

Mobilna aplikacija za praćenje i analizu ribolovnih ulova i putovanja

Macut, Nino

Undergraduate thesis / Završni rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Bjelovar University of Applied Sciences / Veleučilište u Bjelovaru**

Permanent link / Trajna poveznica: <https://um.nsk.hr/um:nbn:hr:144:821258>

Rights / Prava: [In copyright](#)/[Zaštićeno autorskim pravom.](#)

Download date / Datum preuzimanja: **2024-12-21**



Repository / Repozitorij:

[Repository of Bjelovar University of Applied Sciences - Institutional Repository](#)



VELEUČILIŠTE U BJELOVARU
PREDDIPLOMSKI STRUČNI STUDIJ RAČUNARSTVO

**MOBILNA APLIKACIJA ZA PRAĆENJE I ANALIZU
RIBOLOVNIH ULOVA I PUTOVANJA**

Završni rad br. 03/RAČ/2021

Nino Macut

Bjelovar, listopad 2021.



Veleučilište u Bjelovaru
Trg E. Kvaternika 4, Bjelovar

1. DEFINIRANJE TEME ZAVRŠNOG RADA I POVJERENSTVA

Kandidat: **Macut Nino**

Datum: 16.07.2021.

Matični broj: 002001

JMBAG: 0314019371

Kolegij: **PROGRAMIRANJE MOBILNIH APLIKACIJA**

Naslov rada (tema): **Mobilna aplikacija za praćenje i analizu ribolovnih ulova i putovanja**

Područje: **Tehničke znanosti**

Polje: **Računarstvo**

Grana: **Programsko inženjerstvo**

Mentor: **Ante Javor, struč. spec. ing. comp.**

zvanje: **predavač**

Članovi Povjerenstva za ocjenjivanje i obranu završnog rada:

1. **Ivan Sekovanić, mag.ing.inf.et comm.techn., predsjednik**
2. **Ante Javor, struč.spec.ing.comp., mentor**
3. **Tomislav Adamović, mag.ing.el., član**

2. ZADATAK ZAVRŠNOG RADA BROJ: 03/RAČ/2021

U radu je potrebno stvoriti nativnu Android aplikaciju za praćenje i analizu ribolovnih ulova i putovanja.

Aplikacija će podržati akcije za unos, pregled te uređivanje ključnih informacija o ribiču, lokacijama, putovanju, mamcima, ulovima i slično. Aplikacija će također podržati vizualan pregled statistika putem grafova i detaljnih opisa. Korisnik će dobiti mogućnost asistiranja u odabiru lokacije i mamaca za određeni dan s obzirom na prognozu vremena te prošla iskustva korisnika u usporedbi s istim ili sličnim uvjetima.

Zadatak uručen: 16.07.2021.

Mentor: **Ante Javor, struč. spec. ing. comp.**



Sadržaj

1. UVOD	1
2. ANDROID I KORIŠTENE ANDROID BIBLIOTEKE	2
2.1 <i>Android</i>	2
2.2 <i>Android Studio</i>	3
2.3 <i>Google Maps API</i>	4
2.4 <i>Open Weather API</i>	4
3. IMPLEMENTACIJA BAZE PODATAKA	5
3.1 <i>Room Baza Podataka</i>	5
3.2 <i>Sastav Room baze podataka</i>	6
3.2.1 Entity	7
3.2.2 Data Access Object	8
3.2.3 Database	9
3.3 <i>Obrada podataka u bazi</i>	10
3.3.1 Unos podatka	11
3.3.2 Pregled podataka	18
3.3.3 Uređivanje podataka	21
3.3.4 Brisanje podataka	24
4. VANJSKI SERVISI	25
4.1 <i>Implementacija Google Maps sučelja</i>	25
4.1.1 Korištenje Google Maps API za odabir lokacije	26
4.1.2 Korištenje Google Maps API-a za pregled svih lokacija	30
4.2 <i>Implementacija Open Weather sučelja</i>	32
5. STATISTIČKA OBRADA PODATAKA	35
5.1.1 Osnovne statistike	35
5.1.2 Detaljne statistike u obliku tortnog grafa	38
5.1.3 Detaljne statistike u obliku liste	41
5.1.4 Korištenje statistike za sugestiju ulova	42
6. ZAKLJUČAK	44
7. LITERATURA	46
8. OZNAKE I KRATICE	50
9. SAŽETAK	51
10. ABSTRACT	52

1. UVOD

U svijetu sportskog ribolova česta je potreba za evidencijom ribolovnih ulova, putovanja, pregledom statistika istih u svrhe natjecanja, stvaranja baze podataka o vrstama, obične usporedbe i slično. Iz tog razloga te zbog nedostatka ovakvog softvera na tržištu potrebno je popuniti tu prazninu Android aplikacijom. Svrha rada je izraditi android aplikaciju za evidenciju i dokumentaciju ribiča, ribolovnih lokacija, putovanja, vrsta riba, vrsta mamca, ulova, vremena i svih bitnih informacija o istima. Korisnik unosi navedene stavke te pomoću aplikacijsko programskih sučelja (engl. *Application programming interface*, skraćeno API) preko mrežnih servisa automatski dohvaća informacije o lokacijama, putovanjima te vremenu. Korisniku je dostupan unos, pregled, uređivanje te brisanje podataka, detaljan pregled statistika te mogućnost sugestije što loviti s obzirom na vrijeme te prethodne ulove.

Većina ribiča uz hvaljenje voli i jednostavnost, stoga je savršen odabir baš Android aplikacija zbog portabilnosti i lakoće korištenja modernih mobilnih telefona te dostupnosti aplikacije sa same vode što znači dostupnost svih informacija te mogućnost novog unosa u bilo kojem trenu tijekom ribolova. Isto tako odabirom Android aplikacije moguće je uz pomoć lokacije mobilnog uređaja neke podatke dohvatiti i automatski putem mrežnih servisa kao što su podaci o lokaciji te vremenu.

U radu će u drugom poglavlju biti opisan Android operativni sustav zajedno s Android Studio razvojnim okruženjem te mrežni servisi korišteni pri izradi aplikacije. U trećem poglavlju biti će opisana provedba projekta dakle: korištena baza podataka te obrada podataka u samoj bazi, detaljni opisi i korištenja mrežnih servisa te obrada podataka u statističke svrhe. U zadnjem poglavlju nalazi se zaključak.

2. ANDROID I KORIŠTENE ANDROID BIBLIOTEKE

Projekt se sastoji od nativne android aplikacije koja je izrađena u razvojnom okruženju Android Studio. Aplikacija je pisana u programskom jeziku Java te za svoj rad koristi Google Maps API, Open Weather API te Room bazu podataka.

2.1 Android

Prema izvoru [1] Android je široko prihvaćen operacijski sustav otvorenog koda. Tvrtka Google aktivno razvija Android platformu, ali dio koda besplatno nudi proizvođačima hardvera i telefonskim operaterima koji žele koristiti Android na svojim uređajima. Android je krenuo kao softver za video kamere [2]. Andy Rubin i njegov tim počeli su s razvojem Androida u 2003. godini. Ideja iza Androida bila je stvaranje operacijskog sustava kojeg bi koristili svi proizvođači video kamera u svojim proizvodima. Tijekom razvoja Rubin shvaća da su pametni telefoni budućnost te se tako fokus Androida prebacio s video kamera na pametne telefone. U 2005. godini Google kupuje Android za 50 miliona dolara te tim nastavlja razvoj Androida za pametne telefone.

Prema izvoru [3] Android operativni sustav sastoji se od pet odjeljaka:

- Linux jezgra - „srce“ operativnog sustava koje upravlja ulaznim i izlaznim upitima koji dolaze od strane softvera.
- Biblioteke – skup napisanog koda pomoću kojeg korisnici mogu optimizirati zadatke svojih softvera.
- Android izvršitelj (engl. *runtime*) - pruža ključnu komponentu zvanu Dalvik Virtual Machine koja je vrsta Java virtualne mašine. Dalvik Virtual Machine pokreće aplikacije na android uređajima te omogućuje svakoj Android aplikaciji pokretanje vlastitog procesa.
- Programski okvir (engl. *framework*) - pruža programerima usluge više razine operacijskog sustava za korištenje u svojim aplikacijama.
- Aplikacije – sve android aplikacije na uređaju.

Razlikuju se dvije vrste mobilnih aplikacija, nativne aplikacije i hibridne aplikacije. Nativne aplikacije namijenjene su za jedan specifični operativni sustav, primjerice Android. Hibridne aplikacije moguće je koristiti na više operativnih sustava, primjerice Android i iOS. Hibridne aplikacije imaju znatno lošije performanse od nativnih aplikacija jer nisu specijalizirane za samo jedan operativni sustav no stoga su cijene razvoja hibridnih aplikacija znatno niže od cijena razvoja nativnih aplikacije.

Prema izvoru [4] programski jezik Java je objektno orijentirani programski jezik temeljen na klasama. Java je računalna platforma za razvoj aplikacija. Prema tome Java je brz, siguran te pouzdan programski jezik koji se najčešće koristi za razvoj aplikacija koje se mogu pronaći u laptopima, data centrima, igračim konzolama te u ovom slučaju mobilnim telefonima.

2.2 Android Studio

Prema izvoru [5] Android Studio je službeno integrirano razvojno okruženje za razvoj Android aplikacija. Android Studio temeljen je na platformi IntelliJ IDEA tvrtke IntelliJ. Uz moćne uređivače koda i alate za razvoj, Android Studio nudi još više značajki koje povećavaju produktivnost u izradi kao što su:

- Fleksibilan sistem izgradnje aplikacije temeljen na alatu za izgradnju Gradle
- Brz emulator bogat značajkama
- Jedinствeno okružene za razvoj aplikacija za sve Android uređaje
- Predložci koda te integracija alata za verzioniranje koda Git
- Opsežni alati i okviri za testiranje
- Alati za provjeru performansi, upotrebljivosti te kompatibilnosti verzija
- Podrška za C++ i nativni set alata za razvoj (engl. *Native Development Kit*, skraćeno NDK)
- Ugrađena podrška za Google oblak

2.3 Google Maps API

Google Maps je mrežni servis koji pruža detaljne informacije o geografskim regijama i mjestima diljem svijeta. Uz cestovne karte, Google maps nudi i zračne te satelitske poglede na mnoga mjesta. Google Maps API omogućava korisniku pozivanje Google Maps servisa iz svojih aplikacija pomoću API ključa. Time korisnik može u svoju aplikaciju integrirati Google Maps karte koje zatim može koristiti kao prikaz nekog područja, prikaz GPS lokacija uz pomoć GPS-a mobilnog uređaja i mobilnih mreža i slično[6].

2.4 Open Weather API

Prema izvoru [7] Open Weather je mrežni servis koji na brz i elegantan način pruža informacije o vremenu s visokom preciznošću. Pomoću Open Weather aplikacijskog programskog sučelja moguće je preuzeti informacije o vremenu u nekom mjestu pomoću imena mjesta, identifikacijskog broja grada, geografskim koordinatama mjesta ili ZIP kodovima mjesta. Open Weather besplatno pruža podatke o vremenu s bilo koje lokacije uključujući 200,000 gradova u JSON, XML ili HTML formatima. U kombinaciji s Google Maps aplikacijskog programskog sučelja, Open Weather može se koristiti za brzu provjeru vremena na trenutnoj lokaciji korisnika aplikacije.

3. IMPLEMENTACIJA BAZE PODATAKA

Aplikacija služi za unos, pregled, uređivanje, brisanje te pohranu mnogih podataka bitnih korisniku (ribiču) te je dostupna na engleskom ili hrvatskom jeziku. Orijentacija aplikacije može biti portret ili pejzaž. Pomoću Google Maps aplikacijsko programskog sučelja moguće je unijeti novu lokaciju u bazu podataka ili pregledati sve do sada dodane lokacije na jednoj velikoj mapi uz mogućnost odabira opcije otvaranja Google Maps aplikacije u svrhe dobivanja uputa kako doći do navedene lokacije. Na isti način uz pomoć Open Weather aplikacijsko programskog sučelja moguće je dohvatiti, pregledati i tijekom unosa ulova automatski unijeti podatke o vremenu koje su bitne za ulov kao što su npr. temperatura, tlak zraka, jačina vjetrova, oblačnost i slično. Nakon dovoljno unesenih podataka moguće je usporediti trenutno vrijeme s vremenima unesenih ulova te odrediti koju vrstu ribe bi bilo optimalno loviti tijekom trenutnog vremena. Za pohranu navedenih podataka koristi se Room Database.

