Izvedba Andoid mobilne aplikacije s primjenom lokacijskih servisa

Lukaček, Kristijan

Sveučilište
Sjever

Završni rad br. 446/EL/2019

Izvedba Android mobilne aplikacije s primjenom lokacijskih servisa

Kristijan Lukaček, 5043/601

Varaždin, rujan 2019. godine

Sveučilište
Sjever

Odjel za elektrotehniku

Završni rad br. 446/EL/2019

Izvedba Android mobilne aplikacije s primjenom lokacijskih servisa

Student

Kristijan Lukaček, 5043/601

Mentor

mr.sc. Matija Mikac

Varaždin, rujan 2019. godine

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR–42000 Varaždin

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL Odjel za elektrotehniku

STUDIJ preddiplomski stručni studij Elektrotehnika

PRISTUPNIK Kristijan Lukaček | MATIČNI BROJ 5043/601

DATUM 12.07.2019. | KOLEGIJ Mobilne aplikacije

NASLOV RADA Izvedba Android mobilne aplikacije s primjenom lokacijskih servisa

NASLOV RADA NA ENGL. JEZIKU Implementation of Android location services based application

MENTOR mr.sc. Matija Mikac | ZVANJE viši predavač

ČLANOVI POVJERENSTVA

1. Miroslav Horvatić, dipl.ing., predavač
2. mr.sc. Ivan Šumiga, dipl.ing.el., viši predavač
3. mr.sc. Matija Mikac, dipl.ing.el., viši predavač
4. dr.sc. Dunja Srpak, dipl. ing.el. - rezervni član
5.

Zadatak završnog rada

BROJ 446/EL/2019

OPIS

Cilj završnog rada je izvesti potpuno funkcionalnu mobilnu aplikaciju za Android operacijski sustav baziranu na proširenoj primjeni lokacijskih servisa (pohrana GPS lokacija, vizualizacija). Potrebno je teoretski obraditi elemente Android operacijskog sustava i razvojnog okruženja vezane uz samu izvedbu aplikacije. Također je nužno implementirati osnovne elemente web aplikacije koja će preuzimati podatke sa uređaja i omogućiti određenu razinu vizualizacije i u obliku web aplikacije - pri tome voditi računa o identifikaciji korisnika. Kao dodatne mogućnosti (proširena primjena) implementirati funkcionalnost slanja proizvoljnih podataka i fotografija sa geo-pozicijom i dohvat povratnog statusa sa poslužitelja (na primjer, prijave komunalnih problema redarima i slično).

U radu je potrebno:
- obraditi i opisati ključne korištene elemente Android operacijskog sustava - lokacijske servise, pohranu podataka na uređaju, mrežnu komunikaciju i slanje podataka na poslužitelj, pristup vizualizaciji lokacijskih podataka u obliku mapa, kontrolu kamere itd.
- implementirati i dokumentirati aplikaciju koja će omogućiti GPS praćenje
- dodatno implementirati mogućnost slanja podataka i fotografija sa uređaja na server
- implementirati osnovne elemente web aplikacije nužne za funkcioniranje aplikacije ovog tipa

ZADATAK URUČEN 19.08.2019. | POTPIS MENTORA M. Mikac

SVEUČILIŠTE SJEVER

Predgovor

Prije svega bih se htio zahvaliti mentoru koji mi je pomagao tijekom izrade ovog rada savjetima i svojim stručnim mišljenjem.

Također bih se htio zahvaliti svim drugim profesorima koji su mi tijekom studija pomagali i podučavali me. Bez njihovog doprinosa ovaj rad također ne bi bio moguć.

Posebno bih se htio zahvaliti svojim roditeljima koji su mi svojim radom i odricanjem uopće omogućili studij te su mi bili stalna podrška i oslonac. Zahvalio bih se još i ostatku obitelji koji je isto igrao važnu ulogu.

Bilo bi mi drago da ovaj rad i zaključci dobiveni u radu mogu u budućnosti nekome poslužiti odnosno da netko može iskoristiti ono što je obrađeno u ovom radu.

Sažetak

U ovom radu je prikazana i implementirana upotreba lokacijskih servisa koji su sastavni dio najraširenijeg operacijskog sustava za pametne telefone i druge mobilne uređaje, Androida. Gotovo svi mobilni uređaji danas uključuju sklopovska rješenja koja podržavaju korištenje i primjenu lokacijskih servisa, a otvorenost operacijskog sustava omogućava programski pristup i primjenu u aplikacijama. U radu je korišten uobičajeni lokacijski servis koji podatke o lokaciji uređaja preuzima iz GPS sustava. Obzirom da neki uređaji osim standardnog GPS-a podržavaju i druge izvedbe globalnog pozicioniranja, GPS se u radu koristi kao sveobuhvatni naziv za sustav satelitske navigacije. Da bi lokacijski servisi dali neku smislenu, funkcionalnost, osim golog prikazivanja rezultata, aplikacija koja je izrađena kao praktični dio završnog rada proširena je vizualizacijom lokacije pomoću karti i raznih grafičkih elemenata dodanih na karte, pohranom podataka na nekoliko različitih načina. Kao dodatni element proširivosti primjene podataka prikupljenih pomoću lokacijskih servisa, praktični dio završnog rada omogućava i povezanost mobilnog uređaja s mrežnim poslužiteljem na kojem je podatke moguće pohraniti i vizualizirati korištenjem pomoćne web aplikacije, koja je razvijena kao sastavnica praktičnog dijela.

Dodatna mogućnost primjene uključila je izradu podsustava koji omogućava generičko slanje specifičnih podataka (fotografija i lokacija) za neke buduće primjene (npr. prijava komunalnom redarstvu.). Svi elementi korišteni u realizaciji praktičnog dijela rada (mobilna aplikacije i web aplikacija) razmotreni su i opisani u teoretskom dijelu rada, a izvorni su kodovi priloženi kao dodatak.

Popis korištenih kratica

API	sučelje kojim se spaja na vanjske funkcionalnosti, eng. *Application Programming Interface*
APK	predstavlja ekstenziju arhivirane datoteke koja sadrži sve elemente Android aplikacije, eng. *Android Package*
DEX	format izvršnog koda na Android OS-u, eng. *Dalvik EXecutable*
DP	oblik točke koji nije vezan za fizičku veličinu ekrana već se skalira po potrebi, koristi se za grafiku, a manje za tekst, eng. *Density-Independent Pixels*
GLONASS	ruski satelitski navigacijski sustav, eng. *GLObal NAvigation Satellite System*
GPS	američki satelitski navigacijski sustav, prvi te vrste, često se koristi kao naziv za sve sustave satelitske navigacije, eng. Global Positioning System
HTML	jezik za definiranje strukture i sadržaja hipertekstualnih dokumenata, eng. *HyperText Markup Language*
HTTP	komunikacijski protokol za slanje hipertekstualnih dokumenata između klijenta i poslužitelja, eng. *HyperText Transfer Protocol*
IRNSS	indijski regionalni satelitski navigacijski sustav, eng. *Indian Regional Navigation Satellite System*
JPEG	format pohrane slika s omogućenom kompresijom s gubicima u kvaliteti, ima ekstenziju **.jpg*, eng. *Joint Photographic Experts Group*
JSON	format zapisa strukturiranih podataka, eng. *Java Script Object Notation*
JVM	virtualni uređaj na kojem radi Java program, eng. *Java Virtual Machine*
OIB	osobni identifikacijski broj građana
OS	operacijski sustav
PHP	programski jezik za web aplikacije koji se koristi na poslužiteljima često u kombinaciji s HTML-om, eng. *Personal Home Page* nekad, danas eng. *Hypertext Preprocessor*

QZSS	japanski sustav za augmentaciju preciznosti navigacijskih sustava koji će u skorijoj budućnosti postati zasebni samostalni regionalni sustav satelitske navigacije, eng. *Quasi-Zenith Satellite System*
RSSI	nestandardizirana mjera jačine mrežnog WIFI signala, eng. *Received Signal Strength Indicator*
SD	standard vanjskih kartica za pohranu podataka, u obliku malih kartica, na mobilnim uređajima se uobičajeno koristi microSD format, eng. *Secure Digital*
SDK	set programa koji je potreban za izradu Android aplikacije, svi kompilatori, knjižnice, čak emulatori itd., eng. *Software Development Kit*
SMS	kratke tekstualne poruke, eng. *Short Message Service*
SP	oblik točke koji nije vezan za fizičku veličinu ekrana već se skalira po potrebi, koristi se za veličinu teksta, eng. *Scale-Independent Pixels*
SQL	jezik za definiranje strukturiranih upita u relacijskim bazama podataka, eng. *Structured Query Language*
TCP	protokol transportnog sloja za pouzdani prijenosa podataka u mreži, eng. *Transmission Control Protocol*
TLS	protokol za sigurnu komunikaciju preko mreže, eng. *Transport Layer Security*
URI	simbolička adresa nekog resursa, eng. *Uniform Resource Identifier*
URL	simbolička adresa mrežnog resursa, eng. *Uniform Resource Locator*
USB	standard sabirnice za prijenos podataka, eng. *Universal Serial Bus*
USBOTG	standard sabirnice za komunikaciju s mobilnim uređajem, podržava ga Android i nema potrebe za dodatnim pogonskim programima da bi radio, eng. *Universal Serial Bus On-The-Go*
UTF-8	format za zapis znakova, standardizira zapis posebnih znakova, eng. *Unicode Transformation Format 8*
WIFI RTT	mjerenje trajanja vremena potrebnog za dolazak signala od WIFI pristupne točke do mobilnog uređaja, mjerenjem tog vremena dobiva se udaljenost do pristupne točke, eng. *WIFI Round Trip Time*

WIFI	protokol za bežičnu mobilnu komunikaciju, eng. *Wireless Fidelity*
XML	jezik za zapis strukturiranih podataka, eng. *eXtensible Markup Language*

Sadržaj

1. Uvod

Osnovni zadatak završnog rada je izrada mobilne aplikacije koja uključuje funkcionalnost dohvaćanja lokacije uređaja, vizualizacije očitanih i pohranjenih lokacija i mrežne komunikacije. Dodatno je, uz glavni zadatak i spomenuta proširenja je obrađena upotreba resursa uređaja (dohvat fotografije preko kamere), a kompletni sustav je nadopunjen i poslužiteljskom aplikacijom koja omogućava prihvat podataka s uređaja na poslužitelj kao i pratećom web aplikacijom za vizualizaciju tih podataka. Izradi same aplikacije je prethodilo upoznavanje s mogućnostima operacijskog sustava. Obzirom na temu i potrebu implementacije prema zadatku rada, posebna pažnja posvećena je lokacijskim servisima, mrežnim servisima i pohrani podataka. Nakon upoznavanja s osnovama slijedila je izrada aplikacije.

Obzirom da je od samih početaka glavni programski jezik za razvoj Android aplikacija Java, može se pretpostaviti da je većina aplikacija razvijena upravo korištenjem tog programskog jezika. Uz programski jezik Java u novije vrijeme za izradu mobilnih aplikacije koristi se i noviji Kotlin programski jezik. Kotlin je univerzalni programski jezik razvijan od strane tvrtke JetBrains od 2010. godine. Od jeseni 2017. glavno razvojno okruženje, Android Studio (koje je zapravo alat koji razvija upravo JetBrains), dozvoljava korištenje Kotlina, a Google je jezik karakterizirao kao jezik prvog razreda (eng. *First-class language*) za razvoj mobilnih Android aplikacija. Od svibnja 2019. Google definira Kotlin kao poželjni jezik (eng. *Preferred language*) za razvoj novih Android aplikacija [1]. Vrlo je vjerojatno da će Kotlin postepeno Javu za razvoj novih Android aplikacija. Tijekom školovanja je korišten programski jezik Java pa je i mobilna aplikacija napisana u Javi.

Ovaj rad se sastoji od dvije glavne cjeline. Prva cjelina se bavi pregledom osnovnih karakteristika sustava Android, uz posebno izdvojene podsustave bitne za ovaj završni rad (lokacijski servisi, pohrana, vizualizacija, mrežna komunikacija). Taj dio opisan je u drugom poglavlju koje slijedi nakon uvoda. Drugi dio rada opisuje postupak izvedbe praktičnog dijela rada, po poglavljima – u trećem poglavlju daje se pregled razvoja mobilne aplikacije – od kratkog osvrta na razvojno okruženje do detalja vezanih uz implementaciju korisničkog sučelja i pojedinih funkcionalnosti aplikacije. U četvrtom poglavlju opisani su detalji oko serverskog dijela aplikacije – daje se pregled koncepta i potrebnih podsustava, te opisuje izvedba prateće web aplikacije.

U zaključku rada sagledavaju se moguća proširenja razvijane aplikacije. Kao prilog završnom radu daje se programski kod i XML zapisi korišteni za definiranje sučelja – važniji elementi

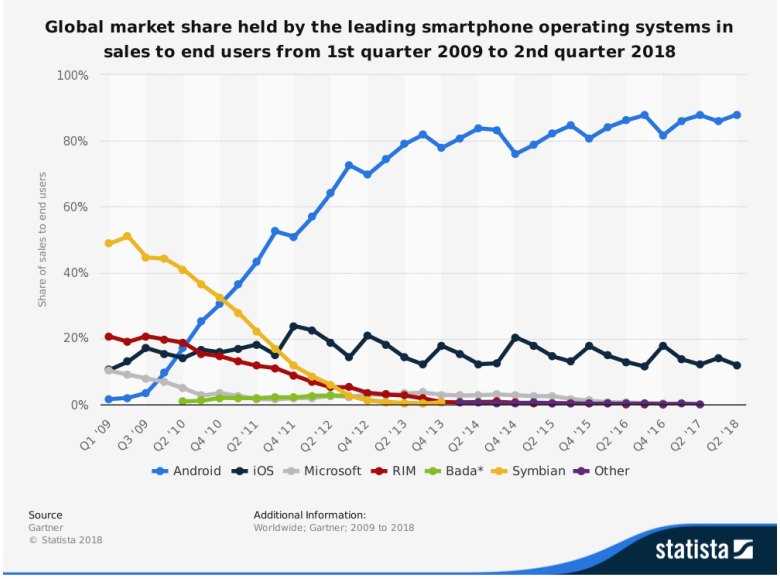
programskog koda korišteni su i u samom tekstu rada, no kompletni kod je izdvojen iz rada i daje se kao prilog.

2. Android i korišteni podsustavi

U ovom poglavlju je dan kratki pregled sustav Android i nekih njegovih značajki. Isto tako su obrađene osnove programskog jezika i razvojnog okruženja u kojem je izrađivana ova aplikacija (zapravo, ukratko je opisana Android razvojna platforma). Posebni je naglasak stavljen na lokacijske servise, kao što i zadatak rada traži. Nadalje, izdvojeni su i kratko opisani svi podsustavi korišten u izvedbi praktičnog dijela rada – pohrana podataka, vizualizacija i mrežna komunikacija.

2.1. O operacijskom sustavu Android

Android svoje korijene vuče iz 2003. godine kada su Andy Rubin, Rich Miner, Nick Sears i Chris White osnovali tvrtku Android Inc. Tvrtka je isprva razvijala operacijske sustave za pametne kamere ali se ubrzo radi premalog tržište prebacuju na razvoj operacijskih sustava za pametne mobilne uređaje. Ti uređaji su tada tek u začetcima. U ono vrijeme glavni suparnici su im bili Symbian i Windows Mobile. Nakon dvije godine razvoja, 2005. ih kupuje Google. Google 2007. inicira osnivanje udruge/organizacije nazvane „*Open Headset Alliance*“ koja okuplja proizvođače pametnih uređaja, tvrtke koje razvijaju aplikacije i telekomunikacijske operatere. Standardiziraju se zahtjevi za Android operacijski sustav i počinje njegova ozbiljnija primjena u mobilnim uređajima. Android je otvorenog koda i dostupan svima, ali Google zadržava mogućnost uskraćenja nekih važnih mogućnosti i usluga ako proizvođač nije certificiran [2].



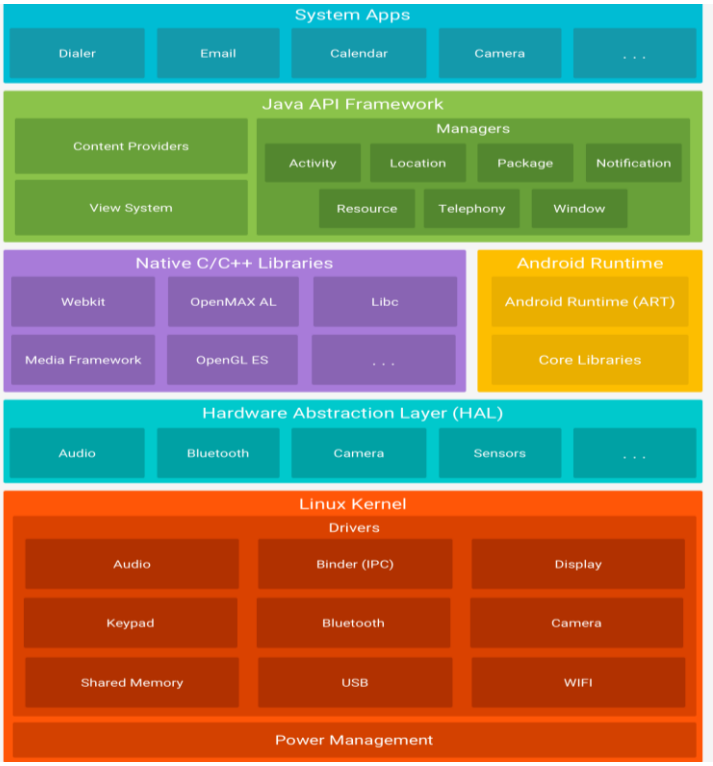
Slika 1: Prodani mobilni uređaji prema OS-u [3]

Android je, kako u okviru nastavnih kolegija, tako i za potrebe završnog rada, odabran kao globalno dominantna mobilna platforma (slika 1) s višegodišnjim udjelima iznad 80% u prodaji novih uređaja. Prema podacima tvrtka Statista, koja je jedna od vodećih svjetskih analitičara za gospodarstvo, Android je u 2018. koristilo oko 88% prodanih mobilnih uređaja! Kao što je vidljivo na slici 1 to je trend koji traje već duže vrijeme. Zbog toga se i tijekom obrazovanja izrađivalo mobilne aplikacije upravo za tu platformu.

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.3%
4.1.x	Jelly Bean	16	1.2%
4.2.x		17	1.5%
4.3		18	0.5%
4.4	KitKat	19	6.9%
5.0	Lollipop	21	3.0%
5.1		22	11.5%
6.0	Marshmallow	23	16.9%
7.0	Nougat	24	11.4%
7.1		25	7.8%
8.0	Oreo	26	12.9%
8.1		27	15.4%
9	Pie	28	10.4%

Slika 2: Popis verzija Android OS-a

Obzirom na stalni razvoj sustava, „problem“ koji se javlja (za programere) jest taj da istovremeno kod korisnika postoje uređaji koji koriste različite verzije sustava Android. Slika 2 prikazuje popis verzija Androida koje su još značajno zastupljene. Jasno, s vremenom se dobar dio podrške za starije verzije sustava ukida, no i dalje postoji mogućnost razvoja aplikacija i za stare verzije sustava. Već od prvih verzija Android ima podršku za lokacijske servise i funkcionalnost tih servisa je većinom ostala nepromijenjena. Danas najraširenije verzije su 8.0 i 8.1 Oreo, kao što je vidljivo na slici 2. Podaci o zastupljenosti verzija prikupljani su od 1. do 7. svibnja 2019. Metoda prikupljanja podataka se mijenjala kroz godine. Danas važeća metoda je praćenje spajanja na Google Play servis [4]. Aplikacija u ovom radu je razvijena za verziju 9.0 Pie, na kojoj je i testirana.



Slika 3: Arhitektura Android OS-a [5]

Android je baziran na jezgri Linux operativnog sustava (eng. *Linux Kernel*). Linux je OS otvorenog koda, besplatno je dostupan. Na slici 3 prikazana je arhitektura sustava. Android ima u svojoj bazi Linux jezgru koja sadrži pogonske programe raznih komponenti uređaja i upravljanje napajanjem. Iznad toga je HAL (eng. *Hardware Abstraction Layer*) koji sadrži knjižnice naredbi koje implementiraju standardizirana sučelja za pristup podsustavima uređaja. To omogućuje jednostavno i lagano upravljanje i korištenje tih sustava u višim slojevima Androida. U sloju iznad postoje dva sustava. Prvi je niz C i C++ knjižnica koje se koriste za povezivanje nižih i viših slojeva. Za pokretanje razvijanih programa neophodan je podsustav za izvršavanje aplikacija, AR (eng. *Android Runtime*) – to je podsustav koji pokreće svaku aplikaciju u zasebnom okruženju, znači da je svaka aplikacija u biti zasebno virtualno računalo koje radi u nekom svojem ograničenom okruženju. To je uobičajeni pristup s ciljem postizanja veće sigurnosti i pouzdanosti – smanjuje se ili eliminira međusobni utjecaj aplikacija, ograničava pristup resursima i slično. Do Android verzije 4.4. okruženje u kojem su se izvodile aplikacije bio je Dalvik VM (prilagođeni JVM s optimizacijama za mobilnu platformu), nakon toga je uveden AR [6]. Za potrebe rada nisu bitni detalji oko načina izvođenja, pa se to niti ne obrađuje. Najvažniji sloj za programere koji razvijaju mobilne aplikacije se odnosi na podsustav koji omogućava programski pristup sučeljima i specijaliziranim Java knjižnicama. Za potrebe razvoja, programerima je omogućen pristup svim dozvoljenim funkcionalnostima uređaja. Primjerice svi oblici manipulacije lokacijskim servisima su također tu. Uobičajeno, za pristup

resursima vezanim uz razne podsustave mobilnog uređaja, Android programerima nudi tzv. menadžere pomoću kojih se može programski dohvaćati ili manipulirati određenim resursima (dostupnim kroz pojam servisa). U ovoj aplikaciji koristi se *LocationManager* koji upravlja lokacijskim servisima. Zadnji vidljiv sloj je sloj sistemskih aplikacija. Primjer su aplikacija SMS poruka, kalendara, sata itd. Sistemske aplikacije nisu fiksne, što znači da ne mora nužno biti samo jedna SMS aplikacija, ili da je aplikacija isporučena s uređajem glavna. Sistemske aplikacije se mogu mijenjati. Koncept sustava i aplikacija omogućava da se unutra aplikacije pozivaju funkcionalnosti drugih aplikacija – primjerice, napiše se zasebna SMS aplikacija koja koristi instaliranu aplikaciju.

2.2. Programiranje za Android

Aplikacije za Android operacijski sustav se najčešće pišu u Java programskom jeziku. Osim Jave, službeni jezik za razvoj za Android je i Kotlin. Uz Javu se koristi i XML. XML je jezik koji služi za strukturirani zapis podataka. U Androidu se koristi za definiciju grafičkih elemenata, sučelja aplikacije i pohranu podataka. Osim Jave i XML-a, u razvoju aplikacije korišten je i SQL – jezik za rad s bazama podataka.

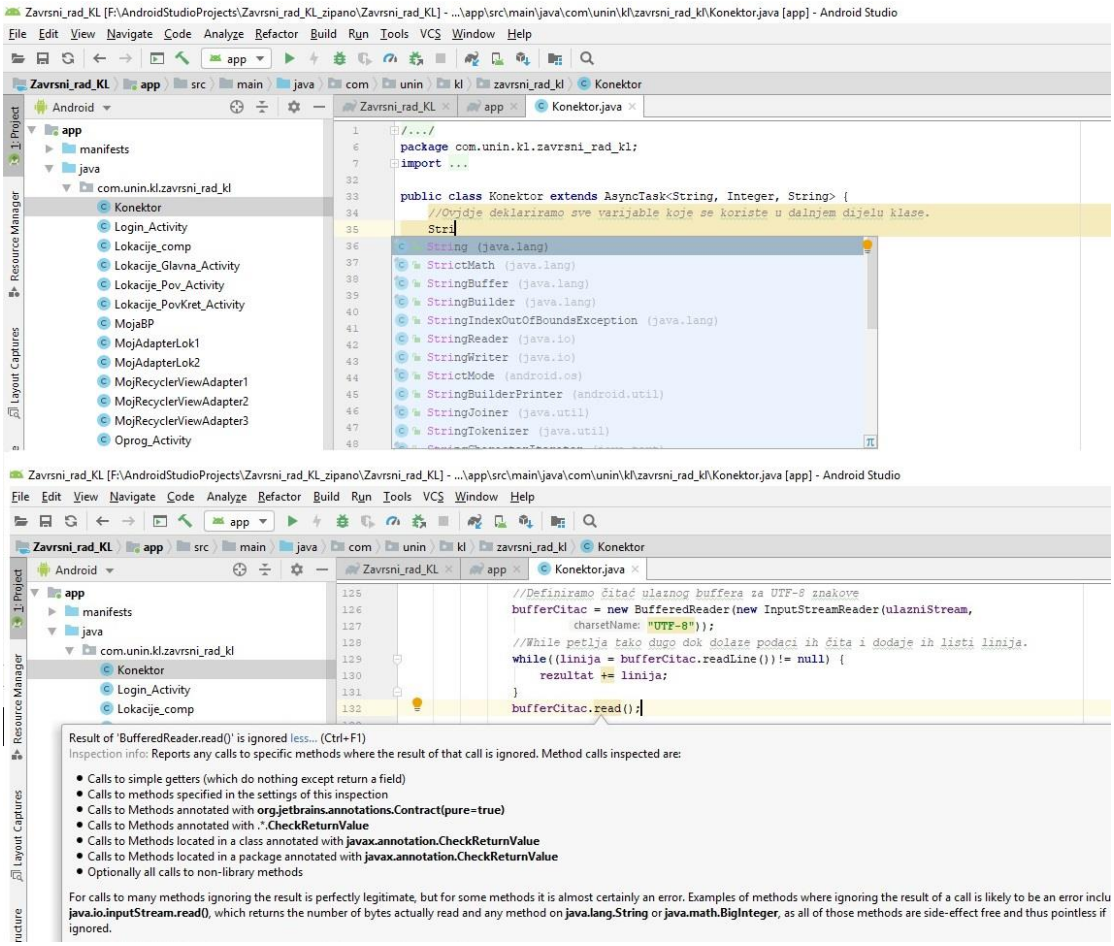
Preporučeno razvojno okruženje za Android platformu je Android Studio [7] – radi se o besplatnom softveru dostupnom za razne platforme, a podržava razvoj za sve tipove Android uređaja (pametni telefoni, tableti, pametni satovi, televizori itd.). To je razvojno okruženje koje Google nudi besplatno i služi za razvoj izvornih *(native)* aplikacija za različite Android uređaje. Ove aplikacije se koriste od pametnih satova do pametnih televizora koji dolaze s Android OS-om. Android Studio prepoznaje i ravnopravno omogućava programiranje u Javi i Kotlinu. Nudi sve potrebno za izradu aplikacija, nudi grafički editor sučelja, ugrađene prevoditelje i sve pomoćne alate potrebne za stvaranje Android aplikacije, emulator za pokretanje virtualnih uređaja i komunikacijsko sučelje za komunikaciju s uređajima (emuliranim ili stvarnim uređajima). Komunikacijsko sučelje omogućava automatsku instalaciju aplikacije na mobilnom uređaju i detekciju kvarova i grešaka.

Aplikacije za Android se mogu izrađivati i kao web aplikacije koje se samo prikazuju u web prozoru na mobilnom uređaju. To se često naziva hibridna aplikacija. Princip rada je da se napravi jedan veliki web pogled u koji se učita stranica koja je pisana HTML5 jezikom. Ona onda može raditi lokalno preko web preglednika, odnosno na način da web pogled učitava sve podatke direktno s poslužitelja. Hibridne aplikacije imaju pristup resursima uređaja pomoću HTML5 i JavaScript koda. Prednost takvih aplikacija je da se mogu relativno brzo i lako razviti

za mnoštvo različitih platformi. Često se podrška za neku novu platformu može implementirati prema gotovim predlošcima i uz minimalnu promjenu programskog koda.

2.2.1. Razvojno okruženje Android Studio

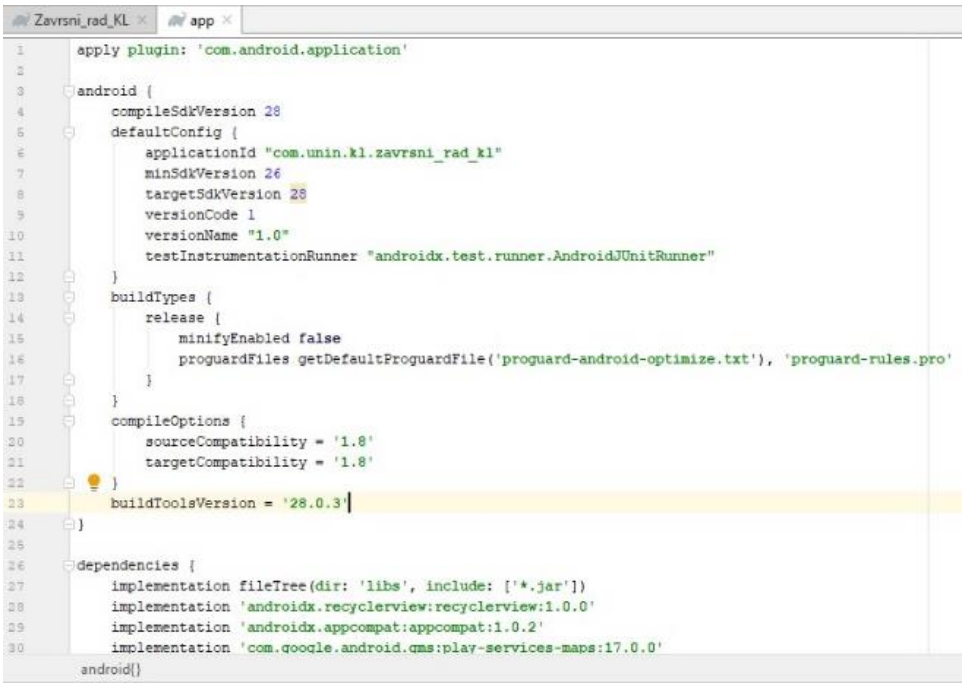
Android Studio (u daljnjem tekstu AS) je kompleksno razvojno okruženje za Android platformu. Tijekom školovanja korištene su samo osnovne mogućnosti – uređivač teksta (programskog koda i koda za opis sučelja i drugih elemenata), prevoditelj i ostali alati za stvaranje izvršne APK datoteke, emulatori i podsustav za pokretanje aplikacije na uređaju ili emulatoru, *debugger* za analizu izvođenja, te niz drugih pomoćnih alata (*LogCat*, *ResourceManager* i slično).



