

**SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN**

Tim AIR1909

DISCOUNT LOCATOR - KOTLIN

PROJEKT IZ KOLEGIJA „ANALIZA I RAZVOJ PROGRAMA“

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Alen Peršić

Kristijan Maodus

Patrik Fumić

Valentino Poljak

Projekt pisanja tutoriala:

- **GitHub:** <https://github.com/AIR-FOI-HR/AIR1909>
- **ZenHub:** <https://app.zenhub.com/workspaces/air1909-5dbc2c4ee93ebe000191a845/board?repos=218985829>

Kotlin Discount Locator projekt (za tutorial)

- **GitHub:** <https://github.com/AIR-FOI-HR/AIR1909-DL>
- **ZenHub:** <https://app.zenhub.com/workspaces/air1909-dl-5dbc2ca6de0163000119b590/board?repos=218985934>

DISCOUNT LOCATOR - KOTLIN

PROJEKT IZ KOLEGIJA „ANALIZA I RAZVOJ PROGRAMA“

Mentor/Mentorica:

Doc. dr. sc. Zlatko Stapić

Varaždin, veljača 2020.

Sadržaj

1. Uvod	1
2. Popis funkcionalnosti	2
3. Baza podataka.....	9
4. Razvoj aplikacije	10
4.1. Instalacija razvojnog okruženja	10
4.2. Postavljanje Android SDK-a.....	13
4.3. Preporučene postavke Android Studio razvojnog okruženja.....	18
4.4. Kreiranje novog projekta	21
4.5. Postavljanje virtualnog uređaja	24
4.6. Kreiranje modula baze podataka.....	26
4.6.1. Firebase	26
4.6.2. SQLLite (Room)	31
4.7. Osnovni UI	34
4.8. Navigacija	38
4.9. Dohvaćanje podataka s webservisa	8
4.10. Kreiranje core modula.....	13
4.11. Kreiranje map view modula.....	14
4.12. RecyclerView	16
4.13. Push notifikacije.....	21
4.14. Google Ads.....	25
4.15. Testiranje i analiza koda	27
Popis literature	33
Popis slika	34
Popis tablica.....	37

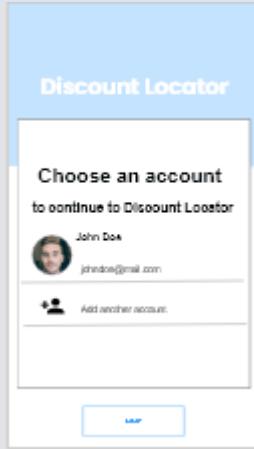
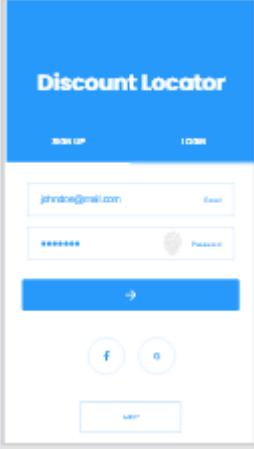
1. Uvod

Tema ovog projekta je Android aplikacija za lociranje popusta, pod nazivom „*Discount Locator*“. Umjesto klasičnog rada u Java programskom jeziku, u ovome projektu koristili smo Kotlin programski jezik i u nastavku će biti detaljno opisan postupak kreiranja potpunog Android projekta od same instalacije razvojnog okruženja do pojedinačnih implementacija raznih featurea koji će kasnije sačinjavati gotov projekt. Cilj ovog rada je da student koji će pratiti isti bude u stanju razviti cijelu Android aplikaciju u Kotlin programskom jeziku uz pomoć Android Studio razvojnog okruženja.

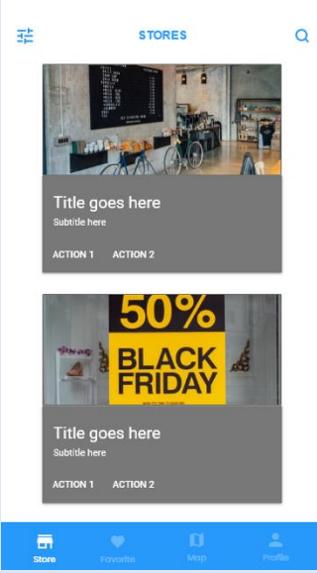
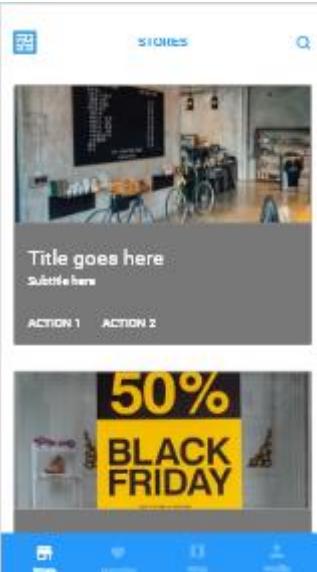
2. Popis funkcionalnosti

