SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1950

DODATAK ZA ANDROID STUDIO ZA LAKŠE SNALAŽENJE U DEKOMPAJLIRANIM APK DATOTEKAMA

Krešimir Ledenko

Zagreb, lipanj 2019.

Prazni list za zadatak

Zahvala

Sadržaj

1. Uvod

Većina ljudi danas koristi pametni telefon i to na način da na njega instaliraju aplikaciju kojom će brže i lakše doći do neke informacije ili obaviti neku radnju. Najčešće korišteni operacijski sustav na pametnim telefonima je Android na kojeg se instaliraju Android aplikacije. Android je mobilni operacijski sustav otvorenog koda temeljen na prilagođenoj verziji jezgre Linux, a razvija ga i održava tvrtka Google. Legalno preuzimanje i instalacija Android aplikacija uglavnom se obavlja preko trgovine aplikacijama Google Play u obliku Android paketa (engl. *Android Package, APK*) [1].

Da bi se saznalo na koji način Android aplikacije rade potrebno ih je analizirati, a ta analiza obično se provodi radi provjere i poboljšanja njihove sigurnosti. Zbog načina distribucije Android aplikacija takva sigurnosna analiza je otežana jer stručnoj osobi nije dostupan izvorni kod, nego samo APK datoteka. Zato je APK datoteku potrebno prebaciti u oblik sličan izvornom, a taj proces naziva se dekompajliranje [2]. S obzirom da rezultat dekompajliranja nije idealan, snalaženje u takvom kodu može biti teško i naporno.

Stoga je cilj ovoga rada razviti dodatak za integrirano razvojno okruženje Android Studio, najčešće korišteno okruženje za razvoj Android aplikacija, koji bi stručnoj osobi olakšao snalaženje u dekompajliranim APK datotekama.

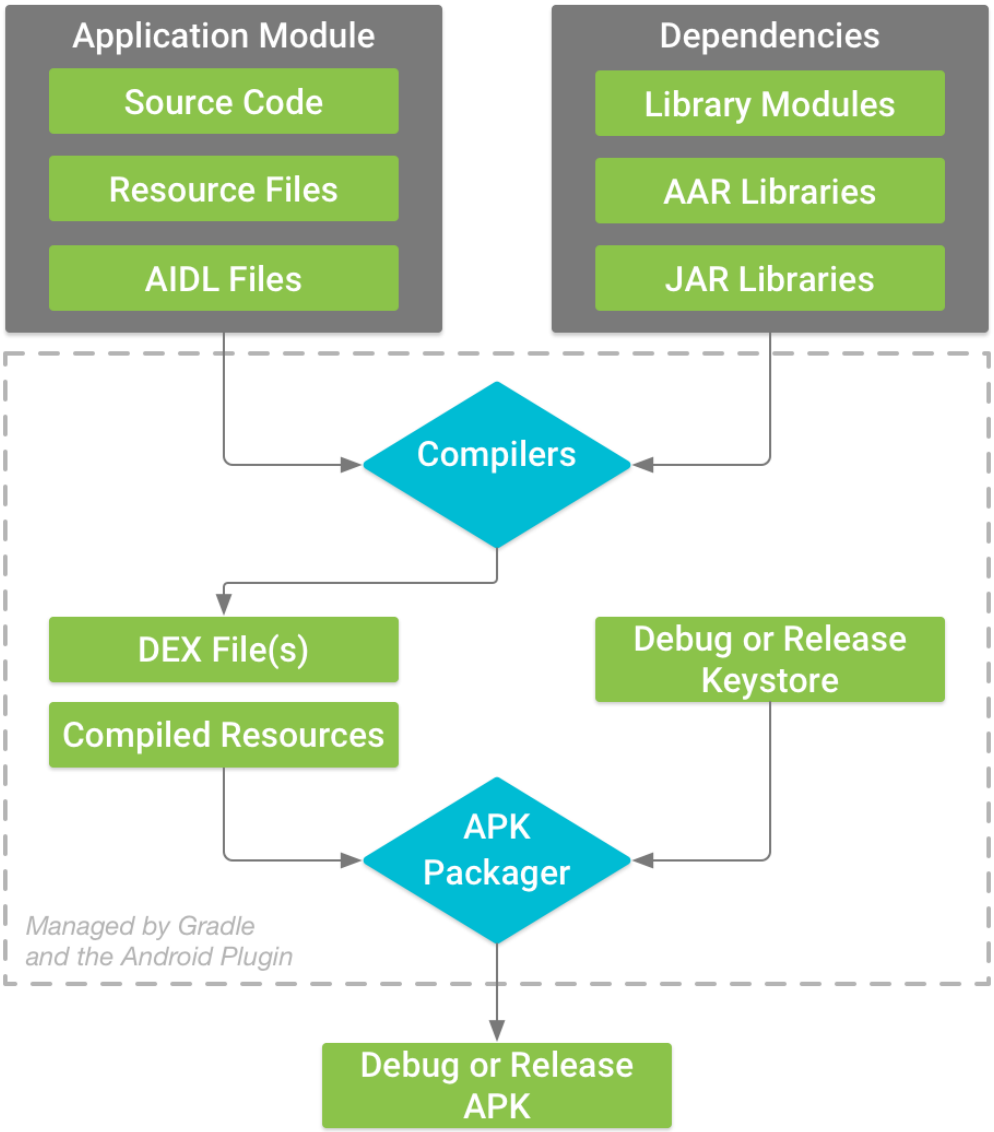
Rad je podijeljen na pet poglavlja. U drugom poglavlju objašnjene su Android aplikacije, njihova izgradnja, struktura i komponente, a na kraju je opisana razlika između izvornog i dekompajliranog koda. Treće poglavlje opisuje razvijeni dodatak za Android Studio, kako radi, koji je rezultat njegovog rada te je dan primjer korištenja dodatka nad testnom aplikacijom. Na kraju je dan zaključak o napravljenom i pregled korištene literature.

2. Općenito o Android aplikacijama

Ovo poglavlje objašnjava proces izgradnje i strukturu Android aplikacije te su u njemu opisane komponente Android aplikacije i način na koji se one aktiviraju. Također je opisano što je to manifest datoteka i kako se u njoj deklariraju spomenute komponente. Na kraju poglavlja objašnjena je razlika između izvornog i dekompajliranog koda.

2.1. Izgradnja i struktura Android aplikacije

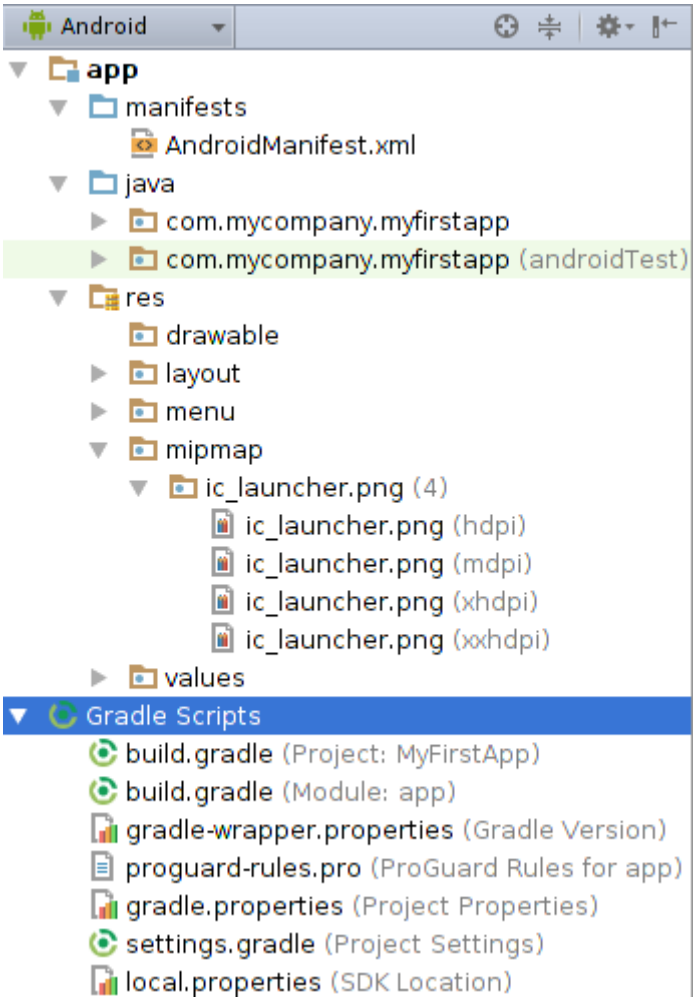
Aplikacije se danas najčešće razvijaju pomoću integriranih razvojnih okruženja (engl. *integrated development environment*), računalnih programa koji ljudima olakšavaju izgradnju softvera. Android Studio je jedno takvo okruženje za razvoj Android aplikacija. Proces izgradnje Android aplikacije, prikazan na slici 1, uključuje mnoge alate i postupke pomoću kojih se napisani kod zajedno s ostalim podacima i resursima pakira u APK datoteku, arhivnu datoteku s ekstenzijom *.apk*. Iz te arhivne datoteke mobilni uređaji s operacijskim sustavom Android instaliraju aplikaciju koja se može koristiti [3].



Slika 1. Proces izgradnje Android aplikacije [4]

Android aplikacijski modul, vidljiv u gornjem lijevom kutu slike 1, obuhvaća sav izvorni kod, resursne datoteke i postavke jedne aplikacije. Skup izvora (engl. *source set*), tj. prostor u kojem su izvorni kod i resursi logički grupirani [5], modula sastoji se od datoteke AndroidManifest.xml i dva direktorija: *java* i *res*. Datoteka AndroidManifest.xml neizostavni je dio svake Android aplikacije u kojoj su zapisane ključne informacije o aplikaciji, direktorij *java* sadrži sve datoteke java izvornog koda, dok direktorij *res* sadrži sve resurse koji nisu izvorni kod [6].

Struktura jednog Android aplikacijskog modula, kada se gleda pomoću Android prikaza Android Studio-a (koji ne odražava stvarnu strukturu direktorija na disku), prikazana je na slici 2.



Slika 2. Struktura Android aplikacijskog modula [7]

Resursi koji se nalaze u direktoriju *res* su dodatne datoteke i statički sadržaj koje koristi kod, poput bitmapa, XML rasporeda (engl. *XML layouts*), znakovnih nizova korisničkog sučelja (engl. *user interface strings*), animacijskih instrukcija i sl. [8].

Direktoriji koji se mogu nalaziti u direktoriju *res* su:

- animator/ - sadrži XML datoteke koje definiraju vlastite animacije
- anim/ - sadrži XML datoteke koje definiraju *tween* animacije
- color/ - sadrži XML datoteke koje definiraju listu boja za stanje
- drawable/ - sadrži bitmap datoteke ili XML datoteke koje su prevedene u neki *drawable* resurs
- mipmap/ - sadrži *drawable* datoteke za različite veličine pokretačkih ikona
- layout/ - sadrži XML datoteke koje definiraju izgled korisničkog sučelja

- menu/ - sadrži XML datoteke koje definiraju aplikacijske izbornike
- raw/ - sadrži datoteke koje trebaju biti pohranjene u izvornom obliku
- values/ - sadrži XML datoteke koje sadrže jednostavne vrijednosti, poput *stringova*, *integera* i boja
- xml/ - sadrži XML datoteke koje se mogu pročitati pri pokretanju
- font/ - sadrži *font* datoteke.

Resursi koji su pohranjeni u navedenim direktorijima su zadani (engl. *default*) resursi i oni definiraju zadani dizajn i sadržaj aplikacije. Budući da postoje Android uređaji koji zahtjevaju drugačiji tip resursa, npr. zato što imaju veći zaslon nego što je uobičajeno, moguće je uključiti alternativne resurse kako bi se podržale različite konfiguracije uređaja [9].