Slika 4: Sučelje razvojnog okruženja – struktura projekta,uređivač teksta i veza s uputama

Uređivač teksta je ključan za programere – u njemu se unosi i uređuje programski kod. Obzirom na kompleksnost jezika i pratećeg Android SDK, mogućnosti automatske provjere i dopunjavanje koda. Te veze s uputama koje opisuju detalje funkcija i načine pozivanja, više su nego dobrodošle i potrebne funkcionalnosti! Primjer izgleda sučelja razvojnog okruženja i uređivača teksta je dan na slici 4.

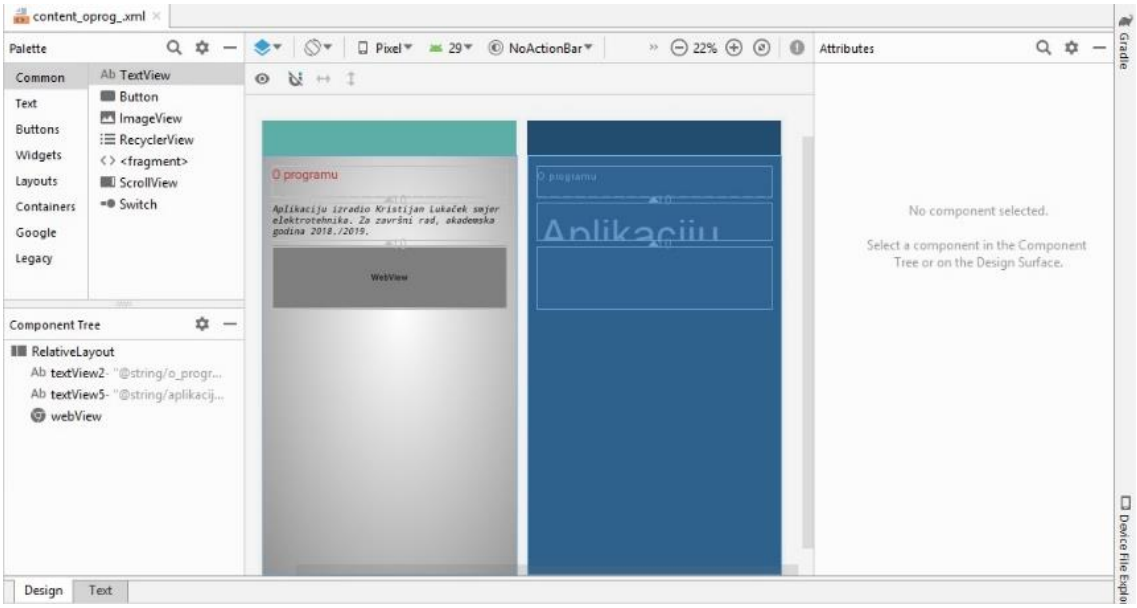
Prilikom pokretanja novog projekta aplikacije AS generira sve datoteke i direktorije koji će kasnije završiti u APK datoteci. Poput direktorija za privatnu pohranu podataka, direktorija za pohranu grafičkih elemenata aplikacije itd. Struktura projekta u AS se formira prema preporukama i vizualizira se unutar sučelja pomoću stabla. AS sadrži sve potrebne SDK-ove za izradu aplikacije, a po potrebi je omogućeno automatsko preuzimanje različitih verzija SDK/API. Općenito, SDK za neku verziju sustava sadrži sve Android API knjižnice, tehničku dokumentaciju itd. Da bi AS ispravno izradio aplikaciju, potrebno je prilikom stvaranja projekta definirati verziju SDK, minimalni podržani SDK i slično. To omogućava izradu aplikacija koje mogu raditi i na starijim verzijama sustava. Između verzija Androida može doći do značajnih promjena pa je potrebno posvetiti veliku pažnju kompatibilnosti aplikacije sa starijim verzijama sustava. Da bi se uspješno formirala aplikacija potrebna je upotreba Gradle sustava. To je napredni sustav upravljanja prevođenjem elemenata Android aplikacije koji na kraju stvara APK arhive koje sadrže tu aplikaciju. Gradle sustavom se upravlja u Gradle datotekama. Verzije SDK-a koji se koriste u aplikacije je moguće mijenjati kasnije u tekstualnom editoru AS-a promjenom postavki u nekolicini Gradle datoteka.



Slika 5: Gradle datoteka izrađene aplikacije

Važan dio Android studija su emulatori. Za svaku od verzija Android SDK dostupne su virtualne slike uređaja, pa je pomoću emulatora moguće pokretati različite verzije sustava. Uz svaki virtualni uređaj moguće je proizvoljno definirati određene parametre – memorija, mogućnosti kao navigacija, veličina ekran itd. Umjesto emulatora mogu se koristiti i mobilni uređaji spojeni na računalo žičano (USB) ili bežično (WiFi). Nakon uspješnog prevođenja i nakon pokretanja aplikacije, AS nudi izbor uređaja (emuliranih ili stvarnih, spojenih na računalo na kojem radimo)

za pokretanje aplikacije. Na odabrani uređaj instalira se aplikacija i započinje izvođenje. Ukoliko je tako podešeno, aktivira se i praćenje izvođenja i otkrivanje pogrešaka (debugger). Osim pomoću AS, moguće je i iz naredbenog retka (terminala) raditi s uređajima, korištenjem ADB (eng. *Android Debug Bridge*) alata. No, u novijim verzijama AS-a većina funkcija je integrirana i programerima dostupna kroz korisničko sučelje. AS uključuje i grafički editor u kojem se izrađuju grafička sučelja aplikacije. Obzirom da se grafičko sučelje strukturirano zapisuje u XML datoteke, grafički uređivač sučelja omogućava prebacivanje i rad u vizualnom režimu, ali i uz pregled i rad s XML datotekama.



Slika 6: Grafički editor sučelja

2.3. Podsustav za lokacijske servise

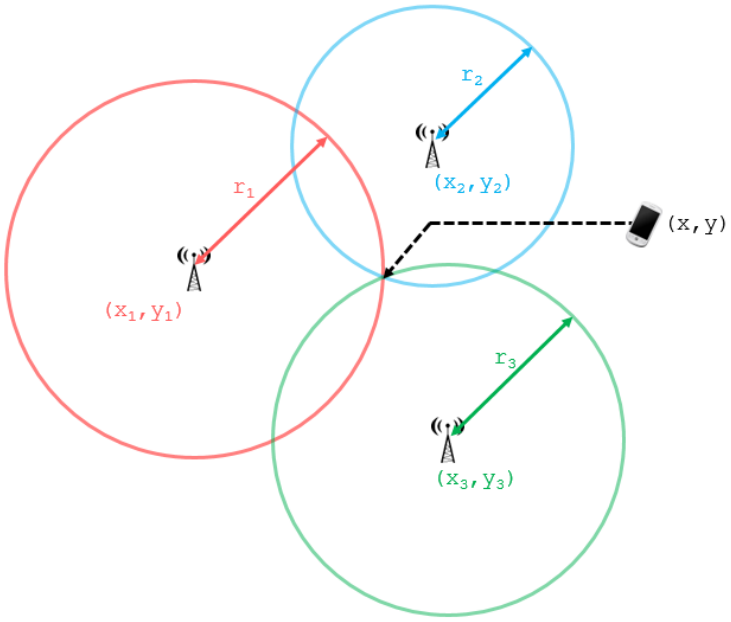
Android koristi više načina dohvaćanja lokacije uređaja. Uobičajeno je za tu namjernu koristiti sustav satelitske navigacije. Najpoznatiji takav sustav je američki GPS, ali postoje i u nekim uređajima su podržani još i ruski GLONASS, kineski DeiDou, europski Galileo, indijski IRNSS. U najavama je i japanski QZSS, s previđenim početkom primjene oko 2023. Uz GPS sustave, o kojima će više riječi biti u sljedećem odjeljku, Android podržava i mogućnost lociranja pomoću lokalnih WIFI mreža, mobilne mreže i određenih senzora ugrađenih u neke uređaje. Principi rada sustava satelitske navigacije opisani su u sljedećim odjeljcima, a ovdje se ukratko daje nekoliko informacija o alternativnim oblicima lociranja i navigacije.

Prilikom korištenja WIFI bežičnih mreža za locirane uobičajena su tri pristupa:

- domet mreže,
- RSSI (eng. *Received Signal Strength Indicator*) pristup,

- RTT (eng. *Round Trip Time*) pristup.

Najjednostavnije i najmanje precizno lociranje je prema dometu mreže – ukoliko je uređaj u dometu neke poznate pristupne točke pretpostavlja se da se nalazi na nekoj udaljenost od nje. Recimo da je domet WIFI mreže vrlo kratak, npr. 50 m i da je pristupna točka poznate lokacije. Može odrediti da ako uređaj vidi pristupnu točku da se nalazi na određenoj udaljenosti od nje. Ovakav način lociranja može se koristiti u gradu jer je lociranje precizno na jednu zgradu, kuću ili dio neke ulice. Drugi pristup lociranja pomoću WIFI sustava je RSSI - ovdje se mjeri jačina signala koji dođe do uređaja. Odašiljač šalje signal maksimalne vrijednosti primjerice 127 i taj signal postaje sve slabiji pa se pretvorbom može smanjiti do 0. Maksimalna snaga signala nije definirana nekom mjernom jedinicom nego je proizvoljno određena od strane proizvođača mrežne opreme. Ovdje je jedinica snage proporcionalna nekoj jedinici udaljenosti. Različiti proizvođači imaju različite raspone snaga, na primjer Atheros ima raspon snage od 0 do 127, a CISCO od 0 do 100 [8]. Da bi se dobila lokacija potrebno je biti u dometu više mrežnih pristupnih točaka. Tada se može trilaterirati lokacija uređaja. Slika 7 prikazuje trilateraciju mobilnog uređaja pomoću WIFI signala ili signala mobilne mreže. Multilateracija će biti detaljnije opisana u sljedećem poglavlju. Taj pristup se primarno koristi za lociranje u zatvorenom prostoru i nije baš pouzdan. Na preciznost se može utjecati zidovima i drugim preprekama koje umjetno slabe signal.



Slika 7: Primjer trilateracije mobilne mreže [9]

Treći pristup lociranju pomoću WIFI-ja je WIFI RTT. Kod ovog sustava se direktno mjeri vrijeme potrebno da signal dođe od pristupne točke do uređaja i obratno. Iz tog vremena se

dobije udaljenost. Kao i kod prijašnje RSSI metode, koristi se primarno za navigaciju u zatvorenom. Treba imati više pristupnih točaka i najnoviji je sustav. Precizniji je od RSSI sustava pa ga od 2019. podržava Google kroz API dodan u Android. Da bi radio pristupna točka i mobilni uređaj moraju podržavati IEEE 802.11mc standard. Podrška na Android platformi je dostupna od verzije Android 9 naviše [10].

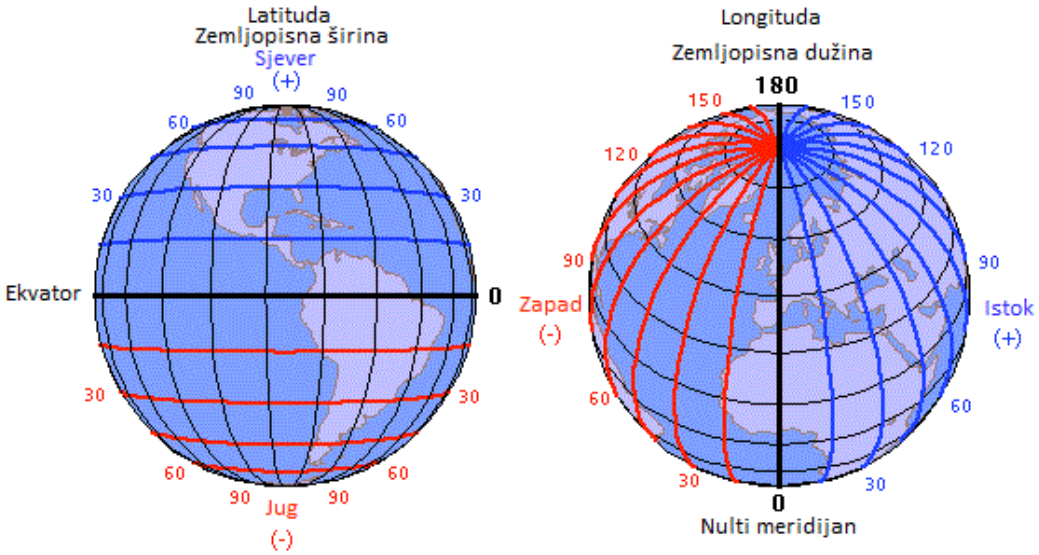
Jedan od načina određivanja lokacije je i lociranje pomoću mobilnih mreža. S tim mrežama se može locirati na većim područjima nego s WIFI mrežama ali je preciznost drastično manja. Preciznost se često kreće u nekoliko stotina metara ili više. Radi na principima kao i prethodni sustavi bazirani na WIFI mrežama. Može odrediti lokaciju ako se uređaj nalazi u dometu dvije ili više mobilne bazne stanice pa je područje gdje se preklapaju signali lokacija uređaja. Može dati i lokaciju samo na bazi dometa jedne bazne stanice ali je to daleko od preciznog lociranja, pogotovo u ruralnim područjima gdje zbog rijetko postavljenih baznih stanica preciznost pada na razinu nekoliko kilometara. Općenito se ova metoda izbjegava. Pozitivna strana dohvaćanja lokacije iz mreža je da puno manje troši bateriju uređaja nego satelitska navigacija, jer uređaj ionako stalno dohvaća informaciju o mobilnoj baznoj stanici, za razliku od dodatne potrošnje energije zbog aktivacije GPS podsustava. Također, pozitivno je da je dostupno i u zatvorenom prostoru i na lokacijama na kojima zbog nekih specifičnosti terena satelitska navigacija ne bi radila dobro (naravno, ne zaboraviti na nižu preciznost). Da bi se mogla dohvatiti lokacija pomoću mreža mora se imati lokacije pojedinih odašiljača. Te lokacije mogu biti dostupne na Internetu ili ih se može pohraniti lokalno na uređaju i koristiti bez pristupa mreži.

Malo egzotičnije metode lociranja koje nemaju nikakvih API-jeva dodanih od strane Google-a su metode lociranja pomoću senzora. Dva primarna pristupa ovdje su pomoću magnetometra i senzorima akcelerometara i žiroskopa. Magnetometar je senzor koji mjeri jačinu i orijentaciju magnetskog polja. U ovom slučaju može mjeriti prirodno magnetsko polje zemlje. Zemlja se može mapirati pomoću prirodnih magnetskih polja pa bi teoretski mogli prema orijentaciji i snazi polja ugrubo odrediti lokaciju uređaja. Ovaj sustav je neisproban, eksperimentalne prirode i nije precizan. Akcelerometar mjeri ubrzanje u jednoj od prostornih osi, a žiroskop mjeri orijentaciju u prostoru u odnosu na neki horizont. Iz akcelerometra je moguće dobiti podatak o orijentaciji jer on može svojim izmjerama utvrditi promjenu orijentacije. Kod starijih ili jeftinijih uređaja akcelerometri služe kao i žiroskopi pošto su jeftiniji ali su i manje precizni od žiroskopa. Noviji uređaji svi imaju odvojeni žiroskop za određivanje orijentacije. Pomoću tih senzora se može implementirati sustav inercijalne navigacije. To je sustav kod kojeg se kreće od poznatog prvog položaja pa se prate pomaci i orijentacija i izračunava se koordinata svake sljedeće lokacije. Problem je da se prilikom svakog izračuna formira određena greška. Ta greška s vremenom raste

do mjere da takvi sustavi postanu neupotrebljivi. Također se kao i prethodni ne upotrebljavaju, ali kad je riječ o svim mogućnostima lociranja na mobilnim uređajima, moraju se spomenuti.

2.3.1. Osnovni princip GPS-a

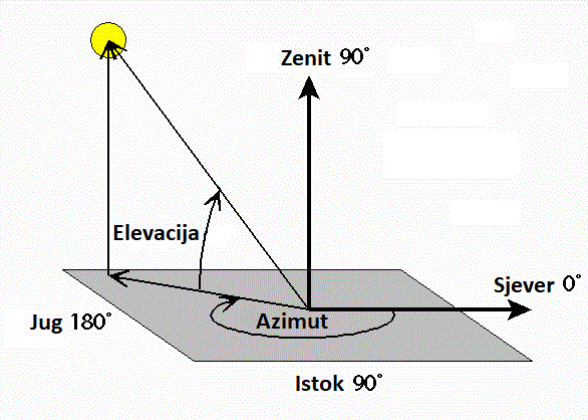
Da bi se shvatio GPS sustav prvo je potrebno objasniti pojmove zemljopisna širina, dužina i primjerice azimut i elevacija. Zemljopisna širina ili latituda je kutna udaljenost neke točke na Zemlji prema sjeveru ili jugu od ekvatora. Mjeri se u stupnjevima u rasponu od 0° do 90°. Zemljopisna dužina ili longituda je kutna udaljenost neke točke prema istoku ili zapadu paralelna s ekvatorom. Mjeri se u stupnjevima u rasponu od 0°do 180°. Obje vrijednosti su u stvari kut koji zatvara spojnica od točke do središta zemlje i ravnina koja je omeđena ekvatorom u slučaju latitude ili nultim meridijanom u slučaju longitude. Prilikom mjerenja se mora obratiti pažnja o smjeru mjerenja u odnosu na ekvator ili nulti meridijan. Sve latitude sjeverno od ekvatora imaju predznak N (eng. *North*) ili S sjever, a sve južno od ekvatora S (eng. *South*) ili J južno. Isto vrijedi za longitudu. Izmjere istočno od nultog meridijan imaju predznak E (eng. *East*) ili I istok, a za mjerenja prema zapadu W (eng. *West*) odnosno Z zapad. Kod Androida i u nekim drugim slučajevima se ne dodaju te oznake nego se stavljaju predznaci na broj koordinata. Na temelju navedenog sustava označavanja su sve sjeverne latitude +, a južne -, odnosno sve istočne longitude su +, a zapadne -. Primjer vizualizacije zemljopisne širine i dužine je dan na slici 8.



Slika 8: Vizualizacija geografske širine i dužine, koncept koordinatnog sustava [11]

Azimut je kut između sjevera i pravca kretanja ili promatranja. Elevacija odnosno visina je kut između horizonta i nekog objekta iznad njega. Horizont može biti prirodni, prividni i pravi. U kontekstu koordinatnih sustava se koristi pravi horizont, to je kružnica koja nastaje presjekom nebeske sfere i ravnine koja je paralelna s površinom Zemlje i koja prolazi kroz promatrača, a

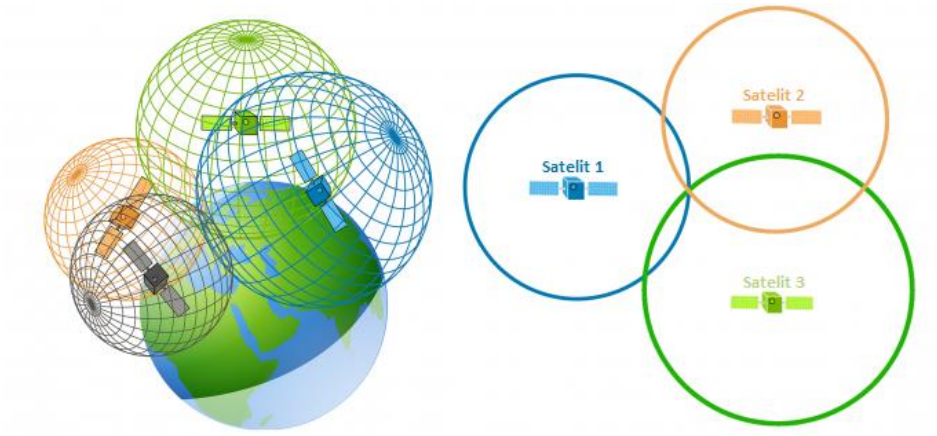
okomita je na vertikalu u njegovu stajalištu. U kontekstu GPS sustava se pojmovi elevacije, azimuta i horizonta koriste za opisivanje kretanja i orijentacije uređaja s GPS prijemnikom te položaja GPS satelita. Primjeri su vidljivi na slici 9.



Slika 9: Primjer azimuta i elevacije [12]

GPS je satelitski navigacijski sustav koji je u vlasništvu oružanih snaga SAD-a, a administrator sustava je Američko ratno zrakoplovstvo. Sustav se sastoji od 31 aktivna satelita u orbiti i zemaljskih kontrolnih postaja. Sateliti se kreću u gotovo savršeno kružnim orbitama na visini od oko 20200 km. Nalaze se u 6 orbitalnih ploha razmaknutih za 60°. Plohe s 24 satelita plus dodatnih 7 satelita osiguravaju da je s bilo kojeg položaja na svijetu vidljivo barem 4 satelita [13].

GPS radi na principu multilateracije. Multilateracija je prikazana na slici 10.

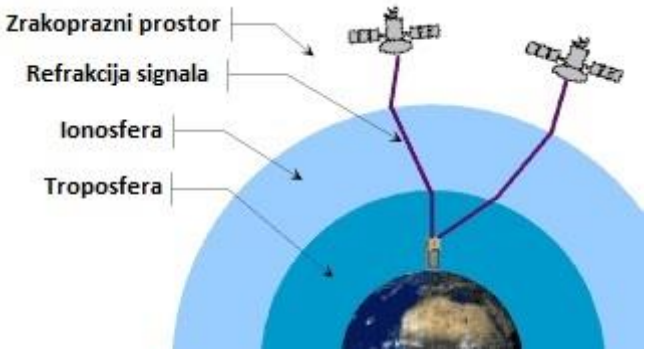


Slika 10: Primjer multilateracije kod GPS sustava [14]

Multilateracija je bazirana na mjerenju udaljenosti. Udaljenost se dobije mjerenjem razlike vremena slanja i prijema. Tako da je za mjerenje udaljenosti ključno utvrđivanje vrlo preciznog vremena slanja i primanja. Udaljenost se dobije kada se to izmjereno vrijeme pomnoži s brzinom

kretanja elektromagnetskih valova. Ta brzina je otprilike ista kao i brzina svjetlosti i iznosi 299.792.458 m/s. Nakon toga se mora utvrditi lokacija odašiljača. Da bi se to napravilo GPS sustav periodički šalje podatke o orbiti satelita. Pošto se orbite mijenjaju rijetko i samo vrlo malo, predvidive su pa se iz tih podataka lako izračuna točna lokacija satelita u vrijeme slanja. Iz tih podataka se može izračunati udaljenost prijemnika od satelita. Lokacija satelita je središte a udaljenost je polumjer zamišljene kugle. Da bi se odredila točna lokacija potrebno je imati 4 satelita. Tih 4 satelita daju 4 kugle koje se onda sve sijeku odnosno dodiruju u jednoj točki. Ta točka je lokacija GPS prijemnika. Da bi se odredile samo zemljopisna dužina i širina dovoljno je imati 3 satelita budući da se određuje lokacija u samo 2 koordinate. To je isto vidljivo na slici 7 te se još naziva trilateracija [14][15].

Uz lociranje kod sustava satelitske navigacije moraju se spomenuti i greške koje se javljaju. Javljaju se greške sata satelita i prijemnika. Satovi mogu akumulirati grešku pa ta greška onda utječe na mjerenje trajanja putovanja signala. Budući da se udaljenost dobiva iz vremena putovanja i fiksne brzine, kod određivanja orbita satelita može doći do greške. Položaj se uz udaljenost određuje položajem odašiljača pa ako je položaj pogrešan za isti iznos greške će biti i lokacija pogrešna.



Slika 11: Greške GPS-a u atmosferi [16]

Jedan od izvora greške može biti i zemljina atmosfera. Kao što je vidljivo na slici 11 kada bilo koji elektromagnetski val putuju kroz različite medije dolazi do refrakcije vala kada prelazi iz jednog medija gibanja u drugi. Taj val se onda, kako je prikazano na slici 11, ne giba ravno između dvije točke. Zbog toga je vrijeme potrebno da bi došao do odašiljača drugačije, što daje grešku udaljenosti i time lokacije. Uz to postoji i greška odbijanja signala i time primanja duplih signala i greška lošeg rasporeda satelita gdje se radi lošeg geometrijskog rasporeda iskrivi područje lokacije prijemnika.

Navedene greške se korigiraju zemaljskim ili svemirskim sustavima. Ti sustavi imaju prijamnike postavljene na vrlo precizno izmjerene lokacije. Onda se proračunava iznos korekcije za koju se

lokacijski signal korigira i to se šalje. Time se omogućuje regionalno korigiranje preciznosti GPS sustava na vrlo visoke razine. Većina država ima neki nacionalni sustava korekcije ili kako se još zove augmentacije. Primjer svemirskog sustava augmentacije je Japanski QZSS koji je počeo kao sustav augmentacije, gdje su prijemnici za umjeravanje sami sateliti, no oni će se preinačiti u samostalni satelitski navigacijski sustav [16].

2.3.2. Programsko očitanje lokacije

Android razvojnim programerima daje i API koji omogućuje rad s lokacijskim servisima (ukoliko uređaj to omogućava, naravno). Kao što je prethodno spomenuto Android ne nudi sve oblike navigacije. Ono što Android nudi je satelitska navigacija i navigacija putem mreža. Kod Android uređaja se koristiti *LocationProvider* (u daljnjem tekstu LP). U izvedenoj aplikaciji se koristi LP koji dohvaća lokacije pomoću satelitskih sustava navigacije. Kao i za druge podsustave, da bi se moglo koristiti lokacijske servise unutar aplikacije, aplikacija od korisnika uređaja mora zatražiti dozvolu za korištenje – prilikom implementacije to se definira u tzv. manifestu aplikacije.

Dohvaćanje, osim što energetski iscrpljuje uređaj ima i određene elemente privatnosti (korisnika uređaja), pa je za to potrebna posebna dozvola kako ne bi došlo do zloupotrebe te mogućnosti od neke zlonamjerne aplikacije. Dozvole za pristup lokacijama su `ACCESS_FINE_LOACTIOIN`, kojom tražimo dozvolu za pristup preciznim lokacijama (sustavi satelitske navigacije) i `ACCESS_COARSE_LOCATION`, kojom traži dozvolu za pristup gruboj lokaciji (mreže). Uz ove dozvole za pristup mrežnoj lokaciji kod WIFI RTT metode su potrebne dozvole `ACCESS_WIFI_STATE` i `CHANGE_WIFI_STATE`. Nakon toga da bi dohvatili lokaciju koristimo *LocationManager* (u daljnjem tekstu LM) koji služi za upravljanje lokacijskim servisima – definira se ponuđač/izvor lokacije (GPS ili mreža), a potom programski dohvaća informacija o trenutnoj lokaciji uređaja. Nakon podešavanja, LM automatski dostavlja definiranu vrstu lokacijske informacije aplikaciji preko predefiniranog programskog događaja (eng. *Event*). Kod novog WIFI RTT sustava je potrebno lokaciju dohvatiti ručno i procedura nije toliko automatizirana.

Da bi se izbjegli problemi sa zastarjelim lokacijama i uvijek unutar aplikacije moglo dohvatiti novu lokaciju mora se implementirati programska struktura za osluškivanje, u ovom slučaju *LocationListener* (u daljnjem tekstu LL). LL omogućava da se unutar aplikacije prilikom svake promjene pokrene funkcija `onLocationChanged` (slika 12) u kojoj se implementira programska logika odnosno operacije koje se žele izvršiti nad primljenim podacima.

```
1   LocationListener ll = new LocationListener() {
2   public void onLocationChanged(Location loc) {
3   lat = Double.valueOf(loc.getLatitude());
4   lon = Double.valueOf(loc.getLongitude());
5   vis = Double.valueOf(loc.getAltitude());}
```

Slika 12: Primjer onLocationChanged funkcije u Javi

Da bi LL mogao čekati promjenu lokacije prilikom gibanja korisnika moguće je definirati proizvoljne uvjete koji će dovesti do registracije promjene lokacije kod uređaja. Ti uvjeti su LP, minimalno vrijeme čekanja na promjenu, minimalna udaljenost pomaka i LL. U izvedenoj aplikaciji je LP satelitski sustavi navigacije. Minimalno vrijeme čekanja na promjenu je definirano u milisekundama, a minimalna udaljenost u metrima. Pošto je programski moguće implementirati više LL, moguće je definirati i više funkcija za obradu primljenih lokacijskih podataka. U optimiziranom okruženju, preporuke su da se s ciljem uštede energije prekine čekanje i praćenje lokacije ukoliko je to moguće [17].

