# FESB, SPLIT

Seminar

OBRADA ID3 TAGOVA NA ANDROID PLATFORMI

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 1 - 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 9 - 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 2 - activity\_main\_menu.xml

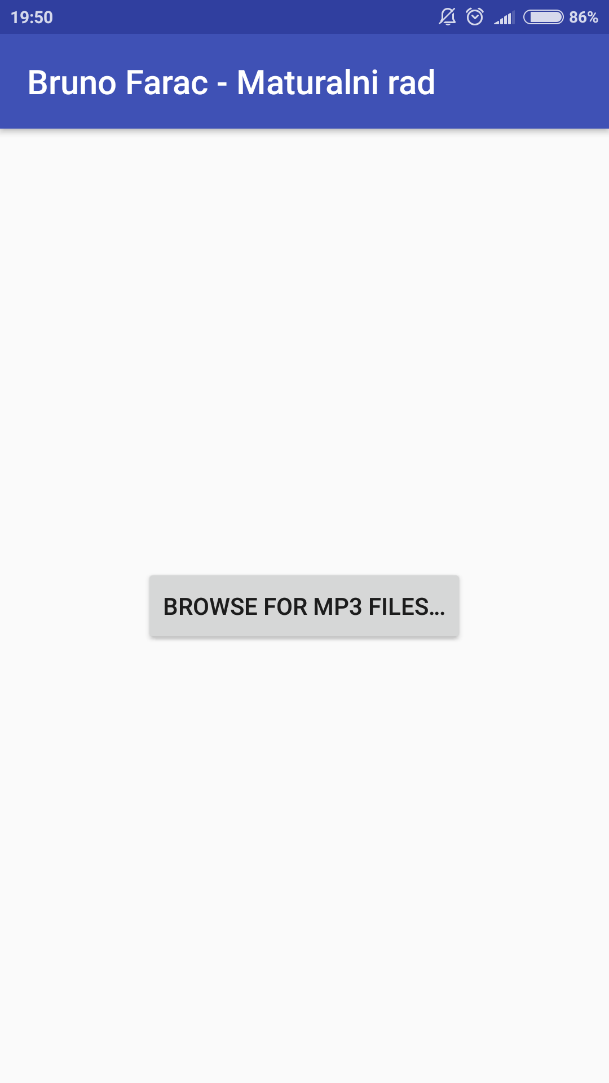


MainMenu (Kod 2) je Activity koji je definiran kao početni pri pokretanju aplikacije u AndroidManifest.xml. Njegov . activity\_main\_menu.xml ga definira da ima jedan gumb linearno po sredini – LinearLayout i općenito vrste Layout-ova unutar .xml datoteka definiraju način na koji su elementi prikaza 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 3 - 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 10 - Izgled MainMenu Activity

## FileListing Activity

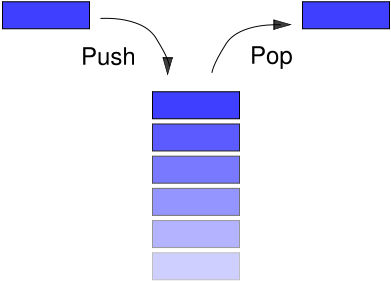
Pozivom iz MainMenu započinje Activity FileListing. Odmah pri kreiranju on poziva i deklarira određene varijable i konstante. (Kod 4)



Kod 4 - 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 11 - 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 5 – FileOperationsModel.java – globalno deklarirane varijable



Kod 6 - 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 7 - 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 8 - FileOperationsModel.java - povratne funkcije

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



Kod 9 - 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 10 - 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:

