ključ je AndroidManifest.xml i ovaj će problem biti analiziran u nekom od sledećih izlaganja.

PRIMER 1 - PRIKAZIVANJE MAPE – UI FRAGMENT

Prikazivanje mapa iz korisničkog interfejsa realizovano je fragmentom u glavnoj aktivnosti.

Nakon dobijanja Google Maps API ključa, sve je spremno za prikazivanje mapa u aplikaciji koja će biti kreirana. Nameću se sledeća dva zadatka:

- Modifikacija AndroidManifets.xml datoteke primenom <uses-library> elementa i Internet privilegije;
- Kreiranje fragmenta korisničkog interfejsa.

Android Studio IDE značajno olakšava pisanje programa koji kao vlastitu funkcionalnost koriste Google mape. Prilikom kreiranja projekta moguće izabrati i šablon Google Maps aplikacije, pri čemu će automatski biti kreiran fragment za učitavanje Google mape u odgovarajuću aktivnost. Izbor šablona Google map aplikacije je prikazan sledećom slikom.

Slika 1.6 Izbor šablona za Google Maps aplikaciju

U nastavku, neophodno je definisati i prikazati korisnički interfejs aplikacije koja se kreira za demonstraciju primene Google mapa u korisničkim aplikacijama. Android Studio IDE, na način prikazan prethodnom slikom, forsira učitavanje mapa u posebnom fragmentu. Upravo tako će i biti generisan korisnički interfejs aplikacije. Sledećim kodom prikazana je activity_maps.xml datoteka korisničkog interfejsa.

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:map="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.vladimirmilicevic.mape.MapsActivity" />
```

PRIKAZIVANJE MAPE – DOZVOLE

AndroidManifest.xml skladišti privilegije i API ključ za korišćenje Google Maps servisa.

Veoma značajnu ulogu u korišćenju Google Mapsservisa u Android aplikaciji vrši datoteka AndroidManifest.xml. Za nove API generacije, Google Maps ključ, za pristup servisima, smešta se u AndroidManifest.xml datoteku (ranije je išao u main.xml). Sledeći element prikazuje deo AndroidManifest.xml datoteke u kojoj se čuva API ključ. Api ključ se čuva elementom koji je obavezno deo XML elementa <application>…. </application>.

Takođe, sve privilegije neophodne za pristup i korišćenje Google mapa i servisa navode se u AndroidManifest.xml datoteci. Ovi elementi ne zavise od drugih XML elemenata, navode se samostalno u tagu <uses permission ...>

```
<meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value= "AIzaSyB4V9JDQBTeQfQgz-2CyxuHHsJly8Puk2w" />
" />
```

Sledeća lista predstavlja najčešće korišćene privilegije u Android aplikacijama za rad sa *Google* mapama:

- ***android.permission.INTERNET*** – Dozvola koju API koristi za preuzimanje mape sa *Google Maps* servera;
- ***android.permission.ACCESS_NETWORK_STATE*** – Dozvola da API proveri status konekcije u cilju donošenja odluke da li podaci mogu biti preuzeti sa servera;
- ***android.permission.ACCESS_COARSE_LOCATION*** – Dozvola da mapa pristupi približnoj lokaciji;
- ***android.permission.ACCESS_FINE_LOCATION*** – Dozvola da mapa pristupi prieciznoj lokaciji;
- ***naziv_paketa.naziv_aplikacije.permission.MAPS_RECEIVE*** – Dozvola da aktuelna Android aplikacija preuzme *Google* mapu.

Za pakovanje mape u aktivnost *Google Map API* koristi *Open Gl ES 2*, a to je takođe određeno sledećim elementom.

```
<uses-feature
        android:glEsVersion="0x00020000"
        android:required="true"/>
```

ANDROIDMANIFEST.XML – PRIVILEGIJE I PODEŠAVANJA

Kompletiranjem AndroidManifest.xml datoteke omogućeno je korišćenje Google Maps servisa aplikaciji.

AndroidManifest.xml datoteka aplikacije nosi u sebi sve neophodne informacije i podešavanja neophodna da Android aplikacija koristi *Google* mape i servise. Sledeći elementi su ugrađeni u AndroidManifest datoteku primera: naziv paketa (crveno), SDK informacije (plavo), privilegije (žuto), informacije o korišćenju *Open GL ES2* za rendanje učitane mape (crno), podešavanja vezana za korišćene biblioteke (zeleno) i glavnu aktivnost (pink), meta podaci o API ključu (braon) itd. Navedeno je prikazano sledećom slikom.

Slika 1.7 AndroidManifest.xml aplikacije koja radi sa Google mapama

PRIKAZIVANJE MAPE – OSNOVNA KLASA AKTIVNOSTI - PRIMER

Učitavanje mape u fragment je zadatak klase aktivnosti.

Sada je moguće kreirati i klasu aktivnosti, sa funkcijama za prikazivanje *Google* mapa. Klasa će da nasleđuje osnovnu klasu FragmentActivity i da implementira interfejs OnMapReadyCallback. Sledeći zadatak je angažovanje fragment menadžera koji će na osnovu *id* elementa iz activity_main.xml datoteke uključiti ftagment u aktivnost. Zatim će biti uključena metoda getMapAsync() čiji je zatatak podešavanje poziva fragmenta.

Na kraju, za učitavanje osnovne *Google* mape, neophodno je, u klasu aktivnosti, dodati metodu onMapReady() za upravljanje GoogleMap objektom. Ovaj objekat će biti, u kasnijim izlaganjima, korišćen da se na mapu implementiraju razne funckionalnosti. Za početak, biće iskorišćen da se u određenoj tački mape postavi marker. Uopšteni kod ove metode onMapReady() prikazan je sledećim listingom.

```
public void onMapReady(GoogleMap mMap) {
mMap.addMarker(new MarkerOptions().position(LatLng(0,0)).title("Marker"));
}
```

U nastavku je neophodno kreirati klasu aktivnosti koja će učitati fragment sa Google mapom i omogućiti izvesne funkcionalnosti. Na primer, na unapred definisanoj lokaciji biće postavljen marker uz odgovarajući prateći komentar.

Sledećim listingom je prikazan početni kod klase aktivnosti. Kako budu uvođene dodatne funkcionalnosti u aplikaciju, ovaj kod će biti proširivan.