2.2. Komponente Android aplikacije

Android aplikacije sastoje se od četiri tipa komponenti:

- aktivnosti (engl. *activities*)
- servisi (engl. *services*)
- višeodredišni primatelji (engl. *broadcast receivers*)
- pružatelji sadržaja (engl. *content providers*)

Aplikacijske komponente su osnovni gradivni elementi Android aplikacije. Svaka komponenta je pristupna točka kroz koju sustav ili korisnik pristupa aplikaciji. Neke komponente ovise o drugima. Svaki tip služi za određenu svrhu i ima različit životni ciklus koji definira kako se komponenta stvara i uništava.

Aktivnosti su komponente za interakciju s korisnikom. Jedna aktivnost predstavlja jedan zaslon korisničkog sučelja. Na primjer, aplikacija za razmjenu elektroničke pošte može imati jednu aktivnost koja prikazuje listu novih poruka i drugu za čitanje poruka. Iako aktivnosti mogu surađivati i stvarati cjelovito korisničko iskustvo, svaka aktivnost je nezavisna od drugih. To omogućuje da jedna aplikacija pokrene aktivnost druge aplikacije, ako joj ona to dozvoljava.

Aktivnosti olakšavaju ključne interakcije između operacijskog sustava i aplikacije na nekoliko načina. One prate što korisnik trenutno radi kako bi osigurale da sustav održava pokrenutim proces u kojem se aktivnost izvršava. Budući da nedavno korišteni procesi sadrže stvari kojima će se korisnik možda vratiti, daju veći prioritet održavanju tih procesa. Pomažu aplikaciji prilikom uništenja njenog procesa kako bi se korisnik mogao vratiti aktivnostima u stanju u kojima su ostavljene. I konačno, omogućuju aplikacijama da međusobno ugrade tok korištenja, a sustavu da koordinira tim tokovima.

Prilikom implementacije aktivnosti potrebno je naslijediti klasu *Activity*.

Servisi ne pružaju korisničko sučelje i koriste se kako bi se aplikacija, zbog različitih razloga, nastavila izvršavati u pozadini. Njihova svrha je u pozadini izvršavati dugotrajne operacije ili odrađivati poslove za druge procese. Servis može pokrenuti neka druga komponenta i ostaviti ga da se izvršava ili se povezati s njim kako bi bili u interakciji.

Postoje dva načina izvršavanja servisa. Prvi način je da servis prilikom pokretanja kaže sustavu da ga ostavi u izvršavanju dok ne završi s poslom. Dvije su vrste tzv. pokrenutih (engl. *started*) servisa koje razlikujemo po načinu na koji ih sustav tretira.

Prednji (engl. *foreground*) servis izvršava nešto čega je korisnik svjestan i ne želi da se to prekine pa sustav održava proces tog servisa aktivnim. Prednji servisi moraju prikazivati obavijesti te će nastaviti s izvršavanjem čak i kada korisnik nije u interakciji s aplikacijom.

U slučaju pozadinskog (engl. *background*) servisa korisnik nije izravno svjestan da se servis izvršava pa sustav ima veću slobodu u upravljanju njegovim procesom te ga može u nekom trenutku prekinuti.

Drugi način izvršavanja servisa su vezani (engl. *bound*) servisi. Oni se izvršavaju jer je komponenta neke aplikacije izrazila želju za njihovim korištenjem. Vezani servis nudi klijent-poslužitelj sučelje koje omogućuje komponentama interakciju s njim, tj. slanje zahtjeva i primanje rezultata. On se izvršava samo onda kada je komponenta vezana na njega. Na servis se može vezati više komponenata odjednom pa će on biti uništen tek kada se sve komponente odvežu od njega.

Iako postoji podjela po načinu izvršavanja, servis istovremeno može biti i pokrenut (u izvršavanju dok ne završi s poslom) i omogućiti vezanje na sebe.

Servis se implementira nasljeđivanjem klase *Service*.

Višeodredišni primatelji su komponente koje omogućuju sustavu da dostavlja događaje (engl. *events*) aplikacijama izvan uobičajenog korisničkog korištenja, što dopušta aplikacijama da reagiraju na te događaje. Zbog načina na koji je definiran njihov pristup aplikaciji, sustav može dostaviti događaje čak i aplikacijama koje se trenutno ne izvršavaju. Aplikacije također mogu inicirati slanje nekog događaja kako bi obavijestile druge aplikacije o nečemu.

Iako višeodredišni primatelji ne prikazuju korisničko sučelje, mogu stvoriti obavijesti u statusnoj traci da bi upozorile korisnika o nekom događaju. Ipak, oni su najčešće poveznica do drugih komponenti i nisu namjenjeni za izvršavanje velike količine posla. Implementiraju se nasljeđivanjem klase *BroadcastReceiver*, pri čemu se svaki događaj dostavlja kao objekt tipa *Intent*.

Pružatelji sadržaja upravljaju zajedničkim skupom aplikacijskih podataka koji se mogu pohraniti u datotečnom sustavu, SQLite bazi podataka, na webu ili bilo kojoj drugoj trajnoj lokaciji za pohranu kojoj aplikacija može pristupiti. Preko pružatelja sadržaja druge aplikacije mogu pretraživati ili mjenjati podatke ako za to imaju dozvolu.

Pružatelj sadržaja sustavu služi za pristup aplikaciji kako bi objavio imenovane podatke, identificirane pomoću URI sheme. Prema tome, aplikacija može odlučiti kako će mapirati podatke koje sadrži u URI prostor imena te zatim podijeliti ta URI imena drugim entitetima koji ih mogu koristiti za pristup podacima. To omogućuje sustavu dvije konkretne stvari u upravljanju aplikacijom.

Prvo, mapirano URI ime ne zahtjeva da aplikacija ostane u izvršavanju pa URI ime postoji i nakon što njegova vlasnička aplikacija bude zatvorena. Jedino što sustav mora osigurati je da se vlasnička aplikacija izvršava kada on prikuplja aplikacijske podatke prema odgovarajućim URI imenima.

Drugo, URI imena predstavljaju važan sigurnosni model na način da aplikacija može omogućiti pristup nekom URI imenu, a istovremeno onemogućiti pristup njegovom pružatelju sadržaja. Tako neka aplikacija može pristupati podacima koji se nalaze iza određenog URI imena (što omogućuje sustav pomoću privremene URI dozvole), ali ne i ostalim podacima druge aplikacije.

Pružatelj sadržaja se također može koristiti za čitanje i pisanje podataka koji su dostupni samo aplikaciji i ne dijele se. Implementira se nasljeđivanjem klase *ContentProvider* i mora

implementirati standardni skup aplikacijskih sučelja koja omogućuju obavljanje transakcija drugim aplikacijama.

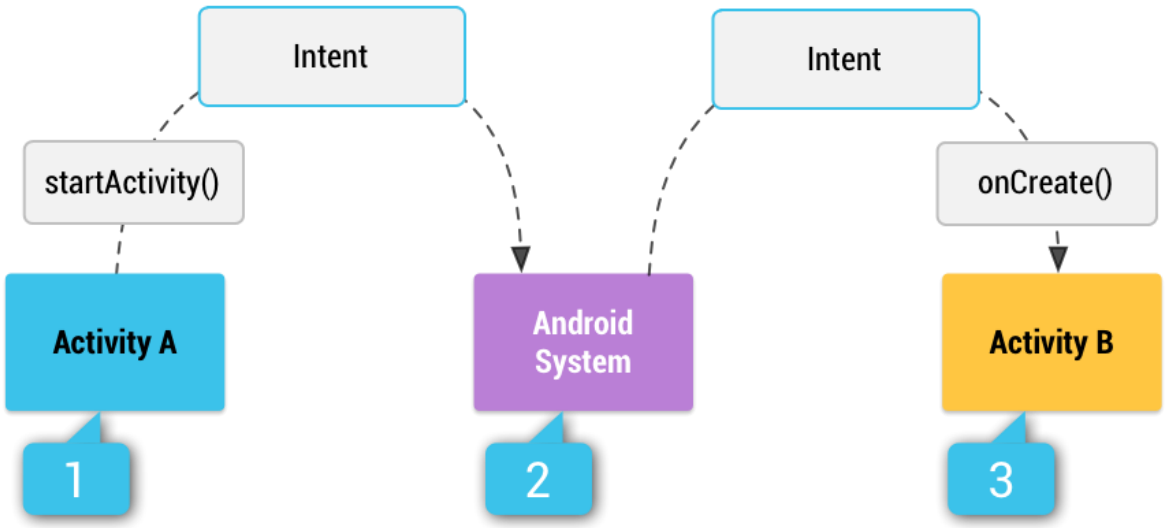
Jedinstvena osobina operacijskog sustava Android je da bilo koja aplikacija može pokrenuti komponentu druge aplikacije. Kada sustav pokrene komponentu, pokreće i proces za njegovu aplikaciju (ako ona već nije pokrenuta) te instancira klase potrebne za tu komponentu. Dakle, komponenta se ne izvršava u procesu aplikacije koja ju je pokrenula, nego u procesu aplikacije koja je vlasnik komponente. Zbog ovog pristupa, Android aplikacije, za razliku od aplikacija na drugim operacijskim sustavima, nemaju jednu pristupnu točku, tj. u njima ne postoji metoda *main* [10].

2.3. Aktivacija komponenti Android aplikacije

Budući da sustav svaku aplikaciju pokreće u odvojenom procesu s dozvolama koje ograničuju pristup drugim aplikacijima, jedna aplikacija ne može direktno aktivirati komponentu druge aplikacije. Za aktivaciju komponente druge aplikacije, aplikacija sustavu mora dostaviti poruku u kojoj se izražava namjera pokretanja određene komponente. Tada sustav aktivira komponentu aplikaciji koja ju je zatražila [10].

Tri od četiri tipa komponente (aktivnosti, servisi i višeodredišni primatelji) aktiviraju se asinkronom porukom zvanom namjera (engl. *intent*). Namjere povezuju pojedinačne komponente prilikom njihova pokretanja. Namjera se stvara pomoću objekta tipa *Intent* u kojem je definiran zahtjev za aktivacijom ili određene komponente (eksplicitna namjera) ili određenog tipa komponente (implicitna namjera) [11].

Na slici 3 prikazano je dostavljanje implicitne namjere kroz operacijski sustav Android kako bi se pokrenula još jedna aktivnost. Pod brojem 1 aktivnost A stvara objekt *Intent* u kojem je opisana akcija i prosljeđuje ga metodi *startActivity*. Na broju 2 operacijski sustav Android pretražuje sve aplikacije kako bi pronašao aktivnost koja zadovoljava akciju iz stvorenog objekta. Broj 3 označava da operacijski sustav pokreće odgovarajuću aktivnost (aktivnost B) pomoću metode *onCreate* kojoj prosljeđuje objekt *Intent* [12].



Slika 3. Primjer korištenja namjere za pokretanje dodatne aktivnosti [13]

Za aktivnosti i servise, namjera definira akciju koja se treba obaviti i može specificirati URI ime podatka nad kojim se akcija obavlja (kao i bilo što drugo što pokrenuta komponenta možda treba znati). Ukoliko se komponenta pokreće da bi se dobio neki rezultat, ona traženi rezultat vraća također u namjeri.

Za višeodredišne primatelje, namjera definira događaj koji se treba dostaviti.

Za razliku od aktivnosti, servisa i višeodredišnih primatelja, pružatelji sadržaja se ne aktiviraju namjerama, nego kada *ContentResolver* pokrene zahtjev nad njima. Primatelj sadržaja (engl. *content resolver*) razmjenjuje sve transakcije direktno s pružateljem sadržaja u ime komponente koja obavlja transakcije. Na taj se način, iz sigurnosnih razloga, stvara sloj apstrakcije između pružatelja sadržaja i komponente koja zahtjeva sadržaj.

