# ELEKTROTEHNIČKA ŠKOLA SPLIT

# TESLINA 2, SPLIT

ZAVRŠNI RAD

OBRADA ID3 TAGOVA NA ANDROID PLATFORMI

Mentor: mag. Ing. Comp. Marin Ivandić Učenik: Bruno Farac

Razred: 4.D

Zanimanje: Tehničar za računalstvo

Split, ožujak 2017.

**SADRŽAJ**

[1. UVOD 3](#_Toc478626249)

[2. Opis korištenih tehnologija i alata 4](#_Toc478626250)

[2.1. Android platforma 4](#_Toc478626251)

[2.2. Java programski jezik 6](#_Toc478626252)

[2.3. MP3 datoteke 7](#_Toc478626253)

[2.4. mp3agic Java library 7](#_Toc478626254)

[3. Razvojna okolina aplikacije 8](#_Toc478626255)

[3.1. Android Studio 8](#_Toc478626256)

[3.2. Visual Studio Team Services 11](#_Toc478626257)

[3.3. Testni uređaj 12](#_Toc478626258)

[4. Struktura aplikacije 13](#_Toc478626259)

[5. Opis koda 16](#_Toc478626260)

[5.1. MainMenu Activity 16](#_Toc478626261)

[5.2. FileListing Activity 18](#_Toc478626262)

[5.3. TagEditor Activity 30](#_Toc478626263)

[6. ZAKLJUČAK 39](#_Toc478626264)

[7. LITERATURA 40](#_Toc478626265)

[8. POPIS SLIKA 41](#_Toc478626266)

# UVOD

Programiranje je, laički rečeno, temeljito objašnjavanje računalu kako obaviti određenu zadaću, prevođenje algoritama u nešto što računalo razumije. Ono je kao stručna disciplina ušlo u sve pore suvremene tehnike. Vrlo je teško pronaći aspekt ljudskog života u kojem se ne služimo u nekoj mjeri nekim oblikom istog. Upravo zbog te sveobuhvatnosti tog područja tehnike smatram da je izbor teme maturalnog rada iz tog područja vrijedan i koristan izbor.

Izbor upravo Android platforme za temelj rada je temeljen na sličnim principima. Android platforma zauzima ~76% tržišta mobilnih uređaja prema podatcima iz 2015 [1.] ., te ne pokazuje nikakve znakove opadanja. Stoga smatram da je perspektivno u odnosu na praktične probleme s kojima bi se mogao susretati nakon srednje škole da se pobliže upoznam s prirodom razvoja aplikacija za nju.

Aplikacija ima za zadaću ponuditi korisniku obradu MP3 datoteka (format audio datoteke), odnosno tzv. ID3 tagova u istima, u kojima se nalaze podatci o izvođaču, albumu, žanru i sl. Također pruža korisniku opciju pretraživanja vlastitog uređaja s svrhom da locira iste. Sama zadaća aplikacije je temeljena čisto na tome što sam htio odabrati nešto što će pokriti što više aspekata razvoja Android aplikacije i pružiti mi svojevrstan uvid u mogućnosti i limitacije Android platforme.

# Opis korištenih tehnologija i alata

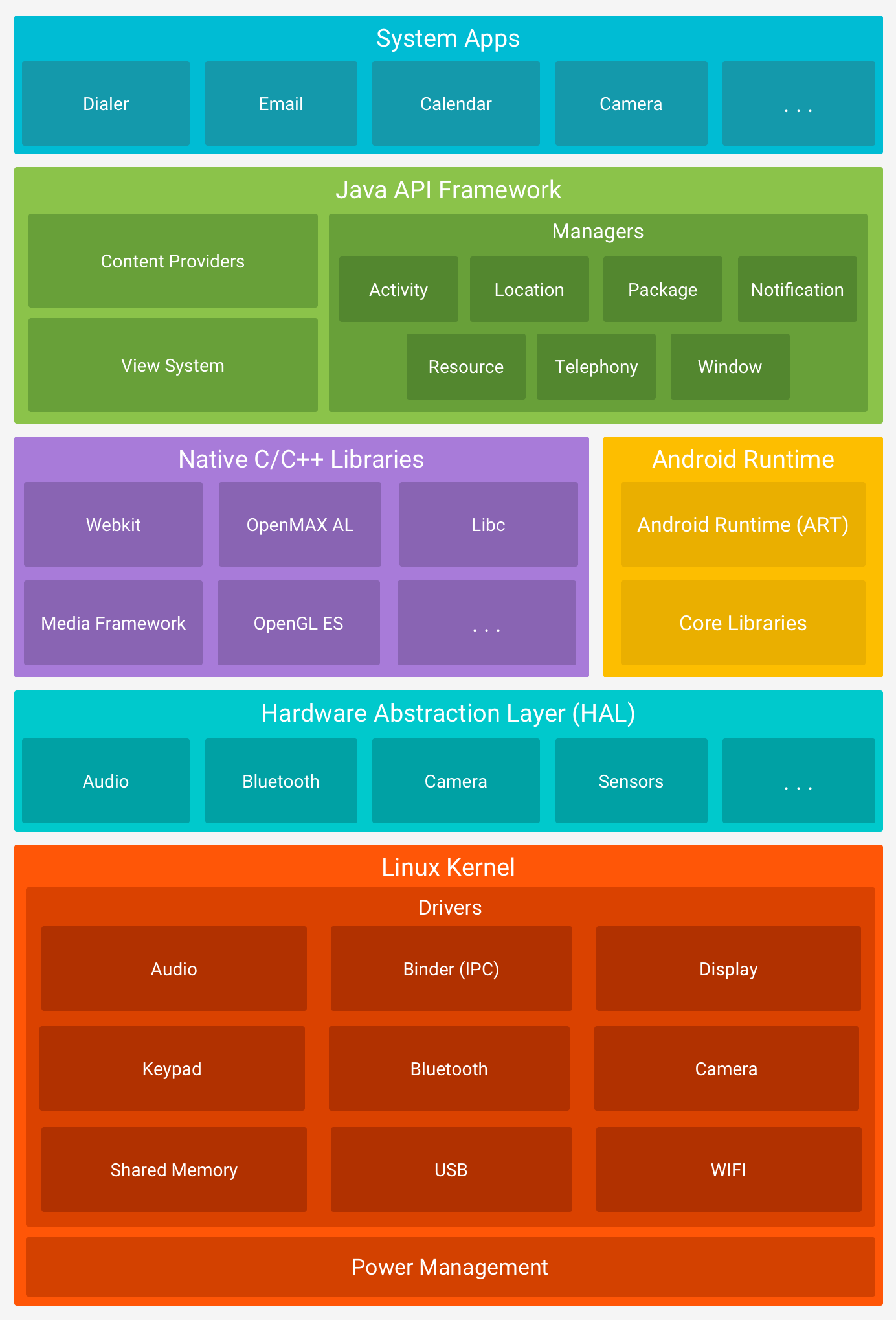
## Android platforma

Android (Slika 1) je otvoreni operativni sustav razvijen za mobilne uređaje od strane Google-a. Izgrađen je na principima otvorenosti („Open Source“ – korisnici imaju uvid u kod sustava i slobodu modificiranja istog) i modularnosti, zbog čega je vrlo brzo postao popularan izbor sustava za mnoge tehnološke kompanije koje se bave proizvodnjom mobilnih uređaja.

Temeljen je na Linux kernelu (monolitni kernel za operativne sustave), i sam framework OS-a je pisan u C/C++ programskom jeziku. Zbog svojeg principa otvorenosti programskog koda, posebna se briga vodi o tome da korisničke aplikacije ne komuniciraju direktno s low-level funkcijama samog uređaja, kao što su korištenje kamere, slanje SMS poruka i sl. (Slika 2). Iako je framework pisan u C/C++, većina aplikacija za Android pisana je u Java programskom jeziku, pomoću Android SDK (Software Development Kit) unutar kojeg je uključen i robusni IDE imena Android Studio kako bi se olakšala izrada aplikacija. Postoji i opcija pisanja programa u C/C++ jeziku, čime se može postići bolje raspolaganje resursima u aplikaciji no to također je dosta složenije te nije u dometu ovog rada, iz kojeg razloga sam koristio Android Studio kao IDE u kojemu je ovaj rad izrađen.