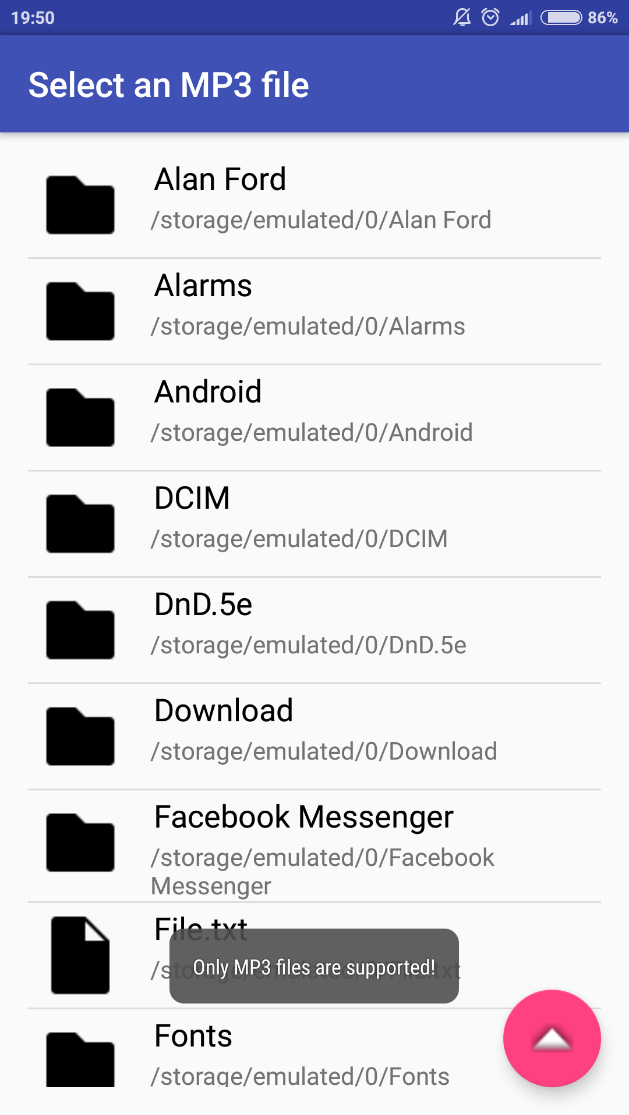


Kod 11 - FileListingAdapter

(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 12 - FileListing



Kod 12 – 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.

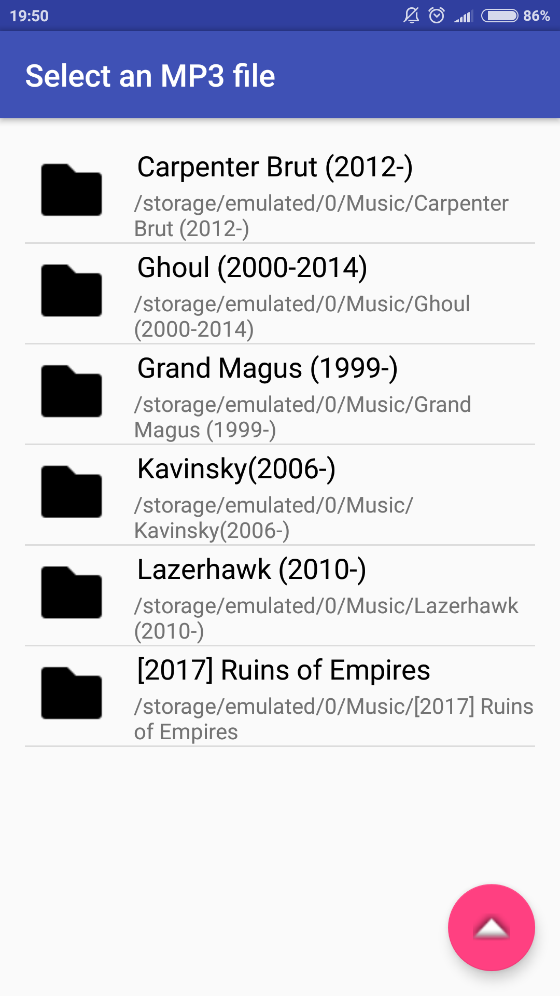
Nakon toga imamo poziv na adapter identičan onome unutar OnClick gumba Toolbara, koji služi za početno izlistavanje. Kao *current\_folder* je uzet *root* iz deklaracija u Kod 4. Također se taj folder gura na stack.

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 13 - FileListing OnItemClickListener



Slika 13 - 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.