Android API uključuje i niz pomoćnih funkcionalnosti vezanih uz lokaciju – primjerice, moguće je dobivati alarme o blizini neke lokacije. Alarmi se pozivaju preko LM i moraju uz namjeru imati definiranu lokaciju preko „kritične točke“ (koordinate, tj. zemljopisnu dužinu i širinu) te radijus upozorenje o blizini (eng. *Proximity Alert*). To upozorenje (alarm) stvara kada se uređaj približi nekoj lokaciji na neku predefiniranu udaljenost. Ti alarmi najčešće pokreću takozvane namjere (eng. *Intent*, specijalni Android objekti koji signaliziraju da postoji namjera obavljanja određene zadaće, u pravilu korištenjem sistemskih modula). Uz namjeru treba definirati i lokaciju alarma i udaljenost oko lokacije u metrima, unutar koje se aktivira „okidač“ i pokreće namjera.

Također jedna od mogućnosti lokacijskih servisa kod Android operacijskog sustava je geokoder. Geokoder omogućava pretvaranje neke koordinate u uličnu adresu ako ona postoji za te koordinate u nekoj bazi podataka ili obrnuto. U izvedenoj aplikaciji koriste se i funkcionalnosti geokodera Uz Google geokoder, koji se uobičajeno koristi, postoje i alternative – QGIS Geocoding Plugins, ESRI Geocoding, HERE Maps Geocoding, PBBI Geocoding, US Census Geocoder, Bing Loaction API [18][19].

API vezan uz lokacijske servise uključuje elementarne funkcije za rad s lokacijama – na primjer, mogućnost uspoređivanja odnosa između dvije lokacijske pozicije. Te dodatne mogućnosti su uključene u razvijanu aplikaciju – svode se na kratki programski kod ili još češće pozivanje jedne jedine funkcije s adekvatnim parametrima (na primjer određivanje udaljenosti između dvije lokacije, definiranje alarma kod približavanja lokaciji i prosljeđivanje u sustav na obradu).

2.4. Vizualizacija podataka o lokaciji i Google karte

Kao što je u prethodnom odjeljku *2.3.* opisano, Android omogućava lako programsko očitanje trenutne lokacije uređaja. No, te brojke većini ljudi neće biti pretjerano korisne, pa je uobičajeni način prikaza lokacije vizualizacija na geografskoj karti. Te karte su grafički elementi koji se lako mogu uključiti u aplikaciju. Karte koje se koriste u ovoj aplikaciju su najraširenija jednostavna opcija – Google karte (eng. *Google Maps*, u daljnjem tekstu GM). Te karte se učitavaju u samo sučelje aplikacije direktno s Google-ovih poslužitelja, kad za to postoji potreba. Da bi se mogao koristiti prikaz karte treba imati vezu na Internet. Dodatno treba postojati dozvola u aplikaciji da bi se mogla spojiti na Internet. Uz dozvolu za pristup aplikacije Internetu u manifestu se definira i takozvani API ključ za karte – to je jedinstveni identifikator koji se generira kod ponuđača karata (Google) i dozvoljava aplikaciji siguran i verificiran pristup Google servisima (karte i/ili ostali servisi). Isti API ključ je potreban za dodavanje polja u koja se učitavaju reklame ili objavu aplikacija na Google Play servisu. U aplikaciji se ključ karti dodjeljuje u manifestu ali sam ključ je definiran u zasebnom XML dokumentu. Taj dokument je generiran automatski kada se u editoru dodaje karta. Obzirom da se karte dohvaćaju s poslužitelja, neće uvijek biti dostupne – programski se status dohvata podataka o karti realizira unutar funkcije `onMapReady`. Funkciju (događaj) poziva Android kad detektira da su dohvaćeni potrebi podaci – programski je moguće definirati parametre prikaza nakon tog događaja. Tu funkciju sustav zove sam automatski kad detektira da su dohvaćeni podaci potrebni za kartu. Onda se dalje definira tip karte koju prikazuje, neke početne postavke te karte i grafičke elemente.

Slika 13: Prikaz osnovne Google karte

Postoje različiti tipovi karata (vizualne interpretacije) i oni mogu varirati od ponuđača – GM omogućava korištenje više tipova karata. Tipovi koji se koriste u ovoj aplikaciji su klasična karta i hibridna karta - prikazuje satelitsku sliku s elementima klasične karte (ceste s nazivom ulica, granice jedinica lokalne samouprave, granice regija i država). Važno je naglasiti da sve karte nisu iste što se tiče utroška resursa mobilnog uređaja. Primjerice satelitske ili hibridne karte su zahtjevne za grafički procesor pogotovo starijih uređaja ali ono gdje su najzahtjevnije je količina podataka koju je potrebno dohvatiti (mrežni promet) prije nego se karte prikažu. Satelitske karte sadrže slike koje trebaju više podataka nego obične karte gdje se koriste jednostavne plohe jednostavnih boja. Kartama se može baratati dodirom znači može ih se pregledavati pomicati pogled ili približavati pogled. Ta opcija može biti isključena odnosno ograničena ili modificirana. Bez dodatnog programiranja, moguća je promjena položaja pogled na karti, rotacija, povećavanje ili smanjivanje (približavanje ili udaljavanje) i slično. Ako korisnik želi može implementirati relativno jednostavno animacije i duge grafičke efekte na karti. Druga bitna i korisna mogućnost je dodavanje grafičkih elemenata na kartu. U radu su korišteni tzv. markeri lokacija i polilinije (eng. *Polyline*). Ti markeri su postavljeni na koordinate primjerice spremljene koordinate ili na trenutne koordinate koje su dohvaćene pa prikazane na karti. Uz unaprijed dane markere moguće je stvoriti posebne markere i grafičke elemente. Na svaki marker se može kliknuti. Klikom na marker se može izvršavati neke naredbe i obrada. Polilinijama se mogu povezivati koordinate u aplikaciji. Kod vizualizacije linija moguće je

podesiti određene postavke za prikaz, na primjer boju i debljinu linije. Uobičajeno je korištenje polilinija za iscrtavane ruta – linije se vuku od jedne do druge lokacije, kronološkim redoslijedom kako su evidentirane [20].

2.5. Pohrana podataka na uređaju

Aplikacija može pohranjivati podatke interno i eksterno. Uz rad s datotekama lociranim unutar datotečnog sustava uređaja Android nudi mogućnost spremanja u tzv. *SharedPreferences* i u bazu podataka. Interna pohrana se odrađuje u memoriji samog uređaja. Eksterna memorija najčešće podrazumijeva pohranu u vanjsku memoriju uređaja. Ta vanjska memorija je u kontekstu Androida logički pojam. Znači nije strogo memorija koja se nalazi izvan uređaja i nije trajno u njemu. Eksterna memorija je memorija kojom svaka aplikacija i korisnik mogu manipulirati i može biti prikazana dijeljenjem na drugom računalu. Uz nju postoje i klasični oblici vanjskih memorijski uređaja koji se mogu spojiti na mobilni uređaj. U pravilu su SD kartice najčešći oblik takve memorije ali se danas javlja mogućnost pohrane podataka na USB stickove pomoću USBOTG standarda. Interna pohrana je stalna i ne može se ukloniti s uređaja dok se neki oblici eksterne mogu po želji uklanjati. Treba obratiti pažnju u koju memoriju se snimaju podaci jer vanjske memorije mogu biti nedostupne aplikaciji, zbog čega može doći do neželjenih situacija za aplikaciju. Svaka aplikacija prilikom instalacije stvori direktorij za internu pohranu u koji se onda spremaju datoteke koje aplikacija stvara. Direktoriji interne memorije su nedostupni drugim aplikacijama i njih može koristiti samo aplikacija koja ih je stvorila, također nisu dostupni ni korisniku. Može se reći da su privatni aplikaciji. Aplikacija može spremati i u javne direktorije primjerice na eksternoj memoriji. Tamo se sprema kada se želi učiniti podatke dostupne drugim aplikacijama ili korisniku. Kada korisnik odluči deinstalirati aplikaciju se s njom brišu svi automatski generirani direktoriji. Znači da se sva interna memorija aplikacije briše s njom. Pa podaci postaju nedostupni. To je neželjeno kada primjerice aplikacija izrađuje slike koje bi korisnik htio zadržati i nakon brisanja aplikacije. Kod takvih slučajeva se te datoteke moraju snimati u eksternu memoriju. Upotreba eksterne memorije se preporučuje i kod instalacije vrlo velikih aplikacija. Za pristup vanjskoj memoriji Android sustava je potrebno dati dozvole za čitanje vanjske memorije `READ_EXTERNAL_STORAGE` i za pisanje u vanjsku memoriju `WRITE_EXTERNAL_STORAGE`. Neki od resursa aplikacija su pohranjeni u internim direktorijima koji nose naslov res, resources ili nešto slično tome, makar moguće je da i sam korisnik definira svoj direktori za takvu svrhu. Ti direktoriji se stvaraju u privatnoj (internoj) memoriji aplikacije prilikom instalacije. Android API uključuje niz funkcija za manipulaciju s datotekama. Dio ih se oslanja na standardni Java File tip podataka, dok je dio realiziran drugačije. Posebno je potrebno obratiti pažnju na funkcije vezane uz rad s direktorijima –

aplikacija mora dohvaćati osnovne informacije o putanji do svog privatnog direktorija u internoj memoriji, ali i o putanji do eksterne memorije. Dodatno, postoje problemi s ovlastima pristupu eksternoj memoriji, što ovisi o verziji operacijskog sustava, ali i internoj arhitekturi uređaja i organizaciji memorije za pohranu [21]. U budućim verzijama Androida će se koristiti ograničena pohrana podataka i očekuju se nove promjene u načinu rada s datotekama i memorijom za pohranu. Najavljuje se da će svaka aplikacija uz privatni direktorij dobiti i javni te će se potpuno ograničiti upotreba javne memorije sustava. Aplikacije koje neophodno moraju imati pristup drugim direktorijima će moći pristupiti, ali novim načinima pristupa koji su dati u zasebnom Android API-u koji je već dostupan [22]. To će zahtijevati često velike promjene kod mnogih postojećih aplikacija [23][24].

2.5.1. Jednostavna pohrana u SharedPreferences

SharedPreferences su posebni oblik pohrane primjenjiv u situacijama jednostavne pohrane para ključ-podatak (eng. *Key-value*). Omogućeno je definiranje nekoliko osnovnih tipova podataka – `boolean, string, float i integer` formata. Ovaj oblik jednostavne pohrane se najčešće koristi za internu pohranu postavki i korisničkih preferencija za aplikaciju. Primjer tih podataka su postavke koje utječu na rad aplikacije, različiti podaci koji se koriste u odvojenim aktivnostima aplikacije i tome slično [25]. Primjer takve pohrane je dat u odjeljku 3.2.4.

2.5.2. Pohrana u jednostavnoj integriranoj bazi podataka

Za pohranu veće količine strukturiranih podataka često se koristi pohrana u baze podataka. SQLite je jednostavna relacijska baza podataka koja je od početaka integrirana u sustav Android. Postoji niz drugih rješenja baza podataka, ali je SQLite vjerojatno najzastupljeniji upravo zbog svoje integriranosti u sustav. Kao i klasične relacijske baze podataka, SQLite podržava pohranu podataka u tablice i definiranje odnosa (relacija) među pohranjenim podacima. SQLite sustav upravljanja bazom podataka vrlo je popularan kod mobilnih uređaja. Popularan je zbog pouzdanosti i zauzimanja malo memorije, a ne zahtijeva ni mnogo procesorskih resursa, što znači da previše ne opterećuje platformu. SQLite ima mogućnost unosa svih tipova podataka u svaki stupac baze podataka. Kod klasičnih baza podataka, u stupac koji je deklariran kao integer stupac, nije moguće unijeti ništa drugo nego cjelobrojni podatak. U SQLite je moguće unijeti bilo koji podatak jer ih aplikacija sprema kao stringove. SQLite sustav upravljanja bazom podataka daje mogućnost da nekolicina aplikacija bude istovremeno spojena na bazu podataka i čita podatke, ili ih upisuje.

Upravljanje bazom podataka se u aplikacije omogućuje dodavanjem knjižnice naredbi vezane uz SQLite. SQLite je privatna baza podataka, što znači da se stvara baza podataka zasebno za svaku aplikaciju koja koristi SQLite i nije dostupna drugim aplikacijama. Baza podataka se pohranjuje

interno na uređaju, ali ju je moguće prebaciti na računalo ili joj pristupiti preko ADB naredbenog retka.

Da bi se omogućio pristup bazama podatka drugih aplikacija mora se omogućiti eng. *Content Provider*, oni mogu učiniti dostupne podatke iz jedne aplikacije spremivši podatke u generički mehanizam ponude podataka drugim aplikacijama. Pristupa im se preko `URI` adresa. Većina baza podataka koja dolaze s aplikacijama uređaja kao što je telefonski imenik su dostupne preko pružatelja sadržaja. `ContentValues` su objekti koje se koristi kod unošenja podataka u bazu podataka. Uz to ima i pokazivače, to su objekti koji se dobiju kada se izvrši neki SQL upit. Pokazivači kao zasebna polja sadrže pojedini red vraćenog upita. Dodatno mogu pristupiti pojedinim stupcima tog reda i na taj način izvlače podatke koje se vratilo upitom iz baze podataka te ih onda kasnije koristi ili dodjeli nekoj varijabli. Kada se barata bazama podataka ponekad treba provjeravati da li postoje baze podataka, da li ih treba ažurirati, isprazniti ili izbrisati.

`SQLiteOpenHelper` je apstraktni razred koji pomaže upravo u tome. Apstraktni razredi su oni razredi koje ne služe stvaranju objekata, nego samo kao osnova za stvaranje podrazreda, a služe samo da bi se izrazile zajednička svojstva svih podrazreda. `SQLiteOpenHelper` sadrži najbolje i najučinkovitije metode koje vrše upravljanje bazom podataka recimo pri pokretanju aplikacije. Tako da `SQLiteOpenHelper` onda jednostavno sa samo linijom dvije naredbi stvori, osvježi ili otvori bazu podataka.

Popis najčešće korištenih naredbi `SQLiteOpenHelper` razreda:

- `.insert()` - stavlja podatke u bazu podataka,
- `.delete()` - briše podatke u bazi podataka,
- `.update()` - naredba se koriti za osvježavanje odnosno promjenu podataka,
- `.execSQL()` - izvršava čisti SQL upit.

Treba naglasiti da bi prilikom svake promjene sadržaja tablice baze podataka treba osvježiti i sadržaj pokazivača s pozivom novog upita za sve pokazivače koji su možda zahvaćeni s tom promjenom podataka [26].
Od 2017. se može koristiti ROOM za pristup bazama podataka. ROOM je sloj apstrakcije koji omogućava lakši i pouzdaniji pristup bazi podataka, omogućava spremanje najvažnijih podataka baze podataka koja je na nekom poslužitelju lokalno na uređaju radi pregledavanja podataka kad mreža nije dostupna, kasnije može odraditi sinkronizaciju promijenjenih lokalnih podataka s poslužiteljem [27].

Kod projektiranja baze podataka treba uzeti u obzir da se u bazu podataka ne mogu unositi veće slike, video, audio datoteke i slični sadržaji. Manje se mogu unositi kao binarne BLOB (eng.

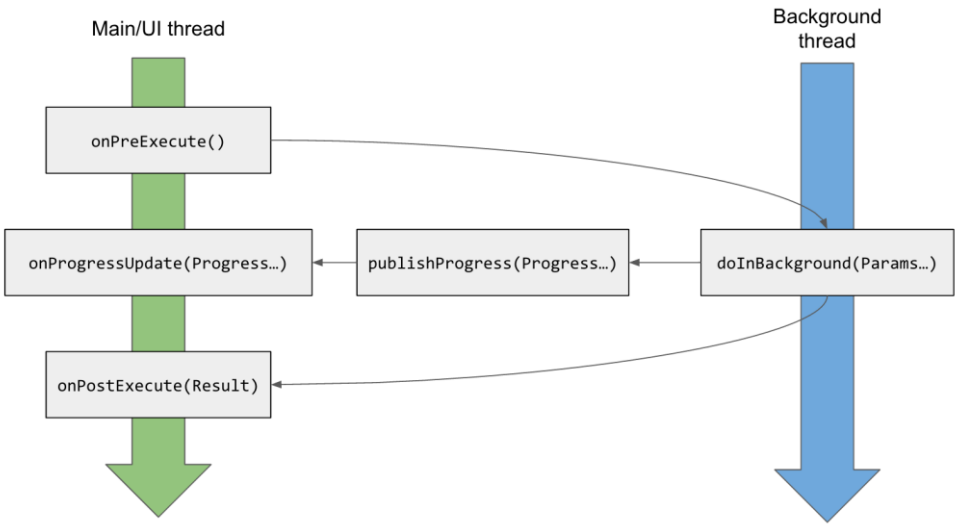
Binary large objects) datoteke. Za veće datoteke kao slike, video i audio se ipak preporučuje da se pohranjuju u neki direktorij uređaja, a u bazi podataka njihov `URI`. Potom se dohvati `URI` i s tim `URI`-jem se obavlja neka manipulacija tim objektom. Također treba uzeti u obzir da je poželjno da svaka tablica ima nekakvi broj identifikatora odnosno ID broj. Taj broj služi kao unikatan identifikator nekog reda tablice. Kada se koristi pružatelj sadržaja koji čini tablicu javno dostupnom drugim aplikacijama uređaja je takvo ID polje obavezno. Također se još preporučuje da je polje auto inkrementirajuće znači da se automatski povećava za jedan dodavanjem svakog novog reda tablice.

2.6. Mrežna komunikacija

Mrežni servisi su integralni dio Android operacijskog sustava. Cijeli operacijski sustav je građen s mrežama na umu. Mrežne operacije se u Androidu odvijaju vrlo „automatizirano“ što znači da se zadaci kao uspostavljanje HTTP konekcije obavljaju u nekoliko linija koda. To omogućuje Android API koji je sadržan u sustavu - iako Java omogućava korištenje svih klasičnih protokola iz TCP/IP složaja u mobilnim aplikacijama najčešće se radi o korištenju HTTP protokola, uz preporuke korištenja sigurne izvedbe odnosno HTTPS protokola. Za rad s HTTP-om zadužen je razred `HttpURLConnection`. Da bi aplikacija mogla raditi s ugrađenim mrežnim resursima mora imati dozvoljene mrežne operacije. Ta se dozvola daje u manifestu aplikacije. Kod mrežnih, ali i drugih procesno/vremensko zahtjevnijih operacija potrebno je pridržavati se preporuka za razvoj aplikacija – sve takve operacije potrebno je prilagoditi i izvoditi u pozadini, tj. konkurentno s glavnim procesom odnosno dretvom aplikacije. Naime, koncept izvođenja Android aplikacije je takav da se osnovne operacije izvode u takozvanoj glavnoj dretvi (eng. *Main thread*) i ona je zadužena za manipulaciju korisničkim sučeljem (odatle u nekim referencama i naziv *UI thread*). Ukoliko zbog određene vremenski zahtjevne operacije sustav detektira da je sučelje „smrznuto“ (nema nikakvih promjena, a očito je da se nešto procesira) zadržava pravo gašenja aplikacije i prijave ANR pogreške (eng. *Application Not Responding*).

S različitim verzijama Androida preporuke su se neznatno mijenjale, no pravilo je uvijek jasno – sve operacije koje su vremensko zahtjevne treba izdvojiti i pokretati ih van glavne dretve u takozvanim radnim ili pozadinskim dretvama (eng. *Worker thread* odnosno eng. *Background thread*), te po potrebi, opet prema propisanim pravilima, osvježavati sučelje u glavnoj dretvi. Moguće je koristiti standardne Java dretve (razred *Thread*), ali i nekoliko specifičnosti koje dodaje Android API.

Za to se uobičajeno upotrebljava predefinirani razred `AsyncTask`. Postupci bazirani na tom razredu izvode se u pozadini, ali omogućavaju i komunikaciju s glavnom dretvom i osvježavanje sučelja (na primjer, prikaz napretka operacije i slično).



Slika 14: Tok funkcija AsyncTask razreda [28]

Na slici 14 se vidi tok izvršavanja određene instance AsyncTask razreda. Jedna od tih funkcija je `doInBackground` koja se vrti u pozadini. U Android sustavu je ta funkcija na čekanju i sluša tako dugo dok se ne pojavi neka operacija koja onda otvori tu funkciju i u njoj nešto obradi. Nakon te obrade odlazi u funkciju koja se nalazi u istom Java razredu `onPostExecute` koja se poziva kada prethodni razred koji se vrti u pozadini nešto izvrši. Tamo se odrađuju zadaće koje su potrebne nakon prijema podataka s poslužitelja. U funkciji `doInBackgrund` nije moguće jednostavno odlaziti u glavni razred aktivnosti da bi se izvršio neki zadatak osim ako ta funkcija nije stavljena u `runOnUiThread`. Ova funkcija predstavlja korisnički prekid glavnog ciklusa izvršavanja aplikacije. Ako se pozove stavlja prekid u listu čekanja gdje mora čekati ako ispred nje postoji neki drugi prekid, ako ne postoji drugi prekid se izvršavaju naredbe koje se nalaze unutar te funkcije. Preporučljivo je da se ne preopterećuje prekid jer bi moglo utjecati na stabilnost aplikacije. Znači samo jednostavne zadaće koje se mogu brzo odraditi [29].

Klasični pristup mrežnoj komunikaciji, kao što je već prethodno istaknuto, svodi se na korištenje razreda `HttpURLConnection` (odnosno `HttpsURLConnection` za sigurnu komunikaciju korištenjem sigurnosnog međusloja i TLS protokola). Alternativno je moguće koristiti dodatne biblioteke poput `okHttp`.

Najbitnije funkcije za rad s razredom `HttpsURLConnection` su:

- `.openConnection()` - otvara konekciju prema poslužitelju,
- `.setDoOutput()` - definira izlazni tip konekcije,
- `.setDoInput()` - definira ulazni tip konekcije,

`.InputStream()` - definira tok podataka kao ulazni,
`.OutputStream()` - definira tok podataka kao izlazni,
`objekt_konekcije.getOutputStream()` - definicija objekta izlaznog toka za konekciju,
`objekt_konekcije.getInputStream()` - definicija objekta izlaznog toka za konekciju,
`.BufferedWriter()` - šalje podatke na poslužitelj tako da uzme cijeli string podataka i pretvori ga u jedan dugi string koji onda prima i čita ili šalje,
`.BufferedReader()` - prima podatke s poslužitelja,
`bufferPisac.write(podatak)` – piše podatak,
`bufferCitac.readLine()` – čita primljeni podatak,
`buffer.flush()` – prisilno ispisuje sve podatke u tok podataka,
`.close()` – zatvara tok podataka i buffer,
`.disconnect()` – prekida konekciju objekta konekcije.

Primjer uspostave konekcije:

```
1   url = new URL("NEKA URL ADRESA");
2   ObjektHTTPKonekcije = (HttpURLConnection)url.openConnection();
3   ObjektHTTPKonekcije.setRequestMethod("POST");
4   ObjektHTTPKonekcije.setDoOutput(true);
5   ObjektHTTPKonekcije.setDoInput(true);
6   izlazniStream = ObjektHTTPKonekcije.getOutputStream();
7   bufferPisac = new BufferedWriter(new OutputStreamWriter(izlazniStream,"UTF-8"));
8   String post_data = URLEncoder.encode("k1","UTF-8")+
9      "="+URLEncoder.encode(v1,"UTF-8")+
10     "&"+URLEncoder.encode("k2","UTF-8")+
11     "="+ URLEncoder.encode(v2,"UTF-8");
12  bufferPisac.write(post_data);
13  bufferPisac.flush();
14  bufferPisac.close();
15  izlazniStream.close();
16  ulazniStream = ObjektHTTPKonekcije.getInputStream();
17  bufferCitac = new BufferedReader(new InputStreamReader(ulazniStream,"UTF-8"));
18  linija = bufferCitac.readLine();
19  bufferCitac.close();
20  ulazniStream.close();
21  httpURLKonekcija.disconnect();
```

Primjer na liniji koda 1 prikazuje deklaraciju i inicijalizaciju nekog `URL` objekta koji sadrži adresu resursa, u našem slučaju poslužitelja. Linija 2 prikazuje definiranje objekta konekcije pomoću `URL` adrese. Linije 3-5 koriste tzv. settere (metode za postavljanje atributa objekta) – u liniji 3 definira se HTTP metoda zahtjeva koja se koristi (POST), a u linijama 4 i 5 definira da se radi o ulazno-izlaznoj vezi. Na liniji 6 se definira izlazni tok podataka, a na liniji 16 ulazni tok podataka. Linija 7 prikazuje definiranje buffer pisača argumentima za tok i tip kodiranja zapisa. Linija 8 prikazuje formiranje podatak koji će se slati. Na liniji 12 se taj podatak šalje u buffera i na liniji 13 se buffer prazni. Linije 14 i 15 prikazuju zatvaranje toka podataka i buffera za izlazni dio konekcije. Na linijama 16 i 17 se definira ulazni tok i buffer te ja na liniji 18 prikazano čitanje primljenog podatka. Linije 19 i 20 prikazuju zatvarane toka i buffera te linija 21 prikazuje prekidanje konekcije.

2.6.1. HTTP zahtjevi – GET i POST

Korištenjem HTTP-a, zahtjevi koji se šalju prema poslužitelju (HTTP upit/zahtjev) mogu koristiti tzv. metode zahtjeva – POST ili GET. Osim adrese (`URL`), zahtjevi će u pravilu uključivati i određene podatke u tekstualnom obliku. Ovisno o metodi, ti podaci se šalju kroz `URL` adresu (GET) ili kao dodatak u formiranom HTTP paketu (POST). Opći oblik GET zahtjeva je : „`http://adresa/stranica.php?k1=v1&k2=v2…`". Zahtjev je od adrese odijeljen znakom upitnika, a podaci se odvajaju znakom &. U prethodnoj adresi podaci se šalju korištenjem ključa/identifikatora k1 i k2, a vrijednosti koje poprimaju su v1 i v2. Problem kod tog oblika zahtjeva je da je cijeli podatak smješten u samu adresu poslužitelja (kad se koristi kroz web preglednik, to znači da je skup poslanih podataka uvijek vidljiv korisniku, no u našoj primjeni, kod internog korištenja unutar aplikacije, to nije značajan nedostatak). Drugi veliki nedostatak je da je takav oblik zahtjeva ograničen u veličini podataka koji se mogu slati na poslužitelj. Druga HTTP metoda zahtjeva, koja je ujedno i korištena u praktičnom radu odnosno mobilnoj aplikaciji, je POST. POST za razliku od prethodnog GET stavlja podatke zahtjeva u sam sadržaj podatkovnog paketa. Posti ima oblik adrese „`http://adresa/stranica.php`". Podaci se nalaze u sadržaju podatkovnog paketa, formata „`k1=v1&k2=v2…`". Prije slanja, podatke je potrebno pretvoriti odnosno prilagoditi u standardni oblik tekstualnih zapisa (na primjer, UTF8), a za to se koristi razred `URLEncode` i metoda `encode` – metoda `encode` kao parametre prima tekst koji je potrebno prilagoditi i format zapisa. To je prikazano u primjeru koda u prethodnom odjeljku *2.6.* na linijama 8-11. Iz odjeljka *2.6.* se može vidjeti da za slanje GET zahtjeva samo treba ugraditi zahtjev prema prethodno definiranom obliku u string koji opisuje `URL` adresu poslužitelja i onda sadržaj same poruke na poslužitelj može ostati prazan. Još je potrebno na liniji 3 definirati zahtjev kao GET.

2.6.2. JSON zapis

U aplikaciji razvijenoj u ovom radu se podaci s poslužitelja u nekoliko slučajeva primaju u JSON (eng. *JavaScript Object Notation*) formatu. JSON je Standardizirani tekstualni oblika zapisa za strukturirane podatke – podaci i strukture se definiraju prema točno određenim pravilima. Većina modernih razvojnih okruženja podržava JSON, pa tako i Android API koji omogućava lako baratanje JSON zapisima i potrebne pretvorbe podataka u JSON i dohvata podataka iz JSON zapisa. Androidu kod baratanja JSON podacima pomaže API koji uzme JSON podatak i stavlja ga u JSON niz lista gdje svaka stavka niza sadrži jedan JSON element. Time automatski rastavi JSON podatak. Tip takvog niza je JSONArray. Konkretni primjer rada s JSON-om se daje u odjeljku *3.2.12.*

3. Razvoj Android aplikacije

Mobilna aplikacija koja predstavlja praktični dio završnog rada uključuje nekoliko ključnih elemenata:

- korištenje lokacijskih servisa za dohvat informacija o lokaciji,
- korištenje pohrane podataka za pohranu lokacija,
- korištenje mapa za vizualizaciju lokacija,
- korištenje kamere za fotografiranje.

Također, kao dodatnu funkcionalnost uključuje sučelje i mogućnosti slanja fotografije i lokacija s ciljem primjene u nekom servisu za prijave npr. komunalnom redaru.