Slika - Trenutni logo Androida



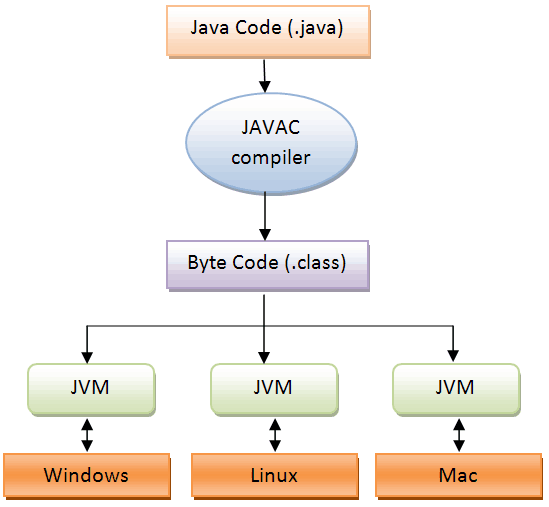
Slika - Shematski prikaz arhitekture Android operativnog sustava. Vidimo podjelu sustava na razine.

## Java programski jezik

Slika - Trenutni logo Jave

Java je objektno orijentiran programski jezik baziran na klasama koji se od svojih suparnika razlikuje po tome što se izvođenje programa pisanih u njoj vrši unutar JVM-a (Java Virtual Machine) – apstraktnom virtualnom stroju, nevezano o platformi na kojoj se sam kod izvodi (engl. „Write once, run anywhere“ – Piši jednom, pokreni bilo gdje).

Funkcionira tako da se Java izvorni kod pretvara u Java bytecode, kojeg JVM čita. (Slika 4) To omogućuje visoki stupanj prenosivosti koda između različitih platformi i u velike olakšava istovremeni razvoj programa za više platformi, smanjenjem vremena potrebnog za prilagođavanje koda platformi. Upravo to je dovelo do široke primjene Jave u raznim područjima informacijske tehnologije kroz širok spektar različitih uređaja, od osobnih računala i mobitela do GPS sustava u autima, bankomata i drugih jednostavnijih uređaja. Oracle Corporation se upravo zato na svojim stranicama hvali s impresivnom činjenicom da „15 bilijuna uređaja koristi Javu“ [2].



Slika - Pojednostavljeni shematski prikaz compile procesa Java koda

## MP3 datoteke

Punog imena MPEG-1 i/ili MPEG-2 Audio Layer III, no poznatije pod imenom MP3, MP3 datoteke su najrašireniji tzv. *lossy* (s gubitcima) format digitalnog audio kodiranja. Korištenje ovog codec-a tipično smanjuje desetostruko veličinu datoteke koju kodiramo, no većini slušatelja zvuči potpuno isto kao i original. Funkcionira tako da smanjuje (aproksimira) dio zvučnog spektra koji se smatra izvan dometa ljudskog sluha. Sama MP3 datoteka je sastavljena od okvira (engl. frame) koji se sastoje od zaglavlja i bloka podataka.

Iako službeno ne postoji standardni format za skladištenje takvih podataka, unutar te strukture su posebno definirani tzv. meta podatci ID3v2 formata (*de facto* standard), tagovi, koji su fokus ovog maturalnog rada. ID3v2 je nastao proširivanjem ID3v1 standarda, i nudi veliki broj dodatnih polja za upis informacija uz one koje je nudio ID3v1 (izvođač, broj pjesme na disku, album, godina itd.) kao što su polje za upis uvjeta korištenja glazbene datoteke, web stranica izvođača, datum distribucije, i slično.

## mp3agic Java library

Library (Hr. programska knjižnica) je naziv za zbirku potprograma koji nude gotova rješenja centrirana oko neke osnovne zadaće, i koja nam se nudi kao modularan dodatak našem kodu. U praksi se programske knjižnice koriste kada bi ručno obavljanje iste zadaće iziskivalo previše vremena. Mnoge su dostupne besplatno i s potpunim uvidom u njihov kod na servisima kao što je GitHub [3], i svaki suvremeni programski jezik ima neki oblik istih.

mp3agic je Java programska knjižnica za manipulaciju ID3 tagovima unutar MP3 datoteka, te je bila nužna u izradi ovog maturalnog rada. Nudi jednostavno i intuitivno upravljanje tagovima, uvodeći novi tip podatka, mp3file, koji deklariramo unutar našeg projekta kao što bi bilo koji drugi, i dolazi s iscrpnom listom funkcija za manipulaciju istima.

# Razvojna okolina aplikacije

## Android Studio

Kao što je prije spomenuto, Google u svome paketu za android programere nudi Android Studio [4.] kao službeni IDE (integrated development environment – integrirano razvojno okruženje). Iako se programerima nudi na izbor široka lepeza drugih IDE-ova, on je prihvaćen kao standard.

Baziran na InteliJ IDEA platformi i razvijen specifično za razvoj Android aplikacija, uz bogat izbor funkcija te dostupan potpuno besplatno, logičan je izbor većine Android programera, od početnika do komercijalnih pothvata.

Neke od funkcija koje on nudi su:

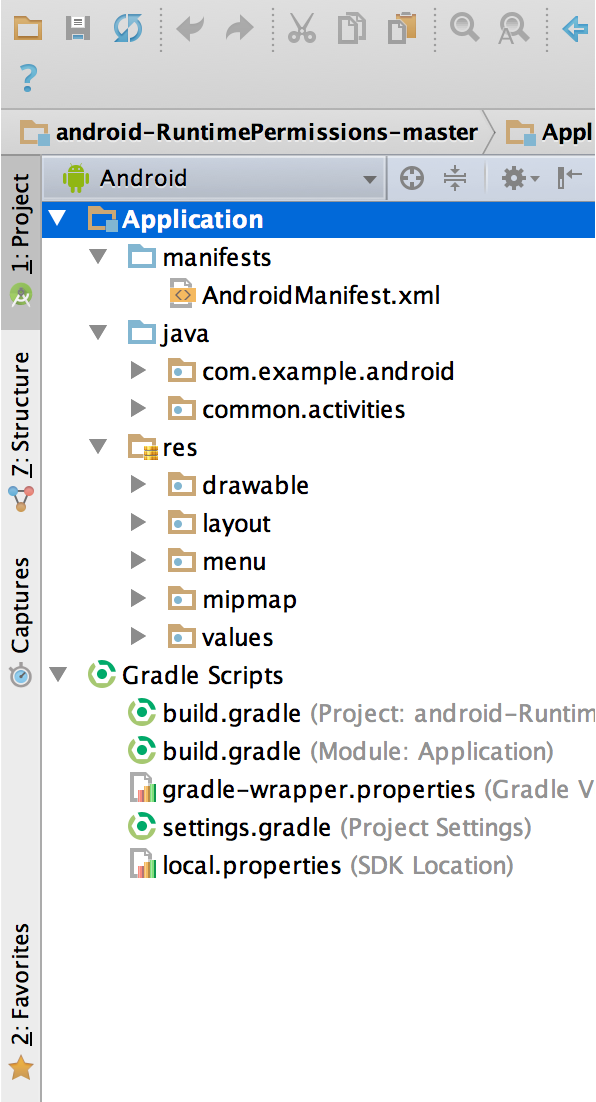
* Integracija s Gradle automatskim sistemom izgradnje programa
* Refaktoriranje Android koda
* Vizualizacija izgleda aplikacije
* Podrška za egzotičnije android uređaje – satove i sl.
* Simulator android uređaja

Za pripremu Android Studio programa za rad potrebno je skinuti instalacijski .exe file s Google-ove stranice i to je sve, svi dodatni alati i SDK će biti skinuti automatski, na izbor korisnika kroz dijalog programa. Specifično za moje računalo je bilo potrebno dodati liniju