```
public class MapsActivity extends FragmentActivity implements OnMapReadyCallback {

    private GoogleMap mMap;
        /**
     * ATTENTION: This was auto-generated to implement the App Indexing API.
     * See https://g.co/AppIndexing/AndroidStudio for more information.
     */
    private GoogleApiClient client;

     @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
        // Obtain the SupportMapFragment and get notified when the map is ready to
be used.
        SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager()
                .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);

        // ATTENTION: This was auto-generated to implement the App Indexing API.
        // See https://g.co/AppIndexing/AndroidStudio for more information.
```

```
         client = new GoogleApiClient.Builder(this).addApi(AppIndex.API).build();
    }

 public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;
        mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);/* ovde će se podešavati tip
mape*/
        // Marker postavljen u Kragujevac i pomerena kamera
        LatLng kg = new LatLng(44.017, 20.917);
        mMap.addMarker(new MarkerOptions().position(kg).title("Marker u
Kragujevcu"));
        mMap.moveCamera(CameraUpdateFactory.newLatLng(kg)); //postavljanje markera
na zadatu lokaciju
     }

 }
```

PRIKAZIVANJE MAPE - FUNKCIONISANJE

GoogleMap klasa je šablon za kreiranje objekata koji daju funkcionalnosti mapi učitanoj android aplikacijom.

GoogleMap klasa je šablon za kreiranje objekata koji daju funkcionalnosti mapi učitanoj android aplikacijom. U okviru UI, mapa će biti realizovana kao *MapFragment* ili *MapView* objekat. Klasa *GoogleMap* automatski upravlja sledećim funkcionalnostima:

- Povezivanje na Google Maps servise;
- Preuzimanje segmenata mapa;
- Prikazivanje segmenata mapa na ekranu uređaja;
- Prikazivanje različitih kontrola za upravljanje mapama (zum, kompas i sl);
- Reakcija na primenu kontrole.

Dalje, MapFragment je potklasa klase Fragment i omogućava da se mapa spakuje u Android fragment.

U metodi onMapReady() inicirani su objekti tipa LatLng, koji ima zadatak da preuzme geografsku širinu i dužinu, i *GoogleMap*, čiji je zadatak postavljanje markera, pozivom metode *addMarker()*, u zadatoj geografskoj širini i dužini.

U Android Studio IDE razvojnom okruženju, izborom opcije Run App ili Shift + F10, prevodi se i pokreće kreirana aplikacija i testira u izabranom emulatoru. Rezultat pokretanja aplikacije je prikazan sledećom slikom.

Slika 1.8 Učitavanje mape i postavljanje markera

ZADATAK 1 - OBJASNITE LISTING METODE ONMAP

Pokušajte sami!!!

Pogledajte pažljivo priloženi listing i dajte njegovo objašnjenje!!!

```
public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;
        mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);/* ovde će se podešavati tip
mape*/
        // Marker postavljen u Kragujevac i pomerena kamera
        LatLng kg = new LatLng(44.017, 20.917);
        mMap.addMarker(new MarkerOptions().position(kg).title("Marker u
Kragujevcu"));
        mMap.moveCamera(CameraUpdateFactory.newLatLng(kg)); //postavljanje markera
na zadatu lokaciju
     }
```

Poglavlje 2

Operacije nad Google mapama

ULAZNO – IZLAZNE KONTROLE

Klasa UISettings omogućava aktiviranje i vidljivost ulazno-izlaznih kontrola Google Map fragmenta.

Google Maps API obezbeđuje rad sa kontrolama koje su slične kontrolama koje se nalaze u ugrađenoj *Google Maps* aplikaciji mobilnog uređaja. Obezbeđivanje vidljivosti ovih kontrola omogućeno je primenom klase UISettings koja se može dobiti i iz klase *GoogleMap* pozivom metode GoogleMap.getUISettings(). Većina ovih kontrola može biti konfigurisana i krozXML atribute ili primenom GoogleMapOptions klase.

Svaka kontrola ima unapred definisanu poziciju u odnosu na ivice ekrana mobilnog uređaja. Pozicija kontrola može biti promenjena upotrebom metode setPadding() na GoogleMap objekat.

Posebno bi trebalo napomenuti da je za upravljanje ulazno - izlaznim kontrolama, u kreiranoj Android aplikaciji koja podržava rad sa Google mapama, zadužen objekat tipa GoogleMap. Ovaj objekat je, prvo, neophodno konstruisati, a zatim pomoću njega pozivati metode za postavljanje i upravljanje ulazno - izlaznim kontrolama na Google mapi. U ovu svrhu, inicijalni projekat, odnosno njegova glavna klasa aktivnosti, biće proširivana novim funkcionalnostima za demonstraciju upotrebe ulazno - izlaznih kontrola Google mapa.

Uz ovu temu priložen je i video materijal koji demonstrira primenu ulazno - izlaznih kontrola u Android aplikaciji.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

PRIMER 2 - ZOOM KONTROLA

Uvećanje i smanjene razmere mape postiže se primenom zoom kontrole.

Prva kontrola koja će biti obrađena i koju je moguće postaviti na Google mapu, učitanu kreiranom Android aplikacijom, jeste *kontrola za uvećanje i smanjenje prikaza Google* mape. Uključivanje ove kontrole je prilično jednostavno. Kreiranim GoogleMap objektom, poziva se metoda setZoomControls(). U zavisnosti da li se želi uključivanje ili isključivanje ove kontrole, ona preuzima logičku konstantu **true** ili **false**, kao argument, respektivno.

Prethodni primer, sa *Google* mapom bez funkcionalnosti, može biti proširen dodavanjem *zoom* kontrole u metodu onMapReady(). Odmah ispod seta naredbi za postavljanje markera na unapred zadatu lokaciju, postavlja se programski kod koji u Google mapu dodaje kontrolu za zumiranje. Navedeno je dokumentovano sledećim listingom koji je izolovan iz glavne klase aktivnosti.

```
@Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;
        mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);/* ovde će se podešavati tip
mape*/
        // Marker postavljen u Kragujevac i pomerena kamera
        LatLng kg = new LatLng(44.017, 20.917);
        mMap.addMarker(new MarkerOptions().position(kg).title("Marker u
Kragujevcu"));
        mMap.moveCamera(CameraUpdateFactory.newLatLng(kg)); //postavljanje markera
na zadatu lokaciju
        mMap.getUiSettings().setZoomControlsEnabled(true);//zoom kontrola
}
```

Da bi ugrađena kontrola za zumiranje mogla da bude ugrađena i prikazana na Google mapi koja je učitana u fragment kreirane aplikacije, neophodno je ponovo izvršiti prevođenje aplikacije. Klikom na opciju Run App ili Shift + F10 u Android Studio IDE razvojnom okruženju, ponovo se prevodi i pokreće aplikacija, u izabranom emulatoru ili realnom mobilnom uređaju. Rezultati pokretanja aplikacije prikazani su sledećom slikom.

Slika 2.1 Kreirana zoom kontrola

PRIMER 3 - KOMPAS KONTROLA

Kompas se aktivira, ili isključuje, pozivanjem metode UiSettings.setCompassEnabled(boolean).

Google Map API obezbeđuje kontolu *Kompas (*Compass) koja se pod određenim uslovima pojavljuje u gornjem levom delu ekrana. Kompas se pojavljuje isključivo u situaciji kada je mapa zarotirana i crvenom strelicom pokazuje stranu sveta *sever*. Kompas se aktivira, ili isključuje, pozivanjem metode UiSettings.setCompassEnabled(boolean), GoogleMap *objektom,* za atribut **true** ili **false,** respektivno. Sledećim kodom je prikazano dodavanje kontrole kompas u metodu *onMapReady()*. Ovaj kod je dodat kao proširenje dosadašnjeg koda glavne klase aktivnosti i nalazi se odmah ispod koda prethodno kreirane kontrole za zumiranje Google mape.