3.1 Room Baza Podataka

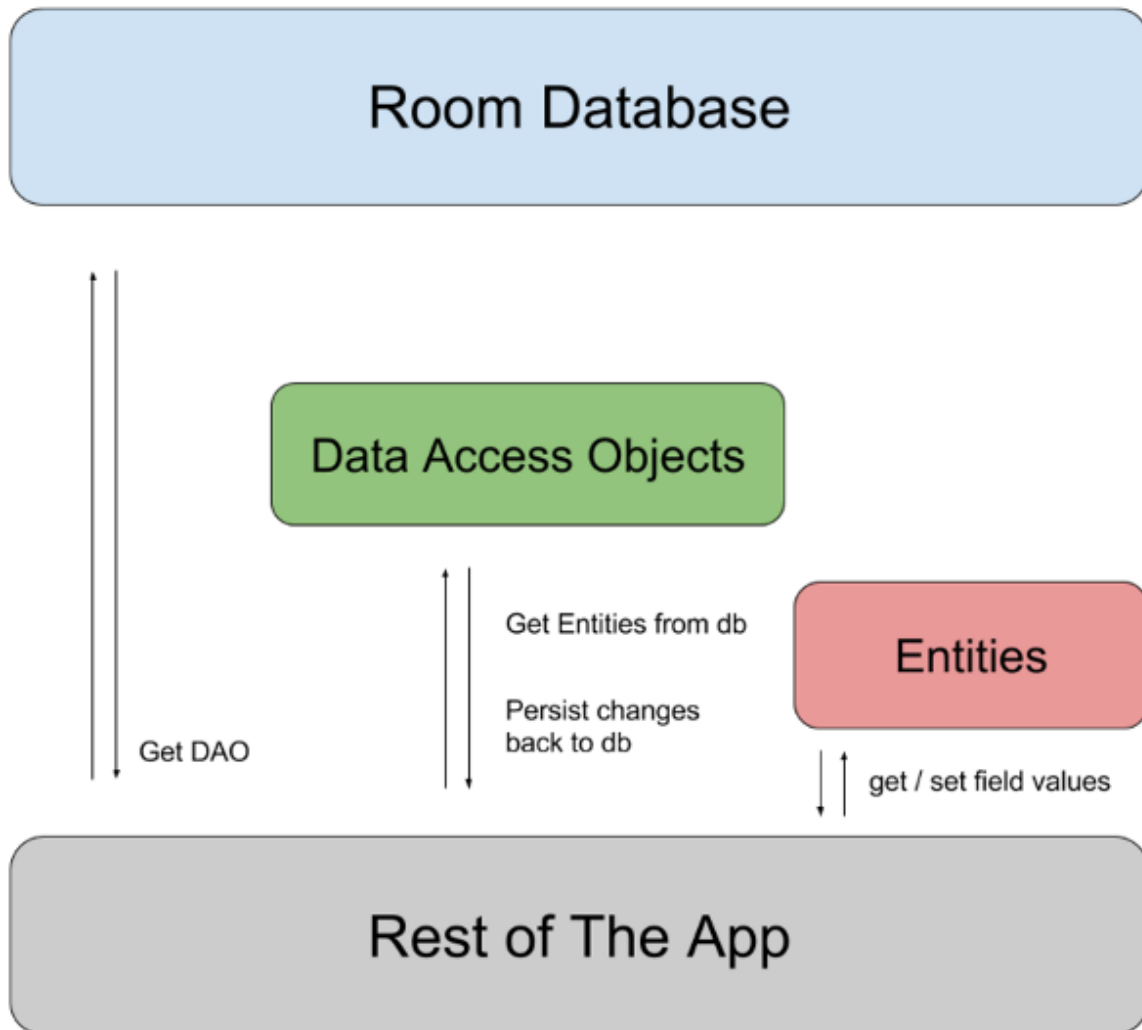
U svrhe pohrane podataka korištena je Room baza podataka. Prema izvoru [8] Room je biblioteka koja služi kao apstrakcijski sloj nad SQLite-om. Room čini korištenje SQLite-a pomoću implementacije anotacija. Room baza podataka sastoji se od sučelja za programiranje aplikacija i prevoditelja (engl. *compiler*).

Glavna prednost Room-a je pouzdanost. Ako dođe do greške prilikom komunikacije između aplikacije i SQLite baze podataka, Room će tu grešku javiti kod prevođenja koda umjesto tijekom izvođenja same aplikacije. Navedeno sprječava nastanak grešaka koje bi zaustavile ili blokirale rad aplikacije. Room je isto tako lagano integrirati s ostalim komponentama. Room je komponenta android arhitekture koja može podatke pretvoriti u oblik razumljiv drugim komponentama android arhitekture kao što su LiveData ili RxJava-in Single Object.

3.2 Sastav Room baze podataka

Tijekom stvaranja Room baze podataka potrebno je proći tri koraka. Prvi korak je stvaranje klase koja sadrži entitet (engl. *Entity*), ta klasa tada predstavlja jednu tablicu u Room bazi podataka. Drugi korak je stvaranje objekta pristupa podacima (engl. *Data Access Object*, skraćeno DAO) koji služi kao pretvornik tipova (engl. *type converter*) te bazi služi za rukovanje podacima. Treći korak je stvaranje klase koja predstavlja samu bazu podataka. Prema tome, Room baza podataka sastoji od tri glavne komponente (Slika 3.1):

- Entity
- DAO
- Database



Slika 3.1 Prikaz arhitekture Room baze podataka[8]

3.2.1 Entity

U kontekstu baza podataka, entitet može biti bilo koja jedna stvar, osoba, mjesto ili objekt. Kada je u pitanju administracija baze podataka, tada se u entitete ubrajaju samo objekti za koje je potrebno spremati podatke, u slučaju da za neki objekt nije potrebno spremati podatke o istom, tada nije potrebno stvarati entitet u samoj bazi podataka. Prema izvoru [9] kako bi se klasa proglasila entitetom, mora joj se dodati anotacija `@Entity`, time će se bazi podataka dati do znanja da se radi o entitetu. Svaki entitet mora imati barem jedno polje koje će predstavljati primarni ključ entiteta. Primarni ključ entiteta se označava anotacijom `@PrimaryKey`. U aplikaciji postoje sedam entiteta, klase: *Fisherman*, *Location*, *Trip*, *BaitCategory*, *Bait*, *Species* i *Catch*. Primjer entiteta *Location* vidljiv je u programskom kodu 3.1. Entitet *Location* sadrži jedinstveni identifikator, skraćeno ID, te podatke o *image*, *name*, *type*, *description*, *latitude* i *longitude* tipa *string*.

Programski kod 3.1: Location Entitet

```
@Entity(tableName = "locations")
public class Location {
    @PrimaryKey(autoGenerate = true)
    public int ID;
    @ColumnInfo(name = "path_to_image")
    public String image;
    @ColumnInfo(name = "name")
    public String name;
    @ColumnInfo(name = "type")
    public String type;
    @ColumnInfo(name = "description")
    public String description;
    @ColumnInfo(name = "latitude")
    public String latitude;
    @ColumnInfo(name = "longitude")
    public String longitude;

    public Location(String image, String name, String type, String
description, String latitude, String longitude){
        this.image = image;
        this.name = name;
        this.type = type;
        this.description = description;
        this.latitude = latitude;
        this.longitude = longitude;
    }
}
```

3.2.2 *Data Access Object*

Prema izvoru [10] DAO je naziv za klase pomoću kojih se definira interakcija s bazom podataka. Kako bi se klasa proglasila Data Access Objektom potrebno joj je dodati anotaciju `@Dao`. Unutar DAO klase moguće je definirati mnogo upita bazi podataka. Preporučeno je imati po jednu DAO klasu za svaku Entity klasu u aplikaciji. DAO klase moraju biti definirane kao sučelja ili kao apstraktne klase. U aplikaciji postoje sedam DAO klasa (jedna za svaki entitet). U programskom kodu 3.2 vidljiv je DAO *Location* i korištenje nekih DAO anotacija. Anotacija `@Query` koristi se kada se želi napisati upit bazi podataka. U navedenom slučaju koristi se za stvaranje upita koji će s baze vratiti podatke o svim lokacijama u obliku liste lokacija te se anotacija pridružuje funkciji `getAllLocations()`. Anotacija `@Insert` koristi se za definiranje funkcije za unos entiteta u bazu podataka, u ovom slučaju za unos nove lokacije. Funkcija kao ulazni parametar prima objekt *Location* te se naziva `insertLocation()`. Na isti način mogu se definirati funkcije za brisanje i uređivanje određenog entiteta u bazi podataka pomoću anotacija `@Delete` i `@Update`. U ovom slučaju navedene anotacije koriste se za definiranje funkcija `deleteLocation()` te `updateLocation()`.

Programski kod 3.2: Location Dao

```
@Dao
public interface LocationDao {
    @Query("SELECT * FROM locations")
    List<Location> getAllLocations();

    @Insert
    void insertLocation(Location... location);

    @Update
    void updateLocation(Location... location);

    @Delete
    void deleteLocation(Location location);
}
```

3.2.3 Database

Prema izvoru [11] pomoću anotacije `@Database` može se definirati klasa kao baza podataka. Takva klasa mora biti apstraktna i mora proširivati klasu `RoomDatabase`. U programskom kodu 3.3 prikazana je implementacija Room baze podataka u aplikaciji.

Programski kod 3.3: Room baza podataka

```
@Database(entities = {Fisherman.class, Location.class, Trip.class,
Species.class, BaitCategory.class, Bait.class, Catch.class}, version =
1)
public abstract class AppDatabase extends RoomDatabase {

    public abstract FishermanDao fishermanDao();
    public abstract LocationDao locationDao();
    public abstract TripDao tripDao();
    public abstract SpeciesDao speciesDao();
    public abstract BaitCategoryDao baitCategoryDao();
    public abstract BaitDao baitDao();
    public abstract CatchDao catchDao();

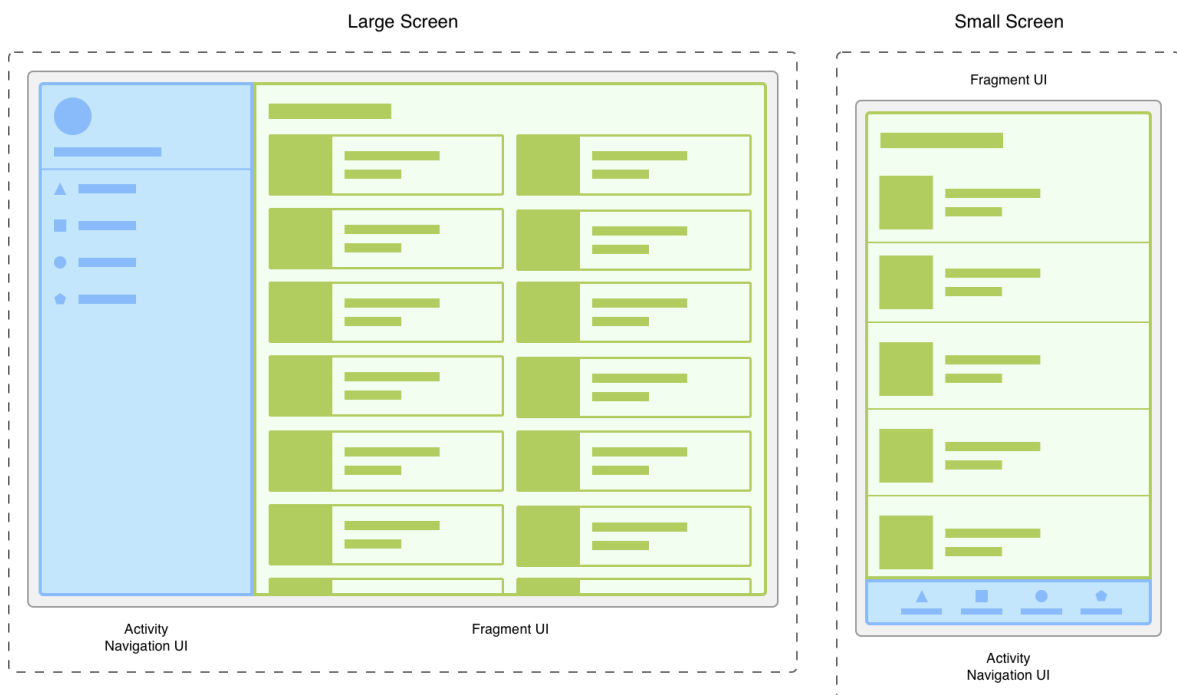
    private static AppDatabase INSTANCE;

    public static AppDatabase getDbInstance(Context context){
        if (INSTANCE == null){
            INSTANCE =
Room.databaseBuilder(context.getApplicationContext(), AppDatabase.class,
"DB_DB16").allowMainThreadQueries().build();
        }
        return INSTANCE;
    }
}
```

Prilikom dodavanja anotacije `@Database` potrebno je navesti sve entitete čiji podaci će se spremati u samu bazu te trenutna verzija baze. Unutar same klase potrebno je navesti te inicijalizirati sve DAO klase koje baza koristi za komunikaciju s entitetima. Za pristup instanci baze iz bilo koje druge klase ili fragmenta poziva se funkcija `getDbInstance()` koja će iz konteksta aplikacije pročitati prethodno inicijaliziranu instancu baze.