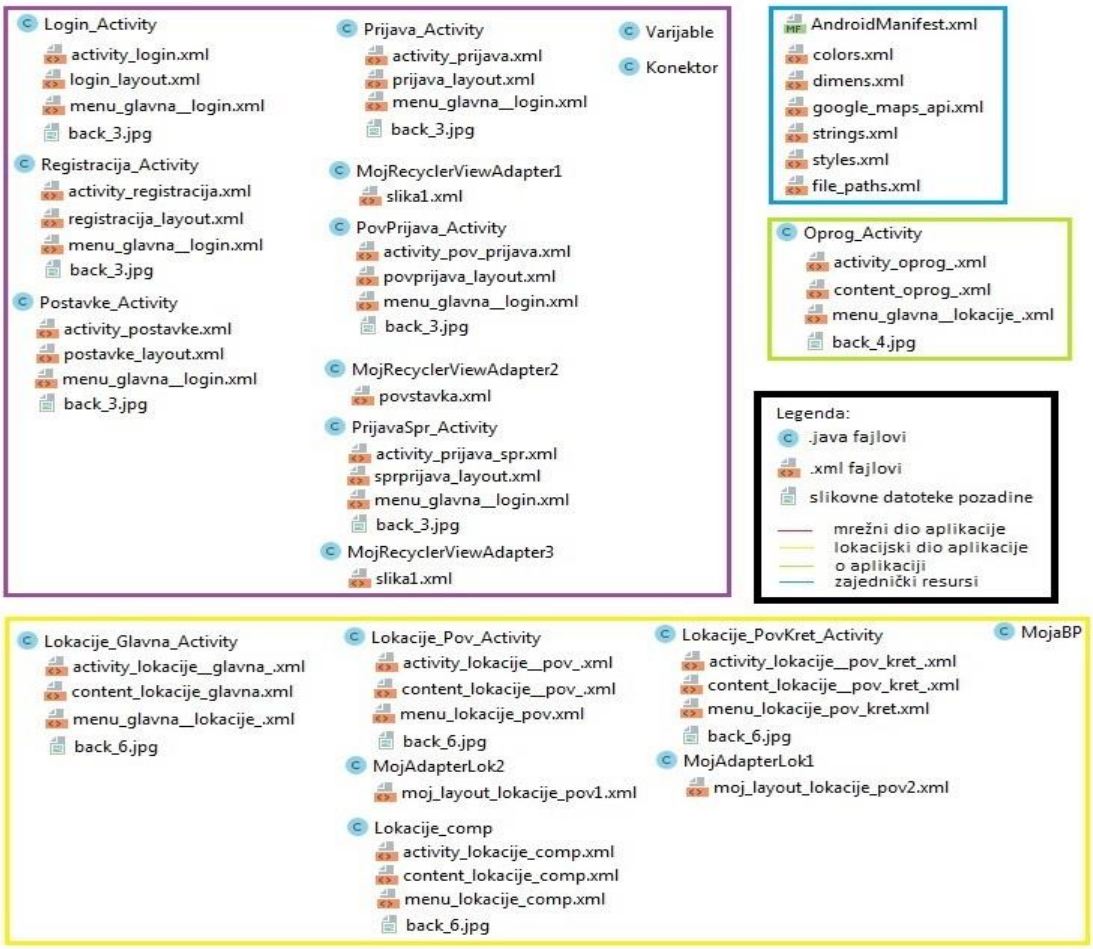
3.1. Koncept aplikacije

Mobilna aplikacija razvijana u okviru završnog rada sadrži dva korisnički orijentirana dijela – osnovni, vezan uz lokacijske servise, pohranu, prikaz i vizualizaciju informacija o evidentiranim lokacijama, te dodatni koji proširuje temeljne funkcionalnosti mogućnostima prijave i prosljeđivanja informacija na poslužitelj.

Lokacijski dio sprema dohvaćenu lokaciju u bazu podataka lokalno na uređaju. Jedna od sposobnost lokacijskog dijela je praćenje kretanja. Spremanje može biti izvedeno ručno ili automatski. Ručno se pohranjuje pojedina lokacija, a automatski lokacije koje su dio neke rute. Također, ima mogućnost brisanja svih spremljenih lokacija i ruta. Svaka aktivnost lokacije ima i prozor u kojem je učitana GM i na njoj se onda prikazuju lokacije ili rute. Dodatno može staviti dvije lokacije kao mete i onda ih uspoređivati. Time pokazuje neke sposobnosti Android operacijskog sustava kao što je mogućnost određivanja udaljenosti između dvije lokacije i određivanje kuta azimuta jedne lokacije u odnos na drugu.

Drugi korisnički dio aplikacije je zamišljen kao modul koji služi za prijavljivanje nekog događaja komunalnom redaru i očitavanje statusa obrade (obrada se provodi korištenje web aplikacije opisane u poglavlju 4). Taj dio aplikacije omogućava interakciju korisnik-poslužitelj – prijavu na poslužitelj, registraciju novog korisničkog računa, dohvat i pregledavanje prethodnih prijava korisnika itd. Prethodne prijave sadrže i odgovor administratora. Opcija brisanja tih prijava nije dana jer je pretpostavka da su te prijave ozbiljne pa bi nadoknadno brisanje prijava moglo biti štetno. Uz to je omogućena kontrola postavki korisničkog računa kad je korisnik prijavljen u sustav. U postavkama se postavlja i IP adresa poslužitelja. Ta adresa je potrebna za komunikaciju s poslužiteljem. Korisničko sučelje aplikacije se razlikuje ovisno o statusu prijave

korisnika. Ako je korisnik prijavljen on ima pristup postavkama korisničkog računa, a ako nije nema pristup. Opet mogućnost pregleda povijesti prijava ovisi o statusu prijavljenosti. Kad korisnik nije prijavljen naravno da ne može pregledavati povijest svojih prijava. Ono što je omogućeno kad korisnik nije prijavljen je puna funkcionalnost mogućnosti izrade anonimne prijave i njezinog slanja na poslužitelj.



Slika 15: Prikaz logičke strukture aplikacije

Slika 15 prikazuje logičku strukturu mobilne aplikacije. Struktura sadrži opet dvije glavne cjeline jedna je mrežna, a duga lokacijska. Uz to zasebno je prikazana i cjelina koja prikazuje podatke o aplikaciji i cjelina koja sadrži elemente koji su dostupni svim cjelinama i potrebni su za funkcioniranje aplikacije.

Datoteka `Konektor.java` koja sadrži mrežne elemente iz odjeljka *2.6.* je napravljena isključivo za mrežnu komunikaciju i nije aktivnost znači nema nikakvog grafičkog prikaza nego se elementi dobiveni iz nje koriste u drugim aktivnostima.

Podsustav za lociranje opisan u odjeljku *2.3.2.* koristi se u više dijelova aplikacija – kako u temeljnom tako i u dijelu za prijave na poslužitelj. Glavna aktivnost gdje se vidi je

`Lokacije_Glavna_Activity.java`, tu je obrađeno neprestano dohvaćanje lokacije. U aktivnosti `Prijava_Activity.java` je vidljiv primjer programskog dohvaćanja lokacije ali samo jedne lokacije, na zahtjev korisnika.

Aktivnosti (prozori) `Lokacije_Glavna_Activity.java`, `Prijava_Activity.java`, `PrijavaSpr_Activity.java`, `Lokacije_Pov_Ativity.java`, `Lokacije_comp.java`, `Lokacije_PovKret_Activity.java` uključuju funkcionalnosti vizualizacije lokacije pomoću GM (Google mapa).

Sve aktivnosti lokacijskog dijela po potrebi koriste pohranu podataka u bazu podataka – pri tome koriste datoteku `MojaBP.java` koja uključuje sav kod vezan uz bazu i izvršavanje operacija nad bazom. Uz pohranu u bazu podataka je prikazana i pohrana u *SharedPreferences*, ne samo u lokacijskom djelu aplikacije nego i u djelu gdje se šalje prijava na poslužitelj. Kod aktivnosti `Prijava_Activity.java` postoji primjer pohrane u datoteku. Pohranjuje se tek napravljena slika. Svi navedeni primjeri pohrane su obrađeni u odjeljku *2.5*.

Na slici 15 ispod svake aktivnosti je dan popis XML datoteka koje sadrže strukturirane informacije i definiciju korisničkog sučelja za prozore, izbornika i drugih resursa. Za lakše snalaženje, kao element sučelja definirana je pozadinska slika koja je vezana uz dijelove aplikacije (osnovni dio aplikacije uključuje jednu, a dio vezan uz prijave drugu pozadinsku sliku).

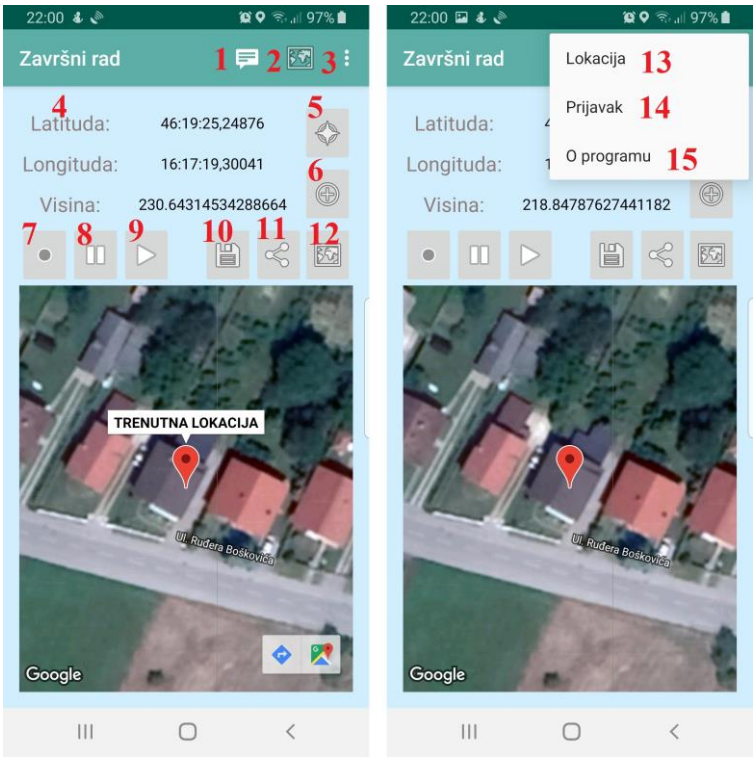
Datoteke `Konektor.java` i `Varijable.java` su na slici 15 zasebno prikazane jer su jednako dostupne svim drugim datotekama. Slično, na slici su izdvojeni dio aplikacije i pomoćni prozor „O programu“. Uz te cjeline prikazani su još i zasebni resursi koji sadrže manifest odnosno dokument koji sadrži sve podatke nužne za funkcioniranje aplikacije kao što su ime aplikacije, dozvole za pristup resursima mobilnog uređaja kao što je kamera, popis svih komponenti aplikacije itd. Tu se još nalaze i XML datoteke koje sadrže razne definicije i postavke najčešće vezane uz korisničko sučelje aplikacije (boje, tekstualni i drugi resursi itd.), XML dokument koji sadrži Google API ključ potreban za pristup Google-ovim servisima (u našem slučaju GM).

3.2. Korisnička sučelja aplikacije

U ovom odjeljku je prikazano korisničko sučelje aplikacije i funkcionalnosti pojedinih elemenata sučelja za sve aktivnosti. Nakon toga ukratko se opisuje princip izvedbe (programerski) ključnih funkcionalnosti. Još su opisani i problemi koji su se javili tijekom razvoja aplikacije i njihova rješenja.

3.2.1. Dohvat i pohrana lokacija i ruta

Na sučelju koje se vidi na slici 16 prikazuje početni prozor aplikacije. Implementacija je dana u `Lokacije_Glavna_Activity.java` datoteci, a resursi i opis sučelja su opisani u XML datotekama `activity_lokacije__glavna_.xml` i `content_lokacije_glavna.xml`.



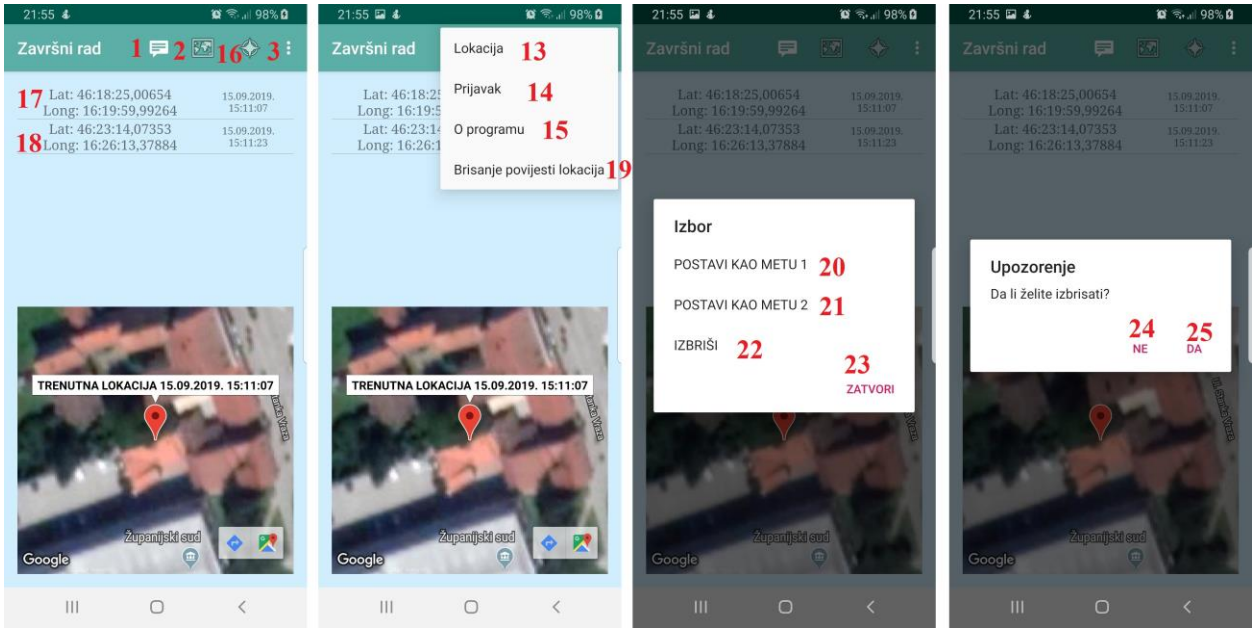
Slika 16: Glavni ekran – lokacijski dio

To je prva aktivnost koja se pokreće pri pokretanju aplikacije. To je ujedno ulazno sučelje za glavni, lokacijski dio aplikacije. Ključni elementi sučelja su na slici označenim brojkama radi lakšeg snalaženja i dokumentiranja! Gornji dio ekrana zauzima naslovni redak koji sadrži i dvije ikone (brojevi 1 i 2) i proširenje za otvaranje izbornika (element broj 3). Brojem 1 je označena ikona za pokretanje sučelja prijavnog dijela aplikacije. Element 5 je vrlo bitan jer aktivira dohvat lokacije preko LM i LL. Nakon aktivacije, mogući je dohvat i pohrana aktualne lokacije elementom 6, odnosno početak snimanja svih lokacija (praćenje kretanja) pomoću elemenata 7 (pokretanje praćenja) i sa elementom 8 zaustavljanja praćenje. Kad se klikne na tipku 12 odlazi u aktivnost gdje je su prikazane spremljene lokacije. Lokacija se sprema u SQLite bazu podataka. Lokacije se privremeno pohranjuju u *SharedPreferences* (interna lokalna pohrana), ali je, naravno, omogućena i trajna pohrana (element sučelja 10) u SQLite bazu podataka. U svakom trenutku je trenutno snimanu rutu moguće vizualizirati klikom na element sučelja 9. Sve pohranjene rute moguće je naknadno analizirati i vizualizirati, a sučelje za to se pokreće elementom sučelja 11. Lokacija je prikazana u obliku latitude, longitude i visine. Te koordinate su pokazane u formatu „stupnjeva:minuta:sekundi.pet decimala sekundi“ (SS:MM:SS.sssss).

Lokacija se dohvaća u *double* formatu te se onda konvertira po potrebi u druge oblike. Preciznost dohvaćene lokacije je varijabilna. Očekivana preciznost sa GPS sustavom je oko 5 m, u lošim uvjetima može pasti na 15 m. Procjenu preciznosti lokacije je moguće dohvatiti pomoću Android API-a. Donji dio osnovnog prozora služi za GM prikaz, uz tip prikaza prema postavkama (na primjeru, hibridan prikaz satelitske snimke s osnovnom kartografijom) – na GM se dodatno markerom označava trenutka lokacija. Na lijevoj strani slike 16 se vidi ista aktivnost kada se otvori izbornik. Elementom 14 se odlazi u glavnu aktivnost (prozor) dijela aplikacije vezanog uz mrežnu komunikaciju i sustav online prijava (`Login_Activity.java`). Sa stavkom 15 se odlazi u dio aplikacije koji sadrži informacije o aplikaciji.

3.2.2. Pregled pohranjenih lokacija

Slika 17 prikazuje sučelje za pregled pohranjenih pojedinačnih lokacija. Aktivnost je implementirana u `Lokacije_Pov_Activity.java` datoteci, a elementi sučelja opisani u `content_lokacije__pov_xml` datoteci.



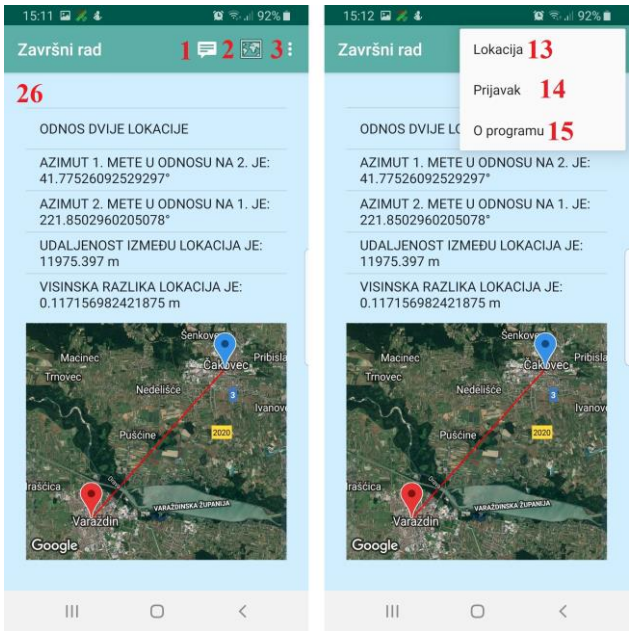
Slika 17: Aktivnost povijesti lokacija

Elementi sučelja označeni sa 1, 2, 3 i 16 služe za navigaciju unutar dijelova aplikacije. Tipkom 16 odlazi u aktivnost usporedbe dviju lokacija. To su lokacije koje su prethodno dodane kao mete. Broj 3 predstavlja tipku za otvaranje izbornika. Središnji (gornji, elementi 17 i 18) dio prozora zauzima lista spremljenih lokacija, dok je u donjem dijelu implementiran prikaz GM. Primjer na slici sadrži samo dvije spremljene lokacije – podatak o lokaciji osim koordinata uključuje i vrijeme snimanja. Podaci pojedine lokacije sadrže koordinate te lokacije i vrijeme snimanja. Klikom na pojedinu lokaciju se prikazuje na karti lokacija i na lokaciju se postavlja marker. Klikom na marker se prikazuje poruka „TRENUTNA LOKACIJA“ te datum i vrijeme

snimanja lokacije. Lokacije su pohranjene u bazi podataka. Dugim klikom na stavku liste otvara se izbornik koji omogućava određene operacije nad označenom lokacijom (vidljivo na trećem dijelu slike 17). Klikom na 23 može se zatvoriti upitnik. Klikom na 22 se briše tu lokaciju. S tipkama 21 i 20 postavlja lokaciju kao metu jedan ili dva ovisno o korisniku. Druga slika s lijeva na slici 17 prikazuje izbornik te aktivnosti. Izbornik je sličan prijašnjem gdje su funkcije tipki 13 do 15 iste kao i tipki s prijašnje aktivnosti samo se razlikuje u tipki 19 koja služi da se istovremeno izbriše cjelokupna povijest svih lokacija. Kod brisanja pojedine lokacije ili brisanja svih lokacija se otvara upitnik vidljiv na četvrtoj slici s lijeva slike 17. Taj izbornik služi kao kontrola da se to ne izbriše slučajno. Izbornik sadrži tipke ne 24 i da 25. Da bi se izašlo iz tog upitnika potrebno je pritisnuti jednu od njih. S brisanjem se briše i lokacija koja je dostupna u aktivnosti usporedbe lokacije, a spremljena u *SharedPreferences*. Brisanjem se stavka uklanja iz baze podataka i osvježava se aktivnost radi promjene sadržaja. Da bi se ušlo u taj izbornik moraju biti dodijeljene te mete inače javlja poruku o grešci koja kaže da moraju postojati dvije lokacije mete.

3.2.3. Usporedba lokacija

Pritiskom na tipku 18 dostupnu samo u aktivnosti spremljenih lokacija se odlazi u aktivnost za usporedbu lokacija - aktivnost je implementirana u `Lokacije_comp.java`, a sučelje opisano u `content_lokacije_comp.xml` datoteci.



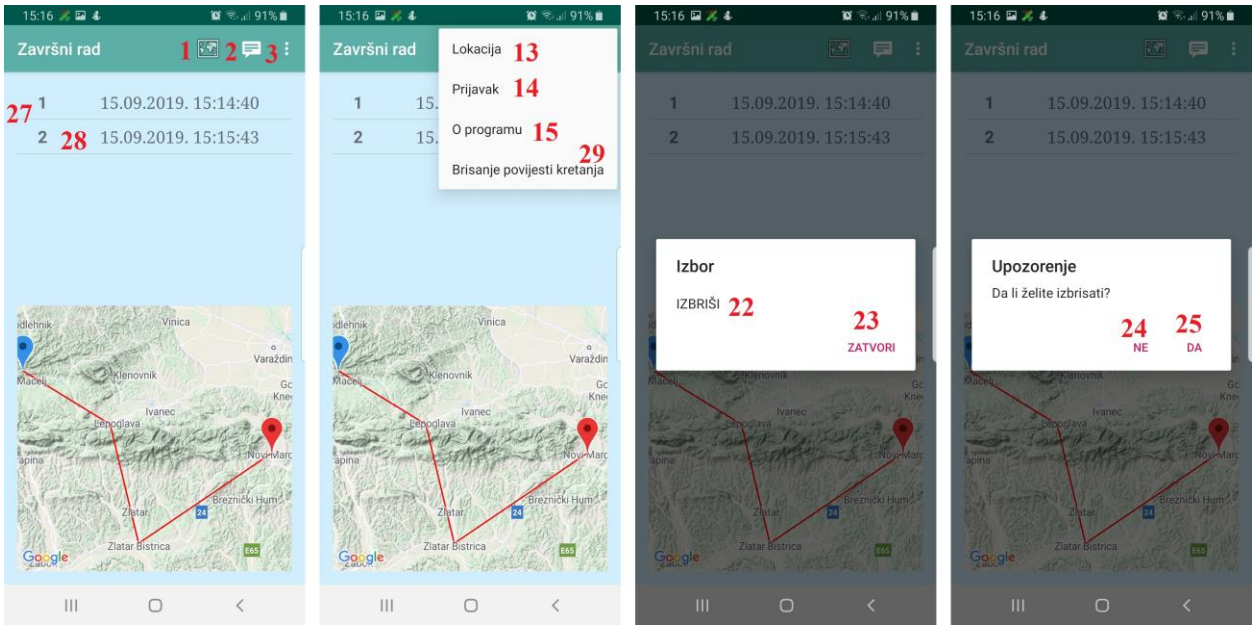
Slika 18: Aktivnost usporedbe lokacija

Informacije o lokacijama koje se uspoređuju su prikazane u obliku liste (26), pa svi podaci nisu vidljivi odjednom (mapa je fiksne veličine u donjem dijelu prozora). Donji dio prozora sadrži kartu sa markerima lokacija i polilinijom koja povezuje lokacije. Na lijevom dijelu slike je

vidljiv izbornik dostupan u ovoj aktivnosti. Na njemu se vide standardne tipke za odlazak u početne aktivnosti mrežnog i lokacijskog dijela aplikacije i za odlazak u dio informacija o aplikacijama.

3.2.4. Pregled i vizualizacija ruta

Na slici 19 se vidi aktivnost povijesti ruta (`Lokacije_PovKret_Activity.java`).

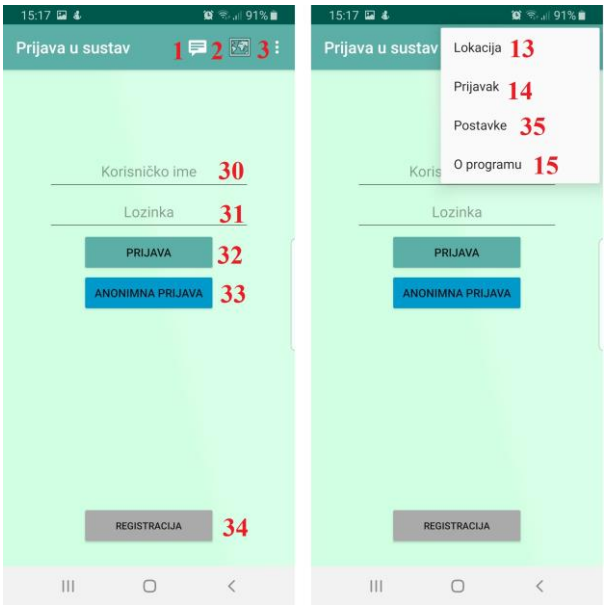


Slika 19: Aktivnost povijesti ruta

Aktivnost ima gumbe 1, 2 i 3 kojima se pristupa dijelovima aplikacije i izborniku. Na dnu prozora se vidi karta na kojoj se vide markeri polazišta i odredišta. Sve točke rute su kronološki povezane crvenom linijom. Broj 27 prikazuje listu koja se može pomicati gore dolje ako ima dovoljno stavki da ispune za listu predviđen prozor. U primjeru su dvije stavke i jedna od njih je označena brojem 28. Stavka rute sadrži redni broj snimanja i datum te vrijeme snimanja. Na drugoj slici s lijeva se vidi izbornik. On ima tipke 13, 14 i 15 kao i u prethodnim primjerima i tipku 29 kojom se brišu sve rute od jednom. Kad se dugo klikne na stavku se otvara upitnik na trećoj slici s lijeva. On ima tipku 22 koju se klikne ako se želi izbrisati tu rutu ili tipku 23 ako se želi zatvoriti. Ako se klikne na 29 ili 22 se otvara upitnik da li se stvarno želi izbrisati kao i u prošloj aktivnosti. Klikom na tipku da 25 se briše, a na tipku ne 24 se ne briše te se zatvara upitnik.

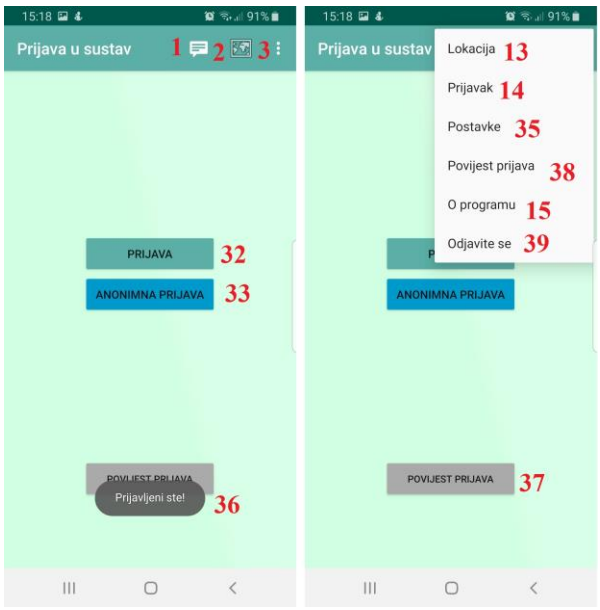
3.2.5. Prijava na poslužitelj, registracija korisnika i postavke

Na slici 20 se vidi početna aktivnost mrežnog dijela aplikacije. Sučelje je implementirano u `Login_Activity.java` datoteci te `activity_login.xml` i `login_layout.xml` datotekama.



Slika 20: Početna aktivnost za prijavu na poslužitelj

U polja 31 i 30 se upisuje lozinka i korisničko ime. Prilikom upisivanja lozinka nije vidljiva nego se prikaže na kratko zadnje upisani znak i onda se pretvara u crnu točku. Korisničko ime je normalno vidljivo. Onda se s unesenim podacima pritiskom na tipku 32 prijavljuje na poslužitelj. Sve mrežne operacije se odrađuju u Java datoteci `Konektor.java` koja nije aktivnosti. Poslužitelj šalje podatke o korisniku. Aktivnost ima standardne tipke za navigaciju po aplikaciji 1, 2 i 3. Na desnoj slici u izborniku su vidljive isto standardne tipke za navigaciju 13, 14 i 15. Pritiskom na tipku 34 odlazi u aktivnost gdje se ispunjava obrazac za registraciju, a pritiskom na tipku 33 odlazi u ispunjavanje anonimne prijave. Tipka 35 na izborniku desno odlazi u aktivnost postavki.



