UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Tamara D. Ivanović

MASTER IZ MATEMATIKE ILI RAČUNARSTVA ČIJI JE NASLOV JAKO DUGAČAK

master rad

Beograd, 2022.

Mentor:

dr Milena Vujošević Janičić, vanredni profesor
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Ana Anić, vanredni profesor
University of Disneyland, Nedođija

dr Laza Lazić, docent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Mami, tati i dedi

Naslov master rada: Master iz matematike ili računarstva čiji je naslov jako dugačak

Rezime: Apstrakt rada na srpskom jeziku u odabranom pismu

Ključne reči: analiza, geometrija, algebra, logika, računarstvo, astronomija

Sadržaj

Glava 1

Uvod

Glava 2

Android

Android operativni sistem (u nastavku Android OS) je operativni sistem zasnovan na Linux jezgru i pripada zajednici otvorenog kôda. U ovom poglavlju biće reči o samom nastanku i razvoju ovog operativnog sistema, arhitekturi i osnovnim kompontentama. Kako je centralna tačka ovog rada aplikacija koja kontroliše set top-box uređaje (STB) koja je pisana u programskom jeziku Java ovo poglavlje će se osvrnuti i na odnos Android OS-a sa njima. Radi boljeg razumevanja rada aplikacije ovo poglavlje će se osvrnuti i na značaj i funkcionisanje Google API-ja, kao i na metod povezivanja dva uređaja sa Android OS-om pomoću mreže.

2.1 Istorijat

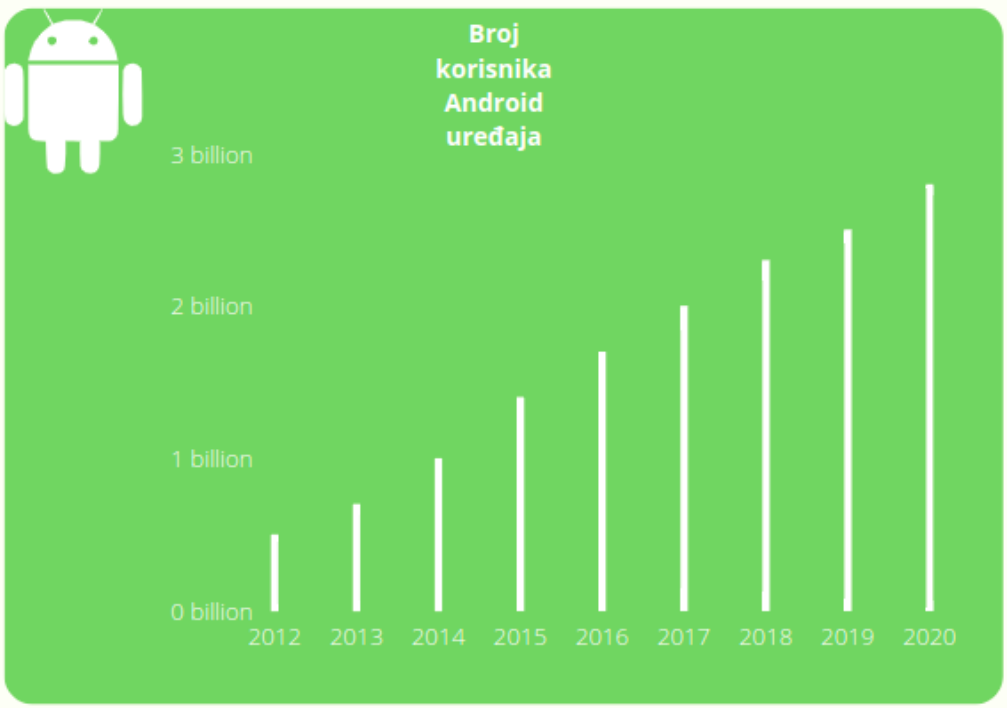
Endi Rubin (Andy Rubin) je 2003. godine sa trojicom kolega u Palo Altu osnovao kompaniju *Android Inc.* sa namerom da kreiraju platformu za kameru sa podrškom za skladištenje u oblaku. Takva ideja nije naišla na podršku investitora i cilj kompanije se preusmerio na pametne mobilne telefone, a vremenom i sve pametne uređaje. Zamisao je bila da sistem bude besplatan za korisnike, a da zarada zavisi od aplikacija i ostalih servisa. To je postalo moguće 2005. godine kada je kompanija *Google* kupila kompaniju i ostavila priliku osnivačima na čelu sa Rubinom da nastave sa kreiranje ovog operativnog sistema. [5]