Postoje odvojene metode za aktivaciju svakog tipa komponente:

- aktivnosti se pokreću proslijeđivanjem namjere metodama *startActivity* ili *startActivityForResult* (ako aktivnost treba vratiti rezultat),
- servisi se pokreću proslijeđivanjem namjere metodi *startService*, a vezanje na servis je moguće proslijeđivanjem namjere metodi *bindService*,
- slanje događaja aplikacijama pokreće se proslijeđivanjem namjere metodama *sendBroadcast*, *sendOrderedBroadcast* ili *sendStickyBroadcast*,
- zahtjev nad pružateljem sadržaja pokreće se pozivom metode *query* objekta *ContentResolver* [11].

2.4. Manifest datoteka i deklaracija komponenti

Da bi operacijski sustav Android mogao pokrenuti komponentu aplikacije, mora znati da ta komponenta postoji, a to sazna čitanjem aplikacijske manifest datoteke. Zbog toga svaka Android aplikacija mora imati datoteku AndroidManifest.xml, s točno tim imenom, koja se nalazi u korijenskom direktoriju skupa izvora. Manifest datoteka opisuje ključne informacije o aplikaciji Androidovim gradivnim alatima (engl. *Android build tools*), operacijskom sustavu Android i Google Play-u.

Manifest datoteka služi za mnogo stvari, a neki dijelovi u njoj moraju biti definirani.

Za početak, to je ime paketa aplikacije, koje je obično jednako nazivu prostoru imena koda. Njega Androidovi gradivni alati koriste za određivanje lokacije entiteta koda pri građenju projekta. Pri pakiranju aplikacije u APK datoteku ime paketa se zamjeni s aplikacijskim *ID*-em iz *Gradle build* datoteka te se koristi kao jedinstveni identifikator aplikacije na uređaju i na Google Play-u.

Zatim komponente aplikacije, tj. aktivnosti, servisi, višeodredišni primatelji i pružatelji sadržaja. Svakoj komponenti moraju biti definirana osnovna svojstva poput naziva njene Java klase. Također im se mogu definirati sposobnosti (engl. *capabilities*) poput konfiguracije uređaja koje podržavaju i filtara namjere (engl. intent filters) koji opisuju kako se komponenta može pokrenuti.

Nadalje, dozvole koje su aplikaciji potrebne kako bi mogla pristupiti zaštićenim dijelovima sustava ili drugim aplikacijama. Također definira dozvole koje druge aplikacije moraju imati ako žele pristupiti sadržaju prvotne aplikacije.

I na kraju, hardveska i softverska svojstva potrebna aplikaciji, što utječe na to koji uređaji mogu instalirati aplikaciju s Google Play-a [14].

Primjer jednostavne manifest datoteke nalazi se na ispisu 1.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:versionCode="1"
  android:versionName="1.0"
  package="com.example.myapp">

  <!-- Beware that these values are overridden by the build.gradle
       file -->
  <uses-sdk android:minSdkVersion="15"
android:targetSdkVersion="26" />

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">

    <!-- This name is resolved to com.example.myapp.MainActivity
         based upon the package attribute -->
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER"
         />
      </intent-filter>
    </activity>

    <activity
      android:name=".DisplayMessageActivity"
      android:parentActivityName=".MainActivity" />
  </application>
</manifest>
```

Ispis 1. Primjer datoteke AndroidManifest.xml [15]

Primarna uloga manifest datoteke je informirati operacijski sustav o komponentama aplikacije, a građena je od elemenata. Sve vrijednosti elementa postavljaju se pomoću atributa. Tehnički, svi atributi su proizvoljni, ali mnogi atributi moraju biti postavljeni kako bi element imao svrhu. Gotovo sva imena atributa počinju prefiksom „*android:*“.

Korijenski element manifesta je <*manifest*> unutar koje se nalazi element <*application*>. To su jedina dva elementa koji moraju biti definirani i svaki od njih se može pojaviti samo jednom. Komponente aplikacije su elementi unutar elementa <*application*> i ne moraju se deklarirati nekim određenim redoslijedom [16].

Za deklaraciju komponenata koriste se elementi: <*activity*> za aktivnosti, <*service*> za servise, <*receiver*> za višeodredišne primatelje i <*provider*> za pružatelje sadržaja.

Aktivnosti, servisi i pružatelji sadržaja koji se koriste u izvornom kodu, a nisu deklarirani u manifestu nisu vidljivi sustavu i zato ne mogu biti pokrenuti. Što se tiče višeodredišnih primatelja, oni mogu biti ili deklarirani u manifestu ili dinamički stvoreni u kodu kao objekti tipa *BroadcastReceiver* i registrirani u sustavu pozivom metode *registerReceiver* [17].

Postoje još dva važna elementa u manifestu: <*intent-filter*> i <*activity-alias*>.

<*intent-filter*> se deklarira unutar elementa komponente i određuje na koje tipove namjera aktivnosti, servisi i višeodredišni primatelji mogu odgovoriti. Ovaj element je važan jer se pomoću njega određuje glavna pristupna točka aplikacije (konkretno, koja aktivnost će se prva pokrenuti pri pokretanju aplikacije) i to na način da se unutar njega dodaju sljedeća dva elementa s odgovarajućim atributima:

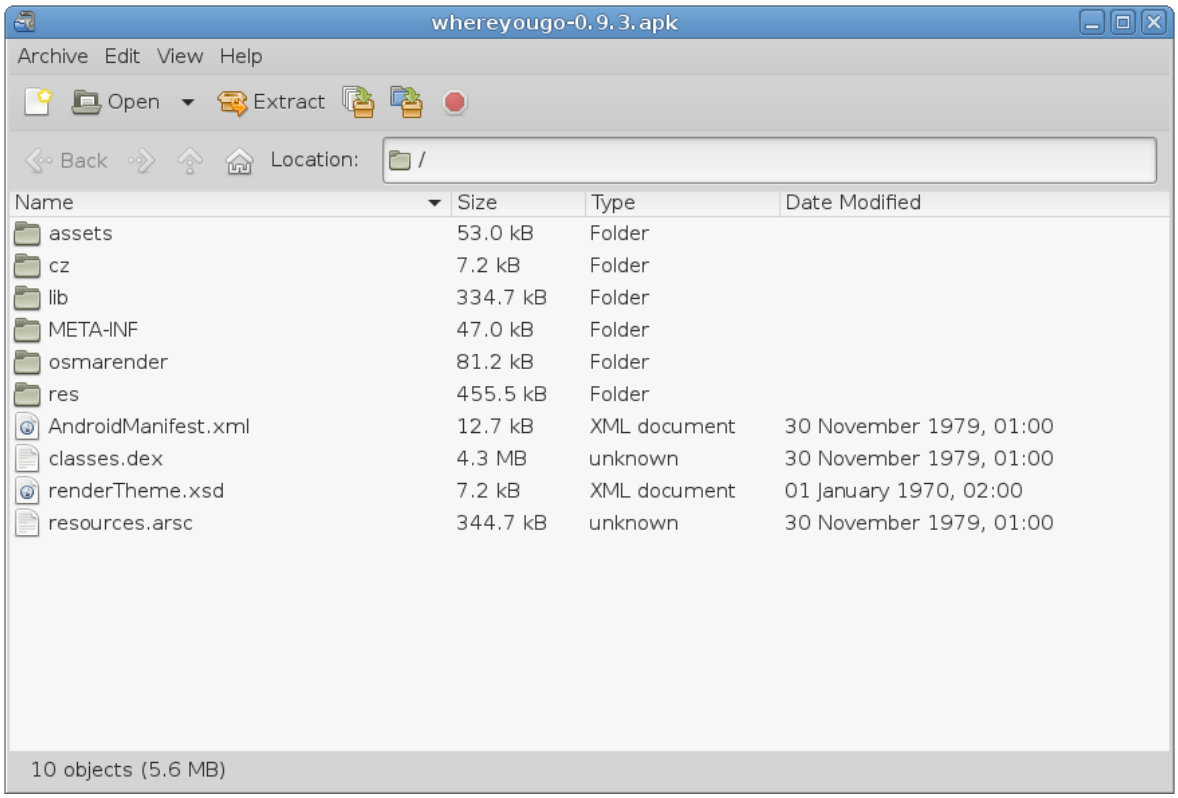
```
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
```

<*activity-alias*>, kao što mu ime kaže, deklarira alias za neku aktivnost, a služi kako bi se ciljana aktivnost prikazala kao nezavisni entitet. Razlog za korištenje ovog elementa je održavanje koda. Ukoliko se dogodi promjena glavne aktivnosti u aplikaciji, potrebno je promjeniti samo aktivnost koju alias predstavlja, a ne svako pojavljivanje glavne aktivnosti u izvornom kodu.

2.5. Razlika između izvornog i dekompajliranog koda

Kompajliranje je postupak prevođenja računalnog koda iz jednog programskog jezika (izvornog jezika) u drugi programski jezik (ciljani jezik). Termin kompajler se primarno koristi za program koji prevodi izvorni kod viših programskih jezika, razumljiv ljudima, u izvršni kod razumljiv računalima. Obrnuti proces, prevođenje izvršnog koda u izvorni kod viših programskih jezika, naziva se dekompajliranje [18].

U slučaju Android aplikacija, izvorni se kod, najčešće pisan u Javi, kompajlira u *bytecode* za javin virtualni stroj (engl. *Java virtual machine*) te se pohranjuje u datoteku s ekstenzijom *.dex* (*Dalvik EXecutable*). U tom se obliku kod aplikacije, zajedno s ostalim resursima i manifest datotekom, pakira u APK datoteku. Primjer jedne APK datoteke prikazan je na slici 4.



Slika 4. Struktura APK datoteke

Dekompajliranjem datoteke s *.dex* ekstenzijom dobit ćemo dekompajlirani kod koji bi trebao imati strukturu sličnu izvornom kodu (u smislu organizacije datoteka). Ukoliko se usporede izvorni i dekompajlirani kod može se vidjeti da oni nisu isti, tj. postoji razlika u načinu na koji su napisani, što je vidljivo na ispisu 2, na kojem je izvorni, te na ispisu 3, na kojem je dekompajlirani kod. Konkretno, nazivi varijabli nisu isti, *for* petlja u izvornom kodu zamijenjena je *while* petljom u dekompajliranom i jedan uvjet iz izvornog pretvoren je u dva uvjeta i labelu u dekompajliranom kodu.

```
private int getAdminCount() {
  if (info == null) {
    return 1;
  }
  int count = 0;
  for (int a = 0, N = info.participants.participants.size();
       a < N; a++) {
  TLRPC.ChatParticipant chatParticipant =
                              info.participants.participants.get(a);
  if(chatParticipant instanceof TLRPC.TL_chatParticipantAdmin ||
     chatParticipant instanceof TLRPC.TL_chatParticipantCreator) {
    count++;
    }
  }
  return count;
}
```

Ispis 2. Metoda izvornog koda

```
private int getAdminCount() {
  final TLRPC.ChatFull info = this.info;
  if (info == null) {
    return 1;
  }
  final int size = info.participants.participants.size();
  int i = 0;
  int n = 0;
  while (i < size) {
    final TLRPC.ChatParticipant chatParticipant =
                        this.info.participants.participants.get(i);
    int n2 = 0;
    Label_0074: {
      if (!(chatParticipant instanceof
                              TLRPC.TL_chatParticipantAdmin)) {
        n2 = n;
        if (!(chatParticipant instanceof
                                TLRPC.TL_chatParticipantCreator)) {
          break Label_0074;
        }
      }
      n2 = n + 1;
    }
    ++i;
    n = n2;
  }
  return n;
}
```

Ispis 3. Metoda dekompajliranog koda

Unatoč tome što su različito napisani, oni rade potpuno istu stvar. Kada bi se kodovi prikazani na ispisima 2 i 3 kompajlirali, nastao bi identičan izvršni kod, što znači da imaju istu funkcionalnost. Poanta je da će se jedan kod pisan u nekom od viših programski jezika uvijek prevesti u identičan izvršni kod, dok se jedan izvršni kod može prevesti u više različitih kodova pisanih u nekom od viših programskih jezika.

Razlog zašto izvorni i dekompajlirani kod nisu isto napisani je taj što dekompajliranje nije postupak obnavljanja izvornog koda, nego stvaranje novog koda. Budući da se prilikom kompajliranja često obavlja optimizacija koda te se u izvršnom kodu nazivi nekih varijabli ne pohranjuju, logično je da dekompajler nema informacija za obnovu koda nego mora krenuti ispočetka. Zato je najuočljivija razlika između izvornog i dekompajliranog koda u nazivima varijabli. Ako se u stvaranje izvornog koda doda i postupak obfuskacije, tj. namjerne izmjene kako bi ga bilo teže čitati i razumjeti, moguće je da dekompajler ne uspije prevesti izvršni kod pa će ga ispisati unutar oznaka komentara. Primjer neuspješnog dekompajliranja je prikazan na ispisu 4.

Zaključak iz navedenog je da kvaliteta dekompajliranog koda ovisi o kvaliteti dekompajlera, dok je kvaliteta izvornog koda konstantna. Zato je iz ljudske perspektive bolje imati izvorni kod jer ga je lakše razumijeti, pretraživati i snalaziti se u njemu.