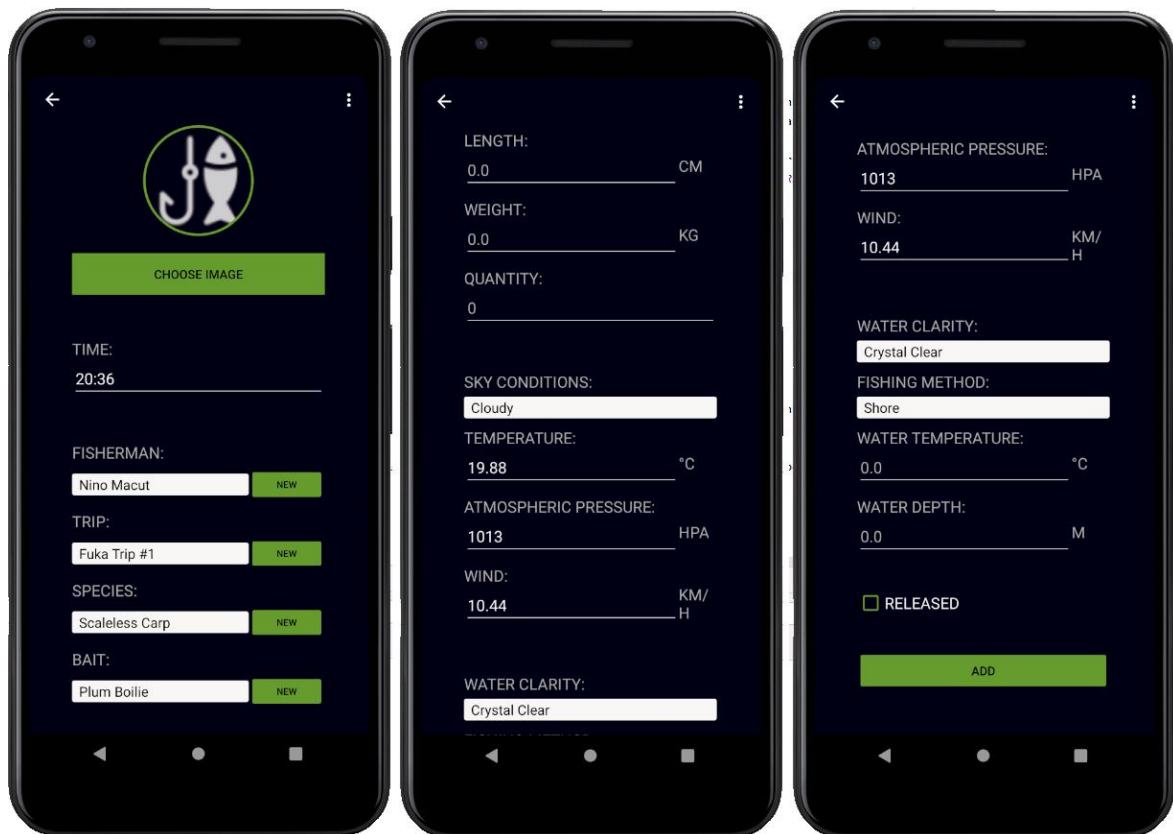
3.3 Obrada podataka u bazi

Korisnik ima pristup dodavanju, čitanju, uređivanju i brisanju svih podataka baze aplikacije. Unutar baze spremaju se podaci o entitetima *Fisherman*, *Location*, *Trip*, *BaitCategory*, *Bait*, *Species* i *Catch*. Sama aplikacija je aplikacija s jednom aktivnošću (engl. *Single Activity Application*) što znači da aplikacija sadrži samo jednu Activity klasu koja se brine o stvaranju i prikazu korisničkog sučelja. U svrhe unosa, čitanja, uređivanja i brisanja postoje odvojeni fragmenti za svaki entitet u bazi. Prema izvoru [12] fragmenti reprezentiraju dijelove korisničkog sučelja koji imaju svoj izgled, svoj odvojeni životni ciklus te sposobnost sami upravljati svojim ulaznim događajima. Fragment ne može postojati sam, nego ga mora ugostiti neka aktivnost ili drugi fragment. Primjeri aplikacija s jednom aktivnošću vidljivi su na slici 3.3.



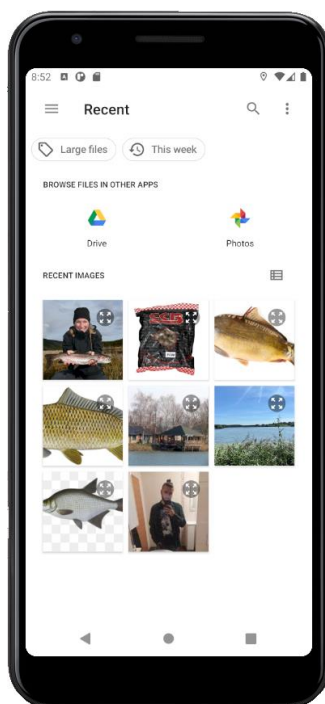
Slika 3.3 Primjeri aplikacija s jednom aktivnošću [12]

3.3.1 Unos podatka



Slika 3.4 Prikaz fragmenta za unos novog ulova.

Na slici 3.4 vidljiv je prikaz korisničkog sučelja fragmenta aplikacije za unos novog ulova. Entitet *Catch* posjeduje najveći broj podataka u usporedbi s ostalim entitetima te su iz tog razloga fragmenti u odnosu s entitetom *Catch* mnogo opsežniji od ostalih fragmenata aplikacije. Entitet *Catch* posjeduje podatke o putanji do slike ulova, vremenu ulova, ribiču, vrsti, putovanju te mamcu s kojim je ulov ulovljen, temperaturi, tlaku zraka, brzini vjetera, oblačnosti, dužini, težini te broju ulova, bistrini vode, dubini vode, temperaturi vode, metodi ulova te je li ulov zadržan ili pušten.



Slika 3.5 Prikaz korisničkog sučelja za odabir slike.

Pritiskom na gumb za dodavanje slike, otvara se izbornik za odabir slike s uređaja pomoću kojeg korisnik može dodati sliku ulovu. Na slici 3.5 vidljivo je korisničko sučelje koje ovisi o modelu korisnikovog mobitela. U programskom kodu 3.4 vidljiva je funkcija za otvaranje sučelja za odabir slike na klik gumba te funkcija za prikazivanje slike u fragmentu nakon samog odabira unutar sučelja.

Programski kod 3.4: Odabir slike u fragmentu

```

buttonUploadCatchImage.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent();
        intent.setType("image/*");
        intent.setAction(Intent.ACTION_GET_CONTENT);
        startActivityForResult(Intent.createChooser(intent, "Title"),
        SELECT_IMAGE_CODE);
    }
});

@Override
public void onActivityResult(int requestCode, int resultCode, @Nullable
Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if(requestCode == SELECT_IMAGE_CODE){
        Uri uri = data.getData();
        previewImage.setImageURI(uri);
    }
}

```

Kako bi se slika spremila u memoriju te njena putanja zapamtila u bazi podataka koriste se funkcije iz programskog koda 3.5

Programski kod 3.5: Spremanje slike u memoriju te bazu podataka

```
private String saveToInternalStorage(Bitmap bitmapImage, String
fileName){
    ContextWrapper cw = new ContextWrapper(getActivity());
    File directory = cw.getDir("Catches", Context.MODE_PRIVATE);
    File mypath=new File(directory,fileName);

    FileOutputStream fos = null;
    try {
        fos = new FileOutputStream(mypath);
        bitmapImage.compress(Bitmap.CompressFormat.PNG, 100, fos);
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        try {
            fos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    return directory.getAbsolutePath();
}

String fileName = "catch" + getRandomNumber(0,100000) + ".jpg";

if(hasPicture == true){
    saveToInternalStorage(bitmap, fileName);
}
else if (hasPicture == false){
    fileName = "nofile";
}
}
```

Sve slike dodane pomoću aplikacije spremaju se u odvojenu mapu aplikacije u memoriji uređaja te u podmape ovisno o entitetu, u ovom slučaju u podmapu *Catches*. Aplikaciji je potrebno dozvoliti pristup memoriji pomoću funkcije u kodu 3.6.

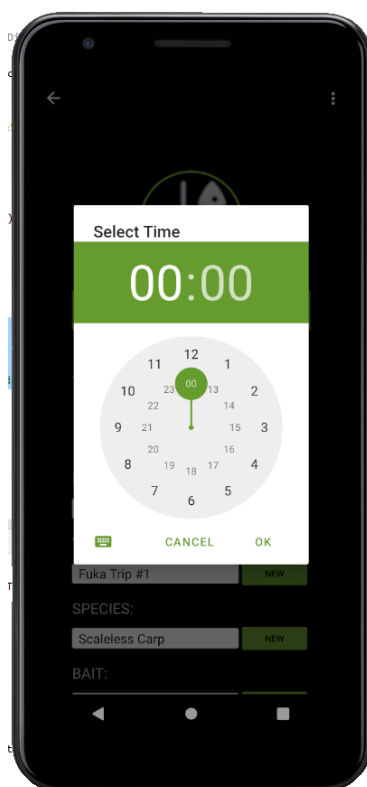
Programski kod 3.6: Funkcija za provjeru i dodjelu dozvola za pristup memoriji.

```
private void verifyPermissions(){
    int permissionRead = ActivityCompat.checkSelfPermission(getActivity(),
Manifest.permission.READ_EXTERNAL_STORAGE);
    int permissionWrite = ActivityCompat.checkSelfPermission(getActivity(),
Manifest.permission.WRITE_EXTERNAL_STORAGE);
    if ((permissionRead != PackageManager.PERMISSION_GRANTED) ||
(permissionWrite != PackageManager.PERMISSION_GRANTED)) {
        ActivityCompat.requestPermissions(getActivity(),STORAGE_PERMISSIONS,
REQUEST_CODE);
    }
}
```

Nakon odabira slike, sljedeći korak je odabir vremena ulova. Tijekom stvaranja fragmenta za dodaju ulova, vrijeme ulova postaviti će se na trenutno vrijeme korisnikovog uređaja prikazano programskim kodom 3.7, no to vrijeme moguće je promijeniti pomoću izbornika na slici 3.6.

Programski kod 3.7: Automatsko dodijeljivanje vremena ulova

```
public void getTime(View root){  
    EditText timeText = root.findViewById(R.id.text_catch_time_input);  
    Calendar calendar = Calendar.getInstance();  
    timeText.setText(String.format(Locale.getDefault(), "%02d:%02d",  
calendar.get(Calendar.HOUR_OF_DAY),calendar.get(Calendar.MINUTE)));  
}
```



Slika 3.6: Prikaz korisničkog sučelja za odabir vremena.

Izbornik na slici 3.6 zapravo je komponenta dijalog za izbor vremena (engl. *Time Picker Dialog*) čiji je izgled promijenjen pomoću teme prilagođene izgledu ostatka aplikacije. Nakon odabira vremena, vrijeme ulova prikazuje se u formatu MM:HH. Programski kod 3.8 prikazuje funkciju za prikaz i upravljanje *Time Picker Dialog* komponentom.

Programski kod 3.8: Prikaz i upravljanje Time Picker Dialog komponentom.

```
public void popTimePicker(View root){
    TimePickerDialog.OnTimeSetListener onTimeSetListener = new
    TimePickerDialog.OnTimeSetListener() {
        @Override
        public void onTimeSet(TimePicker view, int selectedHour, int
selectedMinute) {
            hour = selectedHour;
            minute = selectedMinute;
            time.setText(String.format(Locale.getDefault(),
"%02d:%02d", hour, minute));
        }
    };

    TimePickerDialog timePickerDialog = new TimePickerDialog(getActivity(),
R.style.DialogTheme, onTimeSetListener, hour, minute, true);

    timePickerDialog.setTitle(R.string.input_select_time);
    timePickerDialog.show();
}
```

Sljedeći korak kod dodavanja ulova je odabir ribiča, putovanja, vrste ribe te mamca koji se odnose na ulov. S obzirom da su navedene stavke zapravo drugi entiteti u bazi, odabir se vrši pomoću padajućeg izbornika. Padajući izbornici se ispunjuju podacima prilikom pokretanja fragmenta te prilikom otvaranja samog izbornika. Entiteti u bazi i na interaktivnim komponentama korisničkog sučelja se automatski sinkroniziraju. U programskom kodu 3.9 vidljiv je primjer punjenja padajućeg izbornika.