Sam razvoj operativnog sistema i dalje traje, verzije su mnogobrojne i izlaze često, a sa svakom verzijom napravljeni su značajniji napredci. [8, 11] Svaka verzija je označena brojem, kao i nazivom slatkiša što je ideja projektnog menadžera Rajana Gibsona. U tabeli 2.1 je prikazan pregled najznačajnijih verzija zajedno sa novitetima koje su donele.

Tabela 2.1: Verzije Android OS-a

Naziv verzije	Datum objavljivanja	Najznačajniji noviteti
Android 1.0	Septembar 2008.	Podrška za kameru, internet pregledač, preuzimanje i objavljivanje aplikacija na Android Market-u, integrisani su Google servisi: Gmail, Google maps, Google Calendar, omogućene Wi-Fi i bluetooth bežične komunikacije
Android 1.5 - **Cupcake**	April 2009.	Poboljšana Bluetooth komunikacija, tastatura sa predikcijom teksta, snimanje i gledanje snimaka
Android 2.2 - **Froyo**	Maj 2010.	Poboljšanje brzine, implementacija JIT-a, instaliranje aplikacija van memorije telefona, povezivanje uređaja preko USB
Android 3.x - **Honeycomb**	Februar 2011.	Višeprocesorska podrška, Google Talk video čet, 'Private browsing', uživo prenos preko HTTP-a, USB host API, jednostavnije automatsko ažuriranje aplikacija preko Android Marketa
Android 4.1-4.3 - **Jelly Bean**	Jul 2012.	Glasovna pretraga, WiFi/WiFi-Direct otkrivanje servisa, bezbedno USB debagovanje, 4K podrška, podrška za BLE (Bluetooth Low Energy), WiFi scanning API
Android 6.0 - **Marshmallow**	Oktobar 2015.	Podrška za USB tip C, autentikacija pomoću otiska prsta, MIDI podrška
Android 8.0/8.1 - **Oreo**	Avgust 2017.	Svetla i tamna tema, PIP (Picture-In-Picture) sa opcijom promene veličine, API-ji za neuronske mreže i za deljenu memoriju
Android 9 - **Pie**	Avgust 2018.	Prikaz celog teksta i slike u obaveštenjima o porukama, dugme za gašenje može i da snimi sliku ekrana
Android 10 - **Queen Cake**	Septembar 2019	Bolja podrška za privatnost, pristup sistemskim podešavanjima iz panela, biometrijska autentikacija unutar aplikacija
Android 11 - **Red Velvet Cake**	Septembar 2020.	Snimak ekrana, balončići za poruke, podrška za 5G, bežično debagovanje, bolja podešavanja za dozvole
Android 12 - **Snow Cone**	2021.	Material You jezik za dizajn, podrška za AVIF, Android Private Compute Core