```
/* JADX WARNING: Removed duplicated region for block: B:14:0x003b  */
/* JADX WARNING: Removed duplicated region for block: B:13:0x002c  */
/* JADX WARNING: Missing block: B:9:0x0025, code skipped:
  if (r0.f569id ==
      org.telegram.p004ui.WallpaperActivity.access$500(r8.this$0).f569id)
    goto L_0x0027;  */
public void updateSelected(boolean r9) {
  /*
  r8 = this;
  r0 = r8.currentPattern;
  r1 = 0;
  if (r0 != 0) goto L_0x000d;
L_0x0005:
  r0 = org.telegram.p004ui.WallpaperActivity.this;
  r0 = r0.selectedPattern;
  if (r0 == 0) goto L_0x0027;
L_0x000d:
  r0 = org.telegram.p004ui.WallpaperActivity.this;
  r0 = r0.selectedPattern;
  if (r0 == 0) goto L_0x0029;
L_0x0015:
  r0 = r8.currentPattern;
  if (r0 == 0) goto L_0x0029;
L_0x0019:
  r2 = r0.f569id;
  r0 = org.telegram.p004ui.WallpaperActivity.this;
  r0 = r0.selectedPattern;
  r4 = r0.f569id;
  r0 = (r2 > r4 ? 1 : (r2 == r4 ? 0 : -1));
  if (r0 != 0) goto L_0x0029;
L_0x0027:
  r0 = 1;
  goto L_0x002a;
L_0x0029:
  r0 = 0;
L_0x002a:
  if (r0 == 0) goto L_0x003b;
L_0x002c:
  r2 = org.telegram.p004ui.WallpaperActivity.this;
  r3 = r8.radialProgress;
  r4 = r2.selectedPattern;
  r6 = 0;
  r5 = r8;
  r7 = r9;
  r2.updateButtonState(r3, r4, r5, r6, r7);
  goto L_0x0041;
L_0x003b:
  r0 = r8.radialProgress;
  r2 = 4;
  r0.setIcon(r2, r1, r9);
L_0x0041:
  r8.invalidate();
  return;
  */
  throw new UnsupportedOperationException("Method not decompiled:
org.telegram.p004ui.WallpaperActivity$PatternCell.updateSelected(boolean):v
oid");
}
```

Ispis 4. Neuspješno dekompajlirani kod

3. Razvijeni dodatak za Android Studio

U ovom poglavlju opisan je razvijeni dodatak za Android Studio koji obrađuje dekompajliranu APK datoteku i prikazuje tijek izvršavanja aplikacije. Na početku je dan tijek rada dodatka te su objašnjeni poslovi pripreme. Zatim je objašnjen algoritam koji obavlja temeljnu funkcionalnost dodatka nakon čega slijedi opis dobivenih rezultata i primjer rada dodatka na testnoj aplikaciji.

3.1. Tijek rada razvijenog dodatka i poslovi pripreme

Tijek rada razvijenog dodatka može se podijeliti u pet faza:

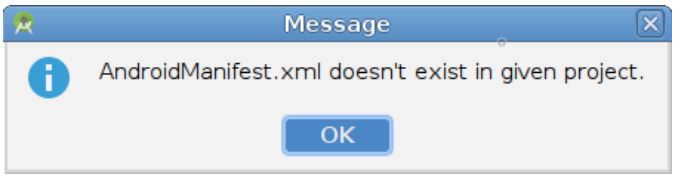
1. Odabir projekta nad kojim će se dodatak izvršiti.
2. Pronalazak datoteke AndroidManifest.xml, izvlačenje aktivnosti koje deklarira i otkrivanje glavne aktivnosti.
3. Pretraga datoteka s kodom kako bi se pronašlo pokretanje aktivnosti.
4. Provjera pronađenih aktivnosti i ispravljanje njihovih naziva.
5. Stvaranje grafova.

Kada se dodatak pokrene otvara se prozor za odabir željenog projekta. Nakon odabira, prvo što se pokreće je metoda *checkForAndroidManifest*, prikazana na ispisu 5.

```
public String checkForAndroidManifest(String currentDirectory) {
  File checkFile = new File(currentDirectory, "AndroidManifest.xml");
  String returnValue = "";
  if (checkFile.exists())
    returnValue = checkFile.toString();
  else {
    File files[] = new File(currentDirectory).listFiles();
    if (files != null)
      for (int i = 0; i < files.length; i++)
        if (files[i].isDirectory())
          if((returnValue =
checkForAndroidManifest(files[i].getPath())).endsWith("/src/main/Andr
oidManifest.xml"))
            break;
  }
  return returnValue;
}
```

Ispis 5. Kod metode *checkForAndroidManifest*

To je rekurzivna metoda koja pretražuje dani projekt dok ne nađe datoteku *AndroidManifest.xml*. Uvjet zaustavljanja rada metode je da putanja (engl. *path*) pronađene datoteke završava s „*/src/main/AndroidManifest.xml*“. Ukoliko takva datoteka nije pronađena, dodatak će završiti s radom i prikazati poruku kao na slici 5.



Slika 5. Obavijest dodatka da nije pronađena datoteka *AndroidManifest.xml*

Ako je datoteka pronađena, dohvaćaju se svi elementi aktivnosti pomoću paketa *javax.xml.parsers*, koji omogućuje obradu XML dokumenata [19], te se spremaju u listu.

Zatim se u listi dohvaćenih aktivnosti pomoću metode *findMainActivity*, koja je prikazana na ispisu 6, traži glavna aktivnost, tj. ona koja će se prva izvršiti pri pokretanju aplikacije.

```
public int findMainActivity(NodeList nodeList, List<ActivityInfo>
activityList) {
  for (int i = 0; i < nodeList.getLength(); i++) {
    Element element = (Element) nodeList.item(i);
    NodeList intentFilter = element.getElementsByTagName("intent-
filter");
    if(intentFilter.getLength() > 0)
      for (int j = 0; j < intentFilter.getLength(); j++) {
        NodeList contents = intentFilter.item(j).getChildNodes();
        for (int k = 0; k < contents.getLength(); k++) {
          org.w3c.dom.Node node = contents.item(k);
          if(node.hasAttributes()) {
            Element elementChild = (Element) node;
            if(elementChild.getAttribute("android:name")
                      .equals("android.intent.action.MAIN") ||
               elementChild.getAttribute("android:name")
                      .equals("android.intent.category.LAUNCHER"))
              for (int l = k+2; l < contents.getLength(); l += 2) {
                elementChild = (Element) contents.item(l);
                if(elementChild.getAttribute("android:name")
                      .equals("android.intent.action.MAIN") ||
                   elementChild.getAttribute("android:name")
                      .equals("android.intent.category.LAUNCHER")) {
                  ActivityInfo activityInfo = new ActivityInfo();
                  activityInfo.name =
                                element.getAttribute("android:name");
                  activityList.add(activityInfo);
                  return i;
                }
              }
          }
        }
      }
  }
  return -1;
}
```

Ispis 6. Kod metode *findMainActivity*

Metoda traži glavnu aktivnost na način da provjerava sadrži li njen element sljedeće elemente s odgovarajućim atributima:

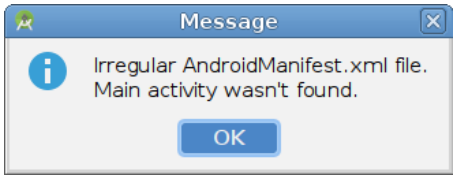
<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

U slučaju da pronađe glavnu aktivnost, vratit će njen položaj u listi, a u slučaju da ne, vratit će broj -1.

Broj -1 je znak da se treba pokrenuti metoda *findActivityAlias* koja funkcionira jednako kao i metoda *findMainActivity* uz razliku da pretražuje dohvaćene elemente <*activity-alias*>.

Ako ni ova metoda nije uspjela pronaći glavnu aktivnost, velika je vjerojatnost da manifest nije ispravan te će dodatak završiti s radom i obavijestiti korisnika o tome porukom kao na slici 6.



Slika 6. Obavijest dodatka da nije pronađena glavna aktivnost

Ako je pretraga uspješna, označava se aktivnost koju alias aktivnosti predstavlja te dodatak nastavlja rad provjerom postoje li datoteke s izvornim kodom koje imaju naziv aktivnosti.

3.2. Algoritam pronalaska aktivnosti i njihova provjera

Središnji dio rada dodatka je algoritam pronalaska aktivnosti u izvornom kodu.

Za svaku aktivnost pronađenu u manifestu, pokreće se metoda *findCodePatterns* nad njenom datotekom s izvornim kodom. Metoda *findCodePatterns* je rekurzivna metoda koja kao rezultat vraća objekt tipa *LinkedHashSet*, listu koja čuva poredak umetanja i kojoj su elementi jedinstveni [20]. Taj objekt se koristi jer se želi izbjeći spremanje pronađenih aktivnosti koje se potencijalno ponavljaju.

Rad algoritma temelji se na traženju regularnih izraza. Taj pristup otkriva konkretne nizove oslanjajući se na nužno poštivanje pravila zadanih programskim jezikom, brži je od parsiranja koda te čak i u slučaju neuspješnog dekompajliranja može pronaći tražene nizove ispisane unutar oznaka komentara.

Trenutna verzija dodatka traži šest različitih regularnih izraza.

Prvo što metoda traži u datoteci koda je pojavljivanje niza „*new ...NazivAktivnosti$... (this*“. Taj dio metode prikazan je na ispisu 7. Pojavljivanje ovog niza znači da je u izvornom kodu postojao neki lambda izraz, anonimna klasa ili nešto slično, što je dekompajler prilikom dekompajliranja izvršnog koda pretvorio u novu, zasebnu datoteku. Nad tom datotekom će se rekurzivno pozvati metoda *findCodePatterns* kako bi se otkrilo je li taj dio koda baratao aktivnostima.

```
if (unchangedLine.contains(activityName + "$"))
  if (unchangedLine.contains("new ") &&
      unchangedLine.contains("(this")) {
    line = unchangedLine.split("new ")[1].split("\\(this")[0];
    File newFile = new File(filePath.replace(activityName, line));
    if(newFile.exists())
      activityInfoActions.addAll(findCodePatterns(newFile,
                                                  mainActivity));
}
```

Ispis 7. Isječak koda metode *findCodePatterns* koji otkriva korištenje koda iz druge datoteke

Ispis 8 prikazuje dio metode koji traži drugi regularni izraz, a to je „new Intent“. Njime se u kodu stvara novi objekt tipa *Intent* kojim će se kasnije aktivirati nova aktivnost. Ako je pronađen niz „new Intent“, u nastavku linije traži se točan naziv aktivnosti koju taj objekt aktivira te se ona sprema u listu pronađenih aktivnosti.