Programski kod 3.9: Ispuna padajućeg izbornika za ribiče

```
private void fillFishermenDropdown(View root){
    Spinner fishermenDropdown =
root.findViewById(R.id.dropdown_catch_fisherman);
    String[] items = new String[fishermenList.size()];

    int i = 0;
    for (Fisherman f: fishermenList
    ) {
        items[i] = f.firstName + " " + f.lastName;
        i++;
    }

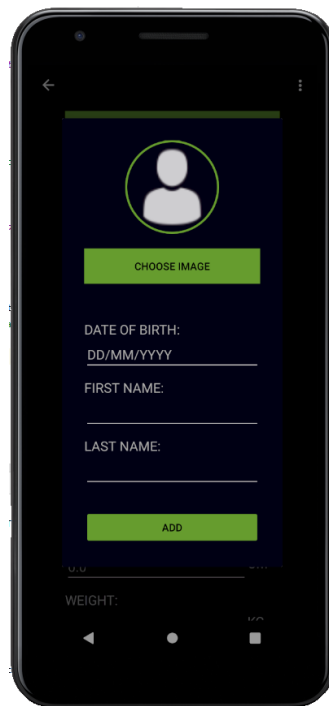
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(getActivity(),
R.layout.support_simple_spinner_dropdown_item, items);

    adapter.setDropDownViewResource(R.layout.support_simple_spinner_dropdown_item);
    fishermenDropdown.setAdapter(adapter);
}
```

Primjer otvaranja prozora za dodavanje novog entiteta izravno iz fragmenta vidljiv je u programskom kodu 3.10. Primjer izgleda prozora za dodavanje novog entiteta vidljiv je na slici 3.7. Nakon dodavanja novog entiteta prilikom sljedećeg otvaranja padajućeg izbornika ponovo će se izvršiti funkcija iz programskog koda 3.9

Programski kod 3.10: Otvaranje prozora za dodavanje novog entiteta

```
buttonAddNewFisherman =  
root.findViewById(R.id.button_add_new_fisherman_from_catches);  
  
buttonAddNewFisherman.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        AddFishermanDialogFragment df = new AddFishermanDialogFragment();  
        df.show(getChildFragmentManager(), "AddFishermanDialogFragment");  
    }  
});
```



Slika 3.7 Primjer prozora za dodavanje novog entiteta iz fragmenta.

Nakon odabira unutar padajućeg izbornika, potrebno je dohvatiti podatak o entitetu koji će se spremirati u bazu. Primjer pripreme podatka iz pojedinog padajućeg izbornika vidljiv je u programskom kodu 3.11.

Programski kod 3.11: Primjer pripreme podatka iz padajućeg izbornika.

```
int fishermanID;
Spinner fisherman = root.findViewById(R.id.dropdown_catch_fisherman);
if (fisherman.getSelectedItem() != null) {
    fishermanID = fishermenList.get(fisherman.getSelectedItemPosition()).ID;
} else {
    Toast.makeText(getActivity(),
getString(R.string.toast_no_selected_fisherman), Toast.LENGTH_SHORT).show();
    return;
}
```

Nakon odabira slike, vremena i entiteta koji se odnose na ulov, slijedi unos osnovnih podataka o ulovu, vremenu, vodi, metodi i je li ulov pušten. Neki od navedenih podataka popunjavaju se pomoću prethodno definiranih padajućih izbornika koji nemaju promijenjive podatke. Nakon konačnog unosa svih podataka, poziva se funkcija koja stvara novi entitet te ga dodaje u bazu podataka. Prilikom pritiska na gumb za dodavanje, poziva se funkcija iz programskog koda 3.12 koja prvo dohvaća instancu baze, a nakon toga stvara entitet te ga dodaje u navedenu instancu.

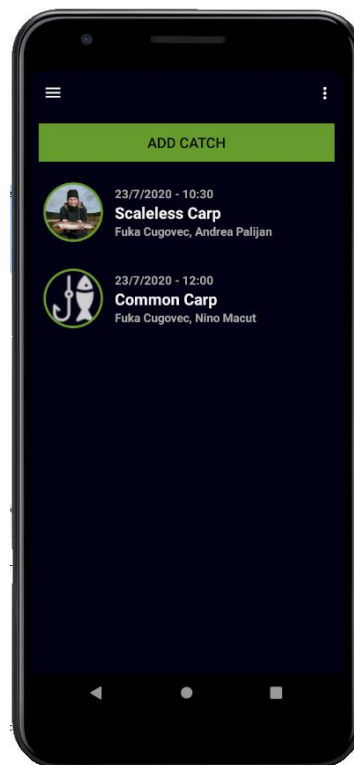
Programski kod 3.12: Stvaranje i dodavanje entiteta u bazu podataka

```
private void addNewCatch(String image, String time, int species, int
fisherman, int trip, int bait, float temperature, float atmosphericPressure,
float wind, String skyConditions, float length, float weight, int quantity,
String waterClarity, float waterDepth, float waterTemperature, String
fishingMethod, boolean released){
    AppDatabase db = AppDatabase.getInstance(this.getActivity());

    Catch catchy = new Catch(image, time, fisherman, species, trip, bait,
temperature, atmosphericPressure, wind, skyConditions, length, weight,
quantity, waterClarity, waterDepth, waterTemperature, fishingMethod,
released);
    db.catchDao().insertCatch(catchy);
}
```

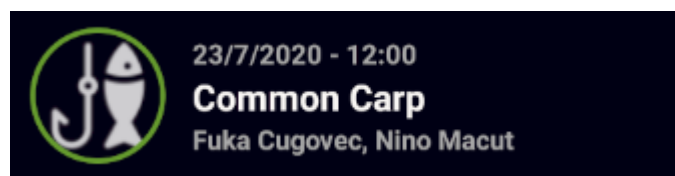
Nakon izvršavanja programskog koda 3.12, entitet *Catch* je uspješno dodan u bazu te spreman za daljnje prikazivanje, uređivanje, korištenje te brisanje.

3.3.2 Pregled podataka



Slika 3.8 Prikaz fragmenta za pregled podataka.

Prikaz podataka vrši se pomoću posebnih fragmenata za svaku vrstu entiteta u bazi. Za prikaz podataka potrebna nam je komponenta *ListView* koja služi za prikaz liste *ListItem* elemenata. Svaki entitet ima *ListItem* element koji je prilagođen podacima tog entiteta. Primjer *ListItem* elementa prilagođenog prikazu entiteta *Catch* vidljiv je na slici 3.9



Slika 3.9 Prikaz *ListItem* elementa za *Catch* entitet

Na *ListItem* elementu vidljiv je prikaz slike, datuma i vremena ulova, imena vrste ulova te mjesto ulova i ribič koji ga je ulovio. U slučaju da ulovu nije dodijeljena slika, prikazati će se prethodno zadana slika iz resursa aplikacije.

Programski kod 3.13: Adapter entiteta Catch

```
public class CatchesAdapter extends ArrayAdapter<Catch> {
    public CatchesAdapter(Context context, List<Catch> catchesList){
        super(context, R.layout.listitem_catch, catchesList);
    }

    @NonNull
    @Override
    public View getView(int position, @Nullable View convertView, @NonNull
    ViewGroup parent) {
        Catch catchy = getItem(position);
        if (convertView == null){
            convertView =
            LayoutInflater.from(getContext()).inflate(R.layout.listitem_catch,parent,false);
        }
        ImageView image =
        convertView.findViewById(R.id.image_catch_image_listitem);
        TextView species =
        convertView.findViewById(R.id.text_catch_species_listitem);
        TextView dateAndTime =
        convertView.findViewById(R.id.text_catch_date_listitem);
        TextView location =
        convertView.findViewById(R.id.text_catch_location_listitem);

        if (!catchy.image.equals("nofile")){
            Bitmap bitmap = loadImageFromStorage(catchy.image);
            image.setImageBitmap(bitmap);
        }
        else if (catchy.image.equals("nofile")){
            image.setImageResource(R.drawable.ic_catches_no_pic);
        }

        AppDatabase db = AppDatabase.getDbInstance(getActivity());
        List<Species> speciesList = db.speciesDao().getAllSpecies();
        List<Location> locationsList = db.locationDao().getAllLocations();
        List<Trip> tripsList = db.tripDao().getAllTrips();
        List<Fisherman> fishermenList = db.fishermanDao().getAllFishermen();

        for(Species s : speciesList){
            if (s.ID == catchy.species){
                species.setText(s.name);
            }
        }
        String locTemp = "";
        for(Trip t : tripsList){
            if (t.ID == catchy.trip){
                locTemp = locationsList.get(t.location).name;
                dateAndTime.setText(t.startDate + " - " + catchy.time);
            }
        }
        for(Fisherman f : fishermenList){
            if (f.ID == catchy.fisherman){
                locTemp += ", " + f.firstName + " " + f.lastName;
            }
        }
        location.setText(locTemp);
        return convertView;
    }
}
```

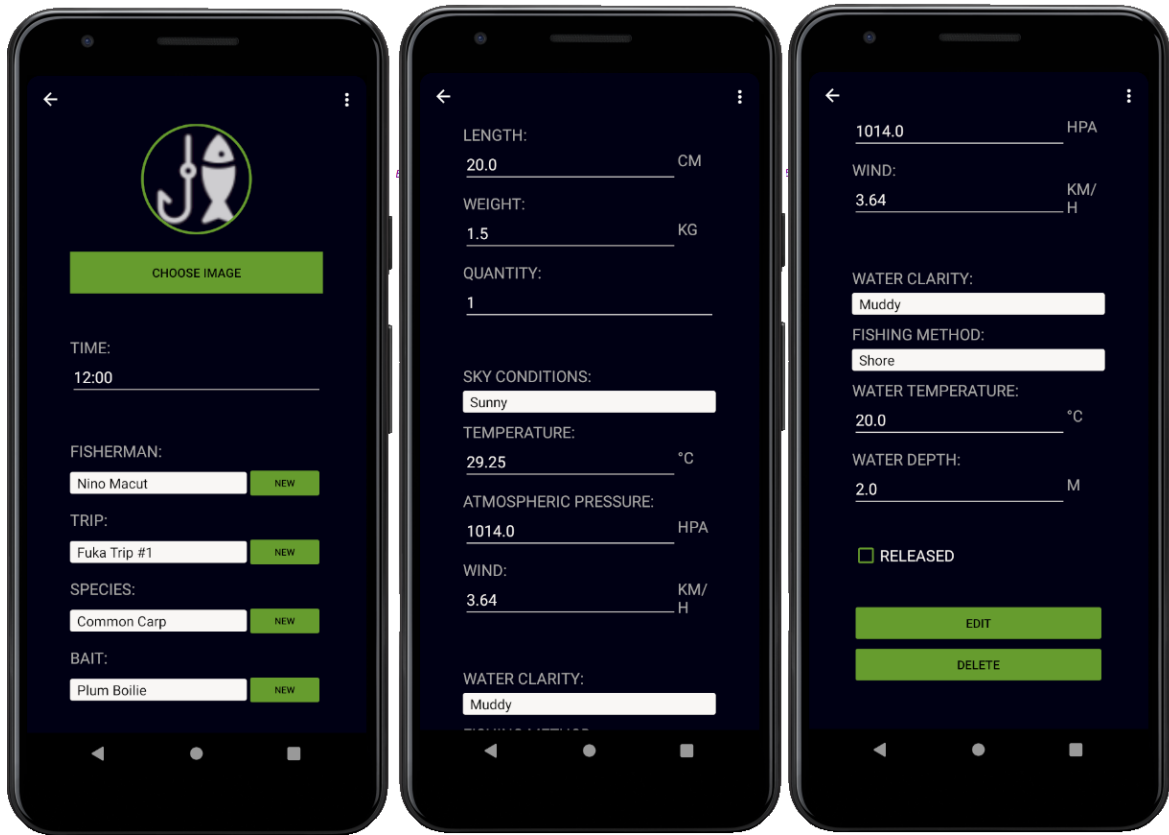

Kako bi izlistali prethodno navedene *ListItem* elemente, potrebno je stvoriti posebnu adapter klasu za navedenu vrstu *ListItem* elementa. Programski kod 3.13 prikazuje primjer adaptera za entitet *Catch*. Adapter klasa diktira komponenti *ListView* kakav *ListItem* element će koristiti za prikaz pojedinog entiteta te kako da ispuni navedeni *ListItem* s podacima o entitetu. Nakon što je stvoren adapter, potrebno je inicijalizirati listu i postaviti njen adapter na stvoreni adapter. Navedeno je moguće postići pomoću programskog koda 3.14. Unutar navedenog koda definirano je i što će se dogoditi ako korisnik pritisne neki element liste prilikom pregleda. U tom slučaju, prikazati će se fragment za puno uređivanje entiteta na čiji je *ListItem* korisnik pritisnuo pomoću identifikatora *editID*. *EditID* služi fragmentu za uređivanje kao indikator o kojem entitetu se radi.

Programski kod 3.14: Inicijalizacija liste

```
private void initializeList(View root){
    listOfCatches = root.findViewById(R.id.list_catches);
    listOfCatches.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int
position, long id) {
        Bundle bundle = new Bundle();
        bundle.putInt("editID", position);
        NavController navController =
Navigation.findNavController(getActivity(), R.id.nav_host_fragment);
        navController.navigate(R.id.nav_edit_catch, bundle);
    }
});
loadList();
}

private void loadList(){
    CatchesAdapter catchesAdapter = new CatchesAdapter(getActivity(),
catchesList);
    listOfCatches.setAdapter(catchesAdapter);
}
```

3.3.3 Uređivanje podataka



Slika 3.10 Prikaz fragmenta za uređivanje podataka.

Nakon odabira entiteta iz liste entiteta kao što je prikazano u prošlom poglavlju, korisniku će se prikazati sučelje za uređivanje podataka o odabranom entitetu. Kako bi se ulazne komponente ispunile podacima, potrebno je pročitati ulaznu informaciju o identifikacijskom broju entiteta iz ulaznih argumenata fragmenta te ispuniti sve komponente podacima o odabranom entitetu kao što je prikazano u programskom kodu 3.15. Paket podataka (engl. *Bundle*) služi za prijenos podataka između fragmenta ili aktivnosti, u ovom slučaju pomoću paketa fragmentu za uređivanje prenesena je informacija o kojem entitetu se radi.