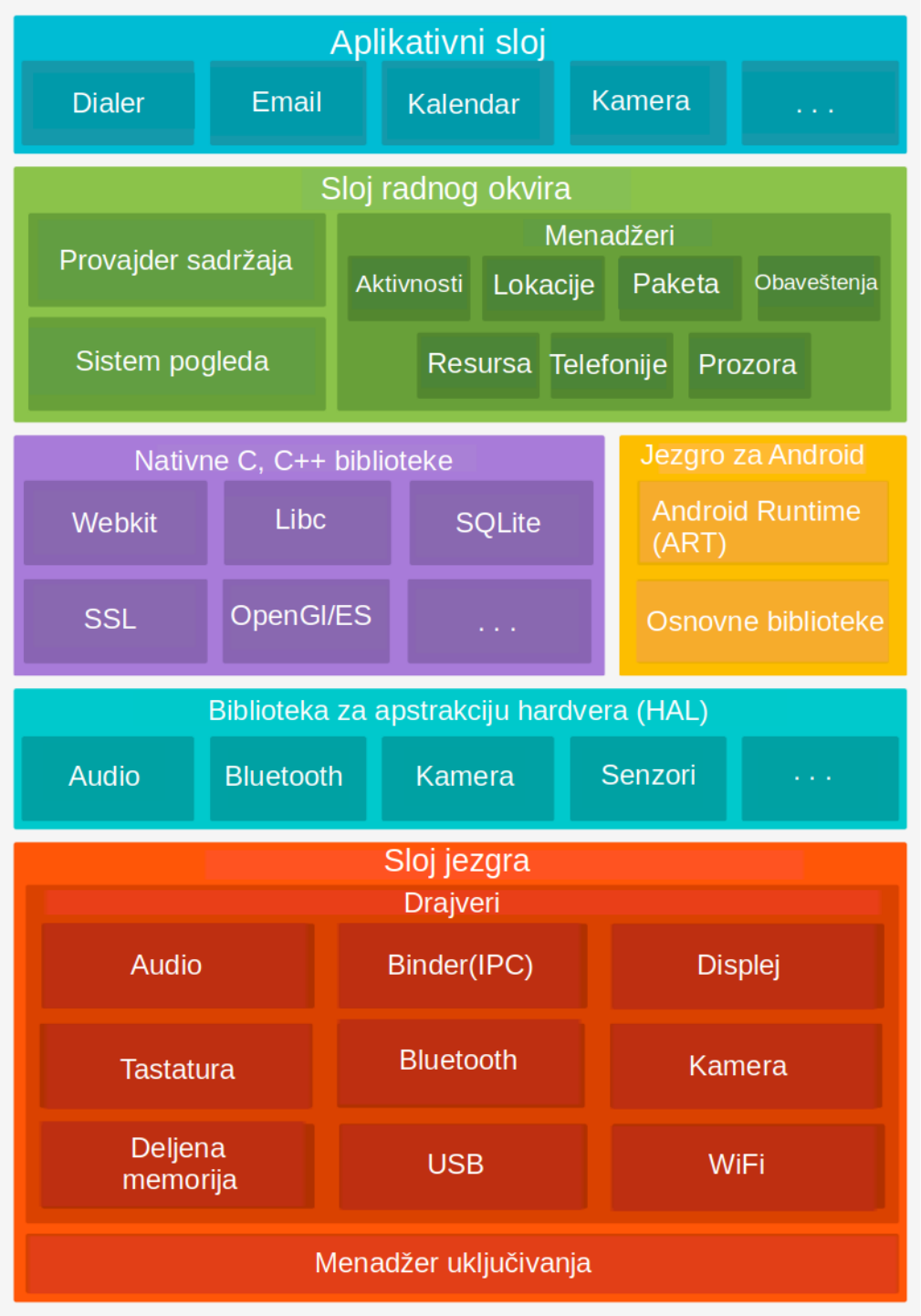
Android OS je svoju primenu na samom startu našao u pametnim telefonima i tabletima. Tokom godina programeri su raširili upotrebu na media plejere (za Android TV), pametne satove i naočare, kućne uređaje, automobile, kamere, konzole za igru... [8] Prema statistici [2], Kineske kompanije drže više od 55% Android tržišta. Od svih kompanija na tržištu najveći udeo imaju: *Samsung* (37.10%), *Xiaomi(*11.20%) i *Huawei*(11%). Sa rastom raznovrsnosti aplikacija koje postoje za Android uređaje, kao i broja različitih uređaja koji se koriste rastao je i broj korisnika. Statistika vezana za trend rasta broja korisnika može se videti na slici 2.1.