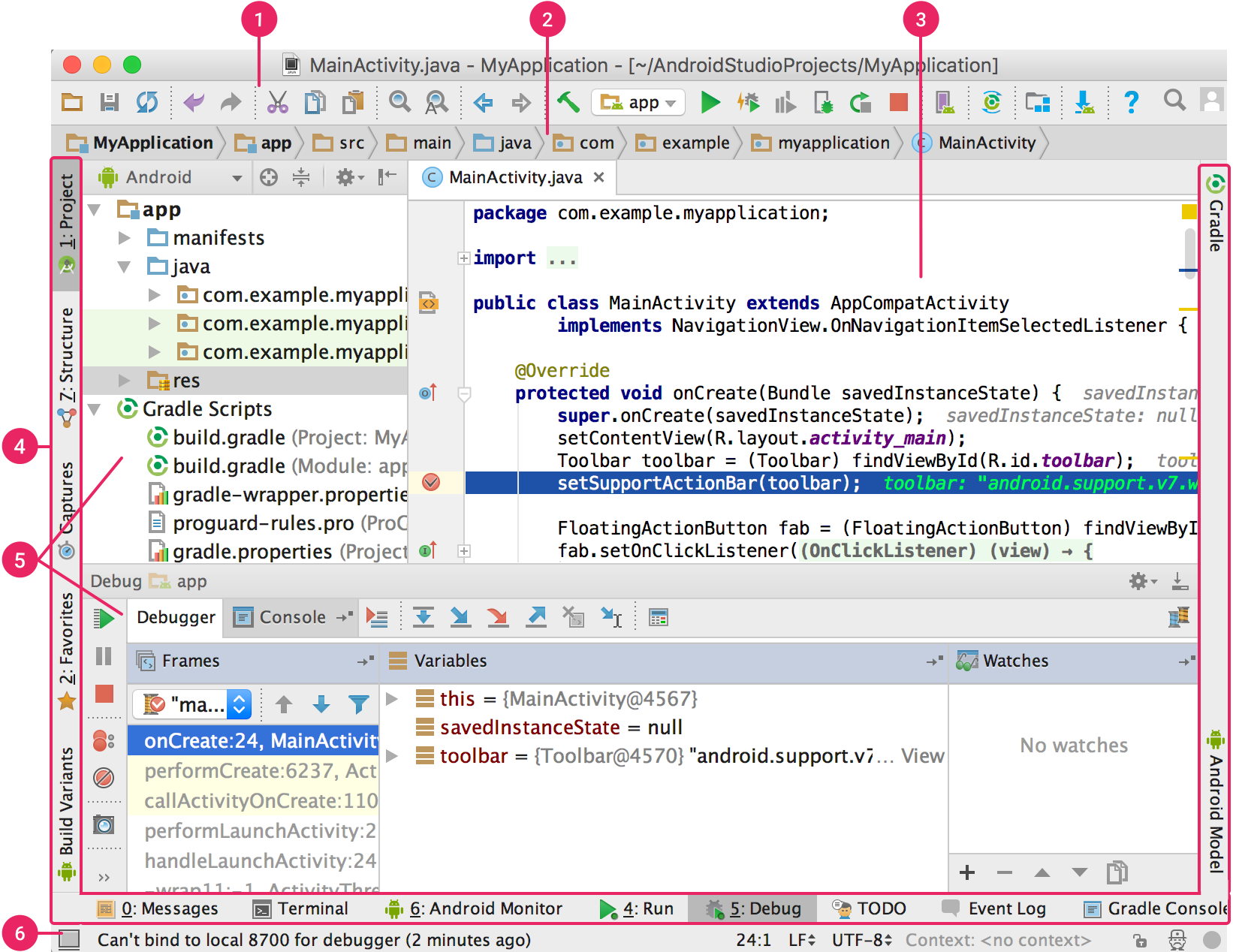
actionSystem.force.alt.gr=true

unutar idea.properties konfiguracijske datoteke koja se nalazi u instalacijskoj mapi Android Studio programa kako bi on funkcionirao s „starim“ tipom tipkovnice koja ima Alt Gr tipku, kakva je korištena za izradu rada.

Sučelje samog Android Studio IDE-a je odmah poznato bilo kome tko je ikada koristio nekakav IDE, ništa drastično nije drukčije. S lijeve strane pri otvaranju nas dočekuje tzv. *tree* dijagram datoteka unutar našeg projekta , koji nudi razne načine sortiranja. Možemo npr. sortirati tako da vidimo samo datoteke koje su relevantne za našu aplikaciju (Android View), čitav projekt (Project View), da vidimo samo datoteke s problemima (Problem View) i dr.



Slika - Izgled tree dijagrama unutar programa

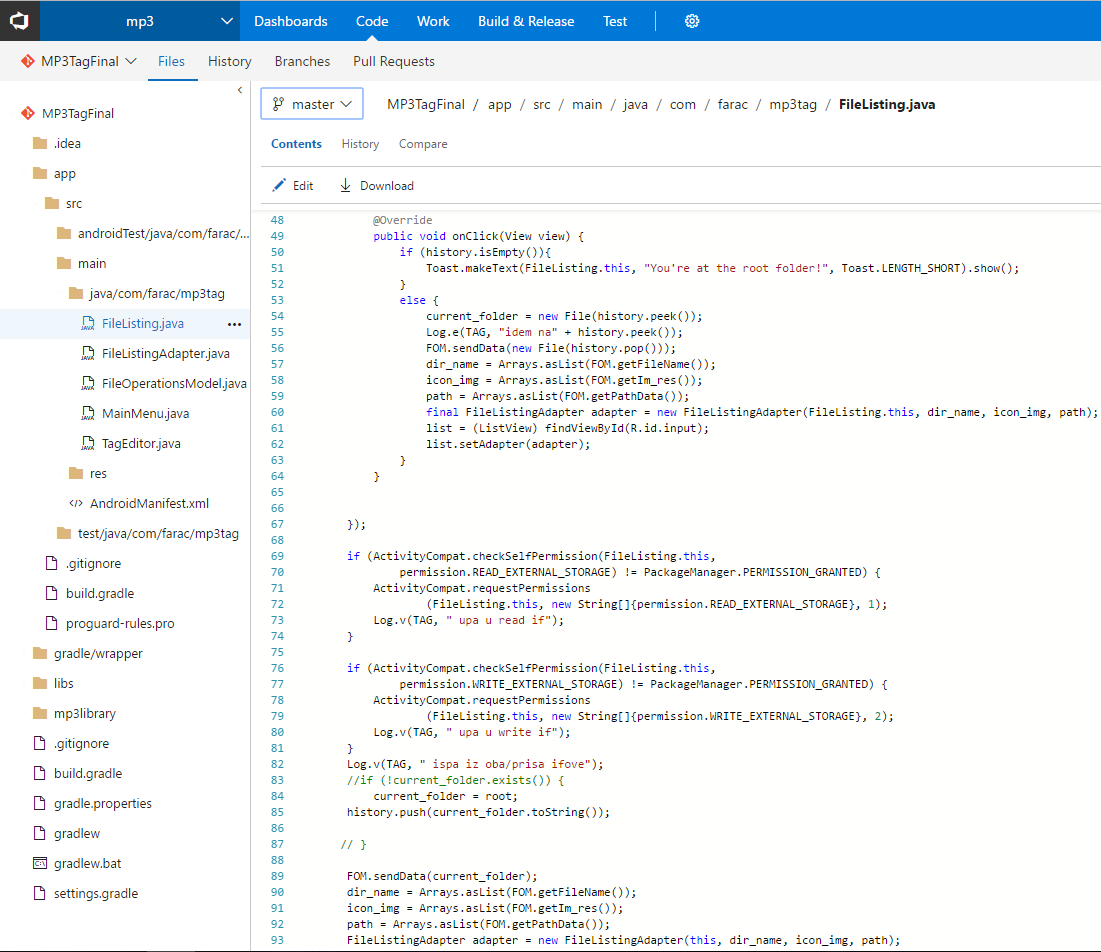
Čitav IDE je podijeljen na više područja, koji korespondiraju sljedećim funkcijama, prema brojevima s Slike 6:

Slika - Podjela Android Studio sučelja

1. Alatna traka – tu se nalaze najčešće korišteni alati te se od tu pokreće aplikacija u svrhu testiranja.
2. Navigacijska traka – prikazuje trenutno aktivnu datoteku i put do nje.
3. Prozor uređivanja – tu se piše i modificira sami kod aplikacije. Mijenja se ovisno o vrsti datoteke otvorene. Podržava bojanje sintakse.
4. Alatna prozorska traka – sadrži gumbove kojima se širi i minimizira različite prozore.
5. Alatni prozori – tu korisnik ima pristup različitim zadaćama, kao što su pretraživanje, upravljanje projektom, upravljanje verzijama programa i sl.
6. Statusna traka – ispisuje različite podatke o trenutnom stanju IDE-a, te upozorenja i poruke.

## Visual Studio Team Services

Visual Studio Team Services ili kraće TFS [5.] je Microsoft-ov proizvod koji nudi upravljanje izvornim kodom projekata. Konkretno, on nudi prostor za spremanje na svojim serverima, specijaliziran za spremanje izvornog koda. Kao takav ima funkcionalnosti grananja koda, sprema sve verzije koda koje na njega stavimo radi lakšeg vraćanja na prethodne verzije u slučaju da nam je to potrebno i slično. On dodjeljuje korisniju vlastiti račun na svojim uslugama, te kreira pod-domenu za njega. Također nudi direktan prikaz koda na bilo kojem računalu koje ima web pretraživač koji to podržava (Slika 7).