Tablica 1. Popis funkcionalnosti

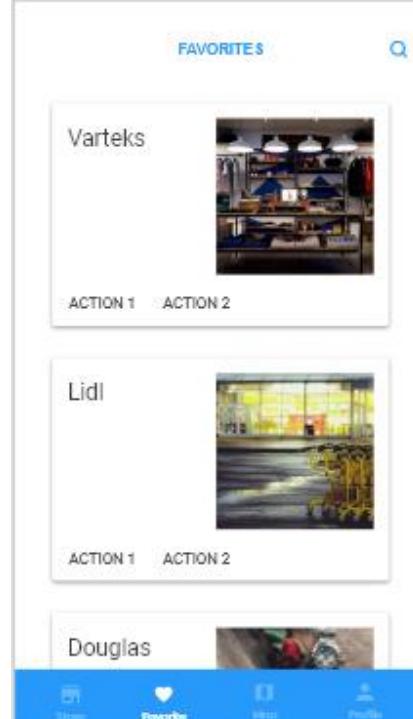
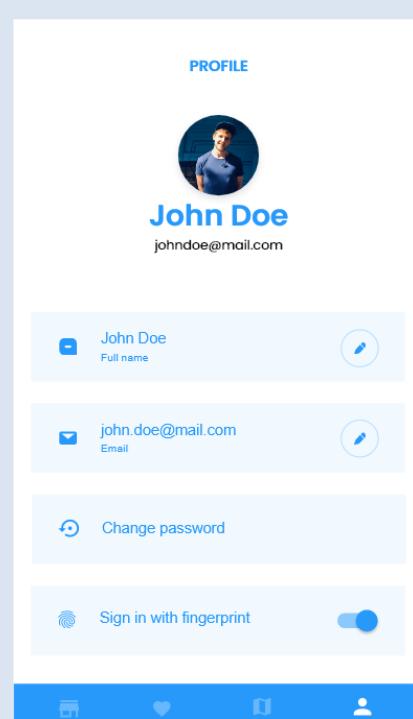
Broj funkc.	Naziv funkcionalnosti	Opis funkcionalnosti	Slika ekrana
1.	Registracija (modularno)	Registriranje korisnika – zapis imena, prezimena, lozinke, korisničkog imena, email-a i godine rođenja korisnika u bazu podataka	
1.1.	Registracija (Facebook)	Registracija korisnika putem njihovog Facebook korisničkog računa	

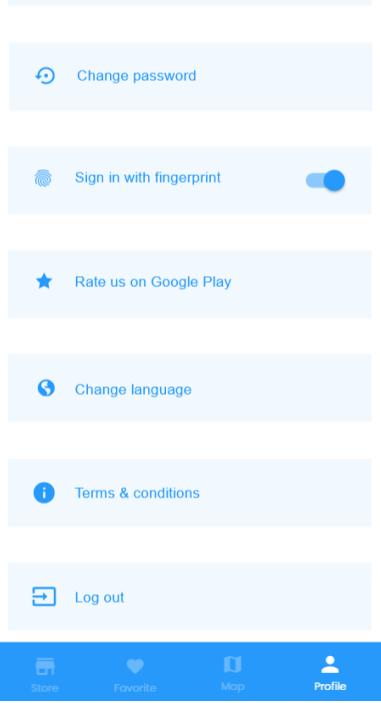
1.2.	Registracija (Google)	<p>Registracija korisnika putem njihovog Google korisničkog računa</p>	
2.	Prijava (modularno)	<p>Mogućnost prijave korisnika u sustav sa svojim podacima za prijavu. Izvedena će biti prijava korisničkim imenom i lozinkom te brza prijava pomoću otiska prsta</p>	
2.1	Prijava (korisničko ime i lozinka)	<p>Prijava je moguća korištenje korisničkog imena i lozinke kojim se korisnik registrirao, te je to osnovna vrsta prijave koju korisnik uvijek može koristiti u slučaju da ostale vrste ne rade ili ako korisnik to preferira. Kada se korisnik prvi puta prijavi sa svojim podacima u našu</p>	

		bazu podataka će se povući njegovi podaci i kreirati njegov korisnički profil u našoj bazi te će se korisniku u budućnosti omogućiti prijava sa tim podacima unutar same aplikacije i korištenje brzog načina prijave.	
2.2	Prijava (Otisak prsta) (modularno)	Prijava putem otiska prsta bit će opcija brze prijave za korisnike koji imaju čitač otiska prsta na svojem mobilnom uređaju. Kada se korisnik prijavi u sustav može u unutar svojeg profila aktivirati mogućnost korištenja otiska prsta te će mu se u budućnosti nuditi ta opcija za prijavu	