Slika 21: Početna aktivnosti prijave nakon prijave na poslužitelj

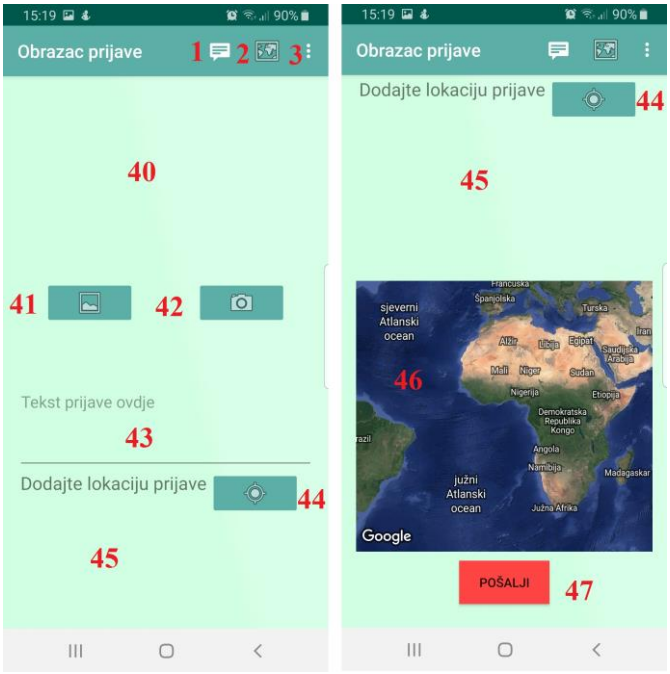
Slika 21 prikazuje početnu aktivnost mrežnog dijela aplikacije nakon prijave na poslužitelj. Aktivnost iz datoteke (`Login_Activity.java`) izgleda drugačije, a i izbornik se izmijenio. Na lijevoj slici se vidi standardne navigacijske tipke 1, 2 i tipku izbornika 3. Ispod njih su nestala polja za unos lozinke i korisničkog imena. Ta polja su programski skrivena u odgovarajućem Java razredu aktivnosti. Isto vrijedi i za tipku registracije. Programski se prikazuje tipka 37 za odlazak u povijest prijava. Tipka prijava pod brojem 32 sad ima drugu funkciju. Njezina funkcija je ulazak u aktivnost gdje se ispunjava prijava koja nije anonimna. Funkcija tipke 33 odnosno anonimna prijava je ostala ista. Ekran je poslikan odmah poslije prijave kad je još vidljiva poruka 36 na kojoj piše da se korisnik uspješno prijavio na poslužitelj. Na desnoj slici je vidljiv izbornik. On sadrži standardne navigacijske tipke 13, 14, 15 i ostala je tipka za odlazak u postavke tog dijela aplikacije 35. Tipkom 38 se odlazi u povijest prijava. To je aktivnost koja sadrži listu svih prethodnih prijava korisnika koje nisu anonimne. Pritiskom na tipku 39 se odjavljuje s poslužitelja. Znači da se brišu svi korisnički podaci koji su učitani prijavom s poslužitelja. Ti podaci su spremljeni u Java datoteku `Varijable.java`. Prilikom odjave se ti podaci vraćaju na početne vrijednosti prije prijave. Također se još ponovno zatvara i otvara početna aktivnost mrežnog dijela aplikacije. Ponovnim otvaranjem se ponovno učitavaju svi elementi te aktivnosti u obrascu kao kad aplikacija nije prijavljena.

Aktivnosti registracije se može pristupiti samo kad korisnik nije prijavljen na poslužitelj. Nema smisla da se može registrirati novi korisnik ako već trenutni korisnik ima korisnički račun. To je mehanizam koji se često javlja na mnogim mrežnim servisima. Ta aktivnost ima polja za unos podataka o korisniku. Unose se imena i prezimena korisnika, OIB broj korisnika, korisničko ime,

dva polja za unos lozinke korisnika. Lozinke moraju biti iste inače neće dozvoliti registraciju. To je često korišteni mehanizam da ne bi došlo do krivog unosa lozinke koja u tim poljima ionako nije vidljiva iz sigurnosnih razloga. Uz lozinke obavezan za slanje registracije je pristanak na uvjete korištenja. Izbornik je u aktivnosti registracije isti standardni izbornik kao u svim drugim aktivnostima kad korisnik nije prijavljen na poslužitelj. U aktivnosti postavka u oba slučaja prijavljenosti se vidi polje za unos IP adrese poslužitelja. Kada je korisnik prijavljen su vidljive postavke korisničkog računa. Te postavke sadrže prikaz svih podataka korisnika koji se onda mogu mijenjati. Primjer spomenutih operacija na poslužitelju je u odjeljku *4.5.*

3.2.6. Obrazac prijave

Na slici 22 se vidi aktivnost obrasca prijave. Sučelje je implementirano u `Prijava_Activity.java` datoteci te `activity_prijava.xml` i `prijava_layout.xml` datotekama.



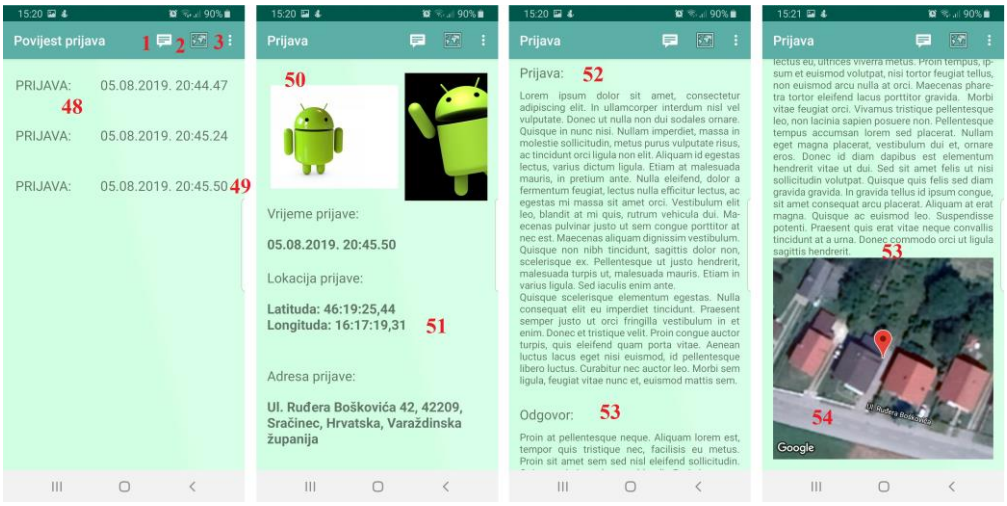
Slika 22: Aktivnost obrasca prijave

Obrazac prijave je isti u svim varijantama postupka (prijavljeni korisnik, anonimna prijava). Gornji dio prozora zauzima najbitniji dio sučelja (element 40) koji služi za prikaz slike ili slika koje se šalju kao dio prijave na poslužitelj. Ukoliko je dodano više slika, moguće je povlačenje lijevo-desno i pregled svih slika. U Androidu su slikovni zapisi resursi – program omogućava korištenje postojećih slikovnih zapisa (gumb 41) ili stvaranje novih korištenjem integrirane kamere (gumb 42). Polje 43 je polje za unos u koje korisnik unosi neki tekst prijave. Klikom na tipku 44 se lokacijskim servisima Androida dohvaća lokacija uređaja. Ta lokacije se dohvaća

samo jednom i onda se čekanje osvježenja lokacije isključuje. Lokacija se potom konvertira u korisniku bliži format od standardnog i prikazuje u polju 45. Prikazuje se u obliku koordinata dohvaćene lokacije i detalja (adrese, ako je dostupna) koje vraća geokoder (odjeljak *2.3.2.*). Po dohvaćanju lokacije se lokacija prikazuje na karti (element 46), karta se pomiče i približava dohvaćenoj lokaciji i stavlja marker s popratnim natpisom (vidljiv na klik miša). Ako pak se klikne na gumb 47 počinje slati podatke na poslužitelj te korisniku vraća obavijesti kao poruku o tijeku slanja. Primjer toga je uspjeh slanja teksta prijave te napredak slanja slika. Ta aktivnost se po uspješno završenom slanju gasi i vraća na početnu aktivnost mrežnog dijela aplikacije.

3.2.7. Prikaz prethodnih prijava

Slika 23 prikazuje aktivnost povijesti prijava i otvorenu poslanu prijavu. Aktivnosti implementirane u `PovPrijava_Activity.java` i `PrijavaSpr_Activity.java` datotekama. Opisi sučelja su dani u `activity_pov_prijava.xml`, `activity_prijava_spr.xml`, `povprijava_layout.xml` i `sprprijava_layout.xml`.



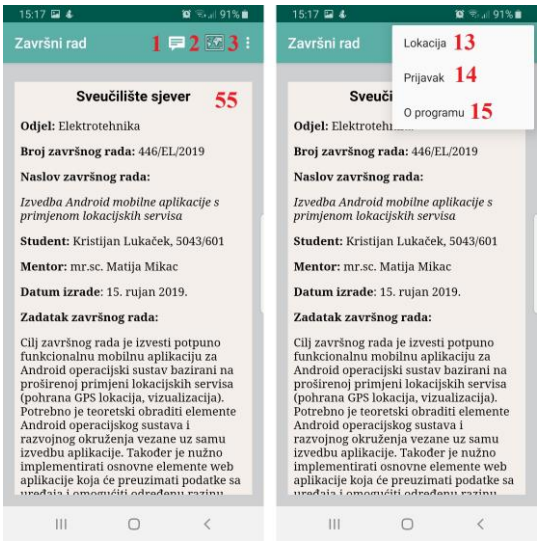
Slika 23: Aktivnost liste povijesti prijava i aktivnost otvorene stare prijave

Na prvoj slici lijevo je vidljiva aktivnost koja sadrži listu polje broj 48 koja ima sve prethodne prijave kronološki poredane i identificirane vremenom kada su poslane. Polje 49 prikazuje jednu od stavki te liste. Na obje aktivnosti slike 23 se vide standardne navigacijske tipke 1, 2 i 3. Pošto ove aktivnosti nisu dostupne kad je korisnik odjavljen izgled izbornika je isti kao kod svih drugih aktivnosti koje su dostupne kad je korisnik prijavljen na poslužitelj. Ako se lista popuni tako da nije moguće sve stavke staviti na ekran može se pomicati gore dolje. Prilikom klika na neku od stavki kao što je stavka 49 se otvara aktivnost koja prikazuje tu prijavu. Slike koje to prikazuju su druga, treća i zadnja slika s lijeva na slici 23. Polje 50 prikazuje slike koje su se učitale s poslužitelja. To je lista koja se može povlačiti lijevo odnosno desno da bi se prikazale slike koje ne stanu na ekran. U ovom primjeru ima šest slika, samo su dvije vidljive s time da je druga tek djelomično vidljiva. Polje prikazuje vrijeme prijave koje se automatski generira slanjem i

obavezno je. Polje 51 prikazuje lokaciju prijave ako postoji i sadrži standardni format lokacije i adrese koji je u ostatku mrežnog dijela aplikacije. Polje 52 prikazuje tekst koji korisnik šalje na poslužitelj. U ovom primjeru to je dio standardnog lorem ipsum teksta. Polje 53 prikazuje odgovor administratora djela aplikacije koji se vrti na poslužitelju. Tu je nekoliko odlomaka lorem ipsum teksta. Kao što je vidljivo na slici 23 cijeli sadržaj ne stane u prozor pa je prikazano u tri dijela. Makar ni to nije cijeli sadržaj prijave jer je lorem ipsum tekst dosta dugačak. Dalje ako postoji lokacija se prikazuje karta na polju 54 i koja je pomaknuta na lokaciju prijave, približena i na lokaciji se nalazi marker. Ako lokacija ne postoji se to polje i karta skriju programski. Klikom na marker se prikazuje tekst „LOKACIJA PRIJAVE“.

3.2.8. Podaci o programu

Aktivnost s slike 24 je implementirana u `Oprog_Activity.java` datoteci. Grafički elementi su definirani u `activity_oprog_.xml` i `content_oprog_.xml` datotekama.



Slika 24: Aktivnost O programu

Aktivnost sadrži kratke informacije o aplikaciji. Informacije o aplikaciji su pokazane u `WebView`-u (55). U `WebView`-u se prikazuje fiksni HTML sadržaj koji je unaprijed definiran i ne može se mijenjati.

3.2.9. Definicija i upotreba standardnih elemenata grafičkog korisničkog sučelja

S obzirom na to da su aplikacije namijenjene krajnjim korisnicima, vrlo je bitno da aplikacije uključuju grafičko korisničko sučelje koje se izrađuje prema nekim manje ili više predefiniranim pravilima. U Android API definirani su neki od elementarnih dijelova sučelja – gumbi, ispis teksta, polja za unos, prikaz slikovnih zapisa i slično. Rad sa svim elementima je unificiran i svodi se na nekoliko jasnih koraka.

U aplikaciji završnog rada se za izradu korisničkog sučelja koriste osnovni elementi tipa `TextView` za ispis teksta na ekran, `EditText` za unos i manipulaciju tekstom, `Button` i `ImageButton` kao elementi koji predstavljaju gumbe koji se mogu dodatno ukrasiti slikovnim zapisima. Nadalje, koriste se GM, izbornik, `RecyclerView` liste, `ListView` liste, `ImageView` pogled, itd.

U najjednostavnijem pristupu programiranju (u razvoju aplikacije nisu korišteni posebni razvojni predlošci) unutar razreda pojedine aktivnosti deklariraju se varijable/objekti koje odgovaraju pojedinim elementima sučelja. Nakon toga se u `onCreate` metodi razreda aktivnosti te varijable povezuju sa elementima sučelja koristeći njihove identifikatore.

Primjer dohvaćanja `TextView` elementa:

```
1  TextView textView = (TextView) findViewById(R.id.id_textView);
```

Za ulazak u neku novu aktivnost (ponavljamo, eng. *Activity*, predstavlja ekran ili prozor korisničkog sučelja) i gašenje trenutne aktivnosti koristi se tzv. namjera (eng. *Inent*) – za to se koristi razred `Intent` – primjer stvaranja namjere i pokretanje aktivnosti (prikazivanje prozora) prikazuje kod u nastavku:

```
1  Intent inte= new Intent(Login_Activity.this, Oprog_Activity.class);
2  Intent inte = new Intent(Login_Activity.this, Oprog_Activity.class);
3  startActivity(inte);
```

Jednostavni elementi korisničkog sučelja dovoljni su za osnovnu interakciju s korisnikom. No, kad postoji potreba za prikazom veće količine podataka, uobičajeno je koristiti prikaz u obliku liste. Zapravo su liste kompleksni elementi sučelja koji sadrže retke i dodatne funkcionalnosti za optimizaciju prikaza. Također, za korištenje lista potrebno je definirati i skup podataka (npr. polja ili vektore i podatkovne liste), ali i, kako API traži, takozvane adaptere koji automatiziraju vezu sučelja (prikaza) i podataka.

U ovoj aplikaciji se koriste dva tipa lista. Prvi su `RecyclerView` liste (razredi: `Prijava_Activity`, `PovPrijava_Activity` i `PrijavaSpr_Activity`) , a drugi klasične `ListView` liste (razredi: `Lokacije_Pov_Activity`, `Lokacije_PovKret_Activity` i `Lokacije_comp`). `RecyclerView` liste su noviji oblik lista. Njihova prednost je da za razliku od drugih lista ne inicijaliziraju na početku sve elemente liste (svaki element liste je zasebni pogled odnosno grafičko sučelje) nego samo one koji će biti vidljivi na ekranu, a one koji se ne vide ima pohranjene. Dodatno omogućava manipulaciju s listom bez da se mora ponovno učitavati cijela lista nego se samo izmjeni željeni element ili elementi. Sve navedeno za posljedicu ima daleko efektivnije korištenje resursa uređaja kao što su procesor i baterija. Za baratanje listama se moraju implementirati adapteri. Oni upravljaju sučeljem koje odgovara

samo jednoj stavki liste. Jedino jednostavne liste bez Izdvojenog (programski definiranog) izgleda sučelja ne trebaju nužno adapter.

Kod ispod je direktno iz aplikacije. Prve dvije linije definiraju objekt `RecyclerView` liste i dinamičko polje u koju će se pohranjivati `URI` adrese slika koje će se učitati u pomičnoj listi.

```
1  RecyclerView recyclerView;
2  ArrayList<Uri> odabraneSlike = new ArrayList<>();
```

Naredbe ispod implementiraju listu `recyclerView` koja će imati slike koje su uz prijavu ako postoje. Ove naredbe se implementiraju u `onCreate` funkciji. Ta funkcija se automatski pokreće kod stvaranja aktivnosti. U njoj se obično dohvaćaju elementi grafičkog sučelja aktivnosti. Prvo dodjeljuje objektu liste identifikatora grafičkog objekta liste.

```
3  recyclerView = findViewById(R.id.rv1);
```

Postavlja horizontalno sučelje u ovoj aktivnosti.

```
4  LinearLayoutManager horizontalLayoutManager
5  = new LinearLayoutManager(Prijava_Activity.this, LinearLayoutManager.HORIZONTAL,
   false);
```

Prethodno definirano sučelje dodjeljuje listi `recycleView`.

```
6  recyclerView.setLayoutManager(horizontalLayoutManager);
```

Postavlja adapter koji će se brinuti za funkcioniranje liste. Dodjeljuje mu podatke kojima će popunjavati listu npr.

```
7  adapter = new MojRecyclerViewAdapter1(this, odabraneSlike);
```

Prijavljuje `listener` adapteru koji će omogućiti brisanje dodanih slika dugim klikom na sliku.

```
8  adapter.setLongClickListener(this);
```

Da bi se pokrenulo osvježavanje liste se koriste naredba ispod. Ona odlazi u adapter s setom podataka spremljenim u listu. Taj će set podataka biti korišten za ponovno formiranje liste.

```
9  adapter.notifyDataSetChanged();
```

Primjer adaptera se nalazi u prilogu u odjeljku `MojRecyclerViewAdapter1.java`.

3.2.10. Dohvaćanje i vizualizacija lokacije

```
1  public GoogleMap mMap;
2  @SuppressLint("MissingPermission")
3  @Override
4  protected void onCreate(Bundle savedInstanceState) {
5     super.onCreate(savedInstanceState);
6     setContentView(R.layout.activity_lokacije__glavna_);
7     MapFragment mapFragment = (MapFragment)
      getFragmentManager().findFragmentById(R.id.map1);
8     mapFragment.getMapAsync(this);
9     LocationManager lm = (LocationManager)
      this.getSystemService(Context.LOCATION_SERVICE);
10    LocationListener ll = new LocationListener() {
11       public void onLocationChanged(Location loc) {
12        lat = Double.valueOf(loc.getLatitude());
13        lon = Double.valueOf(loc.getLongitude());
14        vis = Double.valueOf(loc.getAltitude());
15        lat_pr = String.valueOf(loc.convert(loc.getLatitude(),loc.FORMAT_SECONDS));
16        lon_pr = String.valueOf(loc.convert(loc.getLongitude(),loc.FORMAT_SECONDS));
17        vis_pr = String.valueOf(loc.getAltitude());
18        acc = loc.getAccuracy();
19        bear = loc.getBearing();
20        prov = loc.getProvider();
21        speed = loc.getSpeed();
22        fix_time = loc.getTime();
23        fix_pass_time = loc.getElapsedRealtimeNanos();
```

```
24        LatLng moja_lokacija = new LatLng(lat, lon);
25        mMap.clear();
26        mMap.addMarker(new MarkerOptions().position(moja_lokacija).title("TRENUTNA
          LOKACIJA"));
27        mMap.moveCamera(CameraUpdateFactory newLatLngZoom(moja_lokacija,19));
28      public void onStatusChanged(String provider, int status, Bundle extras) {}
29      public void onProviderEnabled(String provider) {}
30      public void onProviderDisabled(String provider) {}};
31      if (ActivityCompat.checkSelfPermission(this,
32      Manifest.permission.ACCESS_FINE_LOCATION)
33      != PackageManager.PERMISSION_GRANTED
34      && ActivityCompat.checkSelfPermission(this,
35      Manifest.permission.ACCESS_COARSE_LOCATION)
36      != PackageManager.PERMISSION_GRANTED) {
37         return;}
38      lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 100, 0, ll);}
39  @Override
40  public void onMapReady(GoogleMap googleMap) {
41     mMap = googleMap;
42     postaviKartu();}
43  void postaviKartu() {
44     mMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);}
45  public void klikPokazi_RUTU(View v) {
46     ArrayList<LatLng> koordList = new ArrayList<LatLng>();
47     String pom1 = "prazno";
48     LatLng moja_lokacija_zadnja = new LatLng(0, 0);
49     LatLng moja_lokacija_prva = new LatLng(0, 0);
50     mMap.clear();
51     SharedPreferences sp = getSharedPreferences("SPREMANJE_kretanje:",
       MODE_PRIVATE);
52     pom1 = sp.getString("lok_up", "prazno");
53     if (pom1 != "prazno") {
54        lok_up = Integer.parseInt(pom1);
55        LatLngBounds.Builder builder = new LatLngBounds.Builder();
56        for (int i = 1; i <= lok_up; i++) {
57           lat_lon = sp.getString("lat_lon" + Integer.toString(i), "prazno");
58           if (lat_lon != "prazno") {
59              String racun[] = lat_lon.split("\\|");
60              lat = Double.valueOf(racun[0]);
61              lon = Double.valueOf(racun[1]);
62              koordList.add(new LatLng(lat, lon));
63              builder.include(new LatLng(lat, lon));
64              if(i==lok_up){ moja_lokacija_zadnja = new LatLng(lat, lon); }
65              if(i==1){ moja_lokacija_prva = new LatLng(lat, lon); }}}
66     LatLngBounds podrZoma = builder.build();
67     mMap.clear();
68     mMap.addMarker(new MarkerOptions().position(moja_lokacija_zadnja)
69     .title("ZADNJA LOKACIJA").icon
70     (BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_RED)));
71     mMap.addMarker(new MarkerOptions().position(moja_lokacija_prva)
72     .title("PRVA LOKACIJA").icon
73     (BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_AZURE)));
74     PolylineOptions polylineOptions = new
       PolylineOptions().width(5).color(Color.RED);
75     polylineOptions.addAll(koordList);
76     mMap.addPolyline(polylineOptions);
77     mMap.moveCamera(CameraUpdateFactory.newLatLngBounds(podrZoma, 50));
78     }else{Toast.makeText(getApplicationContext(), "NEMATE RUTU",
       Toast.LENGTH_SHORT).show();}}
```

U ovom odjeljku koda je na primjeru iz aktivnosti `Lokacije_Glavna_Activity` prikazana vizualizacija lokacije. Linija 1 je deklaracija objekta `GoogleMap`. Veći dio koda se izvršava unutar `onCreate` funkcije. Na liniji 6 se deklarira koja XML datoteka koja definira izgled aktivnosti. Dalje na liniji 7 i 8 se deklarira fragmenta mape gdje će biti karta i tom fragmentu se dodjeljuje XML element sučelja. Kod na liniji 9 pokreće LM. Linija 10 definira LL koji će čekati promjenu lokacije. Na liniji 11 se otvara funkcija `onLoactionChanged` koja se automatski

poziva prilikom promjene lokacije. Ta se funkcija automatski generira zajedno s funkcijama `onStatusChanged`, `onProviderEnabled i onProviderDisabled`, te funkcije se vide u linijama 28 do 30. Slušanje promjene lokacije je omogućeno na liniji 38. Prva vrijednost funkcije `requestLocationUpadtes` je lokacijski servis koji se koristi, druga vrijednost je vrijeme čekanja u milisekundama, treća minimalna promjena udaljenosti potrebna za detekciju lokacije i zadnja je koji LL se koristi za osluškivanje promjene. Moguće je i onemogućiti slušanje promjene lokacije to se radi s naredbom:

```
1. lm.removeUpdates(ll);.
```

Naredbe 12 do 14 dohvaćaju vrijednost koordinata u `double` brojevnom obliku. Na linijama 15 do 17 se prikazuje dohvat koordinata lokacije i pretvorba formata koordinata koji je prikazan u odjeljku *3.2.1.* Linije 18 do 22 dodjeljuju varijablama dodatne vrijednosti kao preciznost lokacije, smjer gibanja i brzinu (ako se uređaj kreće), starost lokacije i lokacijski servis. Kod na liniji 25 čisti kartu od prethodnih objekata. Te kod na liniji 26 postavlja markera na dohvaćenu koordinatu i postavlja natpis koji će se vidjeti ako se klikne na markera. Linija koda 27 pomiče kameru na lokaciju postavljenog markera i postavljaju maksimalno povećanje pogleda. Kod na linijama 31 do 37 se automatski generira po preporuci AS. Predstavlja if provjeru posjedovanja svih potrebnih dozvola za dohvaćanje lokacije. Ako dozvole nisu date naredba na liniji 37 izlazi iz funkcije. To je dodano da bi dohvaćanje lokacije pomoću LL moglo raditi.

Kada je karta spremna se automatski pokreće funkcija `onMapReady` i `postaviKartu` na liniji 40 do 44. Te funkcije postavljaju objekt i tip karte.

Funkcija na linijama 45 do 78 služi za prikazivanje spremljene rute (ako je ruta snimljena). Linije koda 46 do 49 inicijaliziraju početne vrijednosti varijabli. Karta se čisti na liniji 50. *SharedPreferences* definira datoteku u koju je spremljena ruta (51 i 52) i postavlja početnu vrijednosti koju će *SharedPreferences* imati ako je ona prazna. Koda na linijama 53 do 77 prikazuje if petlju koja se izvršava ako *SharedPreferences* nije prazna. Linija 78 prikazuju else petlju koja se izvršava ako je *SharedPreferences* prazna. Taj kod je jednostavna poruka koja obavještava da nema rute. Linija 54 dodjeljuje vrijednost SharedPreference varijable drugoj varijabli i konvertira ju u `integer` format, a linija 55 definira graditelja objekta koji će pohraniti sve koordinate i temeljem njih odrediti okvire pogleda rute. Na liniji 56 počinje for petlja koja za svaku vrijednosti od 1 do vrijednosti koja odgovora broju lokacije u ruti uzima podatak iz *SharedPreference*s na liniji 57. Potom na liniji 58 petlja if provjerava ako postoji podatak o lokaciji, ako postoji na liniji 59 se rastavlja u listu koja ima 2 polja. Prije je definirano da je svaka latituda i longituda spojena u jedan `string` gdje su međusobno odvojene znakovima „\\|“.

60 i 61 prikazuju dodjeljivanje stavki liste varijablama u `double` brojevnom formatu. Listi koordinata se na liniji 62 dodaju te koordinate i onda se na liniji 63 dodaju i graditelju. Pošto se na polazištu i odredištu rute nalaze markeri ih je potrebno definirati pa se na linijama 64 i 65 vrši provjera da li su to prva ili zadnja koordinata i po potrebi se varijablama dodjeljuje ta vrijednost.

Izvan for petlje se na liniji 66 gradi područje približavanja pogleda iz prethodno punjenog graditelja. Opet se na liniji 67 čisti karta. Na linijama 68 do 73 se dodaju markeri ishodišta i odredišta. Markeri imaju drugačije postavke boje i drugi natpis. Linija 74 definira postavke polilinije koja će spajati pojedine lokacije rute. Linija 75 dodaje listu koordinata objektu polilinija i onda se u liniji 76 te linije crtaju na kartu. Na liniji 77 se pogled kamera na karti pomiče na geometrijsku sredinu rute definiranu pomoću graditelja i objekta područja približavanja s linije 66. Na kraju se definira da je područje približavanja odmaknuto 50 piksela od ruba područja približavanja da se mogu lijepo vidjeti sve lokacije i markeri (da ništa ne izgleda „odrezano“).

U aktivnosti usporedbe lokacije se koriste neke od mogućnosti da bi se uspoređivalo dvije lokacije i njihove odnose. Ti dijelovi koda se nalaze niže.

```
1  double bear1 = dest1.bearingTo(dest2);
2  float dist = dest1.distanceTo(dest2);
3  if(bear1 < 0){bear1 = bear1 +360;}
4  if(bear2 < 0){bear2 = bear2 +360;}
5  if(vis1 > vis2){raz_vis_s=String.valueOf(vis1-vis2);}
6  if(vis2 > vis1){raz_vis_s=String.valueOf(vis2-vis1);}
```

Linija 1 i 2 prikazuju funkciju koja vraća udaljenost dvije koordinate. Na linijama 3 i 4 se prikazuje dohvaćanje azimuta dvije lokacije jedne prema drugoj i provjera azimuta. Ako je azimut negativan se doda 360 i time se dobi format azimuta 0° do 360°. Linije 5 i 6 prikazuju izračun razlike visina dvije koordinate. To su samo neki od podataka koji se mogu dohvatiti. Upravo su ti podaci bili najzanimljiviji u ovoj aplikaciji.

Geokoder omogućava pretvaranje neke koordinate u uličnu adresu ako ona postoji za te koordinate u nekoj bazi podataka ili obrnuto. Da bi geokoder radio mora ga se prvo definirati. Da bi se dohvatilo adresu iz koordinate se treba naredbu koja vrši geokodiranje staviti u try catch petlju. To se radi jer često geokoder ne može vratiti podatke ako ne postoji definirana adresa za neke koordinate, primjerice usred neke šume ili kad pukne Internet konekcija koja treba vratiti podatak za geokodera. Sve operacije vezane uz geokodiranje se odvijaju u `onCreate` funkciji specifično u `onLocationChanged` funkciji lokacijskog servisa.