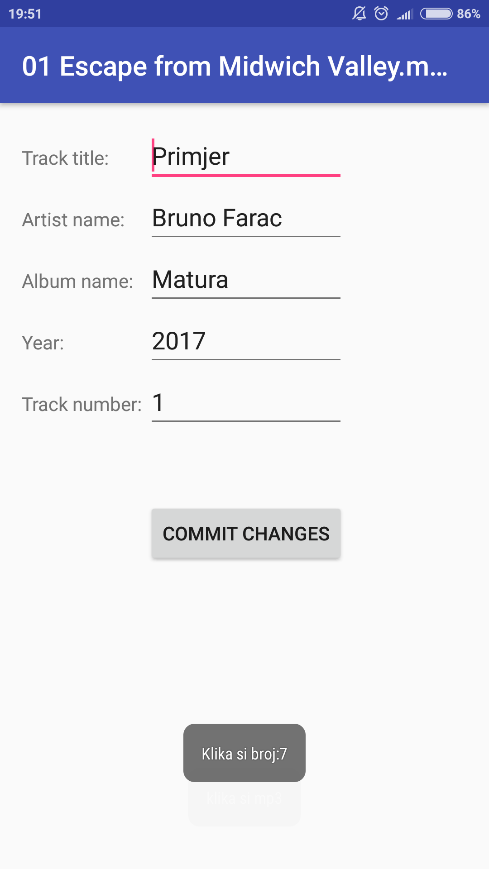
Kod 14 - activity\_tag\_editor.xml

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 15 - 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 14 - TagEditor Activity



Kod 16 - 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 17- 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 18 - TagEditor.java – onCreate

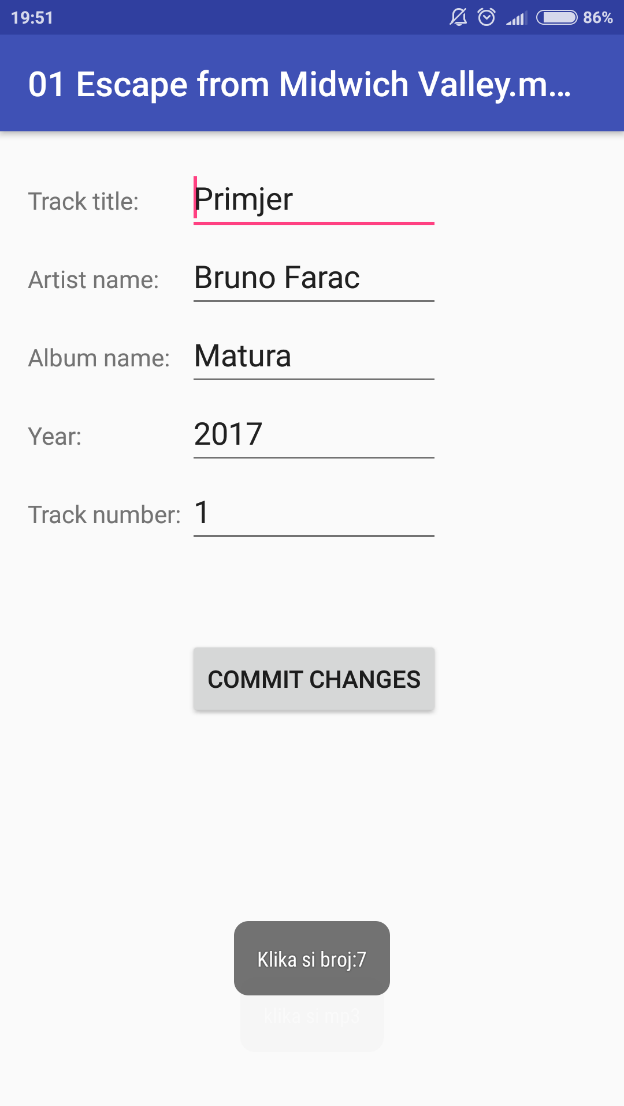
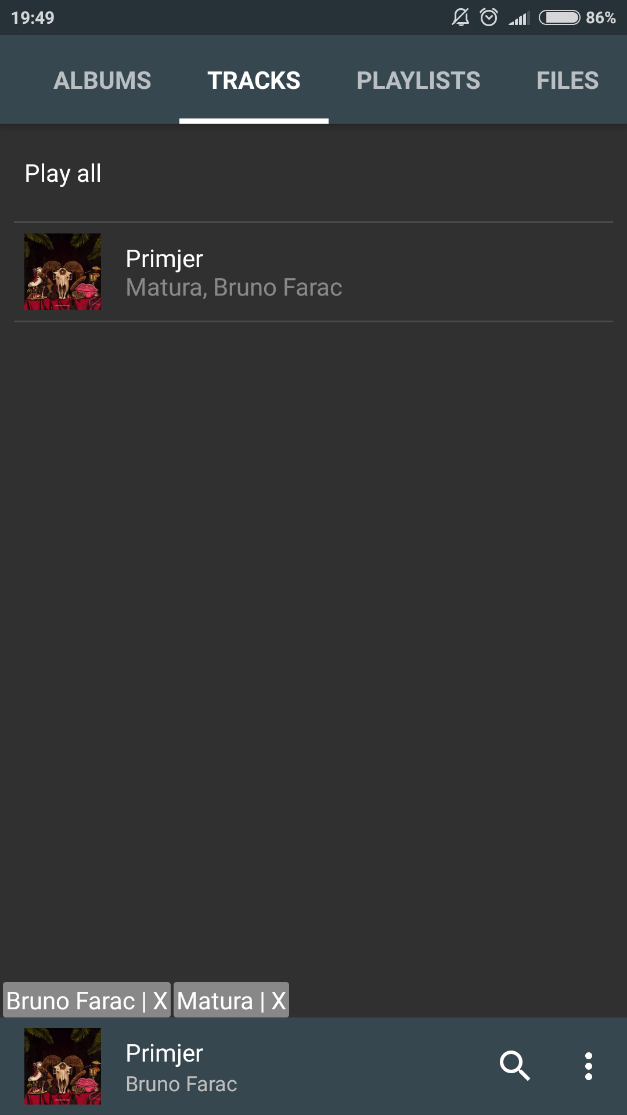
Potom stvaramo novi File od path koji smo prenjeli u TagEditor iz FileListing i njegovo ime postavljamo za naslov Activity instance. Ako *mp3* ima ID3v2 tag, zapisujemo ga u *tag,* ako ne stvaramo novi tag – tako aplikacija može obrađivati mp3 datoteke kojima nikad nisu ID3v2 polja popunjena. Postavljamo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