Slika 2.1: Broj korisnika tokom godina, izvor [2]

2.2 Arhitektura

Android platforma je bazirana na Linux jezgru (eng. *Linux kernel*) pri čemu se jezgro i drajveri nalaze u prostoru jezgra (eng. *kernel space*), a nativne biblioteke u korisničkom prostoru (eng. *user space*). Sve aplikacije se izvršavaju u Java virtuelnoj mašini koja se zove *Android Runtime (ART)*, a postoje Java biblioteke koje povezuju aplikaciju sa bibliotekama napisanim u nativnom jeziku. Sama arhitektura softvera kod Androida je slojevita i postoje četiri sloja koja se naslanjaju na sloj fizičke



Slika 2.2: Arhitektura Android OS, slika kreirana na osnovu [9]

arhitekture. Slojevi arhitekture prikazani su na slici 2.2, a navedeni od viših ka nižim (eng. *top-down*) su[6]:

1. **Aplikativni sloj** (eng. *Application layer*) - Predstavlja skup svih aplikacija koje se izvršavaju na Androidu. Aplikacije mogu biti sistemske, ugrađene ili korisničke. Sistemske su one koje su napisane od strane proizvođača uređaja, ugrađene su one koje su drugi kreirali ali su unapred instalirane na uređaj i ne mogu se obrisati, a sve ostale su korisničke.

2. **Sloj radnog okvira** (eng. *Frameworks layer*)- Predstavlja sloj koji omogućuje da se premoste razlike između aplikativnog i nativnog sloja i koji određuje ograničenja koja moraju da se poštuju pri razvoju Android aplikacija. Ovaj sloj je napisan u programskom jeziku Java, a pomoću JNI(*Java Native Invocation*) komunicira sa nativnim slojem. Najbitniji elementi sloja mogu se videti na slici 2.2, a neki od njih su: menadžer aktivnosti koji upravlja životnim ciklusom aplikacije, menadžer paketa koji poseduje informacije o svim instaliranim aplikacijama na uređaju, menadžer lokacije koji pronalazi geografsku lokaciju uređaja i menadžer telefonije koji omogućava pristup sadržajima telefonije kao što su brojevi telefona.

3. **Nativni sloj** (eng. *Runtime layer*) - Napisan je u nativnom jeziku (C ili C++), sastoji se od nativnih biblioteka, biblioteka za apstrakciju hardvera (*HAL*) i jezgra za Android (eng. *Android runtime*) u koji spadaju osnovne bibilioteke i ART. Do verzije 5.0 ART nije postojao već je korišćena *Dalvik virtuelna mašina (DVM)*.[7]

4. **Sloj jezgra** (eng. *Kernel layer*)- Predstavlja sloj između hardvera i softvera koji poseduje sve drajvere koji su potrebni za hardverske komponente. Takođe, u ovom sloju je moguće pronaći sve vezano za upravljanje memorijom, procesima i uključivanjem, kao i o bezbednosti.

2.3 Komponente Android aplikacije

Pod Android aplikacijom se smatra bilo koja aplikacija koja se pokreće na uređaju sa Android OS-om. Programiranje ovih aplikacija je moguće u mnogim programskim jezicima, dok se zvaničnim smatraju programski jezici Java i Kotlin.[3] U nastavku će biti reči o programiranju u Java programskom jeziku. Kreiranje Android aplikacija ne bi bilo moguće bez njenih osnovnih komponenti. Svaka komponenta ima svoje karakteristike, slučajeve upotrebe kao i funkcije koje vrši. Moguće je i

poželjno kombinovati ih u aplikaciji. Svaka komponenta koje se kreira u aplikaciji mora da se navede u AndroidManifest.xml datoteci. Četiri osnovne komponente su:

1. Aktivnosti (eng. *Activity*)
2. Servisi (eng. *Service*)
3. Prijemnici (eng. *Broadcast receiver*)
4. Provajderi sadržaja (eng. *Content provider*)

Aktivnosti

Aktivnosti predstavljaju jedan prikaz grafičkog korisničkog interfejsa (GUI) na ekranu. Ne postoji ograničeni broj aktivnosti koje jedna aplikacija može imati, takođe moguće je da postoje aplikacije bez aktivnosti. Za razliku od mnogih programskih jezika gde pokretanje aplikacije počinje pozivom *main()* metoda i uvek od istog mesta, Android aplikacije ne moraju uvek započinjati na istom mestu. Uglavnom Android aplikacije imaju jedan početni ekran koji se naziva *Main Activity* i koji se pokreće pri pokretanju aplikacije i više dodatnih koji su logički povezani sa početnim. Logika iza koje stoji ovaj koncept je da je korisniku omogućeno da pokrene različite delove aplikacije u zavisnosti od trenutnih potreba. Jedan primer koji ovo ilustruje je kada korisnik klikne na obaveštenje aplikacije za dostavu hrane da je hrana koju je naručio u putu aplikacija će otvoriti aktivnost koja prikazuje mapu za praćenje, a ne početnu stranu za izbor restorana.