Programski kod 3.15: Inicijalizacija podataka fragmenta za uređivanje

```
Bundle bundle = this.getArguments();
if (bundle != null) {
    editID = bundle.getInt("editID");
    Catch ctch = catchesList.get(editID);

    Bitmap bitmap = loadImageFromStorage(ctch.image);
    portrait.setImageBitmap(bitmap);

    Spinner fishermanSpinner = root.findViewById(R.id.dropdown_catch_fisherman);
    int f = 0;
    for (Fisherman fisherman : fishermenList){
        if (fisherman.ID == ctch.fisherman){
            break;
        }
        f++;
    }
    fishermanSpinner.setSelection(f);

    (...)

    Spinner fishingMethod = root.findViewById(R.id.dropdown_catch_fishing_method);
    fishingMethod.setSelection(((ArrayAdapter)fishingMethod.getAdapter())
        .getPosition(ctch.fishingMethod));

    (...)

    EditText temperature = root.findViewById(R.id.text_catch_temperature_input);
    if (ctch.temperature == 0){
        temperature.setText("");
    } else {
        temperature.setText(String.valueOf(ctch.temperature));
    }

    (...)

    CheckBox released = root.findViewById(R.id.checkbox_catch_released);
    released.setChecked(ctch.released);

    EditText time = root.findViewById(R.id.text_catch_time_input);
    time.setText(ctch.time);
}
```

Unutar funkcije za ispunu podataka potrebno je pobrinuti se da su odabiri padajućih izbornika postavljeni na točan odabir pomoću funkcije komponente *Spinner* naziva *setSelection()*. Ovaj proces moguće je vidjeti u središnjem dijelu programskog koda 3.15.

Nakon pritiska gumba za završavanje uređivanja potrebno je dohvatiti entitet koji se uređuje te izmijeniti njegove podatke u novozadane podatke iz fragmenta što je vidljivo u programskom kodu 3.16.

Programski kod 3.16: Promijena podataka entiteta

```
Catch ctch = catchesList.get(editID);

ctch.image = fileName;
ctch.time = timeString;
ctch.species = speciesID;
ctch.fisherman = fishermanID;
ctch.trip = tripID;
ctch.bait = baitID;
ctch.temperature = temperatureFloat;
ctch.atmosphericPressure = atmosphericPressureFloat;
ctch.wind = windFloat;
ctch.skyConditions = skyConditionsString;
ctch.length = lengthFloat;
ctch.weight = weightFloat;
ctch.quantity = quantityInt;
ctch.waterClarity = waterClarityString;
ctch.waterDepth = waterDepthFloat;
ctch.waterTemperature = waterTemperatureFloat;
ctch.fishingMethod = fishingMethodString;
ctch.released = releasedBoolean;

editCatchInDB(ctch);
```

Nakon promijene podataka poziva se funkcija za uređivanje entiteta u samoj bazi podataka vidljiva u programskom kodu 3.17. Funkcija kao ulazni parametar prima izmijenjeni entitet, pronalazi zastarjelu verziju tog istog entiteta u bazi podataka te ga zamjenjuje s novom verzijom.

Programski kod 3.17: Uređivanje podataka o entitetu u bazi

```
private void editCatchInDB(Catch ctch){
    AppDatabase db = AppDatabase.getInstance(this.getActivity());
    db.catchDao().updateCatch(ctch);
}
```

3.3.4 *Brisanje podataka*

Brisanje podataka vrši se iz fragmenta za uređivanje, vidjeti sliku 3.10. Nakon pritiska gumba za brisanje entiteta potrebno je prikazati dijalog koji će korisnika pitati je li siguran da želi obrisati entitet iz baze podataka što je vidljivo u programskom kodu 3.18. Ukoliko korisnik potvrdi postupak, potrebno je odrediti o kojem entitetu se radi te obrisati ga iz baze podataka.

Programski kod 3.18: Kod pritiska gumba izbriši

```
buttonDeleteCatch.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setTitle(R.string.menu_confirm);
        builder.setMessage(R.string.message_are_you_sure_delete);
        builder.setPositiveButton(R.string.button_yes, new DialogInterface.OnClickListener() {

            public void onClick(DialogInterface dialog, int which) {
                Catch ctch = catchesList.get(editID);
                removeCatchFromDB(ctch);
                dialog.dismiss();
                NavController navController = Navigation.findNavController(getActivity(),
R.id.nav_host_fragment);
                navController.navigate(R.id.nav_catches);
            }
        });
        builder.setNegativeButton(R.string.button_no, new DialogInterface.OnClickListener() {

            @Override
            public void onClick(DialogInterface dialog, int which) {
                dialog.dismiss();
            }
        });
        AlertDialog alert = builder.create();
        alert.show();
    }
});
```

Kako bi se podatak obrisao iz baze podataka potrebno je izvršiti funkciju u programskom kodu 3.19. Funkcija kao ulazni parametar prima entitet kojeg treba obrisati te verziju tog istog entiteta u bazi podataka te zatim briše iz same baze.

Programski kod 3.19: Brisanje entiteta iz baze

```
private void removeCatchFromDB(Catch ctch){
    AppDatabase db = AppDatabase.getInstance(this.getActivity());
    db.catchDao().deleteCatch(ctch);
}
```

4. VANJSKI SERVISI

Za potrebe projekta korišteni su vanjski servisi Google Maps te Open Weather. Google Maps korišten je za prikaz lokacija unutar aplikacije te dohvaćanje trenutne lokacije. Open Weather korišten je u paru s Google Maps servisom za dohvaćanje aktualnih podataka o vremenu.

4.1 Implementacija Google Maps sučelja

Programski kod 4.1: Dodavanje Google Maps API-a u manifest

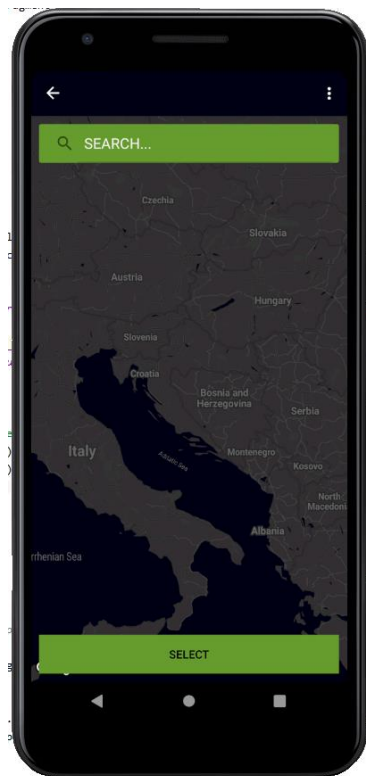
```
<meta-data  
  android:name="com.google.android.geo.API_KEY"  
  android:value="@string/google_maps_key" />
```

Programski kod 4.2: Dodavanje Google Maps API-a u gradle dokument

```
implementation 'com.google.android.gms:play-services-maps:17.0.1'
```

Google Maps API ključ moguće je dobiti na Google Cloud Platformi, no za to je potrebno napraviti i besplatni korisnički račun. Nakon dobivanja API ključa potrebno je implementirati Google Maps API u samom projektu. Kako bi implementirali Google Maps u projektu potrebno je navesti API ključ u manifestu aplikacije kao u programskom kodu 4.1 te implementirati Google Maps u gradle dokumentu aplikacije kao što je vidljivo u programskom kodu 4.2. Nakon implementacije Google Maps aplikacijsko programskog sučelja u projekt moguće je koristiti Google Maps servis za prikaz nekog područja, prikaz GPS lokacija uz pomoć GPS servisa mobilnog uređaja i mobilnih mreža i slično. Za potrebe projekta Google Maps API poslužio je za detaljan odabir te detaljan prikaz svih lokacija na kojima je korisnik do sada pecao. Moguće je i od Google Maps servisa zatražiti da korisniku u Google Maps aplikaciji (u slučaju da je na korisnikovom uređaju instalirana ta aplikacija) pokaže upute kako doći do odabrane lokacije. Isto tako za potrebe aplikacije, Google Maps API korišten je za određivanje korisnikove trenutne lokacije kako bi se kasnije ta informacija u paru s Open Weather aplikacijsko programskim sučeljem koristila za određivanje trenutnog stanja vremena na korisnikovoj lokaciji.

4.1.1 *Korištenje Google Maps API za odabir lokacije*



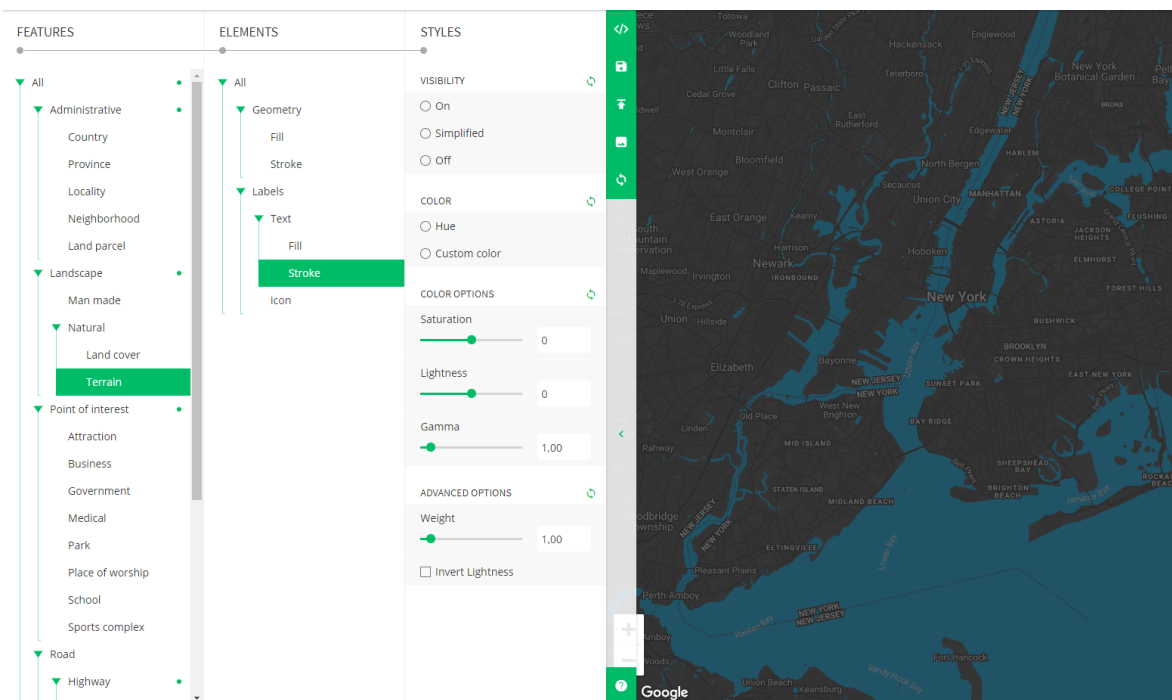
Slika 4.1: Prikaz fragmenta za odabir lokacije.

Prilikom dodavanja novog *Location* entiteta, nakon što korisnik unese osnovne podatke o lokaciji kao što su tip lokacije, naziv, slika te opis, korisnik će biti usmjeren na sučelje za odabir lokacije na Google karti koje je prikazano na slici 4.1. Korisnik tada treba pronaći lokaciju na karti te ju označiti pritiskom prsta na lokaciju. Prilikom odabira dostupna je tražilica koja omogućava lako pronalaženje lokacije jednostavnim unosom imena najbližeg mjesta lokaciji. Nakon što korisnik odabere lokaciju, tada će se zemljopisna širina i dužina lokacije spremi u *Location* entitet za daljnji prikaz. Kako bi se omogućilo prikazivanje karte te njene interakcije, potrebno je inicijalizirati Google Maps kartu unutar programskog koda fragmenta. Programski kod 4.3 prikazuje inicijalizaciju Google Maps karte.

Programski kod 4.3: Inicijalizacija Google Maps karte.

```
SupportMapFragment supportMapFragment =  
(SupportMapFragment)getChildFragmentManager().findFragmentById(R.id.add_google_maps_fo  
r_location);  
  
supportMapFragment.getMapAsync(new OnMapReadyCallback() {  
    @Override  
    public void onMapReady(@NonNull GoogleMap googleMap) {  
        try{  
            boolean success =  
googleMap.setMapStyle(MapStyleOptions.LoadRawResourceStyle(getContext(),  
R.raw.mapstyle));  
        } catch (Resources.NotFoundException e){  
        }  
    }  
}
```

Prilikom inicijalizacije karte moguće je postaviti prilagođeni dizajn karte kojeg je moguće stvoriti u mrežnom uređivaču izgleda Google Maps karte [13]. Na slici 4.2 vidljiv je izgled uređivača. Za potrebe projekta pomoću uređivača stvorena je datoteka koja sadrži podatke o izgledu Google Maps-a koji je prilagođen izgledu ostatka aplikacije.