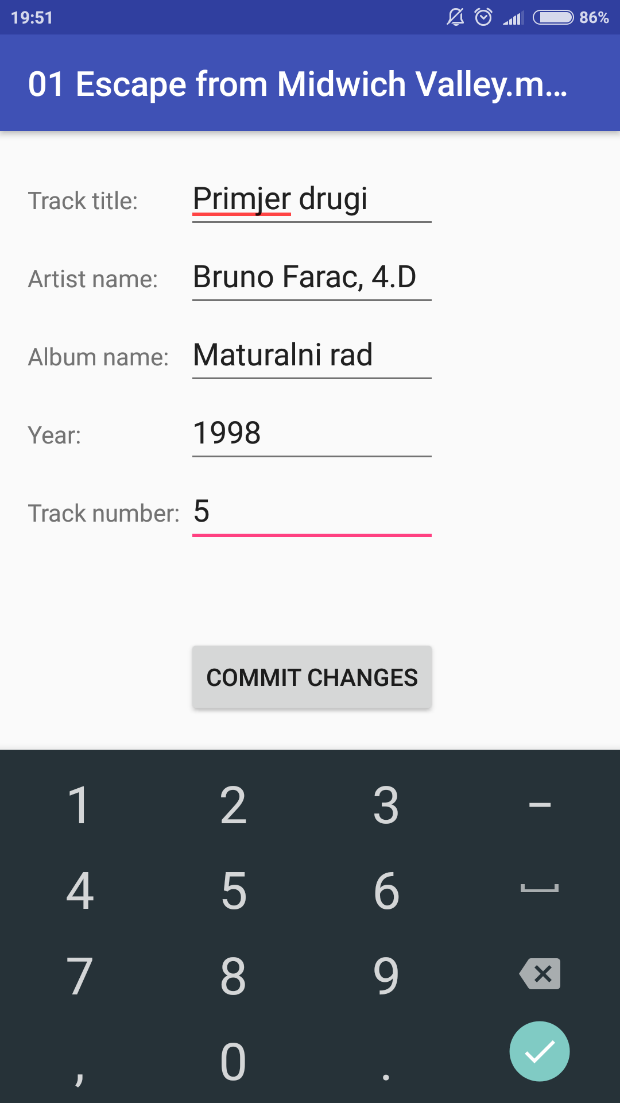
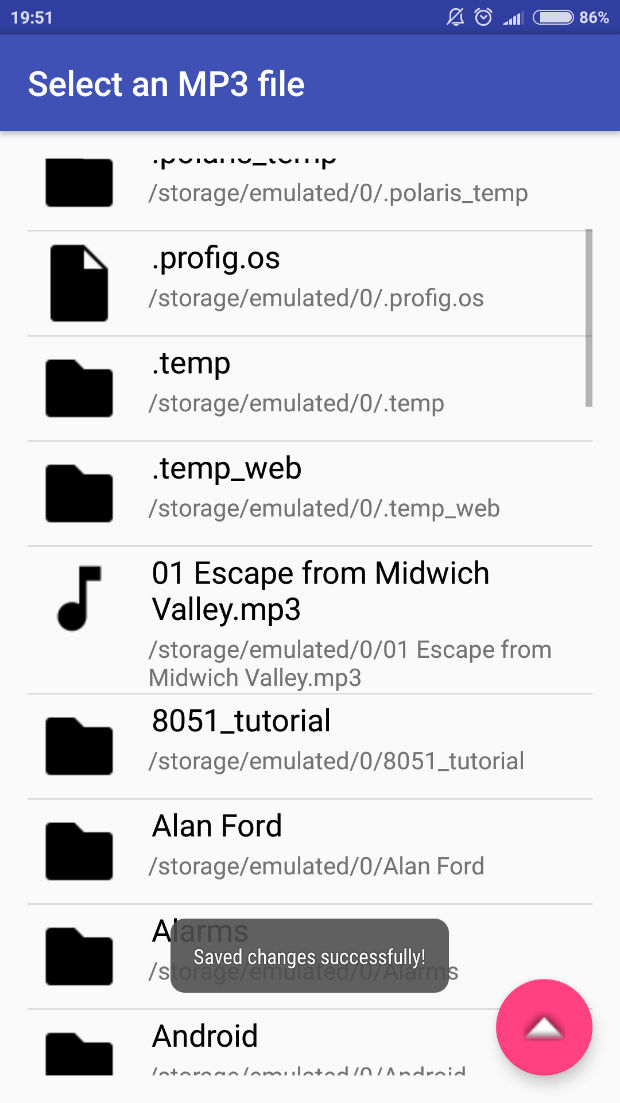
Kod 19 - TagEditor.java – OnClickListener