Pri implementaciji svaka aktivnost mora da ima svoje ime i nasleđuje klasu *Activity*. Samim tim nasleđuje i njene metode koje prate osnovna stanja životnog ciklusa aktivnosti: *onCreate(), onStart(), onPause(), onResume(), onStop(), onDestroy() i onRestart()*[1]. Četiri stanja su:

1. Trajanje (eng. *Running*)
2. Mirovanje (eng. *Paused*)
3. Zaustavljeno (eng. *Stopped*)
4. Uništeno (eng. *Destroyed*)

Metod *onCreate()* je metod koji mora biti implementiran iz razloga što se u njemu nalazi sva logika koja je potrebna da se izvrši pri prvom pokretanju aktivnosti.

U njemu je potrebno uraditi sve inicijalizacije osnovnih komponenti aktivnosti kao i inicijalizaciju statičkih promenljivih, stavljanje podataka u liste... Iz ovog metoda mora se pozvati metod *setContentView()* koji određuje prikaz grafičkog korisničkog interfejsa. Nakon izvršavanja *onCreate()* metoda uvek se poziva *onStart()* metod.

Metod *onStart()* vodi računa o svemu što je potrebno da aktivnost bude vidljiva korisniku. Aplikacija ovde priprema aktivnost da bude prikazana korisniku. Može se registrovati prijemnik da osluškuje promene koje bi izmenile grafički korisnički interfejs. Metodi koji prate *onStart()* su *onResume()* ili *onStop()*.

Metod *onPause()* se poziva u trenutku kada se primeti da korisnik više neće koristiti tu aktivnost. S obzirom da se njeno izvršavanje dešava u momentu kada je aktivnost još uvek vidljiva korisniku sve što se izvršava u metodi mora biti brzo jer sledeća aktivnost neće biti nastavljena dok se metod ne završi. Ovde treba prekinuti sve što nije potrebno da se izvršava kada je aktivnost u stanju mirovanja. Ovaj metod prate metodi *onResume()* ukoliko se fokus vrati na ovu aktivnost ili *onStop()* ukoliko je aktivnost nevidljiva korisniku.

Metod *onResume()* se poziva kada aktivnost počinje da ima interakciju sa korisnikom. Uvek ga prati metod *onPause()*.

Metod *onStop()* se poziva kada god aktivnost više nije vidljiva korisniku što može biti zbog toga što je pokrenuta nova aktivnost ili jer se trenutna aktivnost uništava. Neki od čestih primera kada se koristi implementacija ove metode su osvežavanje korisničkog interfejsa i zaustavljanje animacija ili muzike. Ukoliko se aktivnost vraća interakciji sa korisnikom pozvaće se *onRestart()* metod, u suprotnom *onDestroy()* metod.

Metod *onDestroy()* je poslednji poziv i može se desiti iz dva razloga. Prvi, jer se aktivnost završava. Drugi, da se privremeno gasi aktivnost radi čuvanja memorijskog prostora. Koji se razlog desio može se saznati pomoću metode *isFinishing()*.

Metod *onRestart()* se poziva nakon što je aktivnost stopirana, a pre njenog ponovnog prikaza. Tu možemo uraditi eventualne ponovne inicijalizacije ili neke izmene korisničkog interfejsa pre nego što bude ponovo pozvan *onStart()* metod.

Servisi

Servis je komponenta koja izvršava svoje zadatke u pozadini i najčešće se koriste za zadatke koji se dugo izvršavaju i koji bi usporili aplikaciju ako bi se izvršavali na glavnoj niti. Servisi nemaju grafički korisnički interfejs, ali mogu da komuniciraju

sa ostalim komponentama. [4] U zavisnosti od tipa zadatka koji se očekuje da servis izvrši, kao i dužine trajanja izvršavanja razlikuju se tri vrste servisa:

- **Pozadinski (eng. Background)** - Ovi servisi ne obaveštavaju korisnika ni na koji način o zadacima koje izvršavaju zbog toga što za njihovo izvršavanje nije potrebna nikakva interakcija sa korisnikom. Primer je sinhronizovanje podataka u unapred određeno vreme.
- **??? (eng. Foreground)** - Ovo su servisi za koje korisnici znaju da se izvršavaju tako što servis pomoću obaveštenja obaveštava korisnika o svom izvršavanju. Korisniku se daje mogućnost da pauzira ili u potpunosti zaustavi proces koji se izvršava. Primer ovog servisa je preuzimanje datoteka.
- **Vezani (eng. Bound)** - Ovi servisi se izvršavaju kada je neka komponenta aplikacije povezana sa servisom, odnosno dokle god postoji neka komponenta kojoj je potrebno izvršavanje zadataka koje dati servis izvršava.

Na osnovu životnog ciklusa servisa razlikujemo pokrenute (eng. *Started*) servise i povezane (eng. *Bounded*). Kod pokrenutih servis se inicijalizuje pozivom *startService()* metode, a zaustavlja kada komponenta pozove metod *stopService()* ili ukoliko sam servis pozove *stopSelf()* metod. Povezani se mogu doživeti kao klijent-server struktura iz razloga što komponente mogu da šalju zahteve servisu, kao i da dohvataju rezultate. U trenutku kada neka komponenta pozove *bindService()* metod i time se poveže sa servisom servis se smatra povezanim, a tek kada se sve komponente aplikacije koje su bile povezane sa njim oslobode pozivom *unbindService()* servis prestaje sa radom. Svi navedeni metodi su iz klase *Service* koju je neophodno da svaki servis nasledi pri implementaciji.

Prijemnici

Prijemnici služe da osluškuju sistemska obaveštenja kao i obaveštenja od strane drugih aplikacija na uređaju ili drugih delova iste aplikacije. Da bi mogao da izvršava svoju funkciju potrebno je da prijemnik bude registrovan da osluškuje određene namere (eng. *intent*). Moguće je da jedan prijemnik osluškuje više različitih namera i u zavisnosti od namere da izvršava različite operacije. Neki od primera upotrebe sistemskih prijemnika su prijemnik za procenat baterije, prijemnik za alarm i prijemnik za sms poruke. [11]

Provajderi sadržaja

Provajderi sadržaja obezbeđuju skladištenje podataka aplikacije. Pored samog skladištenja njihova uloga je i da omoguće drugim aplikacijama da pristupe sadržaju ukoliko imaju prava za to. Samo skladištenje je moguće da bude pomoću SQLite baza podataka, datoteka ili na mreži. Sa strane implementacije aplikacija koja želi da deli svoje podatke mora da koristi klasu *ContentProvider* i kreira interfejs prema tim podacima. Druga aplikacija da bi mogla da koristi te podatke mora da napravi instancu objekta klase *ContentResolver* sa svim metodama koje prva aplikacija poseduje.

Namere

Namera (eng. *Intent*) predstavlja objekat koji slanjem poruke zahteva da drugi deo aplikacije ili druga aplikacija izvrši neku akciju. Najčešće su tri upotrebe namera: pokretanje aktivnosti, pokretanje servisa i slanje poruka (eng. *broadcast*). Implementacija se vrši pomoću klase *Intent* i potrebno je kreirati novi objekat. [8]

AndroidManifest.xml

Glavna datoteka bez kog nijedna Android aplikacija ne može da postoji je AndroidManifest.xml. Pomoću ove XML datoteke Android OS i njegovi build tool-ovi dobijaju sve potrebne informacije za instalaciju i pokretanje aplikacije. Kôd počinje navođenjem verzije XML-a i enkodiranja, nakon čega sledi tag *<manifest>* u okviru kog se piše ceo kôd. Obavezni deo kôda je tag *<application>*. Pregled osnovnih elemenata i njihovih opisa može se videti u kôdu 2.1. Više informacija o elementnima i njihovim opcijama može se pronaći na [10]

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest>
3   <!-- Navodi se za svaku dozvolu koju aplikacija zahteva -->
4     <uses-permission android:name="string"/>
5     <!-- Definise bezednosnu dozvolu za ogranicavanje pristupa
    funkcijama ili komponentama-->
6     <permission />
7     <!-- Definise kompatibilnost aplikacije sa verzijama Androida
    -->
8     <uses-sdk/>
```