```
@Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;
        mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);/* ovde će se podešavati tip
mape*/
        // Marker postavljen u Kragujevac i pomerena kamera
        LatLng kg = new LatLng(44.017, 20.917);
```

```
        mMap.addMarker(new MarkerOptions().position(kg).title("Marker u
Kragujevcu"));
        mMap.moveCamera(CameraUpdateFactory.newLatLng(kg)); //postavljanje markera
na zadatu lokaciju
        mMap.getUiSettings().setZoomControlsEnabled(true);//zoom kontrola
        mMap.getUiSettings().setCompassEnabled(true);//postavljanje kompasa
}
```

Analogno prethodnom slučaju, aplikaciju je potrebno ponovo prevesti da bi izmene mogle da budu vidljive krajnjim korisnicima aplikacije. Klikom na opciju Run App ili Shift + F10 u Android Studio IDE razvojnom okruženju, ponovo se prevodi i pokreće aplikacija, u izabranom emulatoru ili realnom mobilnom uređaju. Rezultati pokretanja aplikacije prikazani su sledećom slikom.

Slika 2.2 Kreirana kontrola kompas

PRIMER 4 - DUGME ZA PRIKAZIVANJE MOJE LOKACIJE

Kreiranje dugmeta za pristup vlastitoj lokaciji dešava se pozivom setMyLocationEnabled(true).

Da bi korisnik mogao da pristupi vlastitoj lokaciji na mapi, neophodno je da program uradi dve stvari:

- Uključi servis myLocation ;
- Kreira dugme, koje će, nakon klika, korisniku prikazati njegovu lokaciju.

Kreiranje dugmeta za pristup vlastitoj lokaciji dešava se pozivom metode setMyLocationEnabled(true), objektom klase GoogleMaps. Gašenje, i ponovno uključivanje, ovog dugmeta obezbeđuje se pozivom metode getUiSettings().setMyLocationButtonEnabled(), za vrednosti atributa ***false***ili ***true***, respektivno. Takođe, dodat je kod za izbor tipa Google mape koja će biti prikazana, ali će o tome biti više govora u nekom od narednih izlaganja.

Sledećim kodom data je metoda onMapReady(), izolovana iz glavne klase aktivnosti aplikacije, koja koristi myLocation funkcionalnosti.

```
@Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;
        mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);/* ovde će se podešavati tip
mape*/
        // Marker postavljen u Kragujevac i pomerena kamera
        LatLng kg = new LatLng(44.017, 20.917);
        mMap.addMarker(new MarkerOptions().position(kg).title("Marker u
Kragujevcu"));
        mMap.moveCamera(CameraUpdateFactory.newLatLng(kg)); //postavljanje markera
na zadatu lokaciju
```

```
        mMap.getUiSettings().setZoomControlsEnabled(true);//zoom kontrola
        mMap.getUiSettings().setCompassEnabled(true);//postavljanje kompasa
        mMap.setMyLocationEnabled(true);//vaša lokacija
        mMap.getUiSettings().setMyLocationButtonEnabled(true);//promenom u false,
gasi se dugme
}
```

Takođe, da bi promene bile prikazane krajnjem korisniku, neophodno je ponovo prevesti i pokrenuti aplikaciju izabranim emulatorom ili mobilnim uređajem na poznati način koristeći Android Studio IDE razvojno okruženje. Rezultati pokretanja aplikacije su prikazani sledećom slikom.

Slika 2.3 Dodato dugme za moju lokaciju

PRIMER 5 - MAPTOOLBAR KONTROLA

MapToolbar omogućava korisniku brz pristup ugrađenoj Google Maps aplikaciji mobilnog uređaja.

Zajedno sa predstavljenim kontrolama, još jedna kontrola ima predefinisanu lokaciju na ekranu mobilnog uređaja. Donji desni ugao ekrana, koji je učitao *Google* mapu, rezervisan je za kontrolu MapToolbar. Kontrola omogućava korisniku brz pristup fabrički ugrađenoj *Google Maps* aplikaciji mobilnog uređaja.

Sledećom slikom je prezentovana pozicija ove kontrole u kreiranoj aplikaciji.

Slika 2.4 MapToolbar kontrola

Kada korisnik klinke na ikonicu, u kontroli MapToolbar, Google API kreira nameru *(Intent* objekat) za pokretanje odgovarajuće aktivnosti u ugrađenoj Android aplikaciji *Google Maps.* Kontrolu uključuje, i isključuje, GoogleMap objekat pozivom metode UiSettings.setMapToolbarEnabled(boolean).

Sledećom slikom je prikazan ekran mobilnog uređaja, ili emulatora, nakon izvršenog klika na kontrolu MapToolbar aplikacije koja je kreirana za demonstriranje rada sa Google mapama u Android operativnom sistemu.

Slika 2.5 Ugrađena Google Maps aplikacija

GESTIKULACIJE I/O KONTROLA

Kontrole učitane Google mape ponašaju se analogno kontrolama ugrađene Google Maps Android aplikacije.

Kontrole učitane Google mape ponašaju se analogno kontrolama ugrađene Google Maps Android aplikacije. Međutim, kontrole zahtevaju ponekad i neko specifično ponašanje pa je neophodno upravljati njima iz programskog koda. Kao što je slučaj sa njihovom kontrolama,

gestikulacije je takođe moguće uključiti/isključiti pomoću GoogleMap.getUiSettings, a moguće je njihovo podešavanje prepustiti XML elementima.

Kontrola za zumiranje manifestuje ponašanje u skladu sa promenom nivoa zumiranja kamere:

- Dvostrukim klikom na mapu povećava se nivo zumiranja za 1;
- Klik sa dva prsta smanjuje nivo zumiranja za 1;
- Skupljanje i rastezanje mape pomoću dva prsta, itd.

Pozivom metode *UiSettings.setZoomGesturesEnabled(boolean)* uključuju se, ili se gase, zum gestikulacije.

Takođe, učitana mapa u Android aplikaciju može da dozvoli i skrolovanje povlačenjem mape prstom. Pozivom metode *UiSettings.setScrollGesturesEnabled(boolean)*, ova gestikulacija može biti dostupna ili nedostupna.

Povlačenjem dva prsta preko mape, menja se njen nagib *(tilt)*. Pozivom metode *UiSettings.setTiltGesturesEnabled(boolean)* ova gestikulacija može biti uključena/isključena.

Takođe, klikom na mapu sa dva prsta koji, potom, započinju rotaciju, učitana mapa rotira. Rotacija se uključuje/isključuje pozivom metode *UiSettings.setRotateGesturesEnabled(boolean).*

Kao što je moguće primetiti, svim navedenim metodama se obraća objekat tipa *GoogleMap.* Sledećom tabelom data je lista dostupnih gestikulacija za ulazno - izlazne kontrole Google mapa sa odgovarajućim objašnjenjima.

Slika 2.6 Gestikulacije nad Google mapama

DOGAĐAJI

Za svaki događaj koji se javlja na Google mapi, neophodno je implementirati odgovarajući interfejs.

Događaji, koji su karakteristični za *Google* mapu, su klik i dugački klik. Otuda postoje i dva interfejsa, za osluškivanje događaja, koji su snabdeveni metodama čije bi predefinisanje trebalo da omogući upravljanje događajima. Navedeni interfejsi su OnMapClickListener i OnMapLongClickListener,i postavljaju se pozivom GoogleMap.setOnMapClickListener(OnMapClickListener) ili GoogleMap.setOnMapLongClickListener(OnMapLongClickListener), respektivno. Reakcija na klik na mapu je događaj onMapClick(LatLng)koji ukazuje na lokaciju na mapi na koju je korisnik kliknuo. Ovaj slučaj će biti posebno diskutovan u izlaganju koje sledi. Po analogiji, reakcija na dugački klik je onMapLongClick(LatLng).

Kada kamera menja položaj, za osluškivanje događaja je zadužen interfejs OnCameraChangeListener koji se postavlja na mapu pozivom GoogleMap.setOnCameraChangeListener(OnCameraChangeListener). Osluškivač će biti obavešten o promeni kamere sa pozivom metode onCameraChange(CameraPosition).

Takođe, moguće je osluškivati i događaje koji su karakteristični za markere. U aktuelnom primeru klikom na postavljeni marker učitava se određeni komentar (videti sliku).

Reakcija na klik na marker je prikazana sledećom slikom.

Slika 2.7 Klik na marker

ZADATAK 2 - OBJASNITE DOPUNU LISTINGA METODE ONMAP

Pokušajte sami!!!

Pogledajte pažljivo priloženi listing i dajte njegovo objašnjenje!!!