Primjer Geokodera iz izvedene aplikacije:

```
1  Geocoder geocoder;
2  geocoder = new Geocoder(getApplicationContext(), Locale.getDefault());
```

```
3  double latitude = loc.getLatitude();
4  double longitude = loc.getLongitude();
5  try {
6    adrese = geocoder.getFromLocation(latitude, longitude, 1);
7    adresa = adrese.get(0).getAddressLine(0);
8    zupanija = adrese.get(0).getAdminArea();
9  } catch (IOException e) {
10 e.printStackTrace();}
```

Linija 1 definira objekt geokodera. Na liniji 2 se formira funkcija sustava Geocoder te joj se dodaje kontekst aplikacije. Linije 3 i 4 dohvaćaju koordinate lokacije. Geokoder se radi stabilnosti odrađuje u try catch petlji na linijama 5 do 10. Kod na liniji 6 je funkcija koja dohvaća listu geokodera za definirane koordinate koja sadrži sve podatke koje poslužitelj vraća na uređaj. Geokoder vraća podatke adrese kao listu čija polja čine pojedine elemente neke lokacije kao što su adresa ulice, kućni broj, regija, županija i država Problem leži u tome da geokoder vraća adrese neke koordinate kao pojedina polja neke liste. Svako polje sadrži neki element adrese lokacije. To bi bilo točno da se to radi o SAD-u ali u Hrvatskoj taj sustav nije toliko precizan. U SAD-u u jedno polje je pohranjena ulična adresa, u drugo kućni broj, u sljedeće poštanski broj, pa okrug ili grad, savezna država, a zadnje da je lokacija unutar granica SAD-a. Kod nas je u jedno polje nagurana ulična adresa, kućni broj, grad ili općina te poštanski broj, a u drugom je polju opet grad ili općina, u zadnjim županija i država. Tako da je očigledno da isti standard ne vrijedi za različite države pa treba obratiti pažnju na to. Druga stvar koja se može javiti s geokoderima je da preciznost dohvaćanja adrese ili koordinata može varirati ovisno o državi u kojoj se nalaziti jer podaci nisu jednako precizno definirani za sve države. U ovom primjeru radi toga koriste samo dva polja (linije 7 i 8). Na liniji 7 se varijabli dodjeljuju podaci ulične adrese, a varijabli na polju 8 podaci županije i države. Te podatke onda dalje koristimo za prikaz u aktivnosti i slanje na poslužitelj. Linije 9 i 10 prikazuju dohvat iznimke odnosno greške prilikom dohvaćanja podataka geokodera.

U aktivnostima „`Prijava_Activity`“ i „`PrijavaSpr_Activity`“ postoji problem. Taj problem je da se aktivnosti mogu pomicati gore dolje što znači da i klik na prazni prostor u aktivnosti izaziva neku reakciju. To znači da će gestu kojim će zumirati ili pomicati kartu interpretirati kao gestu za pomicanje cijelog prozora. Pa je stvoren prozor koji se postavlja točno preko fragmenta karte i onda klikom na taj prozor `listener` ispod daje naredbu da svaka gesta u tom prozoru bude zanemarena za ostatak prozora nego samo vrijedi za kartu.

```
1  prozMapPogled.setOnTouchListener(new View.OnTouchListener(){
2     @Override
3     public boolean onTouch(View view, MotionEvent motionEvent){
4       switch (motionEvent.getAction()) {
5         case MotionEvent.ACTION_UP:
6         view.getParent().requestDisallowInterceptTouchEvent(false);
7       break;
8       default:
9         view.getParent().requestDisallowInterceptTouchEvent(true);}
```

```
10      return false;}})
```

Linija 1 prikazuje funkciju koja pokreće `listener` prilikom dodira Ta se funkcija pokreće u `onCreate` funkciji. Funkcija `onTouch` (linija 3) se poziva kod svakog dodira na pogled. Nakon toga se ulazi u switch case petlju na linijama 4 do 10. Ako se desi gesta pomaka ulazi u petlju i provjerava kakav je pomak. Nakon toga na linijama 6 i 8 tu gestu blokira ili ne blokira za glavni prozor aktivnosti. Kad se desio pomak na karti ga sad karta može registrirati bez miješanja glavnog ekrana

3.2.11. Pohrana podataka

Prvi primjer pohrane podataka je pohrana u bazu podataka. Pohrana u bazu podataka se odrađuje u Java datoteci `MojaBP.java` u nastavku slijedi cijeli kod te Java datoteke i objašnjenje.

```
1   package com.unin.kl.zavrsni_rad_kl;
2   import android.content.ContentValues;
3   import android.content.Context;
4   import android.database.Cursor;
5   import android.database.sqlite.SQLiteDatabase;
6   public class MojaBP extends android.database.sqlite.SQLiteOpenHelper {
7       private static final String DATABSE_IME_LOK = "lokacija_bp.db";
8       private static final String TABLICA_IME_LOK = "lokacija_tablica";
9       private static final String STUPAC_ID_LOK = "id";
10      private static final String STUPAC_LAT_LOK = "latituda";
11      private static final String STUPAC_LON_LOK = "longituda";
12      private static final String STUPAC_VIS_LOK = "visina";
13      private static final String STUPAC_PREC_LOK = "preciznost";
14      private static final String STUPAC_SMJER_LOK = "smjer";
15      private static final String STUPAC_IZVOR_LOK = "izvor";
16      private static final String STUPAC_BRZINA_LOK = "brzina";
17      private static final String STUPAC_VRIJEME_LOK = "vrijeme";
18      private static final String STUPAC_VRIJEME_F_LOK = "vrijeme_f";
19      private static final String STUPAC_STAROST_LOK = "starost";
20      private static final String STUPAC_VRIJEME_SPREM_LOK = "datum";
21      private static final String TABLICA_IME_RUT = "rute_tablica";
22      private static final String STUPAC_ID_UNOSA_RUT = "id_unos_rut";
23      private static final String STUPAC_ID_RUTE_RUT = "id_rut";
24      private static final String STUPAC_ID_LOKACIJE_RUT = "id_lok";
25      private static final String STUPAC_TEKST_RUT = "koordinate";
26      private static final String STUPAC_DATUM_RUT = "datum";
27      public static final String TABLE_CREATE_LOK = "create table
    "+TABLICA_IME_LOK+" ("+STUPAC_ID_LOK+
28              " integer primary key autoincrement, "+STUPAC_LAT_LOK+" text,
    "+STUPAC_LON_LOK+" text, "
29              +STUPAC_VIS_LOK+" text, "+STUPAC_VRIJEME_SPREM_LOK+" text,
    "+STUPAC_PREC_LOK+" text, "
30              +STUPAC_SMJER_LOK+" text, "+STUPAC_IZVOR_LOK+" text,
    "+STUPAC_BRZINA_LOK+" text, "
31              +STUPAC_VRIJEME_LOK+" text, "+STUPAC_VRIJEME_F_LOK+" text,
    "+STUPAC_STAROST_LOK+" text  );";
32      public static final String TABLE_CREATE_RUT = "create table
    "+TABLICA_IME_RUT+" ("+STUPAC_ID_UNOSA_RUT
33              +" integer primary key autoincrement, "+STUPAC_ID_RUTE_RUT+" integer,
    "
34              +STUPAC_ID_LOKACIJE_RUT+" integer, " +STUPAC_TEKST_RUT+" text,
    "+STUPAC_DATUM_RUT+" text );";
35      public MojaBP(Context context) {
36          super(context, DATABSE_IME_LOK, null, 1);
37          SQLiteDatabase bp = this.getWritableDatabase();}
38      @Override
39      public void onCreate(SQLiteDatabase bp) {
40          bp.execSQL(TABLE_CREATE_LOK);
41          bp.execSQL(TABLE_CREATE_RUT);}
42      @Override
```

```
43      public void onUpgrade(SQLiteDatabase bp, int oldVersion, int newVersion) {
44          String UPIT1 = "DROP TABLE IF EXISTS "+TABLICA_IME_LOK;
45          bp.execSQL(UPIT1);
46          String UPIT2 = "DROP TABLE IF EXISTS "+TABLICA_IME_RUT;
47          bp.execSQL(UPIT2);
48          this.onCreate(bp);}
49  public boolean spremiPodatke_LOK(String lat_s, String lon_s, String vis_s, String
    vrijeme_sprem, String acc_s, String bear_s, String prov_s, String speed_s, String
    fix_time_s, String fix_time_s_f, String fix_pass_time_s){
50          SQLiteDatabase bp = this.getWritableDatabase();
51          ContentValues cv = new ContentValues();
52          cv.put(STUPAC_VRIJEME_SPREM_LOK, vrijeme_sprem);
53          cv.put(STUPAC_LAT_LOK, lat_s);
54          cv.put(STUPAC_LON_LOK, lon_s);
55          cv.put(STUPAC_VIS_LOK, vis_s);
56          cv.put(STUPAC_PREC_LOK, acc_s);
57          cv.put(STUPAC_SMJER_LOK, bear_s);
58          cv.put(STUPAC_IZVOR_LOK, prov_s);
59          cv.put(STUPAC_BRZINA_LOK, speed_s);
60          cv.put(STUPAC_VRIJEME_LOK, fix_time_s);
61          cv.put(STUPAC_VRIJEME_F_LOK, fix_time_s_f);
62          cv.put(STUPAC_STAROST_LOK, fix_pass_time_s);
63          long ishod = bp.insert(TABLICA_IME_LOK, null, cv);
64          if(ishod == -1){
65              return false;
66          }else{
67              return true;}}
68      public boolean spremiPodatke_RUT(String lat_lon, String vrijeme, Integer
    id_lok_rut, Integer id_rut){
69  Definicija SQLiteDatabase varijable.
70          SQLiteDatabase bp = this.getWritableDatabase();
71          ContentValues cv = new ContentValues();
72          cv.put(STUPAC_TEKST_RUT, lat_lon);
73          cv.put(STUPAC_DATUM_RUT, vrijeme);
74          cv.put(STUPAC_ID_RUTE_RUT, id_rut);
75          cv.put(STUPAC_ID_LOKACIJE_RUT, id_lok_rut);
76          long ishod = bp.insert(TABLICA_IME_RUT, null, cv);
77          if(ishod == -1){
78              return false;
79          }else{
80              return true;}}
81      public Cursor dohvatiPodatke_LOK(){
82          SQLiteDatabase bp = this.getWritableDatabase();
83          Cursor podaci_LOK =  bp.rawQuery("select * from "+TABLICA_IME_LOK,null);
84          return podaci_LOK;}
85      public Cursor dohvatiPodatke_LOK_pojedino(String id){
86          SQLiteDatabase bp = this.getWritableDatabase();
87          Cursor podaci_LOK =  bp.rawQuery("select * from "+TABLICA_IME_LOK+ " where
    id=?",new String[] {id});
88          return podaci_LOK;}
89      public Cursor dohvatiPodatke_RUT_list(){
90          SQLiteDatabase bp = this.getWritableDatabase();
91          Cursor podaci_RUT =  bp.rawQuery("select DISTINCT("+STUPAC_DATUM_RUT+"), "
92                  +STUPAC_ID_RUTE_RUT+" from "+TABLICA_IME_RUT,null);
93          return podaci_RUT;}
94      public Cursor dohvatiPodatke_RUT_pojedino(String id1, String id2){
95          SQLiteDatabase bp = this.getWritableDatabase();
96          Cursor podaci_RUT =  bp.rawQuery("select * from " + TABLICA_IME_RUT + "
    where id_rut=? " +
97                  "and id_lok=?",  new String[] {id1,id2});
98          return podaci_RUT;}
99      public Cursor max_id_rut(){
100         SQLiteDatabase bp = this.getWritableDatabase();
101         Cursor max_id_RUT =  bp.rawQuery("select max(id_rut) from
    "+TABLICA_IME_RUT,null);
102         return max_id_RUT;}
103     public Cursor max_id_rut_lok(){
104         SQLiteDatabase bp = this.getWritableDatabase();
105         Cursor max_id_RUT_LOK =  bp.rawQuery("select max(id_lok) from
    "+TABLICA_IME_RUT,null);
```

```
106         return max_id_RUT_LOK;}
107     public void obrisiPodatke_LOK(){
108         SQLiteDatabase bp = this.getWritableDatabase();
109         String UPIT1 = "DROP TABLE IF EXISTS "+TABLICA_IME_LOK;
110         bp.execSQL(UPIT1);
111         bp.execSQL(TABLE_CREATE_LOK);}
112     public void obrisiPodatke_RUT(){
113         SQLiteDatabase bp = this.getWritableDatabase();
114         String UPIT2 = "DROP TABLE IF EXISTS "+TABLICA_IME_RUT;
115         bp.execSQL(UPIT2);
116         bp.execSQL(TABLE_CREATE_RUT);}
117     public Integer obrisiStavku_LOK(String id){
118         SQLiteDatabase bp = this.getWritableDatabase();
119         return bp.delete(TABLICA_IME_LOK, "ID = ?",
120                 new String[] {id});}
121     public Integer obrisiStavku_RUT(String id){
122         SQLiteDatabase bp = this.getWritableDatabase();
123         return bp.delete(TABLICA_IME_RUT, "id_rut = ?",
124                 new String[] {id});}
125     public Cursor max_id_rut_LOK2(String id){
126         SQLiteDatabase bp = this.getWritableDatabase();
127         Cursor max_id_RUT_LOK =  bp.rawQuery("select max(id_lok) from " +
    TABLICA_IME_RUT + " where id_rut=?",  new String[] {id}); return max_id_RUT_LOK;
128     }}
```

Linije 1 do 5 prikazuju sve knjižnice koje se koriste u ovoj datoteci. Na liniji 6 se deklarira javni razred i nasljeđuje `SQLiteOpenHelper` iz matičnih aktivnosti. Kod na linijama 7 do 26 deklarira varijable koje će se koristiti, a linije 27 do 34 prikazuju deklaraciju `string` varijabli koje sadrže SQL upite za primjerice formiranje tablice lokacija i ruta. Na linijama 35 do 37 se stavljaju podaci u konstruktora razreda. Funkcija `onCreata` u linijama 38 do 41 izvršava formiranje nove tablice ruta ili lokacija. Tu funkciju automatski generira AS, uz nju se automatski generira i funkcija `onUpgrade` (linije 43 do 48) koja se poziva prilikom osvježavanja podataka u tablicama. U njoj se odbacuje nova tablica lokacija ili ruta ako već postoji. Funkcija `spremiPodatke_LOK` između linija 49 i 67 pohranjuje vrijednost lokacije u bazu podataka. Kod na liniji 50 definira objekt baze podataka kao bazu podataka u koju se može upisivati. Na liniji 51 se deklarira objekt vrijednosti sadržaja. Linije 52 do 62 dodaju vrijednosti sadržaja dobivene vrijednosti iz matične aktivnosti redovima tablice lokacije. Naredba na liniji 63 dodjeljuje vrijednost generiranu upisom u bazu podataka, ta vrijednost govori o uspješnosti upisa u bazu podataka. Na linijama 64 do 67 se vrši if else provjera uspješnost, te vraća podatak o uspjehu ili neuspjehu. Funkcija `spremiPodatke_RUT` (linije 68 do 80) u bazu podataka. Radi na istom principu kao i prethodna funkcija. Funkcija `dohvatipodatke_LOK` koja se nalazi u linijama 81 do 84 dohvaća sve podatke iz tablice lokacija. Linija 82 deklarira objekt baze podataka u koji se može upisivati i čitati, linija 83 direktno izvršava upit koji je definiran u zagradi. Na liniji 84 je naredba koja vraća dohvaćene podatke u obliku kursor objekta. Funkcije koje su prikazana na linijama 94 do 106 su varijacije prethodno opisane funkcije. Funkcija `obrisiPodatke` u linijama 107 do 111 briše sve podatke iz tablice lokacija. Linija 109 formira `string` koji sadrži upit. Naredbe na linijama 110 i 111 izvršavaju prethodno definirane upite, 110 briše tablicu, 111 stvara novu praznu tablicu. Sljedeća funkcija radi isto samo za rute.

Funkcije: `obrisiStavku_LOK`, `obrisiStavku_RUT` i `max_id_rut_LOK2` brišu pojedine elemente tablica, odnosno vraćaju maksimalnu vrijednost lokacija jedne rute te su varijacija funkcionalnosti funkcije `spremiPodatke_LOK`. Da bi se pokrenule funkcije iz `MojaBP.java` se koriste naredbe poput ove:

```
1  mojaBP.spremiPodatke_RUT(lat_lon, vrijeme,id_lok_rut, id_rut);
```

Sljedeći primjer pohrane podataka je pohrana podataka u privatni direktorij aplikacije. On se odrađuje u nekoliko funkcija u Java datoteci `Prijava_Activity.java`, te funkcije su dane u nastavku. One dohvate sliku naprave datoteku s unikatnim imenom i pohrane sliku u memoriju.

```
1  private void izradiSliku() {
2  Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
3  if (takePictureIntent.resolveActivity(getPackageManager()) != null) {
4  File photoFile = null;
5  try {
6  photoFile = slikaFile();
7  } catch (IOException ex) {}
8  if (photoFile != null) {
9  Uri photoURI = FileProvider.getUriForFile(this,
10 "com.unin.kl.zavrsni_rad_kl.fileprovider",
11 photoFile);
12 takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, photoURI);
13 startActivityForResult(takePictureIntent, 2);}}}
14 private File slikaFile() throws IOException {
15 String datum = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
16 String strImeSlike = "JPEG_" + datum + "_";
17 File dirPohrana = getExternalFilesDir(Environment.DIRECTORY_PICTURES);
18 File slika = File.createTempFile(strImeSlike,".jpg",dirPohrana);
19 dirSlika = slika.getAbsolutePath();
20 return slika;}
```

Između linija 1 i 13 se nalazi funkcija koja definira u koju će se datoteku spremiti slika. Linija 2 prikazuje naredbu koja definira namjeru stvaranja slike. Namjera je način na koji Android javlja vanjskim aplikacijama da želi da one nešto odrade za aplikaciju. U ovom slučaju je to slikanje. Ako je slika napravljena se preko if provjere na liniji 3 ulazi u if petlju. Na liniji 4 se deklarira objekt tipa `File` koji će sadržavati memorijsku adresu gdje se sprema slika. U try catch petlji (linije 5 do 7) se `File` objektu dodaje adresa buduće slike. Dodaje se preko funkcije `slikaFile()`. Na linija 15 se dohvaća vrijeme spremanja slike, linija formira ime slike koja se sprema (dio imena je vrijeme spremanja). Linija 17 dohvaća adresu direktorija u koji aplikacija sprema slike. Na liniji 18 se formira privremena datoteka u koju će se spremiti sadržaj slike. Argumenti te funkcije su ime slike, ekstenzija odnosno tip slike i direktorij pohrane. Linija 19 dodaje sliku listi koja se kasnije koristi na `RecyclerView` listi. Naredba na liniji 20 vraća podatak o lokaciji stvorene datoteke prethodnoj funkciji. If provjera na liniji 8 provjerava ako je prethodno uspješno stvorena datoteka slike i vraćen podatak. Linije 9 do 11 dodaju `URI` adresu slike koju dohvaća i pružatelja datoteke. On služi da bi omogućio sigurno dijeljene direktorija asociranih s aplikacijom. Odnosno u njemu nalazi u kojem se direktoriju sprema slika. Linije 12 i 13 onda dohvaćaju sliku i pohranjuju ju u direktorij.

Zadnji oblik pohrane podataka koji se koristi uz baze podataka i direktorije je *SharedPreferences*. To je jednostavni oblik zapisa podatka u XML datotekama koje se nalaze u privatnoj datoteci aplikacije. Podaci se zapisuju kao ključ/vrijednost par. Primjer koji se nalazi niže je uzet iz aktivnosti postavki mrežnog dijela izvedene aplikacije. Prikazuje jednostavno čitanje i pisanje *SharedPreferences* pohrane aplikacije.

```
1   SharedPreferences sp = getSharedPreferences("SPREMANJE_POSTAVKI", MODE_PRIVATE);
2   p_ip=sp.getString("IP_server", "127.0.0.1");
3   SharedPreferences.Editor sped = sp.edit();
4   sped.putString("IP_server", ip_add);
5   sped.commit();
```

Lija koda 1 prikazuje deklaraciju *SharedPreferences* objekta. Prvi argument funkcije je ime XML datoteke u koju se sprema, drugi element deklaracije je ovlast pristupa XML dokumentu (u ovom slučaju pristup dokumentu ima samo ova aplikacija). Na liniji 2 se prikazuje dohvaćanje sadržaja XML dokumenta. Prvi argument te funkcije je ključ za pristup podatku, a drugi je podatak koji će biti vraćen ako nije ništa upisano. Linija 3 prikazuje deklaraciju editora sadržaja *SharedPreferences*. Na liniji 4 se objektu *SharedPreferences* dodaje par ključ/vrijednost i linijom 5 se upisuje u XML dokument.

3.2.12. Mrežne operacije

Sve mreže operacije izvedene aplikacije se nalaze u `Konektor.java` datoteci koja se onda poziva iz drugih dijelova aplikacije. Ispod se nalazi primjer i opisi mrežnih operacija u „Konektoru“.

```
1   public class Konektor extends AsyncTask<String, Integer, String> {
2    Context kontekst;
3    String url_str = "";
4    String slika_str;
5    String rezultat = "";
6    String rezultat_s = "";
7    String linija = "";
8    String ip = "";
9    private Bitmap bitmap;
10   Uri uri;
11   String tip = "";
12   String korisnicko_ime, lozinka;
13   Konektor (Context ctx) { kontekst = ctx; }
14   SharedPreferences sp;
15   HttpURLConnection httpURLKonekcija;
16   OutputStream izlazniStream;
17   BufferedWriter bufferPisac;
18   String post_data;
19   InputStream ulazniStream;
20   BufferedReader bufferCitac;
21   URL url;
22   @Override
23   protected String doInBackground(String... parametar) {
24    sp = kontekst.getSharedPreferences("SPREMANJE_POSTAVKI",MODE_PRIVATE);
25    ip=sp.getString("IP_server", "127.0.0.1");
26    tip = parametar[0];
27    if(tip.equals("login")) {
28     url_str = "http://"+ip+"/zavrsnirad/login.php";
29     try {
30      korisnicko_ime = parametar[1];
```

```
31        lozinka = parametar[2];
32        url = new URL(url_str);
33        httpURLKonekcija = (HttpURLConnection)url.openConnection();
34        httpURLKonekcija.setRequestMethod("POST");
35        httpURLKonekcija.setDoOutput(true);
36        httpURLKonekcija.setDoInput(true);
37        izlazniStream = httpURLKonekcija.getOutputStream();
38        bufferPisac = new BufferedWriter(new OutputStreamWriter(izlazniStream, "UTF-
          8"));
39        post_data = URLEncoder.encode("korisnicko_ime","UTF-8")+
40         "="+URLEncoder.encode(korisnicko_ime,"UTF-8")+
41        "&"+URLEncoder.encode("lozinka","UTF-8")+
42         "="+ URLEncoder.encode(lozinka,"UTF-8");
43        bufferPisac.write(post_data);
44        bufferPisac.flush();
45        bufferPisac.close();
46        izlazniStream.close();
47        ulazniStream = httpURLKonekcija.getInputStream();
48        bufferCitac = new BufferedReader(new InputStreamReader(ulazniStream, "UTF-
          8"));
49        while((linija = bufferCitac.readLine())!= null) {
50        rezultat += linija;}
51        bufferCitac.close();
52        ulazniStream.close();
53        httpURLKonekcija.disconnect();
54        return rezultat;
55       } catch (MalformedURLException e) {e.printStackTrace();
56       } catch (IOException e) {e.printStackTrace();}}
57     return null;}
58    @Override
59    protected void onPreExecute() { }
60    @Override
61    protected void onPostExecute(String rezultat) {
62     if(tip.equals("login")) {
63      String[] podaci = new String[0];
64      String split[] = rezultat.split("//");
65     if (split[0].equals("uspjesno_login")) {
66      JSONArray jsonArray = null;
67      try {
68       jsonArray = new JSONArray(split[1]);
69       podaci = new String[jsonArray.length()];
70      for (int i = 0; i < jsonArray.length(); i++) {
71       podaci[i] = jsonArray.getString(i);}
72      } catch (JSONException e) {e.printStackTrace();}
73     Toast.makeText(kontekst, "Prijavljeni ste!",Toast.LENGTH_LONG).show();}
74     if (rezultat.equals("neuspjesno_login")){
75      Toast.makeText(kontekst, "Prijava nije uspjela!",Toast.LENGTH_LONG).show();}}}
76    @Override
77     protected void onProgressUpdate(Integer... values) {
       super.onProgressUpdate(values);}}
```

Odlomak koda na liniji 1 ima deklaraciju javnog `AsyncTask` razreda Konektor. Kod na linijama 2 do 12 deklarira različite varijable koje se ovdje koriste. Na liniji 13 se deklarira konstruktor razreda. Linije koda 14 do 21 sadrže deklaracije različitih Java objekata koji se koriste kasnije. Linije koda 23 do 57 prikazuju funkciju `doInBackground` ta funkcija se odrađuje u pozadini aplikacije i u njoj se rade sve operacije slanja i primanja podataka na poslužitelj. Ona prima iz matičnih aktivnosti `string` listu koja sadrži podatke s kojim će raditi. Linije koda 24 i 25 čitaju IP adresu poslužitelja pohranjenu u *SharedPreferences* pohrani. Nakon toga na liniji 28 pomoću IP adrese se stvara `string` format `URL` adrese poslužitelja te se pohranjuje u varijablu `URL` tipa koja se koristi za spoj (linija 32). Iz `string` liste se na liniji 26 vadi prvi podatak, podatak o tome koji je tip operacije u pitanju. Svaki tip operacije (prijava, slanje, registracija, brisanje

korisničkog računa itd.) ima svoju if petlju u kojoj se izvršava. Linija 27 prikazuje if provjeru odnosno ulazak u if petlju za prijavu na poslužitelj. U linijama 30 i 31 se podaci iz `string` liste dodaju varijablama (napravljeno radi preglednosti). Objekt `httpURLKonekcija` definira HTTP konekciju koja kao adresu koristi `URL` resurs. `HttpURLConnection` je funkcija koja se koristi za brzo i efikasno korištenje mrežnih funkcija Androida (linija 33). Na liniji 34 se definira tip upita koji se šalje kao POST upit, linije 35 i 36 definiraju konekciju kao izlaznu i ulaznu. Linija 37 definira tok podataka kao izlazni tok. Na liniji 38 i 39 se definira pisač koji će pisati znak na izlazni tok bajtova. Linije koda 40 do 42 prikazuje formiranje samog sadržaja upita. Upit se pohranjuje kao `string` podataka. Kodira se s naredbom `URLEncoder.encode`, pojedinim podacima se formira standardni POST upit. Linije koda 43 do 46 šalju taj upit i potom zatvaraju sve elemente izlazne konekcije. Ulazni tok podataka se definira na liniji 47, a na liniji 48 se definira čitač ulaznog toka podataka. U While petlji (linija 49) se dohvaćaju svi podaci poslani na klijenta te se dodaju `stringu` „rezultat“ koji se na liniji 54 vraća iz funkcije `doInBackground`. Na linijama 51 i 52 se zatvaraju svi još otvoreni elementi konekcije, a na 53 se zatvara sama konekcija. Linije koda 55 i 56 prikazuju catch dio try catch petlje u kojem se rade mrežne operacije. Na liniji 57 izlazi iz funkcije `doInBackground`. Funkcija `onPreExecute` na liniji 59 se automatski generira i pokreće se prije funkcije `doInBackground`. U ovom primjeru se u toj funkcije ne mora ništa raditi. Na linijama koda 61 do 75 je funkcija `onPostExecute`, ona iz prethodne funkcije dobiva podatke. If provjera na liniji 62 određuje koja zadaća će se izvršavati, u ovom slučaju prijava na poslužitelj. Linije koda 63 i 64 uzimaju podatak vraćen iz `doInBackground` funkcije i rastavljaju ga na dva dijela. Prvi dio je poruka o uspješnosti operacije, a drugi su podaci o korisničkom računu. Ako if upit na liniji 65 utvrdi da je sve odrađeno uspješno ulazi u if petlju (linije 65 do 73). U toj petlji prvo definira JSON listu na liniji 66. Potom ulazi u try catch petlju (linije 67 do 72) koja je potrebna za siguran rad s JSON objektima. JSON listi dodaje prethodni drugi dio dohvaćenog podatka. Linija 69 definira listu koja će sadržavati rastavljene elemente JSON liste. For petlja na linijama 70 i 71 utvrđuje veličinu JSON liste i svaki element liste pojedinačno dodaje listi „podaci“. Poslije toga na liniji 72 je catch dio try catch petlje. Na liniji 73 vraća korisniku poruku o uspjehu operacije. Ako operacija nije uspjela if petlja na liniji 74 do 75 vraća poruku o neuspjehu. Funkcija `onProgressUpdate` (linija 77) je prazna automatski generirana funkcija koja prati tijek operacija u prethodnim funkcijama. U ovom primjeru nije implementirana.