```
if (unchangedLine.contains("new Intent(")) {
  found = false;
  line = unchangedLine.trim().split("new Intent\\(")[1].trim();
  if (line.startsWith(")"))
    ;
  else if (line.startsWith("Intent.ACTION_MAIN)") ||
           line.startsWith("\"android.intent.action.MAIN\")"))
    activityInfoActions.add(mainActivity);
  else if (line.contains(","))
    try {
      line = line.split(".class")[0].trim();
      String[] helper = line.split(",");
      line = helper[helper.length - 1].trim();
      if(line.contains(".")) {
        helper = line.split("\\.");
        line = helper[helper.length - 1];
      }
      activityInfoActions.add(line);
    } catch (ArrayIndexOutOfBoundsException e) {
      while (true) {
        line = bufferedReader.readLine();
        if(line.contains(".class")) {
          activityInfoActions.add(line.trim()
                                  .split(".class")[0].trim());
          break;
        }
      }
    }
}
```

Ispis 8. Isječak koda koji otkriva stvaranje novog objekta tipa Intent

Treći dio metode, prikazan na ispisu 9, traži regularni izraz „setClassName“. *setClassName* je metoda kojom se objektu *Intent* postavlja naziv komponente koju će aktivirati [21]. Kao i

u prethodnom slučaju, ako je pronađen niz „setClassName“, u nastavku linije traži se točan naziv komponente koju metoda postavlja te se ona sprema u listu pronađenih aktivnosti.

```
if (unchangedLine.contains(".setClassName(")) {
  found = false;
  line = unchangedLine.trim().split(".setClassName\\(")[1];
  try {
    if (line.contains(".class"))
      line = line.split(",")[1].trim().split("\\)")[0]
                                .trim().split(".class")[0];
    else {
      String[] helper = line.split(",")[1].trim().split("\\)")[0]
                        .trim().replaceAll("\"", "").split("\\.");
      line = helper[helper.length - 1];
    }
    activityInfoActions.add(line);
  } catch (ArrayIndexOutOfBoundsException e) {
    line = bufferedReader.readLine().trim();
    if (line.contains(".class"))
      line = line.split("\\)")[0].trim().split(".class")[0];
    else {
      String[] helper = line.split("\\)")[0].trim()
                             .replaceAll("\"", "").split("\\.");
      line = helper[helper.length - 1];
    }
    activityInfoActions.add(line);
  }
}
```

Ispis 9. Isječak koda koji otkriva poziv metode *setClassName*

Četvrti dio metode traži regularni izraz „setClass“. *setClass* je metoda kojom se objektu *Intent* postavlja klasa komponente koju će aktivirati [22]. Ovaj dio metode preslika je trećeg dijela, uz iznimku traženog regularnog izraza, i ima istu funkcionalnost.

Nadalje, peti dio metode koji je prikazan na ispisu 10 u kodu će pronaći korištenje ternarnog (uvjetnog) operatora. Ternarni operator je dio sintakse programskog jezika kojim se pojednostavljuju uvjetni izrazi (engl. *conditional expressions*) [23]. Nakon što je otkriven ternarni operator, iz linije se izvlače aktivnosti i spremaju u listu pronađenih aktivnosti.

```
if (unchangedLine.contains(":") && unchangedLine.contains("?") &&
    unchangedLine.contains(".class"))
  try {
    String[] helper = unchangedLine.trim().split("\\?");
    helper = helper[1].split(" ");
    for (int i = 0; i < helper.length; i++)
      if (helper[i].contains(".class"))
        activityInfoActions.add(helper[i]
                          .split(".class")[0].trim());
  } catch (ArrayIndexOutOfBoundsException e) {
    //activity doesn't exist in this line
}
```

Ispis 10. Isječak koda koji otkriva korištenje ternarnog operatora

Ispis 11 prikazuje zadnji, šesti dio metode koji traži dodijeljivanje vrijednosti varijabli, ukoliko je ta vrijednost naziv neke komponente. Ovo je jedini dio metode čije izvršavanje ovisi o rezultatu prethodnih dijelova, tj. izvršit će se samo ako nije pronađen niti jedan od već navedenih regularnih izraza. Razlog za to je općenitost ovog izraza zbog čega će obuhvatiti i neke izraze koji mu prethode, ali ih neće moći obraditi i dobiti potrebni naziv aktivnosti.

```
if (found && unchangedLine.contains("=") &&
    unchangedLine.contains(".class"))
  try {
    String[] helper = unchangedLine.trim().split(" ");
    for (int i = 0; i < helper.length; i++)
      if (helper[i].contains(".class"))
        activityInfoActions.add(helper[i]
                          .split(".class")[0].trim());
  } catch (Exception e) {
    //activity doesn't exist in this line
}
```

Ispis 11. Isječak koda koji traži dodijeljivanje vrijednosti varijabli

Konačno, kada je algoritam izveden nad svim zadanim datotekama koda slijedi provjera i prilagodba pronađenih aktivnosti. Ovaj korak je nužna nadopuna predstavljenog algoritma jer algoritam zapravo traži nazive komponenti, ali ne može provjeriti tip komponente koju je pronašao. Tako se može dogoditi da algoritam pronađe neki servis i doda ga u listu pronađenih aktivnosti. Budući da bi to pokvarilo rezultat rada dodatka, sve pronađene aktivnosti uspoređuju se s listom aktivnosti dohvaćenih iz manifesta te se neodgovarajuće izbacuju. Prilikom provjere se također prilagođavaju nazivi pronađenih aktivnosti kako bi odgovarali nazivima koji se nalaze u manifestu.

3.3. Prikaz rezultata rada dodatka

Zadnje što će dodatak napraviti je ispis dobivenog rezultata u dvije datoteke: navigation_graph.xml koja se stvara pomoću metode *createNavigationGraph* i graphviz_graph.dot koja se stvara pomoću metode *createGraphvizGraph*. U obje datoteke zapisat će se aktivnosti aplikacije i prijelazi između njih na odgovarajući način.

Datoteka navigation_graph.xml je resursna datoteka koja sadrži sva odredišta i akcije neke aplikacije. Odredišta (engl. *destinations*) predstavljaju aktivnosti aplikacije, dok su akcije (engl. *actions*) logičke veze između odredišta i prikazuju put unutar aplikacije kojim se korisnik može kretati [24].

Početak i kraj datoteke prikazani su na ispisu 12.

Na početku datoteke je XML deklaracija nakon koje slijedi definicija korijenskog elementa <*navigation*>. Korijenski element definira se pomoću pet atributa. Prva tri atributa su uvijek ista i deklariraju prostor imena (engl. *namespace*) elementa <*navigation*>. Četvrtim atributom elementu se definira jedinstveni identifikator, a petim glavna aktivnost aplikacije. Datoteka završava zatvarajućom oznakom korijenskog elementa.

```
<?xml version="1.0" encoding="utf-8"?>
<navigation
      xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:tools="http://schemas.android.com/tools"
      android:id="@+id/graph"
      app:startDestination="@id/menion.android.whereyougo.gui.activit
      y.MainActivity">

(...)

</navigation>
```

Ispis 12. Izgled početka i kraja datoteke navigation_graph.xml

Unutar korijenskog elementa nalaze se elementi <*fragment*> kojima se stvaraju nova odredišta, tj. aktivnosti. Jedan element <*fragment*> predstavlja jednu aktivnost. Na ispisu 13 prikazan je primjer jednog elementa <*fragment*>.

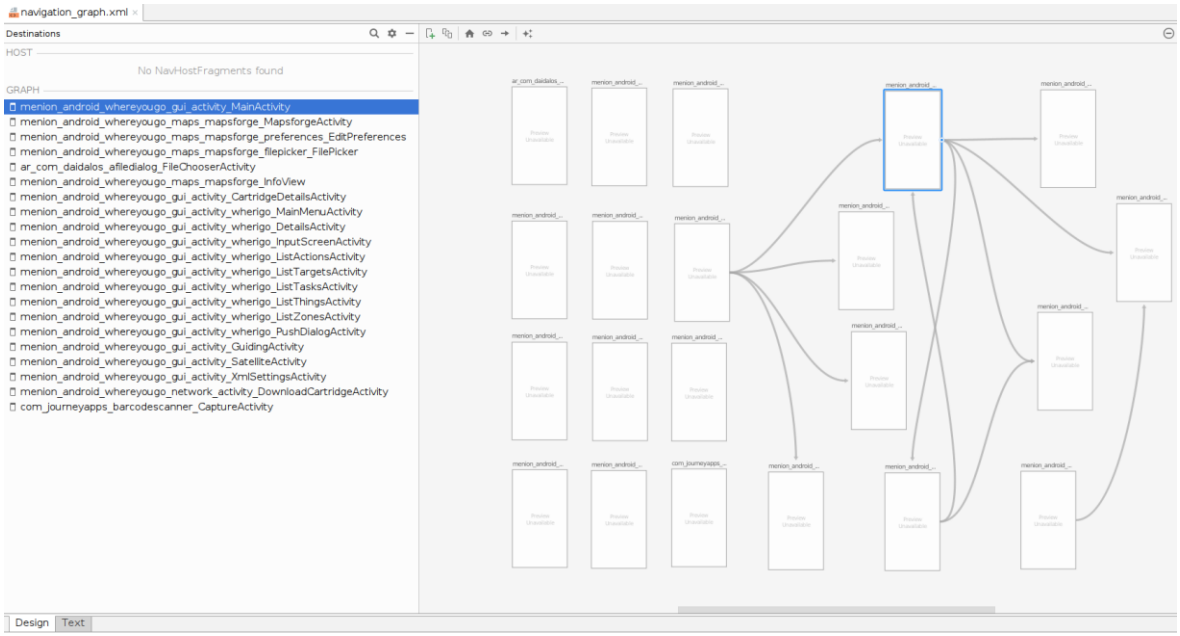
```
<fragment
      android:id="@+id/menion_android_whereyougo_network_activity_D
      ownloadCartridgeActivity"

      android:name="menion.android.whereyougo.network.activity.Down
      loadCartridgeActivity"

      android:label="menion_android_whereyougo_network_activity_Dow
      nloadCartridgeActivity"

      tools:layout="@layout/menion_android_whereyougo_network_activ
      ity_DownloadCartridgeActivity" >

      <action
            android:id="@+id/action_menion_android_whereyougo_netwo
            rk_activity_DownloadCartridgeActivity_to_menion_android
            _whereyougo_gui_activity_MainActivity"

            app:destination="@id/menion_android_whereyougo_gui_acti
            vity_MainActivity" />
      <action
            android:id="@+id/action_menion_android_whereyougo_netwo
            rk_activity_DownloadCartridgeActivity_to_menion_android
            _whereyougo_gui_activity_XmlSettingsActivity"