```
LatLng kg = new LatLng(44.017, 20.917);
        mMap.addMarker(new MarkerOptions().position(kg).title("Marker u
Kragujevcu"));
        mMap.moveCamera(CameraUpdateFactory.newLatLng(kg)); //postavljanje markera
na zadatu lokaciju
        mMap.getUiSettings().setZoomControlsEnabled(true);//zoom kontrola
```

Poglavlje 3

Geokodiranje

GEOCODER KLASA

Android operativni sistem realizuje koncept geokodiranja angažovanjem klase Geocoder.

Za realizaciju koncepta geokodiranja u Android aplikacijama, neohodno je uključiti u rad klasu *Geocoder* . Primenom ove klase omogućeno je pretvaranje obeležene geografske lokacije u adresu i obrnuto. *Geokodiranje* je često korišćena funkcionalnost u Android aplikacijama koje primenjuju Google mape u svojim aktivnostima. Upravo, u ovom delu lekcije će biti prikazan i objašnjen mehanizam *geokodiranja*.

Za demonstraciju primera primene *geokodiranja* u Android aplikacijama, koji koriste *Google* mape, aktuelni primer će nastaviti sa proširivanjem. Biće uvedena nova funkcionalnost, svaki postavljeni marker vratiće na ekranu adresu lokacije u kojoj je postavljen. Aplikacija će koristiti metodu getFromLocation() za konverziju geografske širine *(latitude)* i dužine *(longitude)* markera u adresu.

Klasa Geokoderje članica paketa *Location* i za njenu primenu, u narednom primeru, biće neophodno koristiti sledeće linije koda:

```
import android.location.Address;

import android.location.Geocoder;
```

Takođe, angažovanje geokodera zahteva omogućavanje precizne lokacije kroz posebnu dozvolu u AndroidManifest.xml datoteci (sledeća slika). Upravo je ovo mesto gde je moguće predstaviti sve dozvole koje je moguće sresti prilikom rada sa Google mapama u okviru neke Android aplikacije.

Slika 3.1 Precizna lokacija

AndroidManifest.xml datoteka sa navedenim dozvolama, prikazana je sledećim listingom.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.metropolitan.mape"
   android:versionCode="1"
   android:versionName="1.0" >
   <uses-sdk
      android:minSdkVersion="12"
      android:targetSdkVersion="24" />
```

```
   <uses-permission android:name="com.metropolitan.mape.permission.MAPS_RECEIVE" />
   <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
   <uses-permission android:name="android.permission.INTERNET" />
   <uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
   <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
   <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
   <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

   <uses-feature
      android:glEsVersion="0x00020000"
      android:required="true" />

   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="style/Theme.AppCompat.Light" >
      <uses-library android:required="true" android:name="com.google.android.gms"/>
      <activity
         android:name="com.example.vladimirmilicevic.mape.MapsActivity"
         android:label="@string/app_name" >

         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
       </activity>

      <meta-data
         android:name="com.google.android.maps.v2.API_KEY"
         android:value="AIzaSyAkScBC5onWg72tO9e4UZv4fXPIGLPkFT8" />
   </application>
</manifest>
```

PRIMER 6 - REAGOVANJE MAPE NA DOGAĐAJ – ONMAPCLICK()

Definisanjem metode onMapClick() biće realizovano prevođenje lokacije u adresu.

Sledeći zadatak, prilikom realizovanja geokodiranja, jeste kreiranje metode koja će definisati njegovu logiku. Kreirana metoda reaguje na događaj klik na mapu, kojim se postavlja pokazivač u određenu lokaciju i, potom, kao rezultat obrade na ekranu mobilnog uređaja prikazuje adresu lokacije u kojoj je marker postavljen.

Kreirana metoda ima potpis:

public void onMapClick(LatLng position), a interfejs koji se zadužen za osluškivanje klika na mapu naziva se onMapClickListener i implementiran je objektom klase GoogleMap.