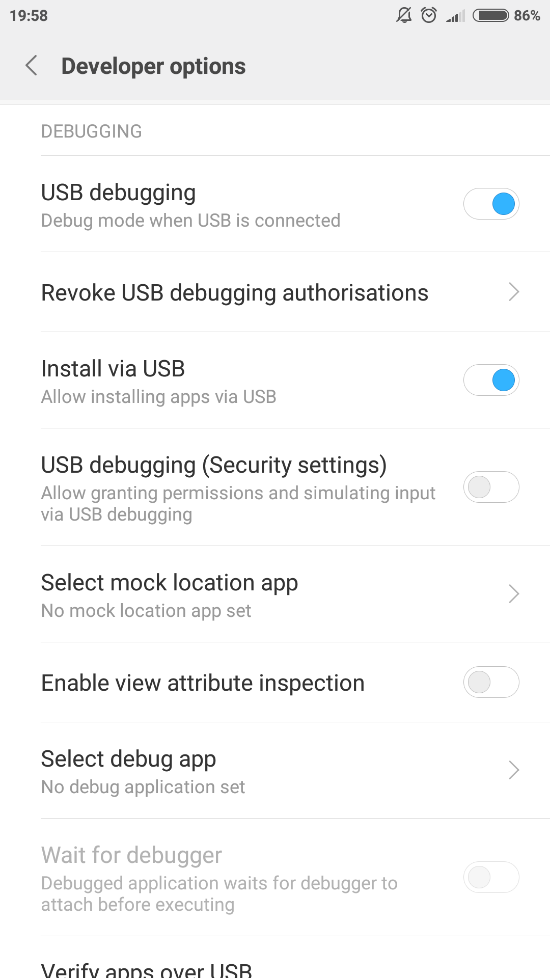


Slika - Prikazivanje koda moje aplikacije u Visual Studio Team Services

## Testni uređaj

Kao testni uređaj korišten je Xiaomi Redmi Note 3 Pro, makar je dobra opcija bila i Android emulator, testiranje na hardverskom uređaju je znatno brže i ne dodatno opterećuje računalo. Mobitel je na Android verziji 6.0.1 što odgovara API razini 23 (Ova aplikacija koristi 22, dakle 5.1 – API razine su naniže kompatibilne)

Prije korištenja mobitela kao testne okoline za aplikaciju potrebno je uključiti USB Debugging u Settings/Developer options/USB Debugging (Slika 8). (Ovisno o proizvođaču mobitela za pristup Developer options može biti potreban dodatan korak.) To omogućuje računalu s Android SDK da direktno se poveže s Android uređajem u svrhu testiranja aplikacije. Pomoću toga dobivamo povratne informacije od uređaja u slučaju grešaka u programu ili ako želimo imati uvid u egzaktno stanje programa. Također je potrebno fizički uređaj povezati USB kablom.



Slika - Developer options

# Struktura aplikacije

Android aplikacija je definirana kao skup Java datoteka kojima se definira jedan ili više Android Activity cjelina (jedinstvena gradivna jedinica aplikacije koja služi kao polazna točka korisnikove interakcije s aplikacijom. [6.] Laički rečeno to je jedan ekran, prikaz android aplikacije.) i njemu pripadajućih .xml datoteka unutar kojih je definiran vizualni izgled onoga što je prikazano korisniku.

Ova aplikacija sadrži 3 razlicite Activity instance, jednu za glavni izbornik, jednu u kojoj se korisniku nudi izbor datoteka na vlastitom uređaju, i jednu u kojoj se .mp3 datotekama modificiraju ID3v2 tagovi.

Od bitnijih ostalih datoteka i mapa koje valja spomenuti imamo:

* colors.xml unutar kojeg se definiraju različite boje koje se koriste unutar aplikacije u hex formatu
* strings.xml unutar kojeg se definiraju različiti stringovi teksta, radi jednostavnosti prevođenja aplikacije. Iako je norma koristiti ga rađe nego direktno u ostale datoteke pisati tekst, malo se koristi u ovom radu
* razne styles.xml datoteke unutar kojih se definira stil aplikacije ili pojedinih Activity instanci
* dimens.xml unutar kojeg se definiraju razne dimenzije, isto pravilo kao i kod strings.xml – norma je pisati dimenzije ovdje, no kako je ovaj projekt predviđen samo za jedan uređaj, nema potrebe
* drawable mapu unutar koje su spremljene slikovne datoteke za prikaz u aplikaciji u različitim rezolucijama zbog različitih gustoća ekrana između uređaja
* AndroidManifest.xml unutar kojeg se definiraju instance Activity, ikona aplikacije, dopuštenja koja aplikacija traži od uređaja i slično.



Kod - AndroidManifest.xml datoteka rada. Vidimo da program traži dopuštenje da čita i piše na vanjsku memoriju. Također vidimo definirane Activity instance – MainMenu, FileListing i TagEditor.

Aplikacija je složena tako da MainMenu sadrži jedan gumb koji poziva FileListing koji ispisuje popis svih direktorija i datoteka na vanjskoj SD kartici uređaja i dopušta pregled istih, korisnik bira željenu .mp3 datoteku. Kada odabere, to poziva TagEditor koji prima odabrani .mp3 i dopušta manipulaciju njenog ID3v2 tag-a. (Slika 9).

MainMenu Activity

FileListing Activity

TagEditor Activity

activity\_main\_menu.xml

activity\_file\_listing.xml

activity\_tag\_editor.xml

file\_list.xml

file\_listing\_layout.xml

Slika - Shema interakcija unutar aplikacije

Sa slike 9 vidimo da MainMenu nasljeđuje svoj vizualni izgled od activity\_main\_menu.xml, te na isti način ostale 2 aktivnosti od svojih .xml datoteka. activity\_file\_listing unutar sebe uključuje file\_listing\_layout. MainMenu jednosmjerno poziva FileListing. FileListing podate šalje u i prima obrađene iz .java klase FileOperationsModel radi preglednosti i višekratnog korištenja. Da bi prikazao popise datoteka FileListing obrađene izliste šalje u FileListingAdapter koji s file\_list.xml kao predloškom generira članove popisa koji se vraćaju u FileListing i ispisuju unutar popisa definiranog unutar file\_listing\_layout.xml. TagEditor pri završavanju vlastitog izvršavanja vraća fokus na FileListing.

# Opis koda

## MainMenu Activity

Kod - activity\_main\_menu.xml

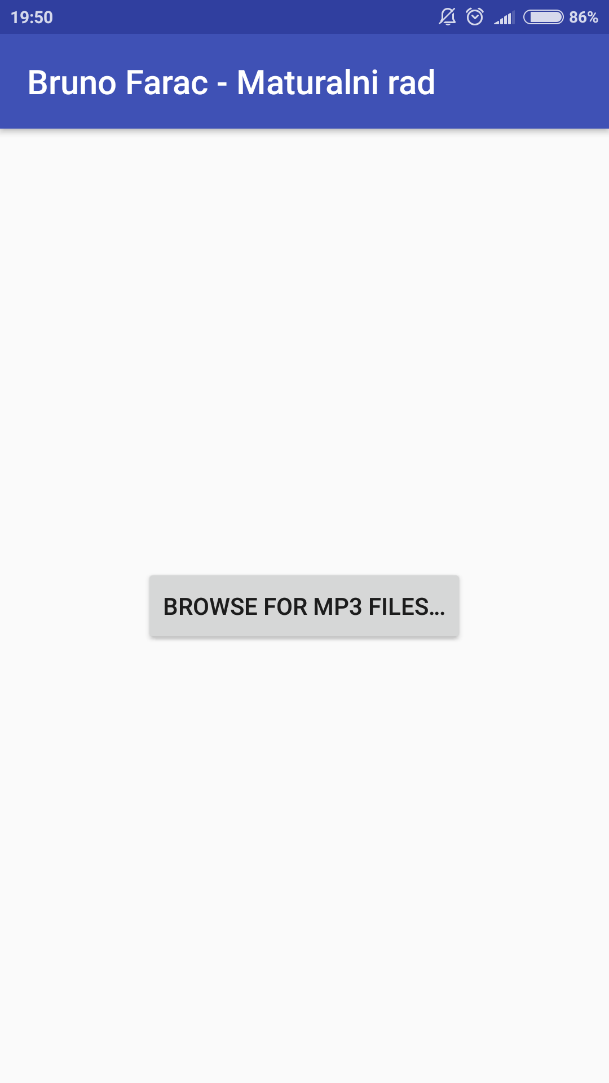


MainMenu (Kod 2) je Activity koji je definiran kao početni pri pokretanju aplikacije u AndroidManifest.xml. Njegov . activity\_main\_menu.xml ga definira da ima jedan gumb linearno po sredini – LinearLayout i općenito vrste Layout-ova unutar .xml datoteka definiraju način na koji su elementi prikaza međusobno smješteni u prostoru. Konkretno LinearLayout svoje elemente iscrtava u jedan red ili stupac, ovisno o *orientation* svojstvu. *Gravity* svojstvo određuje oko čega će biti elementi simetrični, u ovom slučaju oko centra. Tekst ispisan na gumbu je definiran unutar prije spomenutog strings.xml. Primijetimo kako je ID gumba definiran kao *button\_edit\_files.*



Kod - MainMenu.java

Upravo se na taj ID poziva MainMenu.java (Kod 3). Pri početku je definiran gumb *mButtonEditTags* kao Button element. OnCreate (prvo izvršavanje MainMenu instance) postavlja izgled ekrana kao definiran unutar activity\_main\_menu. Naslov ekrana se postavlja kao „Bruno Farac – Maturalni Rad“). Potom se definirani gumb namješta da korespondira gumbom definiranim unutar .xml datoteke pomoću svog ID svojstva. Na njega se smješta tzv. listener (hrv. slušač) koji „osluškuje“ kada će gumb biti pritisnut te kada je pritisnut – onClick – pomoću intent elementa (hrv. namjera) započinje aktivnost FileListing. Pri početku vidimo kako smo sve korištene resurse morali definirati i „uvesti“ import naredbom. Jednostavnosti radi ostatak koda će biti prikazan bez tog dijela.