            app:destination="@id/menion_android_whereyougo_gui_acti
            vity_XmlSettingsActivity" />
</fragment>
```

Ispis 13. Izgled elementa <*fragment*> unutar kojeg se nalaze dva elementa <*action*>

Svaki element <*fragment*> u datoteci ima četiri atributa. Prvi atribut je *id* i on je obavezan, proizvoljne vrijednosti i jedinstven za svako odredište. Atribut *name* predstavlja ime klase s kojom je element povezan. Atribut *layout* određuje kako će odredište biti prikazano u grafičkom prikazu, a atribut *label* naziv koji će biti prikazan iznad odredišta [25].

Unutar elementa <*fragment*> mogu se nalaziti elementi <*action*>. Pomoću njih se definiraju akcije, tj. prijelazi iz jednog odredišta u drugo. Element <*action*> sadrži atribut *id* koji ga jednoznačno određuje i atribut *destination* u koji se zapisuje *id* krajnjeg odredišta. Ako element <*fragment*> nema niti jedan element <*action*> onda se iz njega ne može prijeći u drugo odredište.

Kada se datoteka navigation_graph.xml otvori u Uređivaču navigacije (engl. *Navigation Editor*) Android Studio-a dobije se prikaz kao na slici 7. Na lijevoj strani slike 7 vidi se lista svih odredišta u aplikaciji, a na desnoj strani grafički prikaz odredišta i veza između njih. Lista svih odredišta počinje s glavnom aplikacijom.



Slika 7. Prikaz datoteke navigation_graph.xml u Uređivaču navigacije

Druga datoteka naziva graphviz_graph.dot napisana je u jeziku za opis grafova (engl. *graph description language*) DOT [26]. Prednosti DOT datoteke su jednostavnost definiranja grafa, mala veličina i mogućnost lakog pretvaranja u sliku.

Izgled stvorene DOT datoteke prikazan je na ispisu 14.

U prvom redu ispisa 14 riječju „digraph“ definiran je usmjereni graf naziva application_graph. Nakon toga slijede vitičaste zagrade unutar kojih se definiraju svojstva i čvorovi grafa. Ovaj graf ima jedno svojstvo, a to je „rankdir“ koje označava smjer širenja grafa. Vrijednost „LR“ znači da će se graf širiti s lijeva na desno.

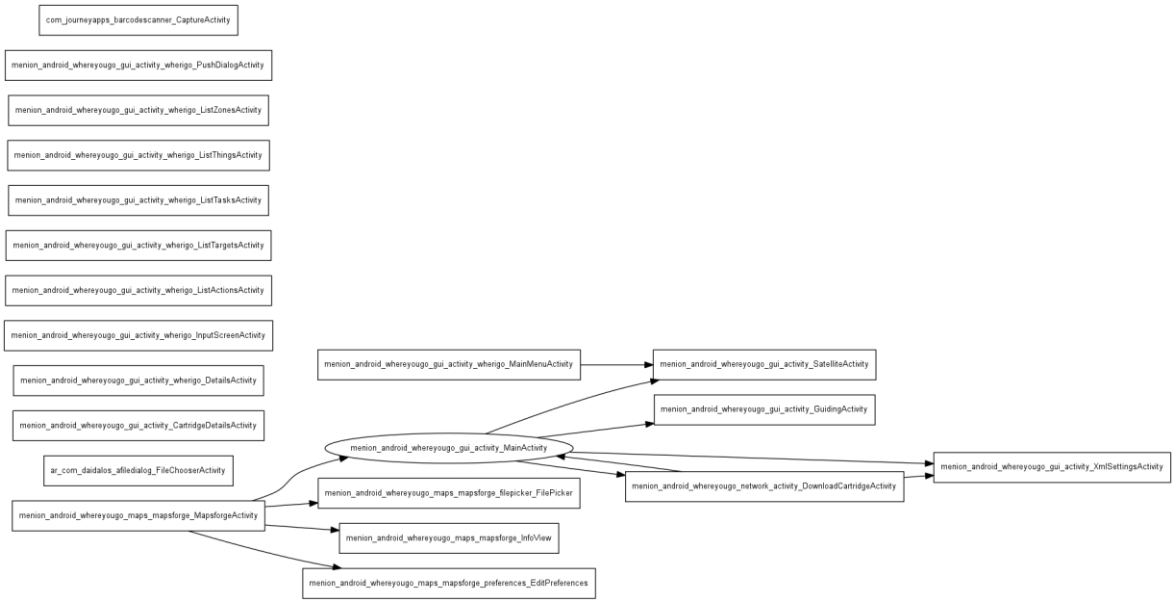
U nastavku se definira izgled čvora grafa nakon čega se navode nazivi čvorova koji će poprimiti taj izgled. Čvor se definira pomoću riječi „node“, a njegov izgled unutar uglatih zagrada različitim ključnim riječima kojima se pridodaju vrijednosti. Desna strelica između dva čvora označava da postoji prijelaz iz prvog čvora u drugi.

U ovom slučaju nastat će jedan čvor u obliku elipse koji će predstavljati glavni čvor, a svi ostali čvorovi bit će u obliku pravokutnika.