Kao što je već napomenuto, gradi se aplikacija koja učitava i upravlja Google mapom u koju se dodaju nove funkcionalnosti. Tako, će metoda onMapClick() biti proširena geokodiranjem kojem odgovara programski kod koji se nastavlja odmah ispod dodatog dugmeta za određivanje vlastite pozicije na Google mapi.

Upravo, u narednom izlaganju, biće prikazan prošireni i dopunjen kod ove metode. Prilaže se sledeći listing.

```
@Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;
        mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);/* ovde će se podešavati tip
mape*/
        // Marker postavljen u Kragujevac i pomerena kamera
        LatLng kg = new LatLng(44.017, 20.917);
        mMap.addMarker(new MarkerOptions().position(kg).title("Marker u
Kragujevcu"));
        mMap.moveCamera(CameraUpdateFactory.newLatLng(kg)); //postavljanje markera
na zadatu lokaciju
        mMap.getUiSettings().setZoomControlsEnabled(true);//zoom kontrola
        mMap.getUiSettings().setCompassEnabled(true);//postavljanje kompasa
        mMap.setMyLocationEnabled(true);//vaša lokacija
        mMap.getUiSettings().setMyLocationButtonEnabled(true);//promenom u false,
gasi se dugme
        //Geokodiranje - prikazivanje adrese iz koordinata
        Geocoder gk = new Geocoder(getBaseContext(), Locale.getDefault());
        mMap.setOnMapClickListener(new GoogleMap.OnMapClickListener() {

            @Override
            public void onMapClick(LatLng position) {
                mMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
                mMap.addMarker(new MarkerOptions().position(position));
                Geocoder gk = new Geocoder(getBaseContext(), Locale.getDefault());
                try {

                    List<Address> adr;
                    adr = gk.getFromLocation(position.latitude, position.longitude,
1);
                    String ad = "";
                    if (adr.size() > 0) {
                        for (int i = 0; i < adr.get(0).getMaxAddressLineIndex();
i++)
                            ad += adr.get(0).getAddressLine(i) + "\n";
                    }
                    Toast.makeText(getBaseContext(),ad,Toast.LENGTH_SHORT).show();

                }
                catch (IOException e){
                    e.printStackTrace();
                }

            }
        });
```

PREVOĐENJE LOKACIJE MARKERA U ADRESU

Geografsku lokaciju preuzima kreirani objekat klase Geocoder.

U nastavku rada neophodno je definisati objekat klase Geocoder čiji će zadatak biti da preuzme lokaciju postavljenog markera izraženu u gografskoj širini i dužini. Angažovanje geokodera zahteva upošljavanje klase Locale iz paketa : Java.util.Locale, čija metoda getDefault() vraća tekuću lokalnu vrednost za instancu JAVA virtuelne mašine. Localevrednosti se razlikuju u zavisnosti od geografskih regiona na kojima se primenjuju.

Uporedno sa kreiranjem objekta, tipa Geokoder, kreirani objekat tipa GoogleMaps dobija novi zadatak – da osluškuje javljanje događaja *klik na mapu*.

Sledećom listingom izdvojen je kod za kreiranje Geocoder objekta i dodelu uloge osluškivaća GoogleMaps objektu.

```
Geocoder gk = new Geocoder(getBaseContext(), Locale.getDefault());
        mMap.setOnMapClickListener(new GoogleMap.OnMapClickListener() {
....
 }
```

Pre demonstriranja kreirane aplikacije, sledećim video materijalom pogledajte naprednije korišćenje markera u svrhu geokodiranja.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

GEOKODIRANJE – FUNKCIONISANJE.

Prevođenje lokacije u adresu realizovano je JAVA blokom try - catch

Program će prvo proveravati da li ima dozvolu za pristup preciznoj lokaciji. Nakon toga čeka će na reakciju korisnika na učitanu mapu. Kreiraće Geokoder objekat, a GoogleMap objekat implementira mehanizam za *oslušivanje klika na mapu*.Kao telo bloka naredbi, koji postavlja osluškivač događaja na mapi, implementirana je metoda onMapClick(). Metoda sadrži try – catchblok koji presreće izuzetke i, ukoliko ih nema izvršava geokodiranje.

Kreiranoj instanci adr tipa *List<Address>*, kreirani *Geocoder* objekat predaje lokaciju markera iskazanu kao par (latitude, longitude). Nakon toga, instanca *List<Address>* upisuje adresu u string koji se prikazuje na ekranu uređaja (sledeći listing).

```
List<Address> adr;
                    adr = gk.getFromLocation(position.latitude, position.longitude,
1);
                    String ad = "";
                    if (adr.size() > 0) {
                        for (int i = 0; i < adr.get(0).getMaxAddressLineIndex();
i++)
```

```
                        ad += adr.get(0).getAddressLine(i) + "\n";
                    }
                    Toast.makeText(getBaseContext(),ad,Toast.LENGTH_SHORT).show();
```

Ovako redefinisanu i modifikovanu aplikacije, neophodno je ponovo prevesti i pokrenuti da bi sve njene funkcionalnosti bile dostupne krajnjem korisniku. Neophodno je u razvojnom okruženju, ovde je u potpunosti akcenat na Android Studio IDE razvojnom okruženju, izabrati opciju Run App ili Shift + F10, debagovati aplikaciju i pokrenuti je u izabranom emulatoru ili realnom mobilnom uređaju povezanom na računar.

Rezultat pokretanja aplikacije je sledeći - na mapi je postavljen novi marker čija se adresa pojavljuje na ekranu (videti sliku), a to je u konkretnom slučaju adresa lokacije na kojoj se nalazi zgrada Metropolitan Univerziteta u Nišu.

Slika 3.2 Geokodiranje

ZADATAK 3 - OBJASNITE DOZVOLE ZA RAD SA GOOGLE MAPAMA

Pokušajte sami!!!