Slika - Izgled MainMenu Activity

## FileListing Activity

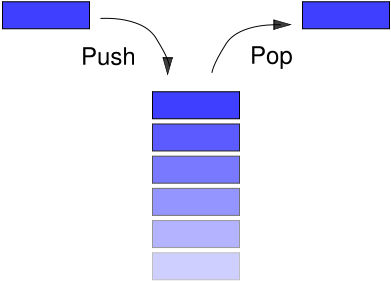
Pozivom iz MainMenu započinje Activity FileListing. Odmah pri kreiranju on poziva i deklarira određene varijable i konstante. (Kod 4)



Kod - FileListing.java – Deklaracije

Deklarira se File objekt koji će služiti kao trenutni folder, *current\_folder*  te se file root postavlja kao root folder vanjske SD kartice. File objekti su objekti koji služe za definiranje datoteka i mapa. Zapravo u sebi sadrže apstraktan path do pojedinog direktorija ili datoteke [7.]. Nakon toga se deklarira ListView objekt *list* na kojeg kasnije namjestimo adapter. Potom imamo 3 List objekta su tu kako bi primali popise imena direktorija, apstraktnih vrijednosti ikone koja će se na listi ocrtati za odgovarajuće datoteke i potpunih path-ova do svakog. List objekti su u suštini polja kojima možemo po potrebi mijenjati veličinu u hodu. Potom imamo Stack koji prima String objekte u kojeg ćemo zapisivati naš put kroz direktorije da bi mogli korisniku pružiti funkcionalnost povratka na prethodni direktorij.

Stack objekti su LIFO (Last In First Out – Zadnji Unutra Prvi Vanka) sekvence (Slika 11) [8.]. S njima možemo upravljati Push i Pop naredbama, koje stavljaju vrijednost na vrh stoga i uklanjaju je vraćajući uklonjenu vrijednost, tim redoslijedom, što ih čini idealnim za funkcionalnost koja nama treba.



Slika - LIFO; Push i Pop naredbe

Na posljetku imamo deklaraciju FileOperationsModela (dalje u tekstu kao FOM), koji je klasa koju smo kreirali da nam olakša baratanje s podatcima i pruži potrebne funkcije. U kodu 6, 7 i 8 su prikazane funkcije koje on obavlja.



Kod – FileOperationsModel.java – globalno deklarirane varijable



Kod - FileOperationsModel.java - sendData funkcija

sendData funkcija se koristi za slanje podataka FOM-u. Prima File podatke koje odmah izlistava unutar nove varijable *f*, potom deklarira 3 polja dužine jednake dužini liste *f*. Lista f biva slagana pomoću Arrays.sort, što ih abecedno slaže. Potom se unutar for petlje za trenutni *i* događa sljedeće:

* za polje *file\_name*, unutar kojeg ćemo spremiti imena datoteka i direktorija, pribavlja ime trenutnog elementa u listi i zapisuje na odgovarajuće mjesto
* za polje *fpath* (puni path) se na isti način pribavlja apsolutni (puni) path i zapisuje u njega
* za polje *im\_res*, koje zapisuje odgovarajuće ikone za prikazivanje u izlistu mapa i datoteka se prvo rezanjem unutar String *ext* zapisuje ekstenzija trenutne datoteke. Potom se kreira novi File *tf* pomoću trenutne path vrijednosti za *i*.
  + Ako je tf direktorij, kao ikona za ovu vrijednost se stavlja *ic folder,* ikona direktorija.
  + Ako je ekstenzija jednaka mp3 za ovu vrijednost se stavlja *ic\_music­\_note*, ikona note
  + Ako nijedna od gornjih tvrdnji nije točna, postavlja se *ic\_insert\_drive\_file*, ikona datoteke

Svi ovi obrađeni podatci se spremaju unutar FOM-a, u varijable deklarirane u Kodu 5.



Kod - FileOperationsModel.java – isMP3 funkcija

isMP3 funkcija prima File vrijednost i vraća Boolean vrijednost (vrijednost koja može imati 2 stanja – istina i laž, 1 i 0) ovisno jeli File primljen .mp3 ili ne. Princip je jednostavnija verzija druge točke for petlje iz Koda 6.



Kod - FileOperationsModel.java - povratne funkcije

Tri funkcije koje vidimo pod Kod 8 služe za dohvat vrijednosti iz FOM-a dobivenih s sendData.



Kod - FileListing – OnCreate

Kada se FileListing aktivnost prvi put pozove, ona obavi isto što i MainMenu u Kod 3, primi vizualni izgled iz activity\_file\_listing.xml koji unutar sebe ima uključen file\_listing\_layout unutar kojeg je deklarirana lista (Kod 10). Kreira se i Toolbar idem pomoću kojeg imamo gumb vidljiv na Slici 12.



Kod - file\_listing\_layout.xml – ListView deklaracija

Na klik tog Toolbar gumba ako je Stack *history* prazan korisnika se Toast porukom (kratke poruke koje se pojave pri dnu ekrana) [9.] obavještava da se on nalazi u najnižoj razini direktorija mogućoj. No ako nije, najgornja vrijednost u Stacku se ubacuje u *current\_folder*, šalje ga se na obradu u FOM te se uklanja iz stoga. Primaju se vrijednosti iz FOM, te se kreira novi adapter kojem se šalju te vrijednosti, te se taj adapter povezuje na ListView iz Koda 10. Adapter radi na sljedeći način, kao što je vidljivo u Kodu 11:

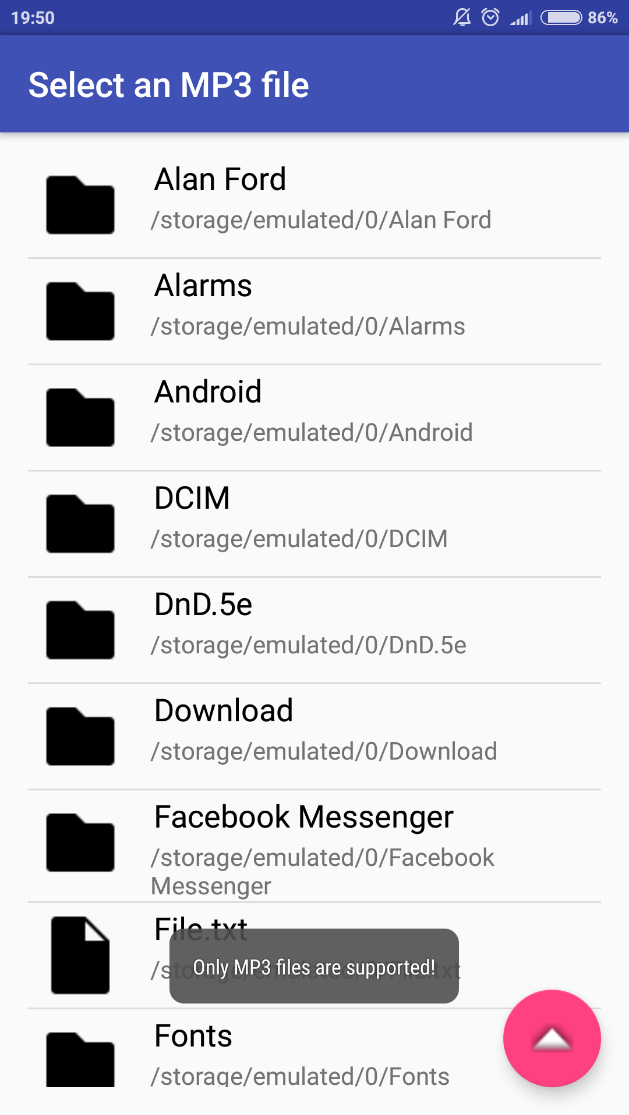


Kod - FileListingAdapter

(Napomena: Kod 11 ne sadrži deklaraciju)

Deklarira se FileListingAdapter koji prima kontekst (odakle je pozvan) te određene podatke u sebe. Pomoću super() se poziva na konstruktor s svojeg parent-a „roditelja“, na klasu koja je jednu razinu iznad njega s vrijednostima koje su uvrštene u konstruktor. U ovom slučaju na FileListingAdapter koji proširuje ArrayAdapter. Postavljaju se vrijednosti varijabli koje on prima.

getView za te iste vrijednosti vrši inflate (hrv. napuhuje), po šabloni danoj mu u file\_list.xml, za elemente svake pojedine liste podataka koje smo mu poslali – vrijednosti obrađene u FOM – i vraća natrag rowView () – redove, pretvorio je podatke u liste koje se mogu prikazati unutar ListView iz Koda 10. Na Slici 12 vidimo kako to izgleda gotovo.