Kod slanja prijava se slike šalju odvojeno od ostatka prijave. Pretpostavka je da ako se zagube slike je manja šteta nego ako se izgubi ostatak sadržaja prijave. Primjer koda za slanje slika:

```
1   if (split[0].equals("uspjesno_prijava")) {
```

```
2     JSONArray jsonArray = null;
3     try {
4      jsonArray = new JSONArray(split[1]);
5      podaci = new String[jsonArray.length()];
6      for (int i = 0; i < jsonArray.length(); i++) {
7       podaci[i] = jsonArray.getString(i);}
8     } catch (JSONException e) {e.printStackTrace();}
9     if (slika_p.equals("da")){
10     String bezZagada = slike.substring( 1, slike.length() - 1);
11     String[] lista = bezZagada.split( ", ");
12     for (int i = 0; i < lista.length; i++) {
13      url_str = "http://" + ip + "/zavrsnirad/slike.php";
14      String st = "Šaljem "+(i+1)+"/"+lista.length+" slika.";
15      ((Prijava_Activity) kontekst).slanje(st);
16      try {
17       uri = Uri.parse(lista[i]);
18       bitmap = BitmapFactory.decodeFile(uri.getPath());
19       ByteArrayOutputStream stream = new ByteArrayOutputStream();
20       bitmap = MediaStore.Images.Media.getBitmap(
21       kontekst.getContentResolver(),Uri.parse(lista[i]));
22       bitmap.compress(Bitmap.CompressFormat.JPEG, 100, stream);
23       bitmap.recycle();
24       byte[] array = stream.toByteArray();
25       slika_str = Base64.encodeToString(array, 0);
26       url = new URL(url_str);
27       httpURLKonekcija = (HttpURLConnection) url.openConnection();
28       httpURLKonekcija.setRequestMethod("POST");
29       httpURLKonekcija.setDoOutput(true);
30       httpURLKonekcija.setDoInput(true);
31       izlazniStream = httpURLKonekcija.getOutputStream();
32       bufferPisac = new BufferedWriter(new OutputStreamWriter(
33       izlazniStream, "UTF-8"));
34       post_data = URLEncoder.encode("slika_str", "UTF-8") +
35        "=" + URLEncoder.encode(slika_str, "UTF-8") +
36        "&" + URLEncoder.encode("ime", "UTF-8") +
37        "=" + URLEncoder.encode("Slika", "UTF-8") +
38        "&" + URLEncoder.encode("id_prijenos", "UTF-8") +
39        "=" + URLEncoder.encode(podaci[0], "UTF-8");
40       bufferPisac.write(post_data);
41       bufferPisac.flush();
42       bufferPisac.close();
43       izlazniStream.close();
44       Log.i("MOJ","Slika_str poslano: "+slika_str);
45       ulazniStream = httpURLKonekcija.getInputStream();
46       bufferCitac = new BufferedReader(new InputStreamReader(ulazniStream,
47        "UTF-8"));
48       rezultat_s = "";
49       linija = "";
50       while ((linija = bufferCitac.readLine()) != null) {
51        rezultat_s += linija;}
52       if (rezultat_s.equals("neuspjesno_slika")) {return rezultat_s;}
53       bufferCitac.close();
54       ulazniStream.close();
55       httpURLKonekcija.disconnect();
56      } catch (MalformedURLException e) {e.printStackTrace();
57      } catch (IOException e) {e.printStackTrace();}}}}
```

Nakon što se prime podaci o uspješnom slanju se ulazi u if petlju u primjeru programskog koda za slanje slika. Prvo što se radi je uzimanje JSON objekt primljenog od poslužitelja i rastavlja se kao u prethodnom primjeru (linije 2 do 8) Na liniji 9 se ulazi u if petlju ako postoji slika za slanje. Linije 10 i 11 uzimaju podatak od matične aktivnosti i rastavljaju ga na pojedine elemente koji svaki sadrži adresu slike koja će se slati. For petlja koja počinje na liniji 12 onda radi slanje slike onoliko puta koliko adresa slika ima u listi. Slanje slike je većim djelom isto kao i slanje u prethodnom primjeru. Jedna od razlika se vidi na liniji 14, tu se formira `string` koji informira

koja od koliko slika se šalje i taj `string` odlazi u prekidnu rutinu u aktivnosti `Prijave_Activity` (linija 15). Ispod se nalazi primjer prekida iz aktivnosti `Prijava_Activity.java`:

```
1  public void slanje(String por) {
2      runOnUiThread(new Runnable() {
3      @Overridea
4      public void run() {
5       Toast.makeText(Prijava_Activity.this, por,Toast.LENGTH_LONG).show();}});}}
```

Linija koda 17 formira `URI` adresu slike iz `stringa`. Linije koda 18 do 23 uzimaju sliku s adrese i formiraju `bitmapu` koja sadrži kod slike. Na liniji 22 se ta `bitmapa` pretvara u sliku JPEG tipa uz zadržavanje pune kvalitete slike. Slika se onda dodaje u bajt polje (linija 24). Bajt polje se potom base64 kodiranjem pretvara u jedan dugi `string` podataka (linija 25) koji se onda klasičnom POST metodom šalje na poslužitelj.

Kad se govori o mrežnim servisima na Androidu se ne može izostaviti web pogled. Pa je u svrhu prikazivanja jednostavne funkcionalnosti web pogleda napisan Java razred aktivnosti koji se nalazi u datoteci `Oprog_activity.java`. Kao što je već prije rečeno web pogled je jednostavan ali jako koristan alat koji je ubiti mali web preglednik.

```
1  WebView wv = (WebView) findViewById(R.id.webView);
2  wv.setWebViewClient(new WebViewClient());
3  wv.setWebChromeClient(new WebChromeClient());
4  String html="ovdje se nalazi HTML kod stranice koja se prikazuje";
5  wv.loadData(html, "text/html", "UTF-8");
```

Naredbe na linijama 1 do 3 inicijaliziraju web pogled koji prikazuje HTML kod slično kao i web preglednik. Te deklarira njegove parametre kao što je klijent ili XML element sučelja. Naredba na liniji 4 deklarira `string` koji sadrži HTML naredbe koje opsuju tekst i dodjeljuju vrijednost. Linija 5 prikazuje učitavanje HTML koda iz `stringa` u web pogled.

Da bi se u izvedenoj aplikaciji moglo pokazati slike neke prethodne prijave ih je potrebno preuzeti s poslužitelja u tu svrhu je implementirana vanjska knjižnica „Picaso“ onda služi za mrežne operacije s slikama na Androidu. Vanjske knjižnice se deklariraju u gradle datoteci. U listi `dependencies` se dodaje web adresa knjižnice. Primjer deklaracije vanjske knjižnice:

```
1  dependencies {
2  implementation 'com.squareup.picasso:picasso:2.71828'}
```

Slika se s poslužitelja dohvaća u adapteru Recycler liste `MojRecyclerViewAdapter3.java` naredbom:

```
1  Picasso.get().load(slika).into(holder.mojIv);.
```

Ta naredba dohvaća sliku preko argumenta „slika“ koji predstavlja njezinu web adresu.

4. Razvoj web aplikacije

Uslijed proširenja funkcionalnosti mobilne aplikacije koja je praktični dio završnog rada javila se potreba za implementacijom određenih funkcionalnosti na poslužitelju! Konkretno, za slanje slikovnih zapisa i prijava (odjeljak *3.2.12.*), na poslužitelju je potrebno izvesti funkcionalnost zaprimanja podataka i pohrane na poslužitelja, korištenjem neke od tehnologija dostupnih na poslužitelju. Za to je odabran programski jezik PHP i mySQL odnosno MariaDB baza podataka na poslužitelju baziranom na Apache web poslužitelju – odabir nije posebno opravdavan već su preuzete tehnologije kojima smo se upoznali tijekom nastave na vezanim kolegijima. Apache poslužitelj je u ovom primjeru dio WAMP programskog paketa koji pomaže kod stavljanja različitih mrežnih resursa na neku mrežu nevažno da li je to lokalna mreža ili Internet.

Kako bi mobilna aplikacija, a i cijeli kompleksni sustav koji uključuje i mobilni i poslužiteljski dio, odradila svoju svrhu, nakon implementacije serverskih skripti zaključeno je da je potrebno razviti i jednostavnu web aplikaciju i korisničko sučelje za zaprimanje, pregled i „rješavanje“ zahtjeva poslanih preko web aplikacije. Ovo poglavlje opisuje upravo taj dio razvijanog sustava – kompletni koncept poslužitelja i samu web aplikaciju. Daljnjim proširenjem kako mobilne tako i web aplikacije praktični dio rada može još dobiti na kompleksnosti i funkcionalnosti, no to nije dio priče o završnom radu.

4.1. Koncept web aplikacije

Poslužiteljska aplikacija se sastoji od više pojedinačnih PHP skripti koje se pokreću na Apache web poslužitelju. U ovom radu se mrežne aplikacije nalaze u direktoriju *„/zavsnirad/“* na poslužitelju. Nastavak teksta prikazuje programski kod napisan u PHP i JavaScript programskim jezicima, PHP radi na strani poslužitelja, a JavaScript na strani klijenta. Za sve važnije funkcije koda web stranice je dan opis. Uz njih se koristi i HTML jezik za definiranje strukture i sadržaja hipertekstualnih dokumenata.

Skripte vezane uz vezu s mobilnom aplikacijom su:

- `spoj.php` - Definira spoj na bazu podataka. Omogućava da u drugim PHP skriptama ne moramo baratati sa svim podacima potrebnim za prijavu na bazu podataka.
- `login.php` - Odrađuje sve poslove potrebne za prijavu korisnika mobilnog uređaja na poslužitelj. To je izvedeno jednostavno samo s vraćanjem podataka o korisniku koji se onda na mobilnoj strani koriste za identificiranje prijave.
- `registracija.php` - Omogućuje registraciju novih korisnika.
- `promjena.php` - Služi za promjenu postavki korisničkog računa ili njegovo brisanje.
- `prijava.php` - Koristi se za slanje prijave na poslužitelj.

- `slike.php` - Prima slike od klijenta i sprema ih u direktorij za slike te dodaje podatak o slici u bazu podataka.
- `dohvat.php` - Skripta šalje sve prijave korisnika na uređaj za pregledavanje starih prijava.

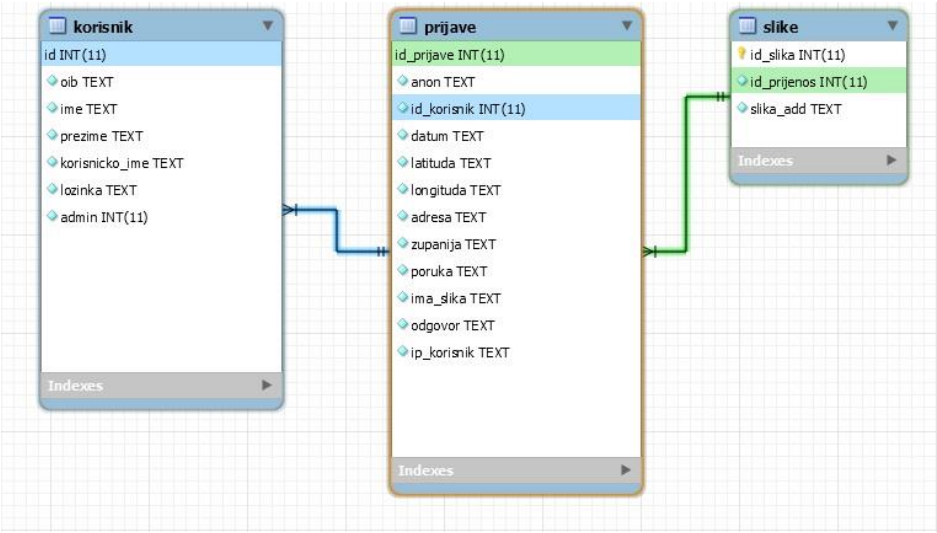
PHP skripte koje su vezane za prikaza i obradu prijava na osobnom računalu:

- `index.php` - Služi za prijavu u sustav. Stvara jedinstvenu sesiju prijave.
- `lista.php` - Prikazuje listu svih prijava prijavljenog korisnika ili svih korisnika ako je prijavljen administrator.
- `prikaz.php` - Prikazuje pojedinu prijavu. Administratoru je još omogućeno brisanje prijave ili pisanje odgovora na prijavu.
- `admin.php` - Skripta koja se koristi da bi se izbrisala prijava ili spremio odgovor na prijavu u bazu podataka.
- `odjava.php` - Odjavljuje korisnika iz sustava (briše podatke sesije).

4.2. Baza podataka

Obzirom da je koncept sustava zamišljen tako da omogućava registriranu ili anonimnu prijavu događaja (slikovni zapisi, lokacija, opis itd.), jasno je da na poslužitelju mora postojati mogućnost pohrane podataka. Za to se koristi klasična relacijska baza podataka – MySQL ili kompatibilno i popularno, besplatno, rješenje MariaDB. [30][31]. Nije provedena nikakva dodatna optimizacija ili analiza koncepta baze podataka, obzirom da to nije bio dio zadatka. U konačnici je ovako definirana baza omogućila realizaciju traženih funkcionalnosti, a u budućnosti se može prilagoditi, sukladno potrebama.

U bazu se pohranjuju korisnički podaci (registrirani korisnici) i svi elementi prijave. Baza podataka koja je korištena u ovom radu sadrži tri tablice (slika 25).



Slika 25: Struktura baze podataka

korisnik

id	oib	ime	prezime	korisnicko_ime	lozinka	admin
1	12345678901	Pero	Perić	user1	pass1	0
2	12345678902	Ivo	Ivić	user2	pass2	0
3	12345678903	Marko	Markić	user3	pass3	0
4	12345678904	Andrija	Andrijanić	user4	pass4	1
5	12345678905	Nikola	Nikolić	user5	pass5	0

Slika 26: Tablica registriranih korisnika aplikacije

Prva tablica je tablica korisnici (slika 26). Ta tablica sadrži podatke o korisniku aplikacije. Ti podaci su identifikacijski ID broj korisnika, taj se broj automatski povećava za svakog novog korisnika koji je stvoren, tzv. funkcija auto-inkrement te je ujedno i ključ te tablice. On služi za pretraživanje prijava. Ima i OIB korisnika, ime i prezime korisnika. Nakon toga dolazi korisničko ime što je jedan od identifikatora kod prijave korisnika na poslužitelj. Lozinka koja se koristi za tajnu potvrdu identiteta korisnika. Račun anonimnog korisnika ima ID broj nula. Predviđeno je da se prijave mogu odraditi i anonimno. Uz to je jadan od korisnika administrator sustava. Identifikacija administratora je omogućena u tablici stupcu admin.

prijave

id_prijave	anon	id_korisnik	datum	latituda	longituda	adresa	zupanija	poruka	ima_slika	odgovor	ip_korisnik
1	ne	1	05.08.2019. 20:44.47	46.3237216012811	16.288753024514346	Ul. Ruđera Boškovića 44, 42209, Sračinec, Hrvatska	Varaždinska županija	Lorem ipsum dolor sit amet, consectetur adipiscing...	ne	Proin at pellentesque neque. Aliquam lorem est, te...	192.168.1.103
2	ne	1	05.08.2019. 20:45.24	0.0	0.0			Lorem ipsum dolor sit amet, consectetur adipiscing...	da	Proin at pellentesque neque. Aliquam lorem est, te...	192.168.1.103
3	ne	1	05.08.2019. 20:45.50	46.32373572033228	16.28869821056234	Ul. Ruđera Boškovića 44, 42209, Sračinec, Hrvatska	Varaždinska županija	Lorem ipsum dolor sit amet, consectetur adipiscing...	da		192.168.1.103
4	ne	2	05.08.2019. 20:46.25	0.0	0.0			Lorem ipsum dolor sit amet, consectetur adipiscing...	ne	Proin at pellentesque neque. Aliquam lorem est, te...	192.168.1.103
5	ne	2	05.08.2019. 20:46.55	46.32373320041271	16.28869258188827	Ul. Ruđera Boškovića 44, 42209, Sračinec, Hrvatska	Varaždinska županija	Lorem ipsum dolor sit amet, consectetur adipiscing...	da	Proin at pellentesque neque. Aliquam lorem est, te...	192.168.1.103
6	da	0	05.08.2019. 20:48.06	46.32370104329895	16.288651980749023	Ul. Ruđera Boškovića 44, 42209, Sračinec, Hrvatska	Varaždinska županija	Lorem ipsum dolor sit amet, consectetur adipiscing...	da		192.168.1.103
7	ne	2	05.08.2019. 21:57.53	0.0	0.0			Lorem ipsum dolor sit amet, consectetur adipiscing...	da		192.168.1.103

Slika 27: Sadržaj tablice prijave

Druga tablica je tablica prijave (slika 27). U njoj se nalaze pojedine prijave i svi podaci koji su poslani na poslužitelj od klijenta. Ova tablica ima polje id_prijave koje sadrži identifikacijski broj prijave. Taj broj je auto inkrement broj. Tablica ima polja koja ispunjava korisnik kao što je polje poruka. To polje sadrži pouku koju je korisnik aplikacije napisao i poslao na poslužitelj. Polje lokacije koje korisnik sam dodaje klikom na gumb za dohvat lokacije. Lokaciju dohvaća kao koordinate, tako i geokoder iz tih koordinata. Dohvaća uličnu adresu asociranu s tim koordinatama. Uz uličnu adresu šalje se i županije. Uz ta polja postoji i polje koje pohranjuje vrijeme slanja prijave. To vrijeme se automatski generira prilikom slanja. Jedno od polja važnih za funkciju sustava je anon. Ono govori da li je prijava koja je primljena poslana kao anoniman.

Drugo polje je id_korisnik koje sadrži identifikacijski broj korisnika iz tablice korisnik koje je važno za dohvat podataka korisniku. Polje ima_slika upozorava da li su s prijavom poslane slike. Polje odgovor ne ispunjava korisnik nego ga ispunjava administrator koji upravlja poslužiteljem. Taj odgovor predstavlja odgovor na neku prijavu. Korisnik odgovor vidi kad pregledava poslane prijave. Zadnje polje koje se dodaje je polje ip_korisnik koje sadrži IP adresu klijenta koji je poslao prijavu.

slike

id_slika	id_prijenos	slika_add
1	2	Slika1.jpg
2	2	Slika2.jpg
3	2	Slika3.jpg
4	3	Slika4.jpg
5	3	Slika5.jpg
6	3	Slika6.jpg
7	3	Slika7.jpg
8	3	Slika8.jpg

Slika 28: Sadržaj tablice slike

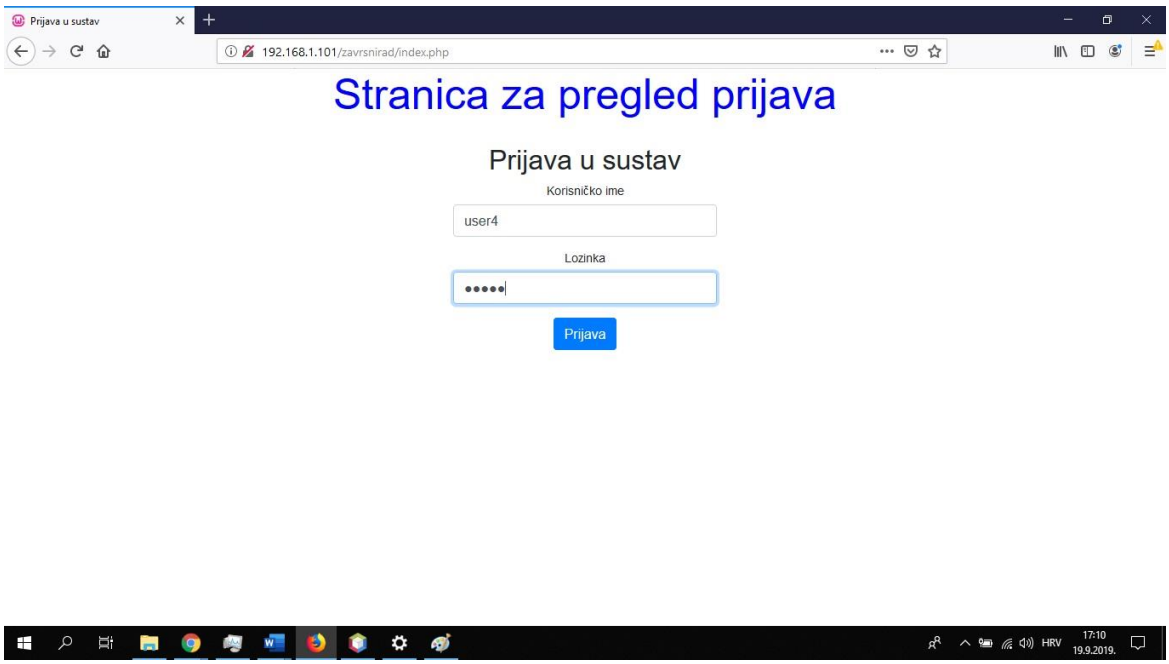
Treća i zadnja tablica baze podataka je tablica slike (slika 28). Ta tablica sadrži polje id_slika koje je auto inkrement polje koje sadrži identifikacijski broj pojedinog reda tablice slike. Polje id_prijenos daje podatak o tome kojoj prijavi neka slika pripada inače bi se slike izmiješale. Slika_add sadrži ime pojedine slike koje slika ima u direktoriju u kojem je pohranjena. Slike se pohranjuju u direktoriju koji se nalazi u istom direktoriju kao i same PHP mrežne aplikacije. Znači adresa slika je: „/zavrsnirad/slike/“. Te su slike spremljene kao JPEG slike. Iz tih podataka se formira mrežna adresa slike koja se vraća klijentu i preko nje dohvaća sliku poslanu uz prijavu.

4.3. Sučelja i skripte web aplikacije

U ovom odjeljku je prikazano korisničko sučelje web aplikacije. Nakon toga ukratko se opisuje princip izvedbe (programerski) ključnih funkcionalnosti PHP skripta.

4.3.1. Skripta za prijavu na poslužitelja

Web stranice kao vizualni dio aplikacije namijenjene su registriranim korisnicima ili administratoru koji obrađuje prijave. Pokreću se, ovisno o postavkama kao inicijalna stranica (`index.php` u korijenskom direktoriju na poslužitelju) koja omogućava kontrolu pristupa za registrirane korisnike. Prilikom prijave se provjerava da li je korisnik administrator sustava. Administrator za razliku od običnog korisnika vidi sve prijave (svih korisnika i anonimne) te može slati odgovor na prijave ili po potrebi ih brisati.



Slika 29: Stranica prijave na web servis

Na prvoj stranici web aplikacije slika 29 `index.php` se unose podaci u polja za upis korisničkog imena i lozinke. Klikom na gumb prijave se pokušava prijaviti. Ako su lozinka ili korisničko ime krivi ponovno otvori stranicu prijave i očisti unosna polja.

```
1   session_start();
2   require "spoj.php";
3   $spoj->set_charset('utf8');
4   if(isset($_SESSION["prijavljen"]) && $_SESSION["prijavljen"] === true){
5       header("Location: lista.php");}
6   if(isset ($_POST['korime']) && isset($_POST['pass'])){
7       $korisnicko_ime = $_POST["korime"];
8       $lozinka = $_POST["pass"];
9       $mysql_upit = "select id, admin from korisnik where korisnicko_ime "
10              . "= '$korisnicko_ime' and lozinka = '$lozinka';";
11      $rezultat = mysqli_query($spoj, $mysql_upit);
12      $row = mysqli_fetch_row($rezultat);
13      if(mysqli_num_rows($rezultat) > 0) {
14          $_SESSION["id"] = $row[0];
15          $_SESSION["prijavljen"] = true;
16          $_SESSION["admin"] = $row[1];
17          header("Location: lista.php");}}
```

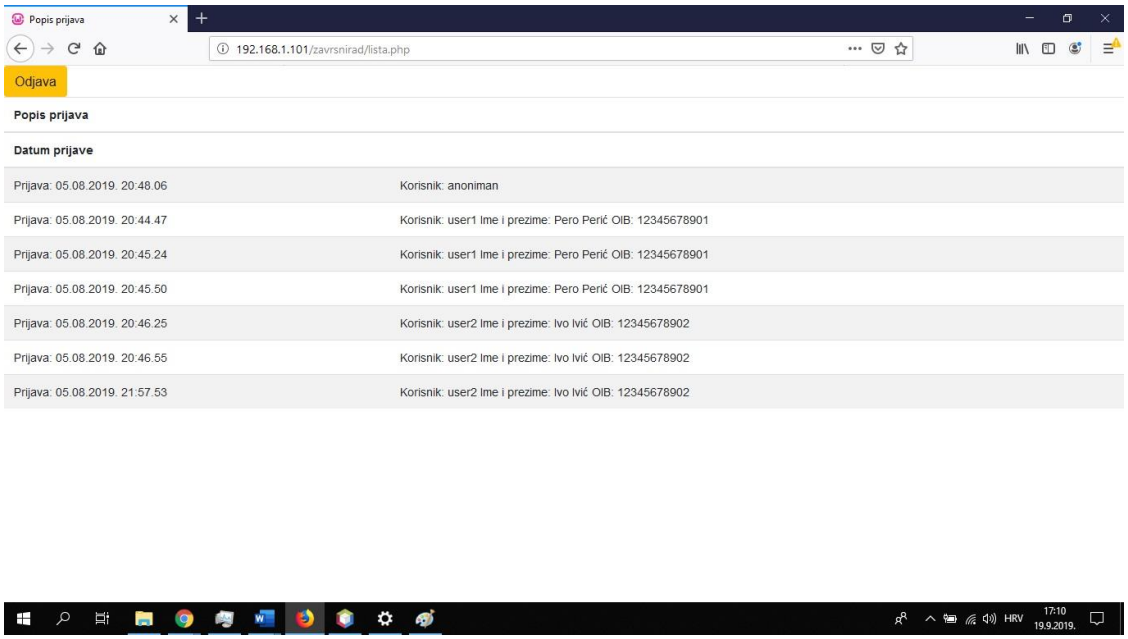
Naredba u liniji 1 pokreće novu sesiju. S naredbom u liniji 2 osigurava vanjski resurs koji sadrži sve podatke potrebne za spoj na bazu podataka. Takva metoda pristupa bazama podataka je sigurnija pa se koristi relativno često. U liniji 3 se definira format zapisa znakova za unos podataka u bazu podataka. If petlja u liniji 4 obavlja provjeru da li je sesija već postavljena, ulazi u petlju ako je postavljeno. Potom (linija 5) odlazi na novu PHP skriptu gdje su liste. Linija 6 provjerava ako su se postavile vrijednosti u unosna polja. Ako jesu ulazi u if petlju. U linijama 7 i 8 se nalazi standardna metoda za dohvaćanje POST zahtjeva za prijenosom podataka. Definira SQL upit kojim će provjeriti da li postoji korisničko ime i lozinka (linija 9 i 10). Naredba u liniji 11 stvara varijablu rezultat koja se popunjava prethodnim upitom koji se izvršio kroz vanjsku

konekciju na bazu podataka. Linija 12 prikazuje dohvaćanje pojedinih redova rezultata iz tablice. Rezultat je zapisan kao JSON objekt. If provjera u liniji 13 provjerava da li ima rezultata i ako ima u linijama 14 do 16 se postavljaju parametri sesije. Varijabli `$_SESSION` dodjeljuje identifikatore kojima će se prepoznavati korisnici. Na kraju u liniji 17 odlazi na novu stranicu `lista.php` gdje je lista prijava.

Uz PHP kod koji je srce cijele stranice se koristi i HTML koji opisuje grafički izgled stranice. HTML kod je standardni i u njemu su definirana polja za unos i gumb za prijavu. Polje za korisničko ime je standardno polje za unos teksta dok je polje za unos lozinke posebno polje prilagođeno unosu lozinke, što znači skriva lozinku kako ju korisnik piše. HTML kod je dostupan u prilogu.

4.3.2. Skripta za prikaz liste prijava

Ako su podaci prijave točni prelazi na stranicu vidljivu na slici 30. To je stranica `lista.php` na njoj je prikazana lista svih prethodnih prijava.