Slika 4.2: Izgled uređivača Google Maps karte.

Nakon inicijalizacije same karte, potrebno je inicijalizirati tražilicu pomoću koje korisnik može lakše pronaći lokaciju. U programskom kodu 4.4 vidljiva je inicijalizacija tražilice. Nakon što korisnik upiše naziv lokacije, tražilica tada stvara upit koji će od Google Maps servisa zatražiti listu adresa koje sadrže upisani naziv. Nakon što se pronađe lista adresa, tada se odabire adresa s najvećim postotkom podudaranja kao uneseni naziv te se lokacija te adrese prikazuje na samoj karti.

Programski kod 4.4: Inicijalizacija tražilice

```
searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
    @Override
    public boolean onQueryTextSubmit(String query) {
        String location = searchView.getQuery().toString();
        List<Address> addressList = null;

        if (location != null || !location.equals("")){
            Geocoder geocoder = new Geocoder(getContext());
            try {
                addressList = geocoder.getFromLocationName(location, 1);
            } catch (IOException e){

            }

            Address address = addressList.get(0);
            LatLng latLng = new LatLng(address.getLatitude(),address.getLongitude());
            googleMap.addMarker(new MarkerOptions().position(latLng).title(location));
            googleMap.animateCamera(CameraUpdateFactory.newLatLngZoom(latLng, 10));

            latitude = String.valueOf(latLng.latitude);
            longitude = String.valueOf(latLng.longitude);
        }
        return false;
    }
});

@Override
public boolean onQueryTextChange(String newText) {
    return false;
}
});
```

U slučaju da korisnik pritisne na neku lokaciju na karti, briše se prethodni odabir lokacije te se na samoj karti prikazuje nova lokacija i u varijable spremaju informacije o istoj što je vidljivo u programskom kodu 4.5.

Programski kod 4.5: Inicijalizacija tražilice

```
googleMap.setOnMapClickListener(new GoogleMap.OnMapClickListener() {
    @Override
    public void onMapClick(@NonNull LatLng latLng) {
        MarkerOptions markerOptions = new MarkerOptions();
        markerOptions.position(latLng);
        markerOptions.title(latLng.latitude + " : " + latLng.longitude);
        googleMap.clear();
        googleMap.animateCamera(CameraUpdateFactory.newLatLngZoom(latLng, 10));
        googleMap.addMarker(markerOptions);

        latitude = String.valueOf(latLng.latitude);
        longitude = String.valueOf(latLng.longitude);
    }
});
```

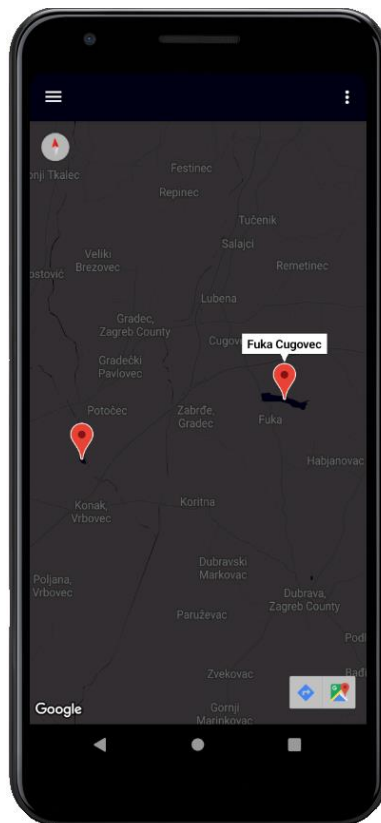
Kada se korisnik odluči za konačnu lokaciju te pritisne gumb za odabir lokacije, tada se stvara potpuni entitet *Location* koristeći prethodno unesene osnovne podatke te odabrane lokacije na Google karti pomoću funkcije u programskom kodu 4.6. Nakon dodavanja entiteta u bazu, korisnik je uspješno izvršio proces dodavanja nove lokacije te je ta lokacija sada spremna za korištenje u ostatku aplikacije.

Programski kod 4.6: Dodavanje Location entiteta u bazu

```
private void addNewLocation(String image, String name, String type, String
description, String latitude, String longitude){
    AppDatabase db = AppDatabase.getDbInstance(this.getActivity());

    Location location = new Location(image, name, type, description, latitude,
longitude);
    db.locationDao().insertLocation(location);
}
```

4.1.2 *Korištenje Google Maps API-a za pregled svih lokacija*



Slika 4.3: Fragment za pregled svih lokacija

Za potrebe projekta stvoren je fragment za prikaz svih prethodnih lokacija na jednoj Google karti vidljiv na slici 4.3. Na taj način je korisniku dostupan jednostavan pregled svih lokacija te mogućnost otvaranja Google Maps aplikacije s ciljem dobivanja instrukcija kako doći do odabrane lokacije. Korisnik može pritisnuti oznaku lokacije kako bi saznao njeno ime te otkrio mogućnosti otvaranja detaljnog pregleda u Google Maps aplikaciji kako bi dobio detaljne instrukcije kako doći do odabrane lokacije. Kako bi se postigla ovakva funkcionalnost potrebno je inicijalizirati Google Maps kartu te na njoj prikazati oznake svih prethodno spremljenih lokacija u bazi podataka, što je moguće vidjeti u programskom kodu 4.7.

Programski kod 4.7: Inicijalizacija karte za Map fragment

```
SupportMapFragment supportMapFragment =
(SupportMapFragment)getChildFragmentManager().findFragmentById(R.id.google_maps_all_Lo
cations);

supportMapFragment.getMapAsync(new OnMapReadyCallback() {
    @Override
    public void onMapReady(@NonNull GoogleMap googleMap) {
        try{
            boolean success =
googleMap.setMapStyle(MapStyleOptions.LoadRawResourceStyle(getContext(),
R.raw.mapstyle));
        } catch (Resources.NotFoundException e){

        }

        googleMap.clear();

        for (Location l: locationsList){
            Double lat = Double.parseDouble(l.latitude);
            Double lng = Double.parseDouble(l.longitude);

            LatLng initialPosition = new LatLng(lat,lng);

            MarkerOptions markerOptions = new MarkerOptions();
            markerOptions.position(initialPosition);
            markerOptions.title(l.name);
            googleMap.addMarker(markerOptions);
        }
    }
});
```

4.2 Implementacija Open Weather sučelja

Za potrebe prikaza i usporedbe vremena na početnom zaslonu aplikacije, unutar projekta korišten je Open Weather API. Korišten je i za automatsku ispunu podataka o vremenu prilikom dodavanja novog ulova u bazu podataka. Kako bi se omogućilo korištenje Open Weather aplikacijsko programskog sučelja potrebno je zatražiti API ključ na web stranici organizacije dostupnoj na izvoru [14]. Nakon dobivanja API ključa moguće je programski pozivati Open Weather servis za dohvaćanje detaljnih podataka o vremenu. Za potrebe projekta korišten je poziv servisa koji koristi geografsku širinu i dužinu lokacije za dohvat informacija [7] vidljiv u programskom kodu 4.8.

Programski kod 4.8: Poziv Open Weather servisa.

```
public void getWeatherDetails(Double lat, Double lng){
    String tempUrl = "";

    tempUrl = url + "?lat=" + lat + "&lon=" + lng + "&appid=" + appid;
    StringRequest stringRequest = new StringRequest(Request.Method.POST, tempUrl, new
Response.Listener<String>() {
        @Override
        public void onResponse(String response) {

(...)

        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                Toast.makeText(getActivity(), error.toString().trim(),
Toast.LENGTH_SHORT).show();
            }
        });

    RequestQueue requestQueue = Volley.newRequestQueue(getActivity());
    Log.d("response", "test");
    requestQueue.add(stringRequest);
}
```

Prilikom odgovora servisa, ako su podaci pronađeni tada će ih servis poslati aplikaciji u obliku JSON (engl. *JavaScript Object Notation*, skraćeno JSON) objekta kojeg je tada potrebno daljnje obraditi u korisne podatke kao što je to prikazano u programskom kodu 4.9.

Programski kod 4.9: Obrada dobivenih podataka.

```
@Override
public void onResponse(String response) {
    Log.d("response", response);
    String output = "";
    try {
        JSONObject jsonResponse = new JSONObject(response);

        JSONArray jsonArray = jsonResponse.getJSONArray("weather");
        JSONObject jsonObjectWeather = jsonArray.getJSONObject(0);

        JSONObject jsonObjectMain = jsonResponse.getJSONObject("main");

        temp = jsonObjectMain.getDouble("temp") - 273.15;
        temperature.setText(df.format(temp) + " " +
getString(R.string.unit_temperature));

        pres = jsonObjectMain.getInt("pressure");
        pressure.setText(pres + " " + getString(R.string.unit_pressure));
        hum = jsonObjectMain.getInt("humidity");
        humidity.setText(hum + getString(R.string.unit_percentage));

        JSONObject jsonObjectWind = jsonResponse.getJSONObject("wind");
        String wnd = jsonObjectWind.getString("speed");
        winds = Float.parseFloat(wnd);
        double ws = winds * 3.6;
        wind.setText(df.format(ws) + " " + getString(R.string.unit_speed));

        JSONObject jsonObjectClouds = jsonResponse.getJSONObject("clouds");
        String clouds = jsonObjectClouds.getString("all");
        int cld = Integer.parseInt(clouds);
        Context context = getContext();
        String[] stringArray =
context.getResources().getStringArray(R.array.array_sky_conditions);

        (...)

        JSONObject jsonObjectSys = jsonResponse.getJSONObject("sys");
        country = jsonObjectSys.getString("country");
        city = jsonResponse.getString("name");

        cityCountry.setText(city + ", " + country);

        loadLists();
        calculateAverages();

    } catch (JSONException e) {
        e.printStackTrace();
    }
}
```

Ovako obrađeni podaci dalje se koriste na početnom zaslonu u svrhe prikaza trenutnog vremena, prilikom računanja za koje vrste ribe je trenutno vrijeme povoljno, te prilikom unosa novog ulova za automatsko dobavljanje potrebnih informacija o vremenu. Informacije na početnom zaslonu te informacije prilikom računanja za koje vrste ribe je trenutno vrijeme povoljno se ažuriraju sa svakom automatski detektiranom promjenom lokacije kao što je to vidljivo u programskom kodu 4.10.

Programski kod 4.10: Automatska detekcija promjene lokacije.

```
@Override
public void onLocationChanged(@NonNull Location location) {
    double lat = location.getLatitude();
    double lng = location.getLongitude();

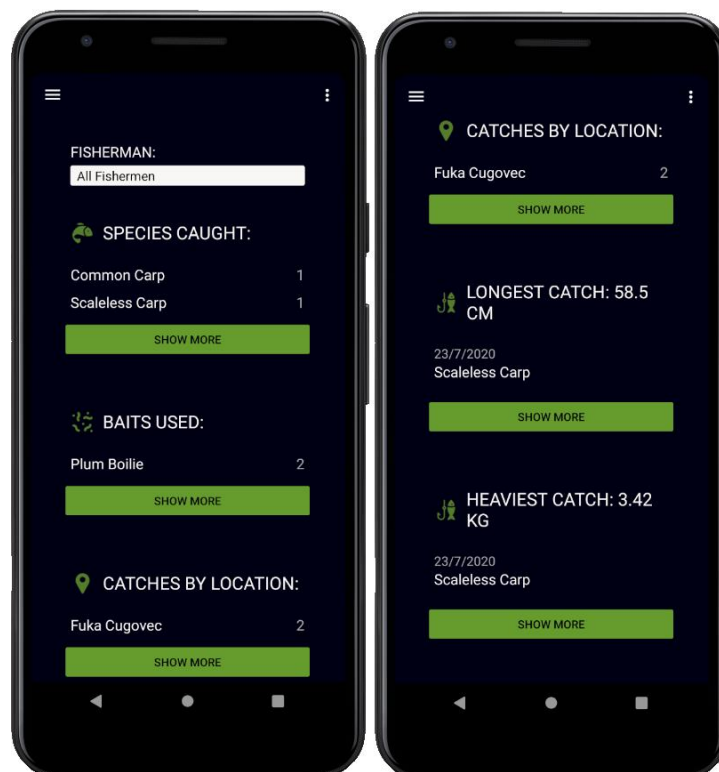
    Log.d("response", String.valueOf(lat));
    Log.d("response", String.valueOf(lng));

    getWeatherDetails(lat, lng);
}
```

5. STATISTIČKA OBRADA PODATAKA

U projektu postoji mnogo primjera obrade podataka, među njima su uvršteni fragment za pregled osnovnih statistika, fragment za pregled detaljnih statistika u obliku grafa, fragment za pregled detaljnih statistika u obliku liste te računanja prosjeka za sugestiju ulova na početnom zaslonu.