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.



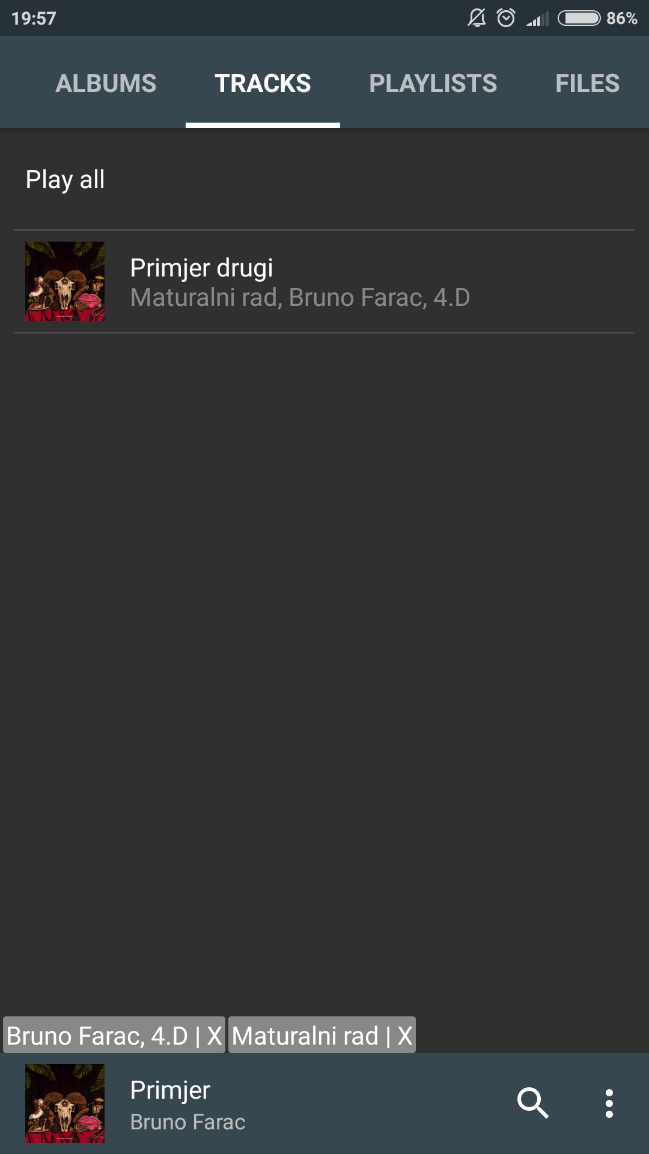
Slika 15 - .mp3 prije uređivanja, u player programu

Slika 16 - .mp3 prije uređivanja



Slika 17 - .mp3 uspješno spremljen

Slika 18 - .mp3 prije finalizacije uređivanja



Slika 19 - .mp3 nakon uređivanja u player programu