Slika 30: Lista web aplikacije svih prijava korisnika

Stranica `lista.php` (slika 30) isto ima jednostavni izgled. Ima gumb za odjavu s poslužitelja. Lista prijava ustvari nije lista nego je napisano kao tablica u HTML kodu. Može se kliknuti na svaku stavku liste. Ovdje se vide sve prijave jer je korisnik prijavljen kao administrator.

```
1       $id = $_SESSION["id"][0];
2       $admin = $_SESSION["admin"][0];
3       $mysql_upit1 = "SELECT datum, id_prijave, id_korisnik FROM `prijave` "
4               . "WHERE id_korisnik = '$id' ORDER BY id_korisnik;";
5       $mysql_upit2 = "SELECT datum, id_prijave, id_korisnik FROM `prijave` ORDER "
6               . "BY id_korisnik;";
```

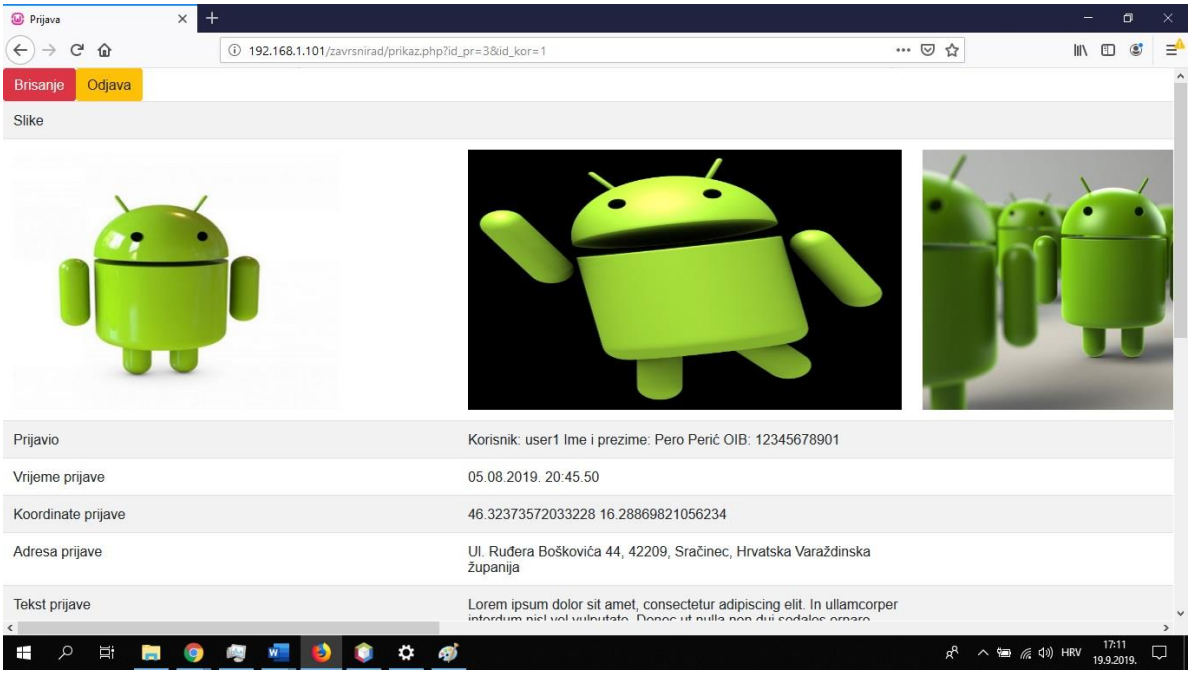
```
7       if($admin==0){$rezultat1 = mysqli_query($spoj, $mysql_upit1);}
8       else{$rezultat1 = mysqli_query($spoj, $mysql_upit2);}
9       $row1 = mysqli_fetch_all($rezultat1);
10      $server_ip = gethostbyname(gethostname());
11      $tablica = "<script>function odjava(){window.location.href "
12                  ."= 'http://".$server_ip."/zavrsnirad/odjava.php';}"
13                  ."function prikaz(id_pr, id_kor) {var pom1 = id_pr.toString();"
14                  ."var pom2 = id_kor.toString();var str = 'http://".$server_ip."/"
15                  ."zavrsnirad/prikaz.php?id_pr='+pom1+'&id_kor='+pom2;window.open"
16                  ."(str, '_top');}</script>";
17      if($rezultat1 == true) {
18          $tablica .= "<table class='table table-striped' align='center'><thead>"
19                  . "<tr><th>Popis prijava</th><th></th></thead><thead><tr><th>"
20                  . "Datum prijave</th><th></th></thead><tbody>";
21          if(empty($row1)){
22              $tablica = "<script>function odjava(){window.location.href "
23                          ."= 'http://".$server_ip."/zavrsnirad/odjava.php';}"
24                      . "</script><h2 align='center'>Nemate prijava.</h2>";
25              echo $tablica;}
26          else{
27              for($i = 0; $i < sizeof($row1); $i++){
28                  $str1 = $row1[$i][0];
29                  $str2 = $row1[$i][1];
30                  $str3 = $row1[$i][2];
31                  $mysql_upit3 = "SELECT korisnicko_ime, ime, prezime, oib "
32                          . "FROM `korisnik` WHERE id = '$str3';";
33                  $rezultat1 = mysqli_query($spoj, $mysql_upit3);
34                  $row2 = mysqli_fetch_row($rezultat1);
35                  $str4 = $row2[0];
36                  $str5 = $row2[1];
37                  $str6 = $row2[2];
38                  $str7 = $row2[3];
39                  if($str3=="0"){
40                      $tablica .= "<tr class='clickable-row' onclick='prikaz"
41                              . "(".$str2.",".$str3.")'><td>Prijava: $str1"
42                              . "</td><td>Korisnik: anoniman</tr>";}
43                  else{
44                      $tablica .= "<tr class='clickable-row' onclick='prikaz"
45                              . "(".$str2.",".$str3.")'><td>Prijava: $str1"
46                              . "</td><td>Korisnik: $str4 Ime i prezime: "
47                              . "$str5 $str6 OIB: $str7</tr>";}}
48              $tablica .= "</tbody></table>";
49              echo $tablica;}}
50      $spoj->close();
```

Početak skripte opet pokreće sesiju i pristupa skripti `spoj.php`. Nakon toga je opet provjera da li je već uspostavljena sesija kao i kod prethodne skripte. U linijama 1 i 2 se uzimaju podaci sesije. Linije 3 do 10 prikazuju formiranje upita i njihovo izvođenje. Razlika ovdje je da postoje dva upita, jedan za administratora, a drugi za običnog korisnika. Varijabla koja se dohvaća u lijini 10 sadrži IP adresu poslužitelja. U linijama 11 do 16 se definira `string` koji je JavaScript funkcija koja će služiti za odjavu sa poslužitelja i odlazak na pregled prijave. JavaScript funkcija za odlazak na drugu stranicu dohvaća u liniji 13 ID brojeve prijave i korisnika. Pomoću njih formira standardni GET upit i šalje ga na stranicu za prikaz prijava koji onda u liniji 16 otvara. Prethodno u linijama 13 i 14 još uzima te varijable pretvara ih u `string` format i definira `URL` adresu poslužitelja. Varijabla `$tablica` sadrži HTML i JavaScript kod koji će se poslati korisniku i učitati web stranicu. If provjera u liniji 17 provjerava da li je dohvat iz baze podataka uspješno obavljen. Nakon toga se formira dio HTML koda koji čini stranicu. U linijama 21 do 25 se provjerava ako postoje prijave i ako ih nema formira i objavljuje naredbom `echo` dio HTML

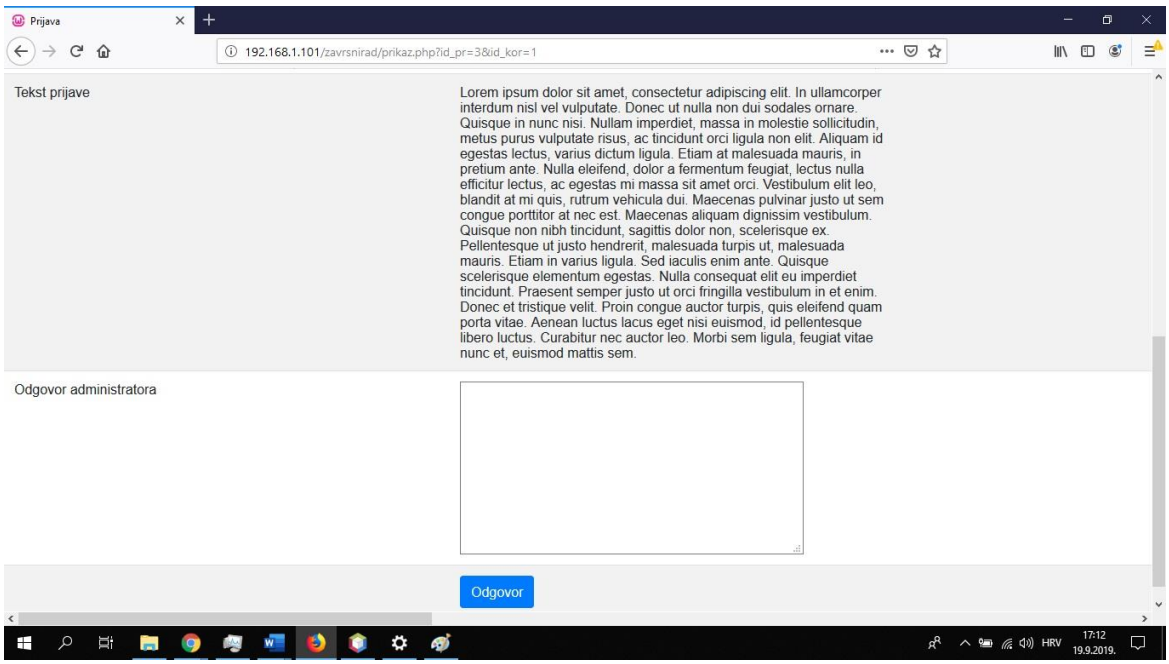
koda koji veli da nema prijava. U naredbi u liniji 26 ulazi u petlju u kojoj se formiraju liste koje sadrže podatke. U for petlji (linija 27) se formira HTML zapis za svaku prijavu koja je vraćena SQL upitom. Varijable `$str1` i `$str2` uzimaju podatak ID broja prijave i datuma, `$str3` podatak o anonimnosti prijave (linije 28 do 30). U linijama 31 do 34 dohvaća podatke o korisniku koji je izradio pojedinu prijavu. Potom u linijama 35 do 38 popunjava podatke iz tog upita u varijable koje sprema u tablicu (linije 44 do 47). If petlja (linije 39 do 42) provjerava ako je korisnik anoniman i shodno s tim formira HTML kod koji pod korisnički račun stavlja „anoniman“. Nakon toga zatvara tablicu u liniji 48 i objavljuje ju u liniji 49. U liniji 50 prekida spoj s bazom podataka.

4.3.3. Skripta za prikaz prijave

JavaScript skripta iz prethodnog odjeljka vodi korisnika na stranicu `prikaz.php` koja je vidljiva na slikama 31 i 32. Ta stranica prikazuje poslanu prijavu.



Slika 31: Prvi dio stranice prijave



Slika 32: Drugi dio stranice prijave

Na ovoj stranici slike 31 i 32 ostaje gumb za odjavu. Uz gumb za odjavu se vide polja koja sadrže podatke prijave. Vidi se tekst koji je korisnik poslao kao prijavu. Ispod teksta korisnika se prikazuje potencijalni odgovor administratora na tu prijavu. U ovom prozoru pošto smo prijavljeni kao administrator je to prazno polje za unos teksta. Ispod njega se nalazi gumb za slanje poruke. Uz gumb za slanje poruke na stranici se nalazi gumb za brisanje prijave (slika 31).

Skripta `prikaz.php` ima funkcionalnosti gotovo isto implementirano kao i prethodna skripta `lista.php` u odjeljku *4.3.2.* pa će se samo spomenuti neke značajne razlike.

```
1  $id_pr = $_GET["id_pr"];
2  <script>
3  function odgovor(id_pr){
4      var pom1 = id_pr.toString();
5      var por = document.getElementById('odg').value;
6      var xhttp = new XMLHttpRequest();xhttp.onreadystatechange = function() {
7          if (this.readyState == 4 && this.status == 200){
8              if(this.responseText=='uspjeh_odgovor'){
9                  alert('Odgovor je poslan!');}}};
10         xhttp.open('GET',
11         'http://".$server_ip."/zavrsnirad/admin.php?tip=odgovor&id_pr='
12                 +pom1+'&por='+por, true);
13         xhttp.send();}
14 </script>
```

Linija 1 prikazuju dohvaćanje podataka poslanih GET upitom. U linijama 3 do 13 je JavaScript skripta koja upisuje odgovor administratora na prijavu. Pokreće se klikom na neki element HTML koda (u ovom slučaju gumba). U liniji 4 dohvaća i pretvara u `string` podatak o ID broju prijave. Linija koda 5 dohvaća upisani tekst. Linije 6 do 9 prikazuju funkciju koja čeka povratak podatka o uspješnoj pohrani iz skripte `admin.php` i po primitku pokreće u liniji 9 upozorenje o uspješnom slanju. U linijama 10 do 12 formira GET upit. Upit šalje 2 podatka podatak o ID

broju prijave i poruku koji je administrator napisao. U liniji 13 šalje taj upit i linijom 14 označava kraj djela koda koji čini JavaScript.

4.3.4. Skripta za obradu prijava

Skripta `admin.php` prima podatke iz skripte `prikaz.php` i onda ovisno o tipu obrade koju mora odraditi obrađuje podatke prijave. Svaki tip obrade se nalazi u zasebnoj if petlji. Prvi tip obrade je brisanje prijave. Kod brisanja prvo dohvati podatak da li postoje slike za neku prijavu. Ako ne nastavi dalje, a ako postoje dohvaća popis slika asociranih sa prijavom te ih briše iz direktorija gdje su slike naredbom: `unlink(„adresa slike“);`. Nakon brisanja samih slika ih se standardnim SQL upitom briše iz baze podataka. Kada se izbrišu slike se opet standardnim SQL upitom brišu podaci same prijave.

Druga if petlja odrađuje spremanje odgovora administratora klasičnim SQL upitom. Po završetku uspješnog brisanja ili spremanja odgovora šalje poruku o uspjehu na skriptu `prikaz.php` iz koje su ove funkcije pozvane.

U naredbama prikazanim niže je vidljiva apstrakcija odrađivanja različitih tipova obrade if petljama.

```
1  $tip = $_POST["tip"];
2  if($tip=="zadatak1"){ echo "neke naredbe 1"; }
3  if($tip=="zadatak2"){ echo "neke naredbe 2"; }
```

U liniji 1 prvo dohvati `$tip` operacije i onda radi if provjere (linije 2 i 3) poslije kojih izvršava neke naredbe.

4.3.5. Skripta za odjavu korisnika sa poslužitelja

Skripta `odjava.php` odrađuje odjavu korisnika.

```
1  session_start();
2  $_SESSION = array();
3  session_destroy();
4  header("Location: index.php");
```

Linija 1 nastavlja postojeću sesiju korisnika na novoj skripti. U liniji 2 se prazne svi podatci trenutne sesije. Linija 3 prekida trenutnu sesiju i u liniji 4 se vraća na inicijalnu stranicu.

4.3.6. Skripta koja sadrži podatke za spoj na bazu podataka

Ova skripta odrađuje spoj na bazu podataka da ostale kripte ne moraju same odrađivati konekciju na bazu podataka nego se onda radi na jednom mjestu gdje se onda po potrebi mogu mijenjati postavke spoja.

```
1  define('HOST','localhost');
2  define('USER','root');
3  define('PASS','');
4  define('DB','Zavrsni_rad');
5  $spoj = mysqli_connect(HOST,USER,PASS,DB);
```

U linijama naredbi 1 do 4 definira sve stavke, podatke potrebne za spoj na bazu podataka. Podaci su ime baze podataka, adresa, korisničko ime i lozinka administratora. `String $spoj` u liniji 5 izvršava spoj na bazu podataka.

4.4. Poslužiteljske skripte za vezu s mobilnom aplikacijom

U ovom odjeljku su ukratko opisane skripte koje dodaju podatke koje prime u bazu podataka, Mijenja po potrebi postojeće podatke ili ih brišu. Opisane su samo funkcionalnosti koje nisu opisivane u prethodnim odjeljcima.

4.4.1. Skripta za prijavu na poslužitelja

Prva funkcija je prijava korisnika mobilne aplikacije na poslužitelj koja se radi u skripti `login.php`. Korisničko ime i lozinka se šalju klasičnim POST upitom i onda se vade iz upita preko ključa. Isto kao kod svih prethodnih primjera formira upit i izvršava ga. Upit iz tablice korisniku vraća sve podatke koje pronađe preko njegovog korisničkog imena i lozinke. Kao i u prethodnim primjerima ima provjere o uspjehu dohvata podataka iz baze podataka i po uspjehu šalje u jednom `stringu` poruku o uspjehu i JSON zapis svih stavki dohvaćenih iz tablice. A ako nije šalje `string` poruku o neuspjehu. S tim porukama mobilna aplikacije upozorava korisnika o uspjehu ili neuspjehu mrežne aplikacije.

```
1   $korisnicko_ime = $_POST["korisnicko_ime"];
2   $str = json_encode($row);
```

Linija 1 prikazuje dohvat podatka poslanog POST metodom, a linija 2 prikazuje JSON kodiranje podatka.

4.4.2. Skripta za registraciju novih korisnika

Druga skripta `registracija.php` se koristi isto isključivo već viđene metode. Ona dohvaća podatke koje šalje korisnik. U ovom slučaju su to svi podaci koji trebaju za stvaranje korisničkog računa. Formira SQL upit za upis tih podataka u bazu podataka. Izvršava upit provjerava uspjeh izvršavanja i prema ishodu vraća poruku o uspjehu ili neuspjehu korisniku na mobilni uređaj.

4.4.3. Skripta za promjenu korisničkih podataka

Nakon registracije je `promjena.php` na toj stranici se obavljaju dvije operacije. Prva je promjena korisničkih podataka. To se radi standardnim SQL upitom za promjenu podataka. Čim izvrši upit za promjenu radi drugi upit gdje vadi podatke i onda po uspjehu prvog upita vraća poruku o uspjehu i JSON zapis koji sadrži nove korisničke podatke za upotrebu na uređaju. Druga operacija koju radi je brisanje korisničkog računa. To je jednostavna funkcija s jednostavnim SQL upitom. Po uspjehu `echo` naredbom vraća poruku o uspjehu uređaju koji ako

je uspješno onda radi proceduru odjave. Različite operacije se odrađuju u različitim if petljama kao što je prikazano u odjeljku *4.3.4.*

4.4.4. Skripta za spremanje primljene prijave

Skripta `prijava.php` dodaje poslanu prijavu u bazu podataka. Ima standardno dohvaćanje POST podatka poslanih od korisnika i onda formira SQL upit koji sve podatke stavi u bazu podataka. Nakon toga dodjeljuje `stringu` najveći ID broj prijave u bazi podataka.

```
1   $id_prijave = getIDprijave();
```

Funkcija `getIDprijave` provodi zasebni upit u bazi podataka. Pomoću najvećeg ID broja uzima upravo dodanu prijavu. Funkcija koja dohvaća najveći identifikacijskog broj prijave i vraća ga za dodavanje:

```
1   function getIDprijave
2       $spoj = mysqli_connect(HOST,USER,PASS,DB);
3       $sql = "SELECT max(id_prijave) as id_prijave FROM prijave";
4       $result2 = mysqli_fetch_array(mysqli_query($spoj,$sql));
5       mysqli_close($spoj);
6       return $result2['id_prijave']; }
```

U liniji 2 uspostavlja spoj na bazu podataka. Linija 3 prikazuje formiranje upit za maksimalni iznos ID broja. Naredba u liniji 4 pokreće upit i pohranjuje rezultat. Linija 5 zatvara spoj na bazu podataka, a linija 6 vraća rezultat upita iz funkcije.

Po izvršenju te funkcije postavlja novi SQL upit i vraća podatke na klijenta. To omogućuje da slike koje se šalju imaju ID ključ prijave i da budu asocirane s jednom specifičnom prijavom.

4.4.5. Skripta za spremanje slika

Slike se na poslužitelj šalju kroz zasebnu skriptu `slike.php`. Prvo se formira `string` koji sadrži ime slike koja će se spremiti. Naredba za to je ispod u liniji 1. Da bi se dobilo ime se poziva funkcije `getBrojSlike()` u liniji 2, ta funkcija vraća najveći ID broj pohranjene slike u tablici slike. Pošto slika nije već spremljena se uzima broj zadnje slike i uveća za jedan, to će biti broj nove slike. Taj se broj dodaje imenu poslanom od klijenta i pomoću tog broja se formira ime slike koja se sprema.

```
1   $ime = $_POST["ime"].getBrojSlike().".jpg";
2   function getBrojSlike(){
3       $spoj = mysqli_connect('localhost', 'root', '','zavrsni_rad');
4       $mysql_upit2 = "SELECT max(id_slika) as id_slika FROM slike";
5       $rez = mysqli_fetch_array(mysqli_query($spoj,$mysql_upit2));
6       mysqli_close($spoj);
7       if($rez['id_slika']==null){return 1;}
8   else{return ++$rez['id_slika'];} }
```

Spoj na bazu podataka s kojom radi je u liniji 3. Upit za dohvat najvećeg identifikacijskog broja slike je prikazan u liniji 4. Linija 5 izvršava upit i pohranjuje podatke u `string`. Linijom 6 zatvara konekciju na bazu podataka. If provjera uspješnosti (linija 7), ako nije uspješno izlazi

daje jedan jer je postojeći broj nula. Ako je uspješan u liniji 8 poveća postojeći broj za jedan i vraća rezultat.

Slike se spremanju u direktorij nazvan „slike“. Ono što se sprema u bazu podataka i kasnije dohvaća je dio `URL` adrese tih slika da bi ih onda aplikacija mogla učitati i postaviti na zaslon. Poslužitelj prima slike u obliku `stringa` nastalog base64 kodiranjem. Base64 kodiranje pretvara bilo koji binarni podatak u tekstualni podatak. Preciznije pretvara 8-bitne podatke u 7-bitne podatke kakve koristi američki format zapisa teksta ASCII (eng. *American Standard Code for Information Interchange*). Primjer dekodiranja i pohrane slika je dan niže:

```
1  $dekodirani_string = base64_decode($slika_str);
2  $add = 'slike/'.$ime;
3  $file = fopen($add, 'wb');
4  $gotovo = fwrite($file, $dekodirani_string);
5  fclose($file);
```

Potrebno je dekodirati base64 `string` koji sadrži sliku, to se radi u naredbi u liniji 1. Linija 2 definira mjesto pohrane i ekstenziju slike. U liniji 2 formira `URL` koji se pohranjuje u bazu podataka kojim će korisnik dohvatiti sliku. Definiranje varijable `$ime` je prikazano u prethodnom isječku koda ovog odjeljka. Linija 3 otvara unos podataka u direktorij za pohranu. U liniji 4 sprema sliku u direktorij i dohvaća rezultat te operacije. Linija 5 prikazuje zatvaranje unosa u direktorij. Ako je slika uspješno pohranjena dodaje `URL` u bazu podataka.

4.4.6. Skripta za dohvat podataka i slanje na mobilni uređaj

Zadnja skripta je `dohvat.php` nju klijent kontaktira kad želi skinuti neku svoju prijavu da bi ju pregledao. Bazirano na ID broju prijave i tipu operacije koju mora odraditi vraća podatke prijave ili slike. Operacija se sastoji od dva djela. Prvi vraća samo tekstualne podatke prijave kao što su datum prijave, tekst prijave, odgovor administratora, lokacija itd. Klijent provjeri da li mu je u dohvaćenim podacima dignuta zastavica za postojanje slika i ako je šalje zahtjev za dohvat pojedinih slika koje pripadaju toj prijavi. Onda klasičnom metodom upita i slanja vraća podatke klijentu. U ovoj skripti je primjer „ručno“ izvedenog JSON kodiranje podataka koji se vraćaju na mobilni uređaj koji onda obrađuje te podatke. Primjer je niže:

```
1  for($i = 0; $i < sizeof($row1); $i++){
2      $str3 = $row1[$i][0];
3      if($i == 0){$str2 = "["; }
4      $str2 .= '["http:\/\/'.$server_ip.'\/zavrsnirad\/slike\/'.$str3.'"]';
5      if($i != (sizeof($row1)-1)){ $str2 .= ","; }
6      if($i == (sizeof($row1)-1)){$str2 .= "]"; } }
7  $str1 = "uspjesno_dohvat_slika//$str2";
8  echo $str1;
```

U liniji 1 počinje for petlja koja popunjava elemente JSON zapisa ovisno o količini dohvaćenih podataka. Podaci su dohvaćeni u prethodnom SQL upitu. Dohvaćeni podatak je ime slike sa „.jpg“ ekstenzijom (linija 2). Linija 3 prikazuje if provjeru da li je to početak JSON zapisa ako je otvara zagradu. U liniji 3 dodaje `stringu` podatak koji je JSON zapis `URL` adrese slike. Adresa

ima `string` koji sadrži IP adresu poslužitelja i na kraju `string` koji sadrži ime slike. Linije koda 5 i 6 provjeravaju da li je kraj dohvaćenih podataka. Ako je zadnji podatak zatvara zagradu, a ako nije stavljaju znak zareza između dva elementa. Po izlasku iz for petlje JSON zapisu dodaje kontrolnu poruku o uspješnom dohvatu slike i u liniji 8 šalje podatke na klijenta.

5. Zaključak

Aplikacija koja je izvedena u ovom završnom radu je podijeljena na dio koji radi na mobilnom uređaju i dio koji radi na poslužitelju te ima web aplikaciju.

Mobilna aplikacija ima dio koji se bavi lokacijskim servisima. Lokacije se mogu kontinuirano dohvaćati i time pratiti kretanje uređaja odnosno korisnika te vizualizirati pomoću karti i kasnije pohraniti sve lokacijske podatke. Moguće je pregledavati pohranjene lokacije ili snimljene rute kretanja te vidjeti u kojem su odnosu dvije lokacije.

Mrežni dio mobilne aplikacije je napravljen po uzoru na servis za slanje prijave komunalnom redarstvu. Ima naslovnu aktivnost gdje se je moguće prijaviti na poslužitelja. Ima obrazac za ispunjavanje novih prijava i njihovo slanje. Moguće je pregledati poslane prijave prijavljenog korisnika.

Uz to aplikacija ima aktivnost gdje je prikazan zadatak završnog rada.

Poslužiteljski dio aplikacije se sastoji od skripta koje obrađuju podatke primljene od korisnika s mobilnog uređaja. Drugi niz PHP skripti koji čine poslužiteljski dio rada služi za prijavu na poslužitelja i pregledavanje poslanih prijava, te administraciju prijava.

Početna ideja mobilne aplikacije završnog rada je proširena dodavanjem mogućnosti slanja proizvoljnih prijava na poslužitelja. Dodavanjem i slanjem slika iz unutarnje pohrane uređaja ili slika napravljenih kamerom uređaja te dodavanjem i slanjem lokacije uz prijavu.

Poslužiteljski dio je proširen dodavanjem administratora koji može brisati prijave ili slati korisniku povratnu informaciju o prijavi.

Aplikacija je potpuno funkcionalna i moguće ju je ubuduće proširiti sa novim mogućnostima. Primjeri mogućnosti proširenja su: slanje video zapisa, obrada prijave na mobilnom uređaju, komunikacija korisnika i administratora itd.

U Varaždinu, 24. rujna 2019. godine
Student:
Kristijan Lukaček

_____
(Vlastoručni potpis)

6. Literatura

[1] https://kotlinlang.org/assets/kotlin-media-kit.pdf, dostupno 01.09.2019.

[2] https://www.techradar.com/news/phone-and-communications/mobile-phones/a-complete-history-of-android-470327, dostupno 01.09.2019.

[3] https://www.netguru.com/blog/android-or-ios-which-one-to-target-first-for-your-next-mobile-app-development, dostupno 01.09.2019.

[4] https://developer.android.com/about/dashboards, dostupno 01.09.2019.

[5] https://developer.android.com/guide/platform, dostupno 01.09.2019.

[6] https://infinum.co/the-capsized-eight/art-vs-dalvik-introducing-the-new-android-runtime-in-kit-kat, dostupno 01.09.2019.

[7] https://developer.android.com/studio, dostupno 01.09.2019.

[8] https://www.metageek.com/training/resources/understanding-rssi.html, dostupno 01.09.2019.

[9] https://www.101computing.net/cell-phone-trilateration-algorithm/, dostupno 01.09.2019.

[10] https://developer.android.com/guide/topics/connectivity/wifi-rtt, dostupno 01.09.2019.

[11] https://journeynorth.org/tm/LongitudeIntro.html, dostupno 01.09.2019.

[12] https://www.celestis.com/resources/faq/what-are-the-azimuth-and-elevation-of-a-satellite/, dostupno 01.09.2019.

[13] https://www.gps.gov/systems/gps/space/, dostupno 01.09.2019,

[14] https://gisgeography.com/trilateration-triangulation-gps/, dostupno 01.09.2019.

[15] https://www.101computing.net/cell-phone-trilateration-algorithm/, dostupno 01.09.2019.

[16] https://www.aboutcivil.org/sources-of-errors-in-gps.html, dostupno 22.08.2019.

[17] https://developer.android.com/training/location, dostupno 01.09.2019.

[18] https://developer.android.com/reference/android/location/Geocoder.html, dostupno 01.09.2019.

[19] https://gisgeography.com/reverse-geocoding-services-addresses-free-paid/, dostupno 01.09.2019.

[20] https://developers.google.com/maps/documentation/android-sdk/intro, dostupno 01.09.2019.

[21] https://developer.android.com/training/data-storage/files, dostupno 01.09.2019.

[22] https://developer.android.com/preview/privacy/scoped-storage, dostupno 01.09.2019.

[23] https://developer.android.com/training/data-storage/files/internal, dostupno 01.09.2019.

[24] https://developer.android.com/training/data-storage/files/external, dostupno 01.09.2019.

[25] https://developer.android.com/training/data-storage/shared-preferences, dostupno 01.09.2019.
[26] https://developer.android.com/training/data-storage/sqlite.html, dostupno 01.09.2019.
[27] https://developer.android.com/training/data-storage/room, dostupno 01.09.2019.
[28] https://medium.com/@trishantsharma1997/asynctask-in-android-f594a565d676, dostupno 01.09.2019.
[29] https://developers.google.com/j2objc/javadoc/android/reference/android/os/AsyncTask, dostupno 01.09.2019.
[30] https://dev.mysql.com/, dostupno 01.09.2019.
[31] https://mariadb.org/about/, dostupno 01.09.2019.
[32] John Horton: Android Programming for Beginners, Packt Publishing, 2015.
[33] Wallace Jackson: Learn Android App Development, Apress, 2013.
[34] Michael Burton: Android App Development for Dummies, , John Wiley & Sons Inc., treće izdanje, 2015.
[35] Reto Meier: Professional Android Application Development, Wiley Publishing, Inc., 2009.
[36] https://stackoverflow.com/, dostupno 01.09.2019.
[37] https://www.w3schools.com/, dostupno 01.09.2019.
[38] https://developer.android.com/, dostupno 01.09.2019.
[39] https://www.androidauthority.com/ dostupno 01.09.2019.

Popis slika

UNIVERSITY NORTH

Sveučilište Sjever

SVEUČILIŠTE SJEVER

IZJAVA O AUTORSTVU
I
SUGLASNOST ZA JAVNU OBJAVU

Završni/diplomski rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, KRISTIJAN LUKAČEK *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom IZVEDBA ANDROID MOBILNE APLIKACIJE S PRIMJENOM LOKACIJSKIH SERVISA *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)

Luka

(vlastoručni potpis)

Sukladno Zakonu o znanstvenoj djelatnost i visokom obrazovanju završne/diplomske radove sveučilišta su dužna trajno objaviti na javnoj internetskoj bazi sveučilišne knjižnice u sastavu sveučilišta te kopirati u javnu internetsku bazu završnih/diplomskih radova Nacionalne i sveučilišne knjižnice. Završni radovi istovrsnih umjetničkih studija koji se realiziraju kroz umjetnička ostvarenja objavljuju se na odgovarajući način.

Ja, KRISTIJAN LUKAČEK *(ime i prezime)* neopozivo izjavljujem da sam suglasan/na s javnom objavom završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom IZVEDBA ANDROID MOBILNE APLIKACIJE S PRIMJENOM LOKACIJSKIH SERVISA *(upisati naslov)* čiji sam autor/ica.

Student/ica:
(upisati ime i prezime)

Luka

(vlastoručni potpis)

Prilog

Cjelokupan programski kod mobilne i mrežne aplikacije razvijene u završnom radu je dostupan na pratećem digitalnom mediju.