Pogledajte pažljivo priloženi listing i dajte njegovo objašnjenje!!!

```
<uses-permission android:name="com.metropolitan.mape.permission.MAPS_RECEIVE" />
   <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
   <uses-permission android:name="android.permission.INTERNET" />
   <uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
   <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
   <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
   <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Poglavlje 4

Promena tipa mape

TIPOVI MAPA

Tip mape upravlja celokupnom reprezentacijom mape.

Svako ko je koristio do sada Google mape mogao je da primeti da mape mogu biti prikazivane u različitim oblicima. Ti oblici, upravo, predstavljaju različite tipove Google mapa o čemu će detaljno biti govora u narednom izlaganju.

Android operativni sistem podržava različite tipove mapa koje obezbeđuje kroz Google Maps Android API. Tip mape upravlja celokupnom reprezentacijom mape. Na primer, geografski atlas može da sadrži *političke mape* čiji je zadatak da se fokusiraju na prikazivanje državnih granica ili *putne mape* u koje su ucrtane saobraćajnice grada ili regiona kojeg mapa prikazuje. U zavisnosti od namene Android aplikacije, ili preferencije programera i/ili korisnika, uređaj će, a na osnovu pokrenutog programa, prikazivati različite tipove mapa.

Google Maps Android APInudi mogućnost rada sa četiri različita tipa mapa, ali i opciju da mapa ne bude prikazana:

- ***Normal*** – Klasična putna mapa koja prikazuje saobraćajnice, izvesne građevinske i prirodne objekte (na primer reke). Obeležavanje puteva i objekata je dostupno na ovom tipu mape;
- ***Satellite*** – Mapa koja predstavlja satelitski snimak. Putevi i objekti nisu dostupni na ovom tipu mape;
- ***Hybrid*** – Predstavlja kombinaciju prethodna dva tipa. Hibridna mapa predstavlja satelitski snimak sa ucrtanim putevima i sa dostupnim obeležavanjima za puteve i objekte;
- ***Terrain*** – Klasična topografska karta. Mapa sadrži boje, konturne linije, obeležavanja i osenčen izgled;
- ***None*** – U fragment će biti učitana prazna mapa.

Posebno je značajno obraditi sve tipove mapa, nakon čega će studenti u vlastitim aplikacijama moći po potrebi da menjaju tip mape. Na primer, nakon klika na dugme za zumiranje mapa menja tip iz Normal u Hybrid i prikazuje više detalja korisniku koji je zumirao određenu teritoriju na Google mapi. Posebno, važno je napomenuti da se definisanje tipa Google mape može obaviti na dva načina: statički (primenom XML koda u GUI datoteci) i dinamički (u okviru JAVA klase aktivnosti).

Sledećim video materijalom prikazano je naprednije upravljanje tipom Google mape u kreiranoj Android aplikaciji.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

TIP MAPE NORMAL

Tip mape može biti određen u main.xml ili klasi aktivnosti uvođenjem konstante MAP_TYPE_NORMAL.

Prvi tip *Google* mapa koji će biti obrađen je ***Normal***. Ovim tipom mapa određena je klasična putna mapa koja prikazuje saobraćajnice, izvesne građevinske i prirodne objekte, kao što su reke, jezera i slično. Obeležavanje puteva i objekata je dostupno na ovom tipu mape. Ovaj tip mape može biti uključen na dva načina:

- Datotekom main.xml (activity_main.xml), instrukcijom ***map:mapType="normal“*** *u* u okviru elementa *<fragment>*;
- Dinamički u JAVA klasi aktivnosti aplikacije, primenom instrukcije ***map.setMapType(GoogleMap.MAP_TYPE_NORMAL);***

Sledećim kodom koji je izolovan iz koda aktuelnog primera moguće je primetiti da tip učitane Google mape se podešava tip ***Normal***.

```
@Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;

        mMap = googleMap;
        mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);/* ovde će se podešavati tip
mape*/
....
}
```

Da bi navedena modifikacija aplikacije bila dostupna i da bi aplikacija mogla da prikazuje mapu definisanu tipom *Normal,* neophodno je izvršiti prevođenje i pokretanje aplikacije iz Android Studio IDE okruženja.

VAŽNO: Posebno je bitno napomenuti da emulatori imaju problema prilikom pokretanja aplikacije sa Google mapom - često nemogućnost učitavanja mape. Zbog toga se preporučuje testiranje krirane aplikacije na realnom mobilnom uređaju.

Sledećom slikom prikazana ja učitana *Google* mapa tipa *Normal* u aktivnost kreirane aplikacije.

Slika 4.1 Google mapa tipa "Normal"

TIP MAPE SATTELITE

Mapa tipa satelite predstavlja satelitski snimak bez istaknutih objekata i labela.

Sledeći tip *Google* mapa koji će biti obrađen je ***Sattelite***. Ovim tipom mapa određena je mapa koja predstavlja satelitski snimak. Posebno je važno napomenuti da putevi i objekti nisu dostupni na ovom tipu mape. Kao i prethodni tip, u kreiranoj Android aplikaciji ovaj tip mape može biti uključen na dva načina:

- Datotekom main.xml (activity_main.xml), instrukcijom ***map:mapType=„sattelite“*** *u* u okviru elementa <*fragment*>;
- Dinamički u JAVA klasi aktivnosti aplikacije, primenom instrukcije ***map.setMapType(GoogleMap.MAP_TYPE_SATTELITE)*** *;*

Sledećim listingom je izolovan deo koda aktuelnog primera u kojem se podešava tip mape *Sattelite*.

```
@Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;

        mMap = googleMap;
        mMap.setMapType(GoogleMap.MAP_TYPE_SATTELITE);/* ovde će se podešavati tip
mape*/
....
}
```

Da bi navedena modifikacija aplikacije bila dostupna i da bi aplikacija mogla da prikazuje mapu definisanu tipom *Sattelite*, neophodno je izvršiti prevođenje i pokretanje aplikacije iz Android Studio IDE okruženja.

Sledećom slikom data je mapa tipa ***Sa ttelite*** *koja* je učitana u aktivnost kreirane Android aplikacije za rad sa Google mapama.

Slika 4.2 Tip Google mape "Sattelite"

TIP MAPE HYBRID

Hibridna mapa predstavlja kombinaciju tipova Normal i Sattelite.

U sledećem izlaganju pažnja će biti usmerena ka tipu *Google* mapa pod nazivom ***Hybrid***. Ovaj tip mapa predstavlja kombinaciju prethodna dva tipa. Hibridna mapa predstavlja satelitski snimak sa ucrtanim putevima i sa dostupnim obeležavanjima za puteve i objekte. Ovaj tip mape može biti uključen na dva načina, baš kao što je to bio slučaj i sa prethodnim tipovima Google mapa:

- Datotekom main.xml (activity_main.xml), instrukcijom *map:mapType=„hybrid“ u u okviru elementa <fragment>;*
- Dinamički u JAVA klasi aktivnosti aplikacije, primenom instrukcije ***map.setMapType(GoogleMap.MAP_TYPE_HYBRID);***

Sledećim listingom je izolovan deo koda aktuelnog primera u kojem se podešava tip mape *Hybrid.*