Slika - FileListing



Kod – FileListing- dopuštenja

Nakon obrade gumba iz Toolbara imamo isječak iz Koda 12 koji se brine za to da nas aplikacija kada se prvi put nađemo u ovoj aktivnosti pita dopuštenje da piše na i čita s eksterne memorije uređaja. Naime, iako je već definirano da su ta dopuštenja potrebna aplikaciji unutar AndroidManifest.xml, od API levela 23 je dodano da korisnici mogu prihvatiti dopuštenja koje aplikacija treba kasnije, ne pri instalaciji, i da potencijalno opasna dopuštenja (kao što je čitanje i pisanje) moraju biti eksplicitno prihvaćena pri pokretanju aplikacije.

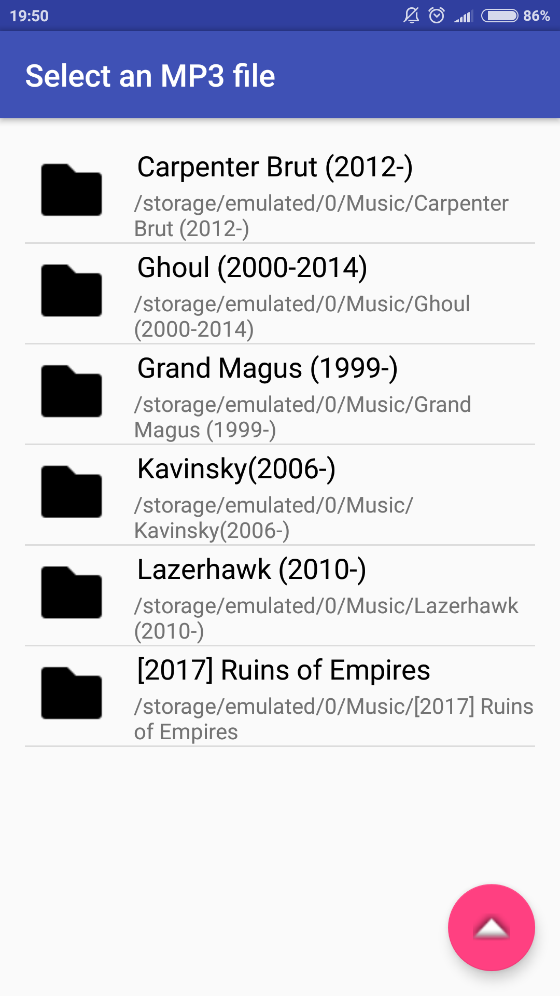
Nakon toga imamo poziv na adapter identičan onome unutar OnClick gumba Toolbara, koji služi za početno izlistavanje. Kao *current\_folder* je uzet *root* iz deklaracija u Kod 4. Također se taj folder gura na stack.

Da bi obradili korisnikov odabir iz izlista, potrebno je namjestiti OnItemClickListener objekt na list (Kod 13). Unutar njega šaljemo u FOM trenutni path (*cpath*). Kreiramo novi file *temp* unutar kojega spremamo File s path vrijednosti odabranog objekta s izlista. Ako je taj File:

* Direktorij – ponovo pozivamo adapter s tom path vrijednosti na već opisan način (Kod 9), kreira se novi izlist podataka unutar njega i čitav se proces ponavlja (Slika 13)
* .mp3 datoteka – započinjemo TagEditor Activity instancu u koju pomoću .putExtra šaljemo path odabrane datoteke (5.3.)
* Neka druga datoteka – pokažemo korisniku poruku koja kaže da su samo MP3 datoteke podržane za obradu (Slika 12)



Kod - FileListing OnItemClickListener



Slika - Izlist direktorija

## TagEditor Activity

TagEditor je Activity unutar kojega obrađujemo .mp3 datoteke koristeći mp3agic library. Izgled joj je određen u activity\_tag\_editor.xml. Složen je tako da unutar vertikalnog LinearLayout definiramo TableLayout (tablicu) s po 2 stupca (Kod 14). TableLayout kao jedan od parametara prima layout\_column koji određuje u kojem će se relativnom stupcu odabrani element prikazati.



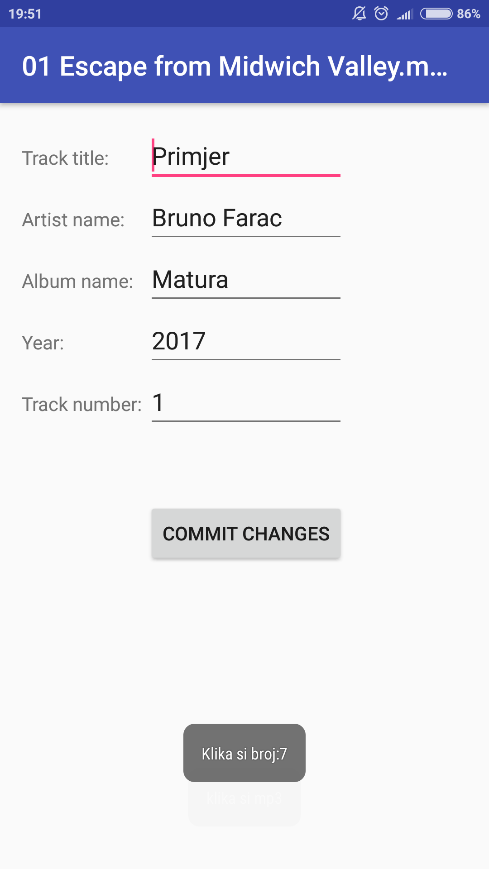
Kod - activity\_tag\_editor.xml

Unutar stupaca imamo jedan TextView koji nas upućuje na koju vrstu podatka aplikaciju tu prima te po jedan EditText koji je jednostavna crta na koju je korisniku dopušten upis podataka. Unutar njega možemo definirati vrstu podataka koju očekujemo, korisno za ograničavanje korisnika npr. na samo brojeve kod upisa godine. Takva konstrukcija je ponovljena onoliko puta koliko podataka tražimo (Slika 14). Zadnji red tablice se nešto razlikuje (Kod 15.)



Kod - activity\_tag\_editor.xml

U njemu smo kreirali gumb kojim će korisnik finalizirati svoj upis i/ili izmjene podataka i signalizira da se promjene upišu u memoriju.



Slika - TagEditor Activity



Kod - TagEditor.java – Deklaracije

U Kodu 16 vidimo deklaracije unutar TagEditor klase. Deklariramo podatke potrebne za daljnju obradu podataka. Primijetimo Mp3File i ID3v2 tip podatka – oni proizlaze iz mp3agic library koju smo uvezli u svoj projekt.



Kod - TagEditor.java – OnCreate

Iz Koda 17 vidimo da odmah nakon primanja izgleda iz activity\_tag\_editor, Activity dobavlja podatke koje smo joj proslijedili pomoću .putExtra (Kod 13). Nakon toga obrađujemo i inicijaliziramo sve interaktivne elemente unutar TagEditor, sve instance EditText i dugme koje nam služi za prihvaćanje unosa. Nakon toga stvaramo novi Mp3File *mp3*, okruženo s try i catch tako da uhvatimo eventualnu grešku u inicijalizaciji, deklaracija Mp3File to zahtjeva.