5.1.1 Osnovne statistike



Slika 5.1: Fragment za pregled osnovnih statistika

Fragment s pregledom osnovnih statistika ima dvostruku ulogu. Prva uloga fragmenta je pregled osnovnih statistika. U te statistike spadaju broj ulovljenih primjeraka ovisno o vrsti ribe, mamac korišten prilikom ulova, ulovi ovisno o lokaciji, najduži ulov te najteži ulov. Te statistike mogu se odnositi na samo jednog ribiča ili samo na određenog odabranog ribiča. Druga uloga fragmenta je izbornik za daljnji detaljni pregled prethodno navedenih statistika.

U slučaju ulovljenih primjeraka ovisno o vrsti ribe, mamca korištenog prilikom ulova te ulova ovisno o lokaciji, prikazana su tri podatka koji se najviše podudaraju s navedenom kategorijom (npr. kod ulova ovisno o lokaciji biti će prikazane prve tri lokacije s najviše zabilježenih ulova). Kod slučaja najdužeg ulova te najtežeg ulova, prikazani su osnovni podaci o ulovu ovisno o kategoriji. Za ispunjavanje sučelja s osnovnim statistikama koristi se funkcija vidljiva u programskom kodu 5.1.

Programski kod 5.1: Definicija funkcije za ispunu podataka na sučelju.

```
private void showStatistics(View root, String query){  
  
    getSpeciesCaught(root, query);  
    getBaitsUsed(root, query);  
    getCatchesByLocation(root, query);  
    getLongestCatch(root, query);  
    getHeaviestCatch(root, query);  
  
}
```

Funkcije `getSpeciesCaught()`, `getBaitsUsed()`, `getLongestCatch()` i `getHeaviestCatch()` su vrlo opsežne funkcije čija je svrha dohvaćanje, računanje, obrada te sortiranje relevantnih informacija ovisno o temi. Najjednostavniji primjer obrade podataka vidljiv je u programskom kodu 5.2, a radi se o obradi podataka kako bi se saznalo koji od ulova u bazi ima najveću masu. Nakon obrade podataka, podaci se prikazuju na sučelju fragmenta. Moguće je na pritisak gumba *Pokaži više* prikazati detaljan pregled statistika u obliku kružnog grafa ili u obliku liste.

```
if(query == getString(R.string.statistics_no_fisherman)){
    for(Catch c : catchesList){
        if (c.weight > heaviestCatch.weight){
            heaviestCatch = c;
        }
    }
} else {
    Log.d("test", "getHeaviestCatch: " + query);

    Fisherman fisherman = null;
    for (Fisherman f : fishermenList){
        if ((f.firstName + " " + f.lastName).equals(query)){
            fisherman = f;
            break;
        }
    }

    for(Catch c : catchesList){
        if ((c.fisherman == fisherman.ID) && (c.weight >
heaviestCatch.weight)){
            heaviestCatch = c;
        }
    }
}
```

5.1.2 Detaljne statistike u obliku tortnog grafa



Slika 5.2: Fragment za pregled detaljnih statistika u obliku tortnog grafa

U svrhu prikaza nekih od statistika poželjno je bilo koristiti tortne grafove kao na slici 5.2. lijevo. Kod primjera detaljnog pregleda korištenih mamaca, korisniku je pomoću tortnog grafa jasno vidljivo koji mamac je ovisno o prošlim ulovima imao najveći uspjeh na terenu. Na slici 5.2. lijevo vidljiv je opći prikaz statistika koji se odnosi na sve mamce. Najbitniji podatak je lista vrsti riba koje su ulovljene tim mamcima. Lista se puni dinamički i njen vizualni dio se dinamički povećava s obzirom na broj instanci u listi. U slučaju da korisnika zanimaju podaci koji ovise o samo jednoj vrsti mamca, tada korisnik može odabrati neki mamac na grafu pritiskom na njegov segment u grafu. Tim postupkom će se lista vrsti riba ažurirati s aktualnim podacima za odabrani mamac te će se u sredini grafa prikazati ime mamca i broj ulova postignut istim kao na slici 5.2 desno.

```
chartName = bundle.getString("chartName");
ArrayList<Integer> values = bundle.getIntegerArrayList("values");
ArrayList<String> strings = bundle.getStringArrayList("strings");

ArrayList<PieEntry> entries = new ArrayList<>();

IntStream.range(0, values.size()).forEachOrdered(n -> {
    entries.add(new PieEntry(values.get(n), strings.get(n)));
});

PieDataSet pieDataSet = new PieDataSet(entries, chartName);
pieDataSet.setColors(ColorTemplate.COLORFUL_COLORS);
pieDataSet.setValueTextColor(Color.WHITE);
pieDataSet.setValueFormatter(new DecimalRemover(new
DecimalFormat("###,###,###")));
pieDataSet.setValueTextSize(16f);
pieDataSet.setSliceSpace(3f);

PieData pieData = new PieData(pieDataSet);

pieChart.setData(pieData);
pieChart.setEntryLabelColor(Color.WHITE);
pieChart.getDescription().setEnabled(false);
pieChart.setCenterText(chartName);
pieChart.setCenterTextColor(Color.WHITE);
pieChart.setCenterTextSize(22f);
pieChart.setHoleColor(Color.TRANSPARENT);
pieChart.setDrawHoleEnabled(true);
pieChart.setHoleRadius(60);
pieChart.setTransparentCircleRadius(65);
pieChart.setTransparentCircleAlpha(50);
pieChart.getLegend().setEnabled(false);
pieChart.animate();
pieChart.setOnChartValueSelectedListener(this);
```

Prije korištenja tortnog grafa, potrebno ga je inicijalizirati kao što je vidljivo u programskom kodu 5.3. Tijekom inicijalizacije moguće je postaviti izgled grafa i vrijednosti koje će prikazivati.

```
@Override
public void onValueSelected(Entry e, Highlight h) {
    PieEntry pe = (PieEntry) e;
    RelativeSizeSpan largeSizeText = new RelativeSizeSpan(1f);

    RelativeSizeSpan mediumSizeText = new RelativeSizeSpan(.8f);

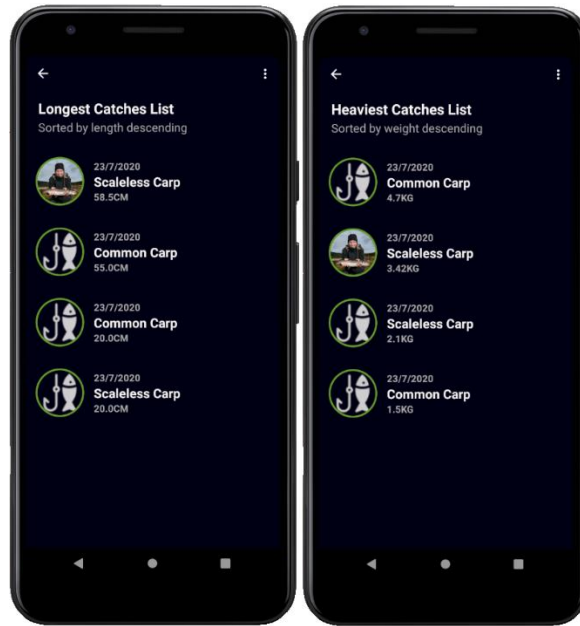
    RelativeSizeSpan smallSizeText = new RelativeSizeSpan(.5f);

    SpannableString spannableString = new SpannableString(pe.getLabel() + "\n"
+ df.format(pe.getValue()) + " " + getString(R.string.statistics_catches));
    spannableString.setSpan(new
ForegroundColorSpan(getResources().getColor(R.color.palette_darker_gray)),
pe.getLabel().length(), spannableString.length(),
Spannable.SPAN_EXCLUSIVE_INCLUSIVE);
    spannableString.setSpan(largeSizeText, 0, pe.getLabel().length(),
Spannable.SPAN_INCLUSIVE_INCLUSIVE);
    spannableString.setSpan(mediumSizeText, pe.getLabel().length(),
spannableString.length(), Spannable.SPAN_EXCLUSIVE_INCLUSIVE);

    pieChart.setCenterText(spannableString);
    selectedItem = pe.getLabel();
    fillStatistics(root);
}
```

Nakon inicijalizacije potrebno je navesti što će se dogoditi ako korisnik pritisne neku krišku na grafu što je vidljivo u programskom kodu 5.4. U slučaju da korisnik pritisne krišku u sredini grafa prikazati će se naziv kriške te vrijednost na koju se odnosi. Kod slučaja pregleda korištenih mamca, prikazati će se naziv mamca i broj ulova postignut tim mamcem. Nakon toga će se ponovo ispuniti lista ulova, ovaj puta samo s ulovima koji se odnose na odabrani mamac, vidljivo na slici 5.2 desno. U slučaju da korisnik ponovo pritisne već odabranu krišku, graf će se vratiti u prvobitno stanje.

5.1.3 Detaljne statistike u obliku liste



Slika 5.3: Fragmenti za pregled detaljnih statistika u obliku liste

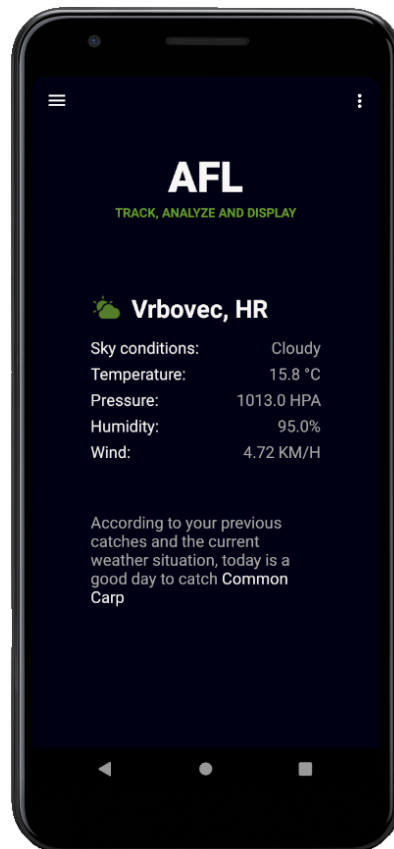
U svrhu prikaza nekih od jednostavnijih statistika bilo je poželjno za njihov prikaz koristiti liste. Na slici 5.3 lijevo vidljiva je primjena liste za prikaz najdužih ulova sortirano od najdužeg do najkraćeg, a na slici desno najtežih ulova od najtežeg do najlakšeg.

Programski kod 5.5: Funkcija za ispunu liste

```
private void loadList(int selection){
    if (selection == 1){
        catchesList.sort(Comparator.comparing(Catch::getWeight).reversed());
    } else if (selection == 2){
        catchesList.sort(Comparator.comparing(Catch::getLength).reversed());
    }
    CatchStatisticsListAdapter tripsAdapter = new
    CatchStatisticsListAdapter(getActivity(), catchesList);
    list.setAdapter(tripsAdapter);
}
```

Za popunjavanje liste korištena je funkcija iz programskog koda 5.5 koja kao ulazni parametar prima broj koji služi kao selektor načina sortiranja liste. Nakon selekcije i sortiranja, sortirani podaci prosljeđuju se prilagođenom adapteru za prikaz entiteta u listi.

5.1.4 *Korištenje statistike za sugestiju ulova*



Slika 5.4: Home fragment s pregledom vremena i sugestijom ulova

Na slici 5.4 vidljiv je *Home* fragment na kojemu se nalazi sugestija ulova ovisno o trenutnom stanju vremena. Korisniku je dostupan i pregled trenutnog vremena u slučaju da sam želi odlučiti na što će ciljati. Ukoliko jedna vrsta ribe ima najveće šanse za ulov ovisno o vremenu, tada će ime te vrste biti prikazano u sugestiji ulova. Ukoliko više vrsta dijeli prvo mjesto s istim šansama za ulov, tada će sve vrste biti prikazane u obliku popisa. Računanje sugestija funkcionira na principu sakupljanja bodova. Vrsta ribe s najviše bodova ima najveću šansu za ulov s obzirom na trenutno stanje vremena.

```
if(speciesAverage.temperatureAverage > temp){
    if ((speciesAverage.temperatureAverage - temp) < 1){
        speciesAverage.points += 3;
    }
    else if ((speciesAverage.temperatureAverage - temp) < 2){
        speciesAverage.points += 2;
    }
    else if ((speciesAverage.temperatureAverage - temp) < 3){
        speciesAverage.points += 1;
    }
} else if (speciesAverage.temperatureAverage < temp){
    if ((temp - speciesAverage.temperatureAverage) < 1){
        speciesAverage.points += 3;
    }
    else if ((temp - speciesAverage.temperatureAverage) < 2){
        speciesAverage.points += 2;
    }
    else if ((temp - speciesAverage.temperatureAverage) < 3){
        speciesAverage.points += 1;
    }
} else if (speciesAverage.temperatureAverage == temp){
    speciesAverage.points += 3;
}
```

Primjer sakupljanja bodova vidljiv je u programskom kodu 5.6. Prije izvršavanja navedenog koda se za svaku vrstu ribe računa prosjek svih ulova te vrste za svaku stavku (npr. temperatura, pritisak, brzina vjetra i slično.) Zatim se kao u programskom kodu 5.6 svaki prosjek uspoređuje s trenutnim vremenom te ovisno o sličnosti dodjeljuju bodovi za tu vrstu ribe. Nakon izračuna bodova za svaku vrstu u sugestiji ulova prikazuje se vrsta ili vrste s najviše osvojenih bodova.