```
@Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;

        mMap = googleMap;
        mMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);/* ovde će se podešavati tip
mape*/
....
}
```

Da bi navedena modifikacija aplikacije bila dostupna i da bi aplikacija mogla da prikazuje mapu definisanu tipom ***Hybrid***, neophodno je izvršiti prevođenje i pokretanje aplikacije iz Android Studio IDE okruženja.

Sledećom slikom data je mapa tipa ***Hybrid*** *koja je učitana u aktivnost kreirane Android aplikacije za rad sa Google mapama.*

Slika 4.3 Google mapa tipa "Hybrid"

TIP MAPE TERRAIN

Tip Terrain je topografska slika regiona kojeg prikazuje.

Poslednji tip *Google* mapa koji će biti obrađen je ***Terrain***. Ovim tipom mapa određena je klasična topografska karta regiona kojeg prikazuje. Drugim rečima, mapa prikazuje detaljan reljef regiona na koji je fokusirana. Ovaj tip mape može biti uključen na dva načina, kao što je bio slulaj sa svim ostalim prethodnim tipovima mapa:

- Datotekom main.xml (activity_main.xml), instrukcijom ***map:mapType=„terrain“*** *u* u okviru elementa <*fragment*>;
- Dinamički u JAVA klasi aktivnosti aplikacije, primenom instrukcije *map.setMapType(GoogleMap.MAP_TYPE_TERRAIN);*

Sledećim listingom je izolovan deo koda aktuelnog primera u kojem se podešava tip mape ***Terrain.***

```
@Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;

        mMap = googleMap;
        mMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);/* ovde će se podešavati tip
```

```
mape*/
....
}
```

Da bi navedena modifikacija aplikacije bila dostupna i da bi aplikacija mogla da prikazuje mapu definisanu tipom ***Terrain***, neophodno je izvršiti prevođenje i pokretanje aplikacije iz Android Studio IDE okruženja.

Sledećom slikom data je mapa tipa ***Terrain*** *koja je učitana u aktivnost kreirane Android aplikacije za rad sa Google mapama.*

Slika 4.4 Google mapa tipa "Terrain"

PRIMER 7 - APLIKACIJA SA GOOGLE MAPOM I LOKACIJSKIM SERVISIMA

Kreiranje odgovarajuće aplikacije na časovima vežbi.

Zadatak:

1. Koristeći Android Studio IDE razviti Android aplikaciju za rad sa Google mapama primenom Google API za Android 6.0 verziju operativnog sistema;

2. Koristeći Google konzolu nabaviti Google API ključ za aplikaciju;

3. Omogućiti korišćenje različitih tipova mapa;

4. Koristiti markere i geokodiranje;

5. Koristiti Android Studio šablon za kreiranje Google Maps aktivnosti.

5. Dodati sve potrebne dozvole, API ključ i ostalu podršku u AndroidManifest.xml datoteku.

Budući da su prihvaćena osnovna podešavanja, kod GUI datoteke i datoteke fragmenta Google mape ovde neće biti prikazani.

AndroidManifest.xml datoteka, sa odgovarajućim dozvolama i podrškom radu sa Google mapama priložena je sledećim listingom:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.metropolitan.gmapedemo">

    <!--
         The ACCESS_COARSE/FINE_LOCATION permissions are not required to use
         Google Maps Android API v2, but you must specify either coarse or fine
         location permissions for the 'MyLocation' functionality.
    -->
    <uses-permission android:name="com.metropolitan.mape.permission.MAPS_RECEIVE" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission
```

```
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <!--
             The API key for Google Maps-based APIs is defined as a string resource.
             (See the file "res/values/google_maps_api.xml").
             Note that the API key is linked to the encryption key used to sign the
APK.
             You need a different API key for each encryption key, including the
release key that is used to
             sign the APK for publishing.
             You can define the keys for the debug and release targets in src/
debug/ and src/release/.
        -->
        <meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="AIzaSyApLtSTzkZuam2Qg7Mhl6lem9YkKawpFZk" />

        <activity
            android:name=".MapsActivity"
            android:label="@string/title_activity_maps">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Navedeni zahtevi aplikacije su realizovani sledećom klasom aktivnosti.

```
package com.metropolitan.gmapedemo;

import android.location.Address;
import android.location.Geocoder;
import android.net.Uri;
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;
import android.widget.Toast;
import com.google.android.gms.appindexing.AppIndex;
import com.google.android.gms.common.api.GoogleApiClient;
import com.google.android.gms.maps.CameraUpdateFactory;
```

```
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;
import com.google.android.gms.appindexing.Action;
import java.io.IOException;
import java.util.List;
import java.util.Locale;

public class MapsActivity extends FragmentActivity implements OnMapReadyCallback {

    private GoogleMap mMap;
    /**
     * ATTENTION: This was auto-generated to implement the App Indexing API.
     * See https://g.co/AppIndexing/AndroidStudio for more information.
     */
    private GoogleApiClient client;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
        // Obtain the SupportMapFragment and get notified when the map is ready to
be used.
        SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager()
                .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);

        // ATTENTION: This was auto-generated to implement the App Indexing API.
        // See https://g.co/AppIndexing/AndroidStudio for more information.
        client = new GoogleApiClient.Builder(this).addApi(AppIndex.API).build();
    }



    @Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;
        mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);/* ovde će se podešavati tip
mape*/
        // Marker postavljen u Kragujevac i pomerena kamera
        LatLng kg = new LatLng(44.017, 20.917);
        mMap.addMarker(new MarkerOptions().position(kg).title("Marker u
Kragujevcu"));
        mMap.moveCamera(CameraUpdateFactory.newLatLng(kg)); //postavljanje markera
na zadatu lokaciju
        mMap.getUiSettings().setZoomControlsEnabled(true);//zoom kontrola
        mMap.getUiSettings().setCompassEnabled(true);//postavljanje kompasa
        mMap.setMyLocationEnabled(true);//vaša lokacija
        mMap.getUiSettings().setMyLocationButtonEnabled(true);//promenom u false,
gasi se dugme
```

```
        //Geokodiranje - prikazivanje adrese iz koordinata
        Geocoder gk = new Geocoder(getBaseContext(), Locale.getDefault());
        mMap.setOnMapClickListener(new GoogleMap.OnMapClickListener() {

            @Override
            public void onMapClick(LatLng position) {
                mMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
                mMap.addMarker(new MarkerOptions().position(position));
                Geocoder gk = new Geocoder(getBaseContext(), Locale.getDefault());
                try {

                    List<Address> adr;
                    adr = gk.getFromLocation(position.latitude, position.longitude,
1);
                    String ad = "";
                    if (adr.size() > 0) {
                        for (int i = 0; i < adr.get(0).getMaxAddressLineIndex();
i++)
                            ad += adr.get(0).getAddressLine(i) + "\n";
                    }
                    Toast.makeText(getBaseContext(),ad,Toast.LENGTH_SHORT).show();

                }
                catch (IOException e){
                    e.printStackTrace();
                }