Kod - TagEditor.java – onCreate

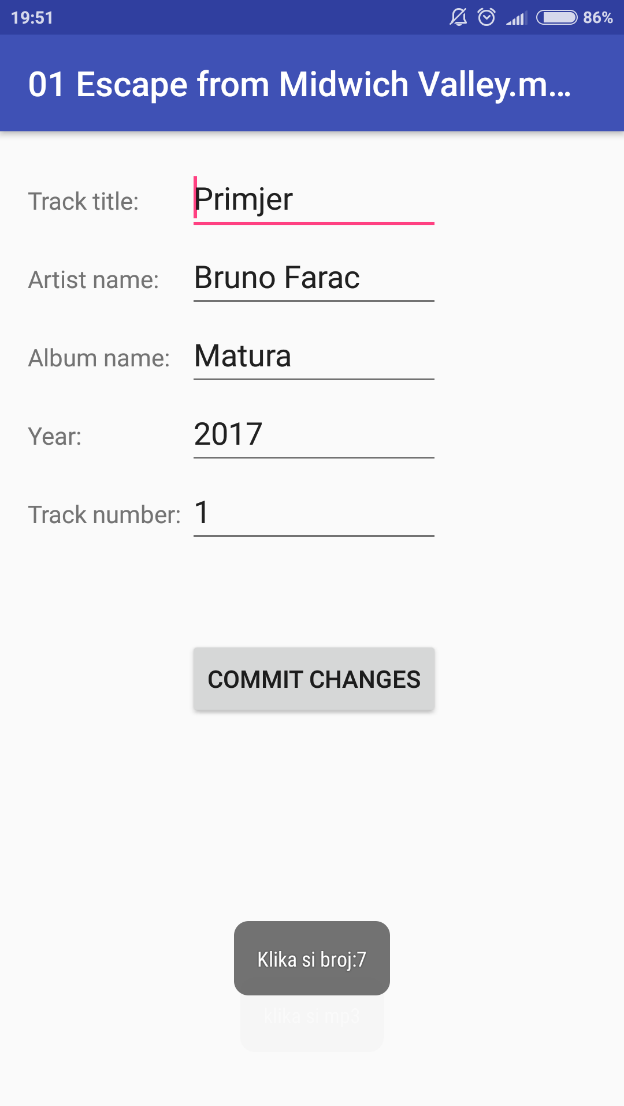
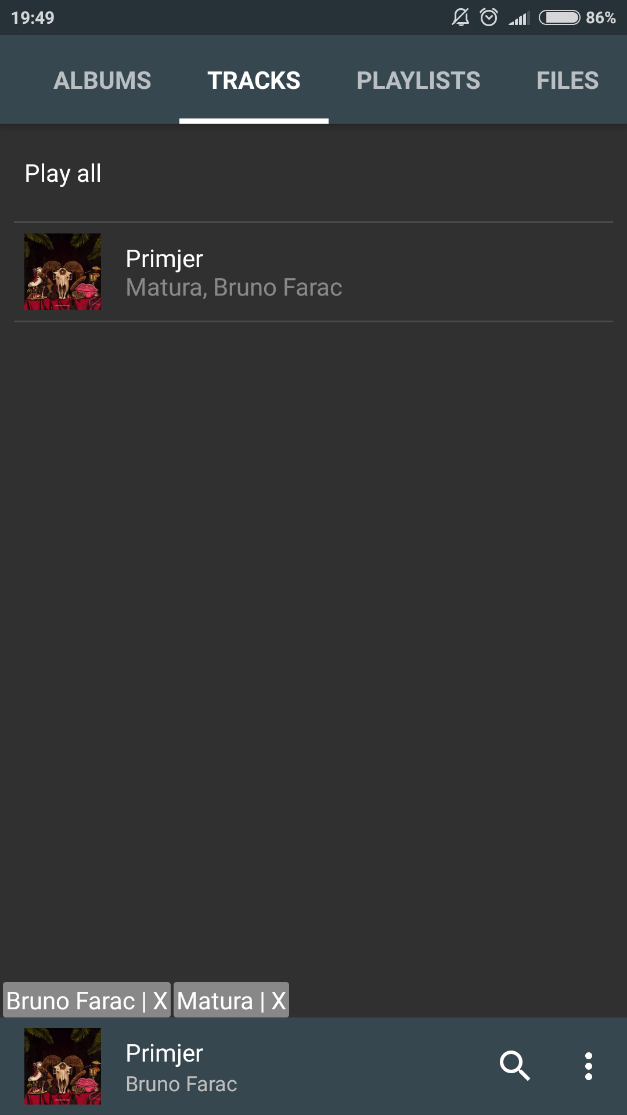
Potom stvaramo novi File od path koji smo prenjeli u TagEditor iz FileListing i njegovo ime postavljamo za naslov Activity instance. Ako *mp3* ima ID3v2 tag, zapisujemo ga u *tag,* ako ne stvaramo novi tag – tako aplikacija može obrađivati mp3 datoteke kojima nikad nisu ID3v2 polja popunjena. Postavljamo trenutne tag vrijednosti za podatke koje aplikacija može mijenjati kao trenutne vrijednosti unutar EditText polja.

Slijedi postavljanje onClickListener na *commit* gumb (Kod 19) kao što smo i u Kodu 3. Kada je pritisnut, upisane podatke dobivamo u obliku String na kojem vršimo .trim() kako bi eliminirali prazne znakove prije i poslije String. Postavljamo ih na za to predviđena mjesta u ID3v2. Konačni mp3 pokušamo spremiti, opet okružen s try i catch jer to zahtjeva funkcija.



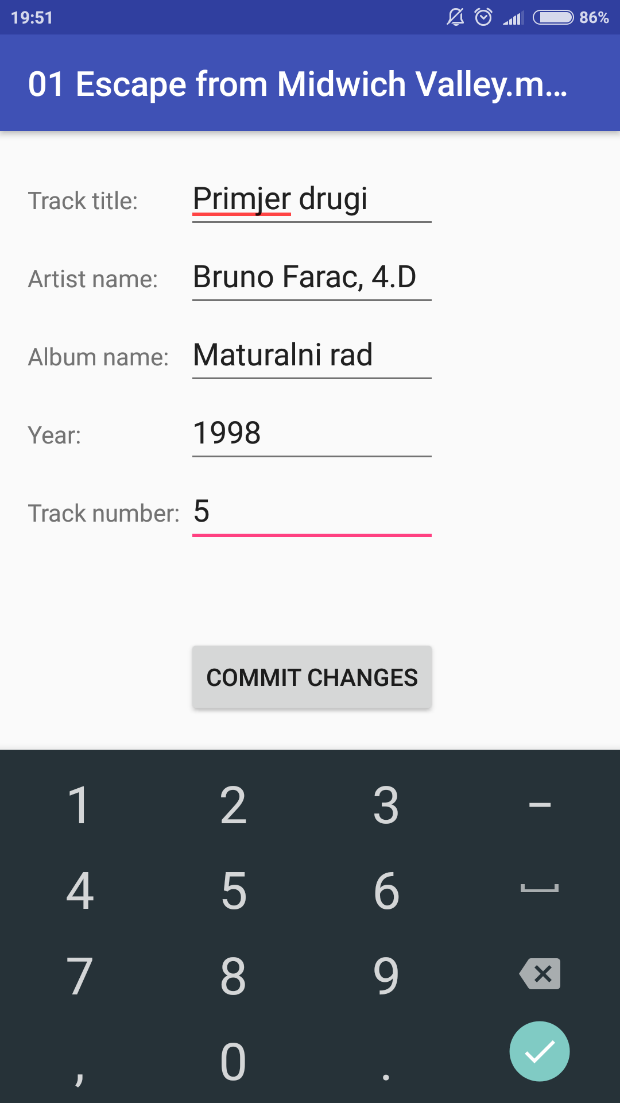
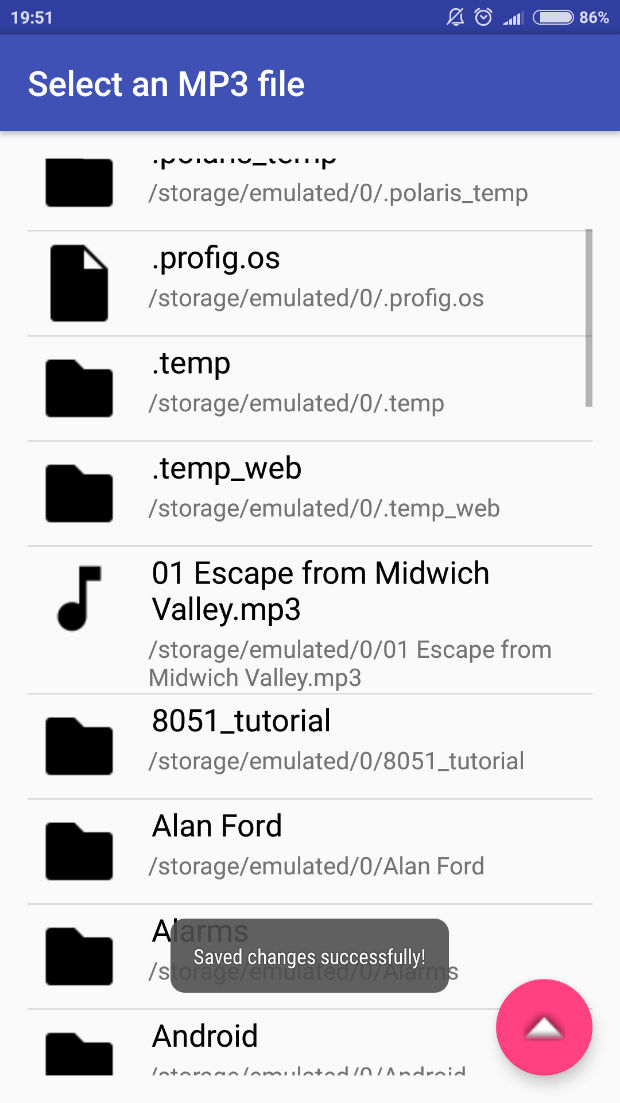
Kod - TagEditor.java – OnClickListener