```
9     <!--Definise hardverske i softverske karakteristike koje
      aplikacija zahteva  -->
10    <uses-configuration />
11    <!-- Deklarise aplikaciju i sve njene elemente-->
12    <application>
13    <!-- Definise aktivnost -->
14        <activity>
15        <!-- Definise tipove namera koje aktivnost podrzava -->
16            <intent-filter> . . . </intent-filter>
17            <!-- Par ime vrednost za dodatne podatke koji se
      dodeljuju roditeljskoj komponenti-->
18            <meta-data />
19        </activity>
20    <!-- Alias za aktivnost, moze imati svoje drugacije postavke u
      odnosu na aktivnost -->
21        <activity-alias> . . . </activity-alias>
22    <!-- Deklarise servis i njenove intent-filter i meta-data -->
23        <service> . . . </service>
24    <!-- Deklarise prijemnik-->
25        <receiver> . . . </receiver>
26    <!-- Definise provajdera sadrzaja-->
27        <provider> . . . </provider>
28    <!-- Definise deljenu biblioteku sa kojom aplikacija mora biti
      vezana. -->
29        <uses-library />
30    </application>
31 </manifest>
```

Listing 2.1: Primer AndroidManifest.xml, izvor: [10]

2.4 Android i STB

2.5 Android i Java

2.6 Google API

2.7 Povezivanje Android uređaja preko mreže

Glava 3

Implementacija aplikacije

Glava 4

Zaključak

Bibliografija

[1] Rick Boyer. *Android 9 Development Cookbook, Third Edition.* Packt, 2018.

[2] BusinessOfApps David Curry. Android Statistics (2022), 2022. on-line at: `https://www.businessofapps.com/data/android-statistics/`.

[3] Kotlin Foundation. Kotlin programski jezik. on-line at: `https://kotlinlang.org/docs/android-overview.html`.

[4] Erik Hellman. *Android Programming, Pushing the Limits.* Wiley, 2014.

[5] Darren Cummings Iggy Krajci. *Android on x86, An Introduction to Optimiying for Intel Architecture.* Apress, 2013.

[6] Nemanja Lukić Ištvan Papp. *Projektovanje i arhitekture softverskih sistema: Sistemi zasnovani na Androidu.* FTN Izdavaštvo, 2015.

[7] Google LLC. Dalvik VM, 2020. on-line at: `https://source.android.com/devices/tech/dalvik`.

[8] Google LLC. Android Developers, 2022. on-line at: `https://developer.android.com/`.

[9] Google LLC. Android Developers Arhitektura, 2022. on-line at: `https://developer.android.com/guide/platform`.

[10] MIT. Android Manifest. on-line at: `https://stuff.mit.edu/afs/sipb/project/android/docs/guide/topics/manifest/manifest-intro.html`.

[11] Miodrag Živković. *Razvoj mobilnih aplikacija, Android Java programiranje.* Univerzitet Singidunum, 2020.

Biografija autora

Vuk Stefanović Karadžić (*Tršić, 26. oktobar/6. novembar 1787. — Beč, 7. februar 1864.*) bio je srpski filolog, reformator srpskog jezika, sakupljač narodnih umotvorina i pisac prvog rečnika srpskog jezika. Vuk je najznačajnija ličnost srpske književnosti prve polovine XIX veka. Stekao je i nekoliko počasnih mastera. Učestvovao je u Prvom srpskom ustanku kao pisar i činovnik u Negotinskoj krajini, a nakon sloma ustanka preselio se u Beč, 1813. godine. Tu je upoznao Jerneja Kopitara, cenzora slovenskih knjiga, na čiji je podsticaj krenuo u prikupljanje srpskih narodnih pesama, reformu ćirilice i borbu za uvođenje narodnog jezika u srpsku književnost. Vukovim reformama u srpski jezik je uveden fonetski pravopis, a srpski jezik je potisnuo slavenosrpski jezik koji je u to vreme bio jezik obrazovanih ljudi. Tako se kao najvažnije godine Vukove reforme ističu 1818., 1836., 1839., 1847. i 1852.