6. ZAKLJUČAK

Tema završnog rada je Android mobilna aplikacija za praćenje i analizu ribolovnih ulova i putovanja. Aplikacija u svom radu koristi Google Maps te Open Weather servise za dohvaćanje određenih informacija. Korisniku aplikacije moguće su akcije dodavanja, pregledavanja, uređivanja i brisanja podataka. Korisniku se pruža mogućnost vizualnog pregleda statistika putem grafova i detaljnih opisa. Pružena je i mogućnost asistiranja u odabiru ulova s obzirom na prognozu vremena te prošla iskustva korisnika u usporedbi s istim ili sličnim uvjetima. Tijekom razvoja aplikacije pojavili su se samo manji problemi koje nije bilo teško ispraviti. Najveći problem stvarao je Google Maps servis zbog potrebe stvaranja korisničkog računa kojeg je potom trebalo spojiti s aplikacijom kako bi se servis mogao koristiti. Nakon razvoja same aplikacije vidljiva je moguća primjena aplikacije za osobno praćenje ulova i putovanja, za ribolovna natjecanja, kao službeni zapisnik nekog ribičkog društva i slično. Aplikacija skriva puno potencijala za daljnji razvoj. Neki od dodatnih ideja uključuju korištenje tehnologije detekcije objekata za detekciju vrste ulovljene ribe, dodavanje opcije za detaljnu kreaciju i odabir primame (što uključuje sve sastojke, arome i boje), širenje raspona aplikacije u ostale elemente ribolova (ribolovna oprema, ribolovni sistemi i slično), detaljnije postojeće i naknadno dodane statistike, globalne statistike svih korisnika, dodavanje rang liste te stvaranje verzije aplikacije isključivo stvorene za korištenje u svrhe administracije ribičkog društva ili natjecanja. Daljnji cilj razvoja aplikacije je stvoriti u potpunosti doradenu verziju aplikacije bez gore navedenih ideja koja bi potom bila postavljena na Google Trgovinu kako bi postala dostupna široj javnosti. Nakon toga slijedi daljnji razvoj prethodno navedenih ideja počevši od opcija za primamu te naprednih statistika.

7. LITERATURA

Primjeri:

- [1] M. Karch, »Lifewire« 1 Rujan 2021. [Mrežno] Dostupno na: <https://www.lifewire.com/what-is-google-android-1616887> [Pokušaj pristupa 10 Rujan 2021]
- [2] C. Scott Brown »Android Authority« 1 Kolovoz 2021 [Mrežno] Dostupno na: <https://www.androidauthority.com/what-is-android-328076/> [Pokušaj pristupa 10 Rujan 2021]
- [3] »El-pro-cus« 2020. [Mrežno] Dostupno na: <https://www.elprocus.com/what-is-android-introduction-features-applications/> [Pokušaj pristupa 10 Rujan 2021]
- [4] J. Hartman »Guru99« 28 Kolovoz 2021 [Mrežno] Dostupno na: <https://www.guru99.com/java-platform.html> [Pokušaj pristupa 10 Rujan 2021]
- [5] »Android Developers« 2021 [Mrežno] Dostupno na: <https://developer.android.com/studio/intro> [Pokušaj pristupa 10 Rujan 2021]
- [6] »TechTarget« 2021 [Mrežno] Dostupno na: <https://whatis.techtarget.com/definition/Google-Maps> [Pokušaj pristupa 10 Rujan 2021]
- [7] »OpenWeather« 2021 [Mrežno] Dostupno na: <https://openweathermap.org/guide> [Pokušaj pristupa 10 Rujan 2021]
- [8] C. H. Parcal »Medium« 25 Lipanj 2020 [Mrežno] Dostupno na: <https://medium.com/huawei-developers/room-database-with-kotlin-mvvm-architecture-477c3ad3c264> [Pokušaj pristupa 10 Rujan 2021]
- [9] »Android Developers« 2021 [Mrežno] Dostupno na: <https://developer.android.com/reference/androidx/room/Entity> [Pokušaj pristupa 16 Rujan 2021]
- [10] »Android Developers« 2021 [Mrežno] Dostupno na: <https://developer.android.com/reference/android/arch/persistence/room/Dao> [Pokušaj pristupa 16 Rujan 2021]
- [11] »Android Developers« 2021 [Mrežno] Dostupno na: <https://developer.android.com/reference/android/arch/persistence/room/Database> [Pokušaj pristupa 16 Rujan 2021]
- [12] »Android Developers« 2021 [Mrežno] Dostupno na: <https://developer.android.com/guide/fragments> [Pokušaj pristupa 16 Rujan 2021]

[13] »Snazzy Maps« 2021 [Mrežno] Dostupno na: <https://snazzymaps.com> [Pokušaj pristupa 17 Rujan 2021]

[14] »OpenWeather« 2021 [Mrežno] Dostupno na: <https://openweathermap.org> [Pokušaj pristupa 17 Rujan 2021]

Tablica programskih kodova

<i>Programski kod 3.1: Location Entitet</i>	7
<i>Programski kod 3.2: Location Dao</i>	8
<i>Programski kod 3.3: Room baza podataka</i>	9
<i>Programski kod 3.4: Odabir slike u fragmentu</i>	12
<i>Programski kod 3.5: Spremanje slike u memoriju te bazu podataka</i>	13
<i>Programski kod 3.6: Funkcija za provjeru i dodjelu dozvola za pristup memoriji.</i>	13
<i>Programski kod 3.7: Automatsko dodijeljivanje vremena ulova</i>	14
<i>Programski kod 3.8: Prikaz i upravljanje Time Picker Dialog komponentom.</i>	15
<i>Programski kod 3.9: Ispuna padajućeg izbornika za ribiče</i>	15
<i>Programski kod 3.10: Otvaranje prozora za dodavanje novog entiteta</i>	16
<i>Programski kod 3.11: Primjer pripreme podatka iz padajućeg izbornika.</i>	17
<i>Programski kod 3.12: Stvaranje i dodavanje entiteta u bazu podataka</i>	17
<i>Programski kod 3.13: Adapter entiteta Catch</i>	19
<i>Programski kod 3.14: Inicijalizacija liste</i>	20
<i>Programski kod 3.15: Inicijalizacija podataka fragmenta za uređivanje</i>	22
<i>Programski kod 3.16: Promijena podataka entiteta</i>	23
<i>Programski kod 3.17: Uređivanje podataka o entitetu u bazi</i>	23
<i>Programski kod 3.18: Kod pritiska gumba izbriši</i>	24
<i>Programski kod 3.19: Brisanje entiteta iz baze</i>	24
<i>Programski kod 4.1: Dodavanje Google Maps API-a u manifest</i>	25
<i>Programski kod 4.2: Dodavanje Google Maps API-a u gradle dokument</i>	25
<i>Programski kod 4.3: Inicijalizacija Google Maps karte.</i>	27
<i>Programski kod 4.4: Inicijalizacija tražilice</i>	28
<i>Programski kod 4.5: Inicijalizacija tražilice</i>	29
<i>Programski kod 4.6: Dodavanje Location entiteta u bazu</i>	29
<i>Programski kod 4.7: Inicijalizacija karte za Map fragment</i>	31
<i>Programski kod 4.8: Poziv Open Weather servisa.</i>	32
<i>Programski kod 4.9: Obrada dobivenih podataka.</i>	33
<i>Programski kod 4.10: Automatska detekcija promjene lokacije.</i>	34
<i>Programski kod 5.1: Definicija funkcije za ispunu podataka na sučelju.</i>	36
<i>Programski kod 5.2: Primjer jednostavne obrade podataka.</i>	37
<i>Programski kod 5.3: Inicijalizacija tortnog grafa</i>	39
<i>Programski kod 5.4: Odabir kriške grafa</i>	40
<i>Programski kod 5.5: Funkcija za ispunu liste</i>	41
<i>Programski kod 5.6: Primjer sakupljanja bodova</i>	43

Tablica slika

<i>Slika 3.1 Prikaz arhitekture Room baze podataka[8]</i>	6
<i>Slika 3.3 Primjeri aplikacija sa jednom aktivnošću [12]</i>	10
<i>Slika 3.4 Prikaz fragmenta za unos novog ulova.</i>	11
<i>Slika 3.5 Prikaz korisničkog sučelja za odabir slike.</i>	12
<i>Slika 3.6: Prikaz korisničkog sučelja za odabir vremena.</i>	14
<i>Slika 3.7 Primjer prozora za dodavanje novog entiteta iz fragmenta.</i>	16
<i>Slika 3.8 Prikaz fragmenta za pregled podataka.</i>	18
<i>Slika 3.9 Prikaz ListItem elementa za Catch entitet</i>	18
<i>Slika 3.10 Prikaz fragmenta za uređivanje podataka.</i>	21
<i>Slika 4.1: Prikaz fragmenta za odabir lokacije.</i>	26
<i>Slika 4.2: Izgled uređivača Google Maps karte.</i>	27
<i>Slika 4.3: Fragment za pregled svih lokacija</i>	30
<i>Slika 5.1: Fragment za pregled osnovnih statistika</i>	35
<i>Slika 5.2: Fragment za pregled detaljnih statistika u obliku tortnog grafa</i>	38
<i>Slika 5.3: Fragmenti za pregled detaljnih statistika u obliku liste</i>	41
<i>Slika 5.4: Home fragment sa pregledom vremena i sugestijom ulova</i>	42

8. OZNAKE I KRATICE

API – Application programming interface (aplikacijsko programsko sučelje)

DAO – Data Access Object (objekt za pristup podacima)

HTML – Hypertext Markup Language (prezentacijski jezik za izradu web stranica)

JSON – JavaScript Object Notation (format za razmjenu podataka)

XML – Extensible Markup Language (jezik za označavanje podataka)

ZIP – Zone Improvement Plan (poštanski broj)

9. SAŽETAK

Naslov: Mobilna aplikacija za praćenje i analizu ribolovnih ulova i putovanja

U ovom radu opisana je izrada native Android mobilne aplikacije za unos, prikaz, uređivanje, brisanje i obradu podataka o ribičima, ribolovnim lokacijama, ribolovnim putovanjima, ulovima, mamcima i slično. Aplikacija je izrađena koristeći Java programski jezik te Android Studio razvojno okruženje. Unutar aplikacije moguće je dodati, pregledavati, uređivati i brisati mnogo podataka bitnih u ribolovnom svijetu. Uz to moguće je pregledati i koristiti statistike kao prikaz stvarnih situacija. U radu je korišteno i objašnjeno korištenje Google Maps servisa te Open Weather servisa koji su korišteni za dohvaćanje podataka o lokaciji i vremenu na istoj.

Ključne riječi: Android, Google Maps, Open Weather, Room, statistika, obrada podataka, baza podataka

10. ABSTRACT

Title: Mobile application for monitoring and analysis of fishing catches and trips

This paper describes the development of a native Android mobile application for entering, displaying, editing, deleting and processing data on fishermen, fishing locations, fishing trips, catches, baits and the like. The application was created using the Java programming language in the Android Studio development environment. Within the application, it is possible to add, view, edit and delete a lot of data important in the fishing world. In addition, it is possible to view and use statistics as representations of real situations. The paper explains the use of the Google maps service and the Open weather service which are used to capture location and weather data respectfully.

Keywords: Android, Google Maps, Open Weather, Room, statistics, data processing, database

IZJAVA O AUTORSTVU ZAVRŠNOG RADA

Pod punom odgovornošću izjavljujem da sam ovaj rad izradio/la samostalno, poštujući načela akademske čestitosti, pravila struke te pravila i norme standardnog hrvatskog jezika. Rad je moje autorsko djelo i svi su preuzeti citati i parafraze u njemu primjereno označeni.

Mjesto i datum	Ime i prezime studenta/ice	Potpis studenta/ice
U Bjelovaru, <u>29.9.2021.</u>	NINO MACUT	Nino Macut

Prema Odluci Veleučilišta u Bjelovaru, a u skladu sa Zakonom o znanstvenoj djelatnosti i visokom obrazovanju, elektroničke inačice završnih radova studenata Veleučilišta u Bjelovaru bit će pohranjene i javno dostupne u internetskoj bazi Nacionalne i sveučilišne knjižnice u Zagrebu. Ukoliko ste suglasni da tekst Vašeg završnog rada u cijelosti bude javno objavljen, molimo Vas da to potvrdite potpisom.

Suglasnost za objavljivanje elektroničke inačice završnog rada u javno dostupnom nacionalnom repozitoriju

NINO MACUT

ime i prezime studenta/ice

Dajem suglasnost da se radi promicanja otvorenog i slobodnog pristupa znanju i informacijama cjeloviti tekst mojeg završnog rada pohrani u repozitorij Nacionalne i sveučilišne knjižnice u Zagrebu i time učini javno dostupnim.

Svojim potpisom potvrđujem istovjetnost tiskane i elektroničke inačice završnog rada.

U Bjelovaru, 29.9.2021.

Nino Macut

potpis studenta/ice