            }
        });
        //Adresa postavljenog markera
        try {

            List<Address> adr;
            adr = gk.getFromLocation(kg.latitude, kg.longitude, 1);
            String ad = "";
            if (adr.size() > 0) {
                for (int i = 0; i < adr.get(0).getMaxAddressLineIndex(); i++)
                    ad += adr.get(0).getAddressLine(i) + "\n";
            }
            Toast.makeText(getBaseContext(),ad,Toast.LENGTH_SHORT).show();

        }
        catch (IOException e){
            e.printStackTrace();
        }
    }

    @Override
    public void onStart() {
        super.onStart();

        // ATTENTION: This was auto-generated to implement the App Indexing API.
        // See https://g.co/AppIndexing/AndroidStudio for more information.
```

```
        client.connect();
        Action viewAction = Action.newAction(
                Action.TYPE_VIEW, // TODO: choose an action type.
                "Maps Page", // TODO: Define a title for the content shown.
                // TODO: If you have web page content that matches this app
activity's content,
                // make sure this auto-generated web page URL is correct.
                // Otherwise, set the URL to null.
                Uri.parse("http://host/path"),
                // TODO: Make sure this auto-generated app deep link URI is correct.
                Uri.parse("android-app://com.example.vladimirmilicevic.mape/http/
host/path")
        );
        AppIndex.AppIndexApi.start(client, viewAction);
    }

    @Override
    public void onStop() {
        super.onStop();

        // ATTENTION: This was auto-generated to implement the App Indexing API.
        // See https://g.co/AppIndexing/AndroidStudio for more information.
        Action viewAction = Action.newAction(
                Action.TYPE_VIEW, // TODO: choose an action type.
                "Maps Page", // TODO: Define a title for the content shown.
                // TODO: If you have web page content that matches this app
activity's content,
                // make sure this auto-generated web page URL is correct.
                // Otherwise, set the URL to null.
                Uri.parse("http://host/path"),
                // TODO: Make sure this auto-generated app deep link URI is correct.
                Uri.parse("android-app://com.example.vladimirmilicevic.mape/http/
host/path")
        );
        AppIndex.AppIndexApi.end(client, viewAction);
        client.disconnect();
    }

}
```

ZADATAK 4 - KREIRAJTE LISTU SA VAŠIM OMILJENIM MESTIMA ZA PRIKAZIVANJE NA MAPI

Pokušajte sami!!!

Kreirajte listu sa vašim omiljenim mestima. Klikom na stavku liste prikazuje se lokacija mesta Google mapi.

Poglavlje 5

Domaći zadatak 8

ZADACI SA SAMOSTALNI RAD

Vežbanje rada sa Google mapama u vlastitim Android aplikacijama.

Kreirati Android aplikaciju po sledećim instrukcijama:

1. Polazni korisnički interfejs sadrži dva polja za unos teksta i jedno dugme;
2. U polja se unose geografska širina [-90,0;90,0] i dužina [-180,0;180,0] ;
3. Klikom na dugme učitava se *Google mapa,* tipa *Normal*,sa postavljenim markerom u tački određenoj unetim vrednostima za geografsku širinu i dužinu;
4. Učitavanjem mape na ekranu se prikazuje naziv ulice (lokacije) u kojoj je marker postavljen;
5. U mapu ugraditi kontrole za zumiranje, prikazivanje vlastite lokacije i kompas;
6. Klikom na dugme za vlastitu lokaciju, tip mape se menja u *Hybrid*;
7. Klikom na mapu postavlja se marker u tački u kojoj je mapa dodirnuta;
8. Za svaki kreirani marker omogućiti prikazivanje podataka o lokaciji (geokodiranje).

Pregled Lekcije09

PREGLED LEKCIJE08

U lekciji su obrađeni osnovni koncepti rada Android aplikacija sa Google mapama.

Lekcija je stavila akcenat na osnovne koncepte rada sa Google mapama koje se učitavaju u aktivnost kreiranih korisničkih Android aplikacija. Savladavanjem navedenih koncepata, studenti će biti u stanju da ih kombinuju sa ciljem građenja složenijih Android programa koji upravljaju *Google* mapama, njihovim elementima i izabranim lokacijskim servisima.

Prvi, uvodni deo lekcije, govori o lokacijskim servisima i načinu kako mapa može biti učitana u Android aplikaciju koja se kreira. Posebno, istaknuto je da, ukoliko se želi da se koriste *Google* mape u Android programima, neophodno je izvršiti registrovanje na *Google API* konzoli sa ciljem dobijanja *Google API* ključa. Potom, uvodni deo lekcije diskutuje o tome kako je neophodno izvršiti određena podešavanja i dodeliti privilegije za rad sa *Google* mapama, u AndroidManifes.xml datoteci.

U nastavku lekcije govori se o kontrolama koje izvode određene operacije nad *Google* mapama, poput: zumiranja mape, pokazivanja strana sveta i trenutne lokacije korisnika, a potom su analizirane gestikulacije mape i događaji. Svaka od navedenih ulazno - izlaznih kontrola je detaljno obrađena sa akcentom na njihove predefinisane pozicije na ekranu u kojem je prikazana Google mapa. Posebno je, kako su kontrole postepeno uvođene u aplikaciju, za svaku od njih izvršeno testiranje aplikacije i demonstracija funkcionalnosti i primene datih kontrola.

Lekcija, potom, govori i o konceptu geokodiranja kao alatu koji omogućava pretvaranje geografske lokacije u adresu i obrnuto. Na veoma jednostavan i efikasan način, u aplikaciji je markerom birana željena lokacije, a potom je primenom geokodiranja vraćana adresa lokacije u kojoj je postavljen marker. Posebnom primenom klase Locale, podaci o lokaciji se vraćaju u formatu koji zavisi od lokalnih standarda lokacije na koju ukazuje marker.

Izlaganje se završava diskusijom i primerima o različitim tipovima *Google* mapa koje je moguće koristiti u Android aplikacijama. Svaki od tipova Google mapa je posebno i detaljno analiziran. Istaknuto je da se tipovi mapa u Android aplikacijama mogu podešavati na dva načina: statički - pomoću XML koda i dinamički - pomoću JAVA koda. Aplikacija je testirana i demonstrirana za različite tipove Google mapa učitanih u njenu aktivnost.

LITERATURA

Za pripremanje lekcije korišćena je sledeća literatura

1. Lee W. M. 2012. *Android 4 – razvoj aplikacija*, Wiley Publishing, INC

2. http://developer.android.com/training/index.html
3. http://www.tutorialspoint.com/android/
4. http://www.vogella.com/tutorials/android.html
5. https://developers.google.com/maps/documentation/android-api/start
6. https://developers.google.com/maps/
7. Literatura određena linkovima priloženih video materijala