```
digraph application_graph {
rankdir = LR;
node [     fontname = "Arial",
           fontsize = 10,
           shape = oval];
     menion_android_whereyougo_gui_activity_MainActivity;
node [     fontname = "Arial",
           fontsize = 10,
           shape = box];
menion_android_whereyougo_gui_activity_MainActivity ->
     menion_android_whereyougo_gui_activity_GuidingActivity;
menion_android_whereyougo_gui_activity_MainActivity ->
     menion_android_whereyougo_gui_activity_SatelliteActivity;
menion_android_whereyougo_gui_activity_MainActivity ->
     menion_android_whereyougo_gui_activity_XmlSettingsActivity;
menion_android_whereyougo_gui_activity_MainActivity ->
     menion_android_whereyougo_network_activity_DownloadCartridgeAct
ivity;
menion_android_whereyougo_maps_mapsforge_MapsforgeActivity ->
     menion_android_whereyougo_gui_activity_MainActivity;
menion_android_whereyougo_maps_mapsforge_MapsforgeActivity ->
     menion_android_whereyougo_maps_mapsforge_filepicker_FilePicker;
menion_android_whereyougo_maps_mapsforge_MapsforgeActivity ->
     menion_android_whereyougo_maps_mapsforge_InfoView;
menion_android_whereyougo_maps_mapsforge_MapsforgeActivity ->
     menion_android_whereyougo_maps_mapsforge_preferences_EditPrefer
ences;
menion_android_whereyougo_maps_mapsforge_preferences_EditPreferences
;
menion_android_whereyougo_maps_mapsforge_filepicker_FilePicker;
ar_com_daidalos_afiledialog_FileChooserActivity;
menion_android_whereyougo_maps_mapsforge_InfoView;
menion_android_whereyougo_gui_activity_CartridgeDetailsActivity;
menion_android_whereyougo_gui_activity_wherigo_MainMenuActivity ->
     menion_android_whereyougo_gui_activity_SatelliteActivity;
menion_android_whereyougo_gui_activity_wherigo_DetailsActivity;
menion_android_whereyougo_gui_activity_wherigo_InputScreenActivity;
menion_android_whereyougo_gui_activity_wherigo_ListActionsActivity;
menion_android_whereyougo_gui_activity_wherigo_ListTargetsActivity;
menion_android_whereyougo_gui_activity_wherigo_ListTasksActivity;
menion_android_whereyougo_gui_activity_wherigo_ListThingsActivity;
menion_android_whereyougo_gui_activity_wherigo_ListZonesActivity;
menion_android_whereyougo_gui_activity_wherigo_PushDialogActivity;
menion_android_whereyougo_gui_activity_GuidingActivity;
menion_android_whereyougo_gui_activity_SatelliteActivity;
menion_android_whereyougo_gui_activity_XmlSettingsActivity;
menion_android_whereyougo_network_activity_DownloadCartridgeActivity
     -> menion_android_whereyougo_gui_activity_MainActivity;
menion_android_whereyougo_network_activity_DownloadCartridgeActivity
     -> menion_android_whereyougo_gui_activity_XmlSettingsActivity;
com_journeyapps_barcodescanner_CaptureActivity;
}
```

Ispis 14. Izgled DOT datoteke

Pretvaranjem DOT datoteke iz ispisa 14 u sliku nastala je slika 8, koja prikazuje isti rezultat kao i slika 7.



Slika 8. DOT datoteka pretvorena u sliku

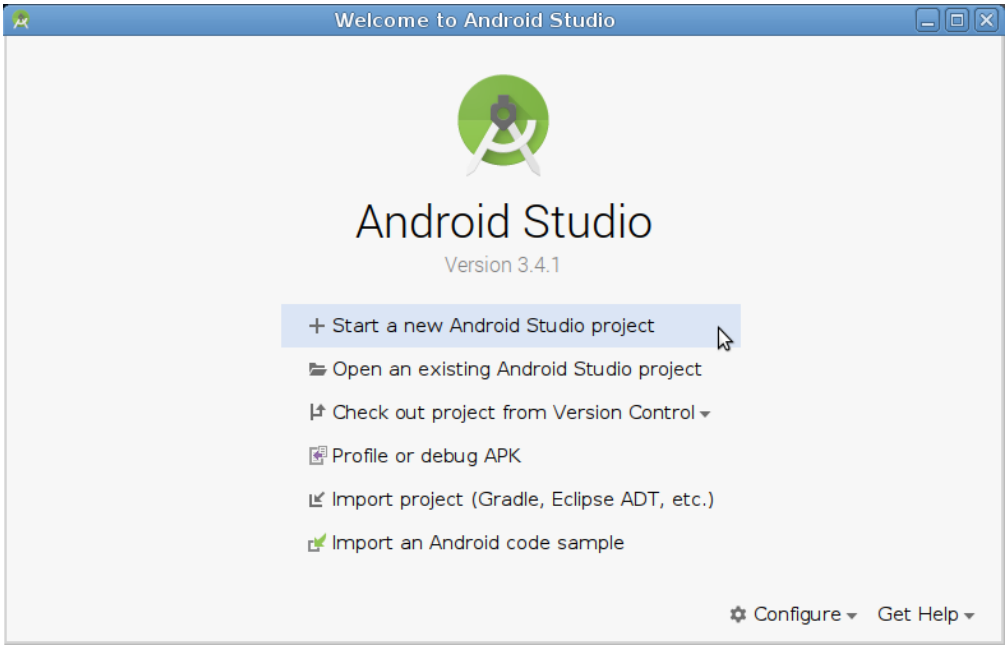
Razlog za stvaranje dvije datoteke koje služe istoj svrsi je u komplementarnosti njihovog korištenja. Datoteka navigation_graph.xml služi za rad u Android Studio-u i lakše praćenje svega što aplikacija sadrži u jednom alatu. Ukoliko postoji potreba za prikazom izvan Android Studio-a može se koristiti datoteka graphviz_graph.dot.

3.4. Primjer rada dodatka na testnoj aplikaciji

U ovom primjeru opisane su pripremne radnje koje uključuju pokretanje alata Android Studio, rad samog dodatka na testnoj aplikaciji i kako se rezultat može prikazati u Android Studio-u.

Za ovaj primjer koristi se već dekompajlirana APK datoteka privacy-friendly-netmonitor-2.0.apk koja je pohranjena na mediju koji je priložen uz rad. Dakle, podrazumijeva se da na početku ovog primjera već postoji direktorij koji sadrži skup izvora testne aplikacije.

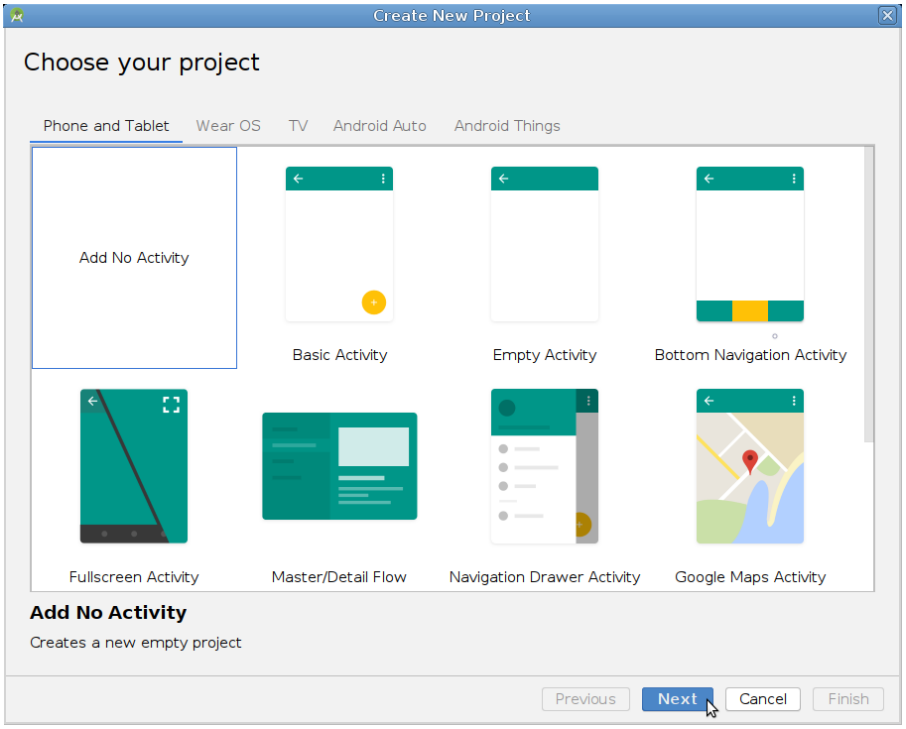
Na početku je potrebno pokrenuti alat Android Studio pri čemu će se pokazati prozor kao na slici 9. U tom prozoru ponuđeno je nekoliko mogućnosti za rad na projektu.



Slika 9. Početni prozor pokrenutog Android Studio-a

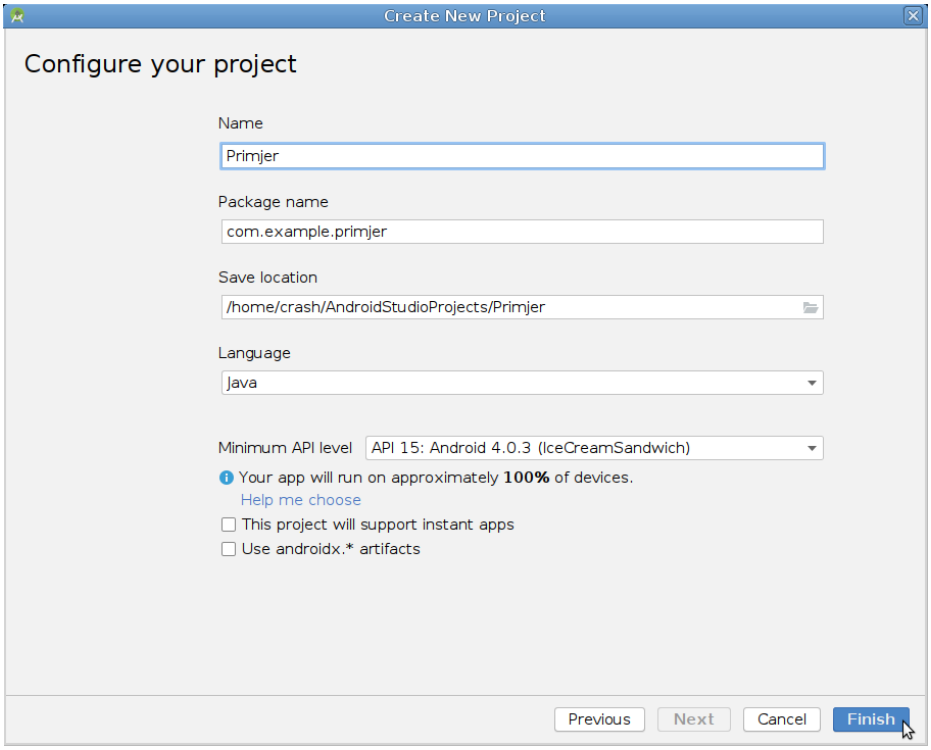
Odabere se opcija „Start a new Android Studio project“ budući da će se time stvoriti okolina potrebna za projekt koja je nužna za prikaz rezultata, a ne bi bila dostupna kada bi se skup izvora aplikacije samo otvorio ili uvezao.

Zatim će se otvoriti prozor prikazan na slici 10 u kojem je moguće odabrati kakva će se aktivnost automatski generirati u projektu koji će se stvoriti.



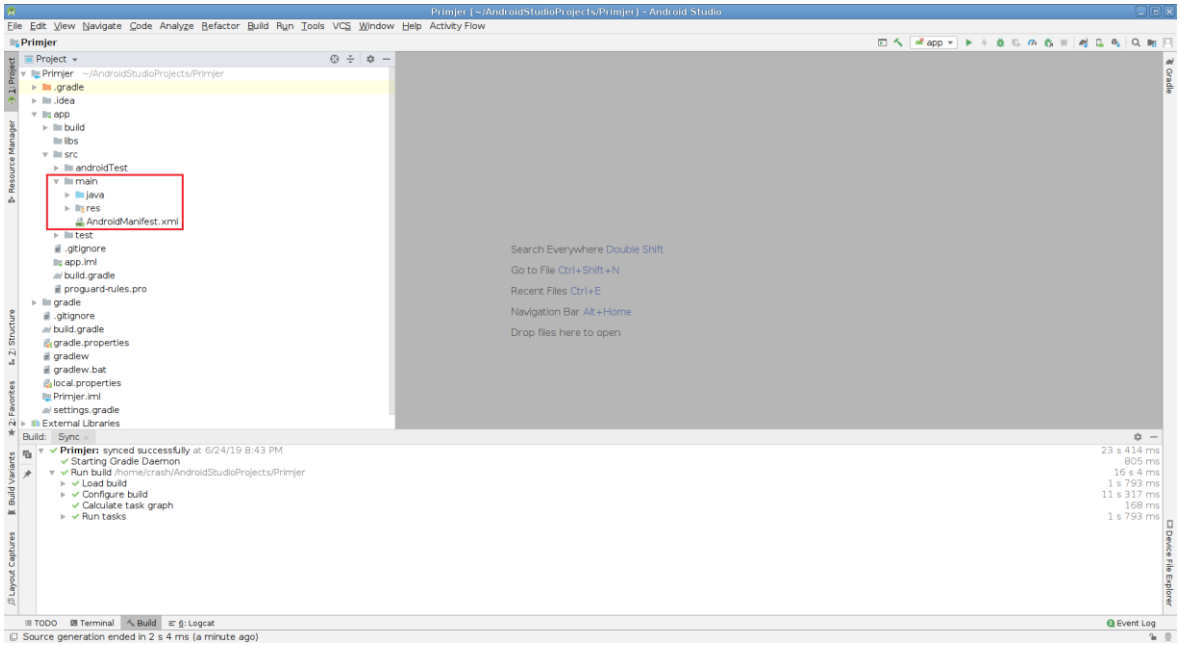
Slika 10. Prozor za odabir aktivnosti koje će se automatski generirati u projektu

Za potrebe ovog primjera dovoljno je odabrati opciju „Add No Activitiy“ i nastaviti dalje čime će se otvoriti prozor kao na slici 11. U njemu se definira ime i neka druga svojstva projekta. Upiše se ime projekta, a ostala svojstva ostave se zadana i odabere se završetak.



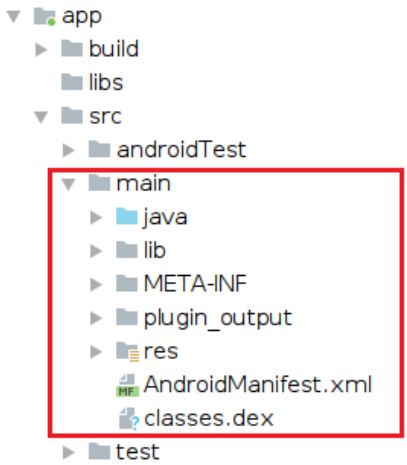
Slika 11. Prozor za postavljanje svojstava projekta

Tada će se otvoriti okruženje za rad s projektom kao na slici 12. Na njoj je osim elemenata okruženja vidljiv i označeni dio koji ističe glavni skup izvora testne aplikacije.



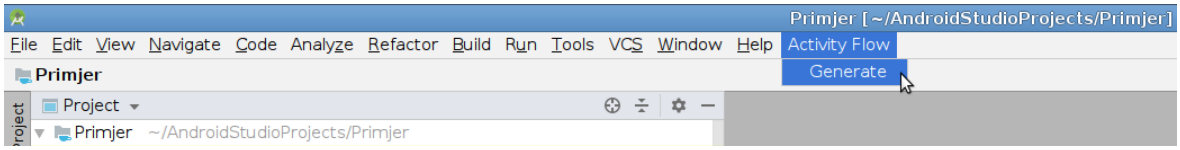
Slika 12. Okruženje Android Studio-a za rad s projektom

Zatim se u datotečnom sustavu računala pronađe direktorij sa skupom izvora testne aplikacije te ga se kopira na mjesto glavnog skupa izvora stvorenog projekta. Na slici 13 označena je promjena strukture projekta koju je uzrokovalo kopiranje skupa izvora u odnosu na označeni dio na slici 12. Kopiranje skupa izvora je posljednji pripremi korak.



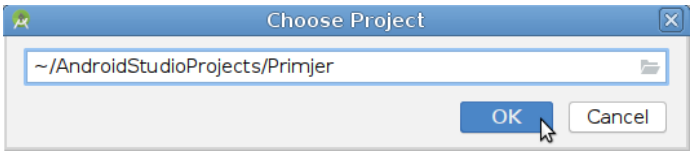
Slika 13. Promjena strukture projekta nakon kopiranja skupa izvora

Nakon što se obave svi pripremni koraci, dodatak se pokreće kako je prikazano na slici 14.



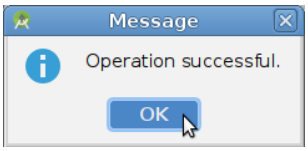
Slika 14. Pokretanje dodatka

Na početku rada dodatka otvara se prozor za odabir projekta nad kojim će se dodatak izvršiti. Slika 15 prikazuje navedeni prozor, ali na njoj se može vidjeti i da je zadana vrijednost u prozoru putanja do trenutno otvorenog projekta.



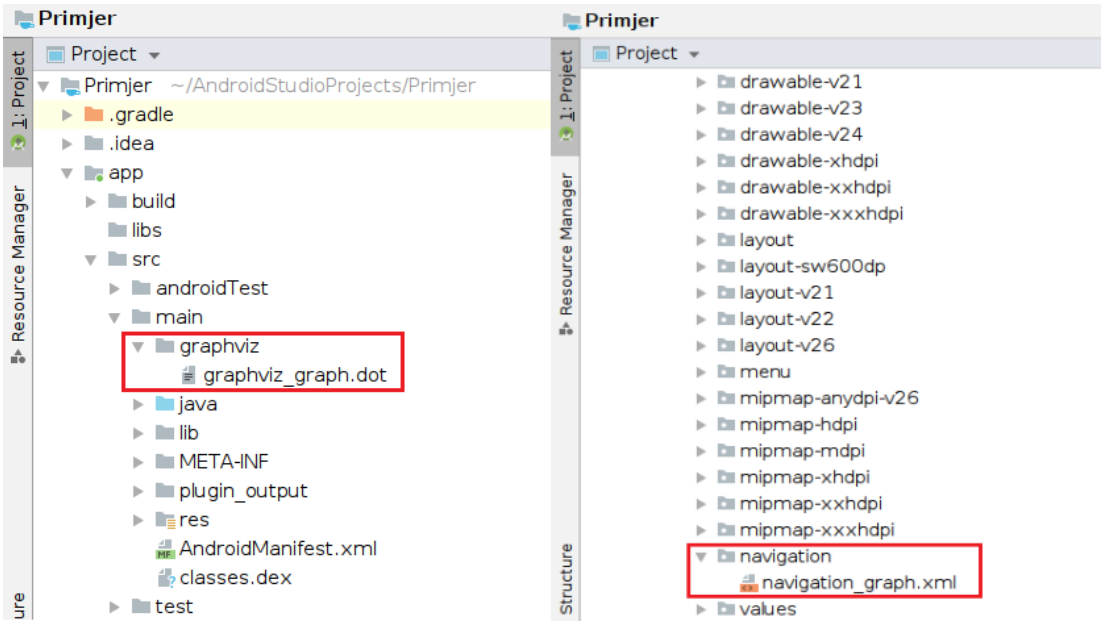
Slika 15. Odabir projekta nad kojim će se dodatak završiti

Zadana vrijednost se potvrdi i dodatak prikazuje obavijest o uspješnom izvršenju dodatka kao na slici 16.



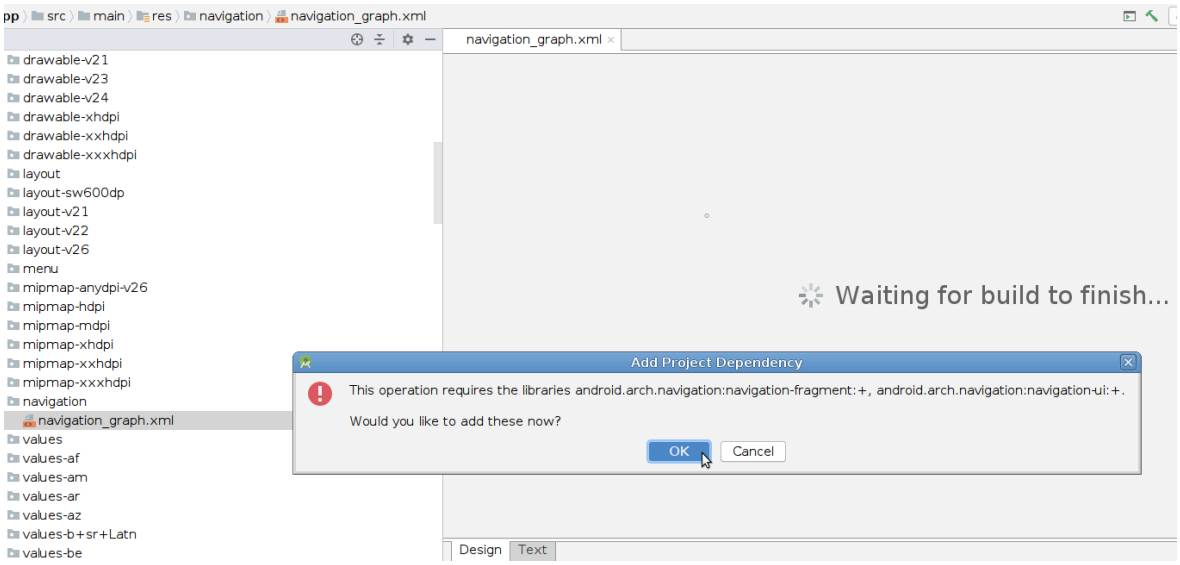
Slika 16. Obavijest o uspješnom izvršenju dodatka

Budući da u projektu postoji puno poddirektorija koji ne stanu na zaslon, na slici 17 paralelno su prikazane i prigodno označene dobivene datoteke.



Slika 17. Paralelni prikaz dobivenih datoteka

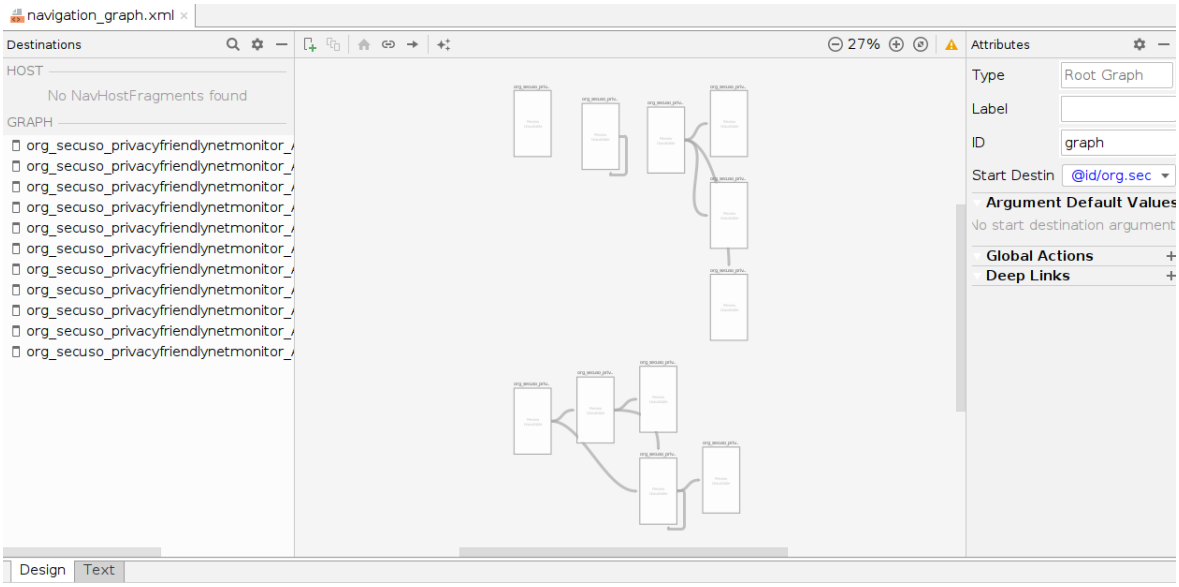
Nakon završetka rada dodatka, datoteka navigation_graph.xml može se otvoriti dvoklikom na nju. Slika 18 prikazuje početak otvaranja datoteke u Uređivaču navigacije i upozorenje o potrebi dodavanja nekih biblioteka (engl. *libraries*).



Slika 18. Početak otvaranja datoteke i upozorenje upozorenje o potrebi dodavanja nekih biblioteka