No međutim mp3 ne spremamo direktno, nego ga prvo spremamo kao [ime].new, potom brišemo stari mp3 i novi preimenujemo tako da odgovara starome. Razlog tomu je zapravo vrlo jednostavan, spremanje datoteke preko same sebe je u većini slučaja nemoguće, ili u najmanju ruku nije preporučeno na većini platformi, pa rabimo ovaj „trik“. Ako obje funkcije brisanja i preimenovanja uspiju, *a* i *b* poprime vrijednost 1 što vraća poruku korisniku da je operacija uspjela, u suprotnom javlja grešku. Primjer uspješnog korištenja aplikacije ispod, Slike 15, 16, 17, 18 i 19.



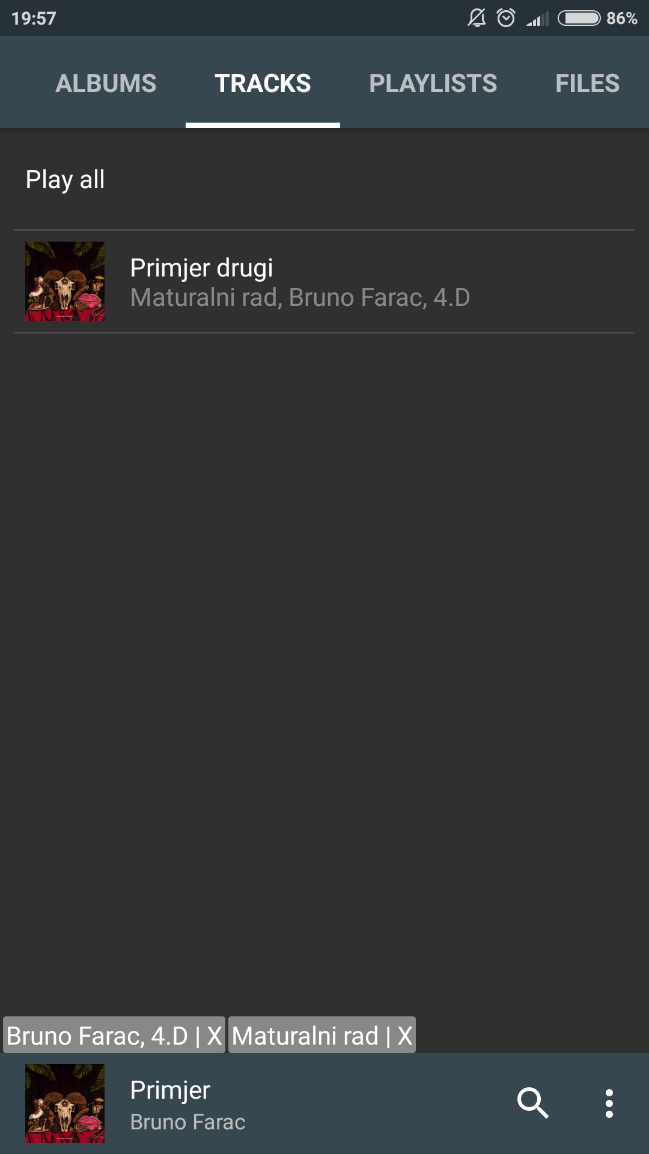
Slika - .mp3 prije uređivanja, u player programu

Slika - .mp3 prije uređivanja



Slika - .mp3 uspješno spremljen

Slika - .mp3 prije finalizacije uređivanja



Slika - .mp3 nakon uređivanja u player programu

# ZAKLJUČAK

Kroz ovaj rad sam se detaljno upoznao s Android razvojnim okruženjem te sam stekao korisna praktična znanja o istoj. Ponovio sam Java programski jezik, što bi mi moglo pomoći u budućim pothvatima u struci i daljnjem školovanju. Iz prve ruke sam iskusio kako kreativno rješavanje problema u programiranju izgleda, te na kakav način tražiti potrebne informacije ukoliko mi zatrebaju usred programskog projekta. Također sam naučio na koji način je najbolje otklanjati pogreške i druge probleme s radom programa.

Jedan od neotklonjivih problema s kojim sam se susreo je taj da I/O (input/output, ulazne/izlazne) operacije iziskuju veliku količinu energije iz baterije uređaja, no to je jednostavno u naravi android platforme [10.]. Program nije namijenjen za stalan rad, nego je više osmišljen s brzim ispravama i sl. na pameti.

Sam rad je uspješno dovršen, i ostaje još dosta prostora za eventualno poboljšanje njegovih svojstava. Moguće je dodati još polja koja bi program čitao i pisao iz ID3v2 standarda. Također čitav izgled aplikacije bi se moglo unaprijediti, to mi nije bio primarni fokus, nego funkcionalnost i demonstrativnost samog projekta.

# LITERATURA

[1.] <https://www.statista.com/statistics/385001/smartphone-worldwide-installed-base-operating-systems/>

[2.] <https://go.java/index.html>

[3.] <https://github.com>

[4.] <https://developer.android.com/studio/index.html>

[5.] <https://www.visualstudio.com/team-services/>

[6.] <https://developer.android.com/reference/android/app/Activity.html>

[7.] <https://developer.android.com/reference/java/io/File.html>

[8.] <https://docs.oracle.com/javase/7/docs/api/java/util/Stack.html>

[9.] <https://developer.android.com/guide/topics/ui/notifiers/toasts.html>

[10.] <http://pages.cs.wisc.edu/~swift/papers/TR1808.pdf>

# POPIS SLIKA

[Slika 1- Trenutni logo Androida 4](file:///G:\sk\maturalni\zavrsni.docx#_Toc478589320)

[Slika 2 - Shematski prikaz arhitekture Android operativnog sustava. 5](#_Toc478589321)

[Slika 3- Trenutni logo Jave 6](file:///G:\sk\maturalni\zavrsni.docx#_Toc478589322)

[Slika 4 - Pojednostavljeni shematski prikaz compile procesa Java koda 6](#_Toc478589323)

[Slika 5 - Izgled tree dijagrama unutar programa 9](file:///G:\sk\maturalni\zavrsni.docx#_Toc478589324)

[Slika 6 - Podjela Android Studio sučelja 10](#_Toc478589325)

[Slika 7 - Prikazivanje koda moje aplikacije u Visual Studio Team Services 11](#_Toc478589326)

[Slika 8 - Developer options 12](file:///G:\sk\maturalni\zavrsni.docx#_Toc478589327)

[Slika 9 - Shema interakcija unutar aplikacije 15](#_Toc478589328)

[Slika 10 - Izgled MainMenu Activity 18](file:///G:\sk\maturalni\zavrsni.docx#_Toc478589329)

[Slika 11 - LIFO; Push i Pop naredbe 20](#_Toc478589330)

[Slika 12 - FileListing 26](file:///G:\sk\maturalni\zavrsni.docx#_Toc478589331)

[Slika 13 - Izlist direktorija 29](file:///G:\sk\maturalni\zavrsni.docx#_Toc478589332)

[Slika 14 - TagEditor Activity 31](file:///G:\sk\maturalni\zavrsni.docx#_Toc478589333)

[Slika 15 - .mp3 prije uređivanja, u player programu 35](file:///G:\sk\maturalni\zavrsni.docx#_Toc478589334)

[Slika 16 - .mp3 prije uređivanja 35](file:///G:\sk\maturalni\zavrsni.docx#_Toc478589335)

[Slika 17 - .mp3 uspješno spremljen 36](file:///G:\sk\maturalni\zavrsni.docx#_Toc478589336)

[Slika 18 - .mp3 prije finalizacije uređivanja 36](file:///G:\sk\maturalni\zavrsni.docx#_Toc478589337)

[Slika 19 - .mp3 nakon uređivanja u player programu 37](file:///G:\sk\maturalni\zavrsni.docx#_Toc478589338)