Zahtjev za dodavanje biblioteka mora se prihvatiti da bi se datoteka prikazala. Kada se zahtjevi prihvate, Uređivač navigacije prikazuje stvorenu datoteku navigation_graph.xml. Na slici 19 prikazan je rezultat rada dodatka nad testnom aplikacijom u Uređivaču navigacije. S lijeve strane je lista svih pronađenih aktivnosti, u sredini je grafički prikaz

tijeka aplikacije, a s desne strane je prikaz atributa pomoću kojeg se mogu dobiti detalji o nekom elementu iz grafičkog prikaza.



Slika 19. Prikaz datoteke navigation_graph.xml u Uređivaču navigacije

4. Zaključak

Analiza dekompajlirane APK datoteke težak je i dugotrajan posao. Automatizacija dijela ovog procesa stručnoj osobi može značajno olakšati snalaženje i ubrzati rad. Na početku rada opisano je kako se Android aplikacije izgrađuju, kakva je njihova struktura i od kojih komponenti se mogu graditi. Razvijeni dodatak tijekom izvršavanja koristi sve te informacije kako bi proizveo željeni rezultat. U radu je također prikazan i objašnjen algoritam na kojem se temelji funkcionalnost razvijenog dodatka te su opisani formati datoteka u koje se zapisuju rezultati. Uz to je dan primjer kako se koristi razvijeni dodatak.

Budući da se glavni algoritam razvijenog dodatka temelji na traženju regularnih izraza, dodatak će zadane izraze, ako postoje, brzo i sa sigurnošću pronaći u kodu jer napisani kod mora poštovati pravila zadana programskim jezikom. Također, pristup s traženjem regularnih izraza zaobilazi problem loše dekompajliranog koda.

Razvijeni dodatak je ograničen na pronalazak samo onih nizova koji su mu zadani, ali ga je moguće nadograditi dodavanjem novih regularnih izraza koji će se tražiti tijekom izvršavanja.

5. Literatura

[1] Wikipedia: *Android (operating system)*, s Interneta, https://en.wikipedia.org/wiki/Android_(operating_system), 9. lipnja 2019.

[2] Wikipedia: *Decompiler*, s Interneta, https://en.wikipedia.org/wiki/Decompiler, 9. lipnja 2019.

[3] Android Developers: *Application Fundamentals*, s Interneta, https://developer.android.com/guide/components/fundamentals, 10. lipnja 2019.

[4] Android Developers: *Configure your build*, s Interneta, https://developer.android.com/images/tools/studio/build-process_2x.png, 10. lipnja 2019.

[5] Android Developers: *Configure your build*, s Interneta, https://developer.android.com/studio/build#sourcesets, 10. lipnja 2019.

[6] Android Developers: *Projects overview*, s Interneta, https://developer.android.com/studio/projects#ProjectFiles, 11. lipnja 2019.

[7] Android Developers: *Projects overview*, s Interneta, https://developer.android.com/images/tools/projectview-p1.png, 11. lipnja 2019.

[8] Android Developers: *App resources overview*, s Interneta, https://developer.android.com/guide/topics/resources/providing-resources, 11. lipnja 2019.

[9] Android Developers: *App resources overview*, s Interneta, https://developer.android.com/guide/topics/resources/providing-resources#ResourceTypes, 11. lipnja 2019.

[10] Android Developers: *Application Fundamentals*, s Interneta, https://developer.android.com/guide/components/fundamentals.html#Components, 12. lipnja 2019.

[11] Android Developers: *Application Fundamentals*, s Interneta, https://developer.android.com/guide/components/fundamentals#ActivatingComponents, 13. lipnja 2019.

[12] Android Developers: *Intents and Intent Filters*, s Interneta, https://developer.android.com/guide/components/intents-filters.html#Types, 15. lipnja 2019.

[13] Android Developers: *Intents and Intent Filters*, s Interneta, https://developer.android.com/images/components/intent-filters_2x.png, 15. lipnja 2019.

[14] Android Developers: *App Manifest Overview*, s Interneta, https://developer.android.com/guide/topics/manifest/manifest-intro.html, 15. lipnja 2019.

[15] Android Developers: *App Manifest Overview*, s Interneta, https://developer.android.com/guide/topics/manifest/manifest-intro.html#example, 15. lipnja 2019.

[16] Android Developers: *App Manifest Overview*, s Interneta, https://developer.android.com/guide/topics/manifest/manifest-intro.html#filec, 15. lipnja 2019.

[17] Android Developers: *Application Fundamentals*, s Interneta, https://developer.android.com/guide/components/fundamentals#DeclaringComponents, 16. lipnja 2019.

[18] Wikipedia: *Compiler*, s Interneta, https://en.wikipedia.org/wiki/Compiler, 16. lipnja 2019.

[19] Oracle: *Package javax.xml.parsers*, s Interneta, https://docs.oracle.com/javase/8/docs/api/index.html?javax/xml/parsers/package-summary.html, 17. lipnja 2019.

[20] Oracle: *Class LinkedHashSet<E>*, s Interneta, https://docs.oracle.com/javase/8/docs/api/index.html?java/util/LinkedHashSet.html, 17. lipnja 2019.

[21] Android Developers: *Intent*, s Interneta, https://developer.android.com/reference/android/content/Intent.html#setClassName(java.lang.String,%2520java.lang.String), 17. lipnja 2019.

[22] Android Developers: *Intent*, s Interneta, https://developer.android.com/reference/android/content/Intent.html#setClass(android.content.Context,%2520java.lang.Class%3C?%3E), 17. lipnja 2019.

[23] Wikipedia: *?:*, s Interneta, https://en.wikipedia.org/wiki/%3F:, 17. lipnja 2019.

[24] Android Developers: *Get started with the Navigation component*, s Interneta, https://developer.android.com/guide/navigation/navigation-getting-started#create-nav-graph, 18. lipnja 2019.

[25] Android Developers: *Get started with the Navigation component*, s Interneta, https://developer.android.com/guide/navigation/navigation-getting-started#anatomy, 18. lipnja 2019.

[26] Wikipedia: *DOT (graph description language)*, s Interneta, https://en.wikipedia.org/wiki/DOT_(graph_description_language), 18. lipnja 2019.

Sažetak

Dodatak za Android Studio za lakše snalaženje u dekompajliranim APK datotekama

Ovaj diplomski rad bavi se razvojem dodatka za Android Studio za lakše snalaženje u dekompajliranim APK datotekama. U radu je također opisana izgradnja, struktura i komponente Android aplikacija te razlika između izvornog i dekompajliranog koda. Funkcionalnost razvijenog dodatka temelji se na traženju određenih regularnih izraza u dekompajliranom kodu. Rezultat izvršavanja dodatka su dvije datoteke u kojima su u različitom formatu zapisane aktivnosti aplikacije i prijelazi između njih. U radu je dan primjer korištenja dodatka nad testnom aplikacijom.

Ključne riječi: Android Studio, dodatak, APK datoteka, Android aplikacija, dekompajliranje, regularni izraz

Abstract

Android Studio plugin for easier analysis of decompiled APK files

This thesis is about development of Android Studio plugin for easier analysis of decompiled APK files. It contains description of Android application's structure, components and how it is built and difference between source code and decompiled code. Plugin functionality is based on search for certain regular expressions in decompiled code. Plugin result is list of application's activities and connections between them, which are stored in two files written in different formats. Example of plugin use on test application is given.

Keywords: Android Studio, plugin, APK file, Android application, decompilation, regular expression