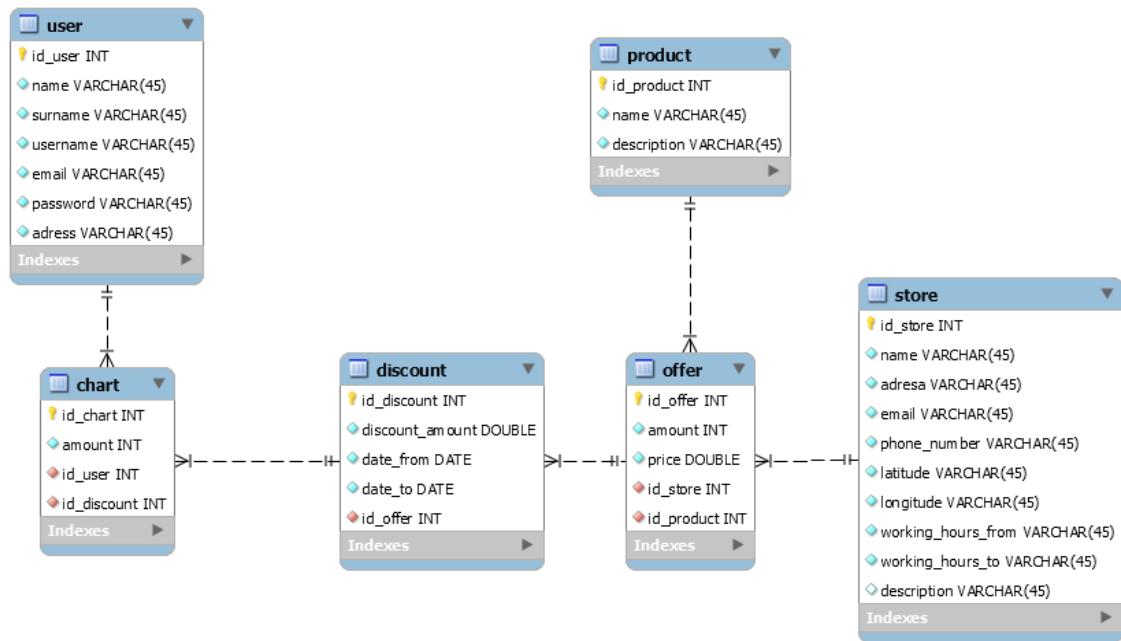
3.	Početni zaslon aplikacije	Ovo je osnovni zaslon koji svaki korisnik vidi kada otvorí aplikaciju	
3.1	Pregled trgovina	Pregled svih trgovina iz baze podataka te je moguće te trgovine na razne načine sortirati ili filtrirati	
3.1.1	Sortiranje trgovina	Mogućnost sortiranja trgovina po abecednom redu, kategorijama artikala koje trgovine prodaju, udaljenosti od trenutne lokacije korisnika	

3.1.2	Filtriranje trgovina	Mogućnost filtriranja trgovina prema prvom slovu naziva trgovine	
3.1.3	Pregled proizvoda odabrane trgovine	Pritiskom na određenu trgovinu otvara nam se popis svih proizvoda te trgovine s njihovim cijenama	
3.2	Pregled svih proizvoda	Mogućnost pregleda svih proizvoda koji su zapisani u bazi podataka, moguće ih je sortirati te filtrirati na razne načine	
3.2.1	Sortiranje proizvoda	Mogućnost sortiranja proizvoda prema kategorijama, datumu dodavanja, cjeni, popustu	
3.2.2	Filtriranje proizvoda	Mogućnost filtriranja proizvoda prema prvom slovu proizvoda ili nazivu proizvoda.	
3.2.3	Pregled odabranog proizvoda	Pritiskom na određeni proizvod otvara se novi prozor u kojem je moguće vidjeti detalje o	

		određenom proizvodu te trgovine koje taj proizvod prodaju.	
3.2.4	Favoriti	Mogućnost da korisnik doda neku trgovinu kao favorita.	
3.5	Korisnički račun i postavke korisničkog računa	Ekran korisničkog računa sadrži razne informacije o korisniku te također i postavke koje korisnik može mijenjati	

3.5.1	Postavke korisničkog računa	<p>Pritiskom na postavke otvara se novi prozor u kojem je moguće mijenjati jezik aplikacije, odjaviti se prikazati uvjete korištenja</p>	
-------	-----------------------------	--	--

3. Baza podataka



Na slici iznad nalazi se osnovni ERA model naše aplikacije. Sada ćemo pojasniti svaku od tablica modela kako bi bilo lakše shvatiti model podataka.

User – Tablica koja se odnosi na sve registrirane korisnike aplikacije sa osnovnim podacima svakoga od njih

Product – Tablica product predstavlja sve proizvode koje su trenutno u ponudi aplikacije (neovisno o tome koja trgovina ima proizvod u ponudi)

Store – Predstavlja sve trgovine koje koriste aplikaciju, te njihove osnovne podatke

Offer – Tablica koja sadrži ponudu proizvoda jedne trgovine, sa definiranom količinom i cijenom(1 kom) proizvoda

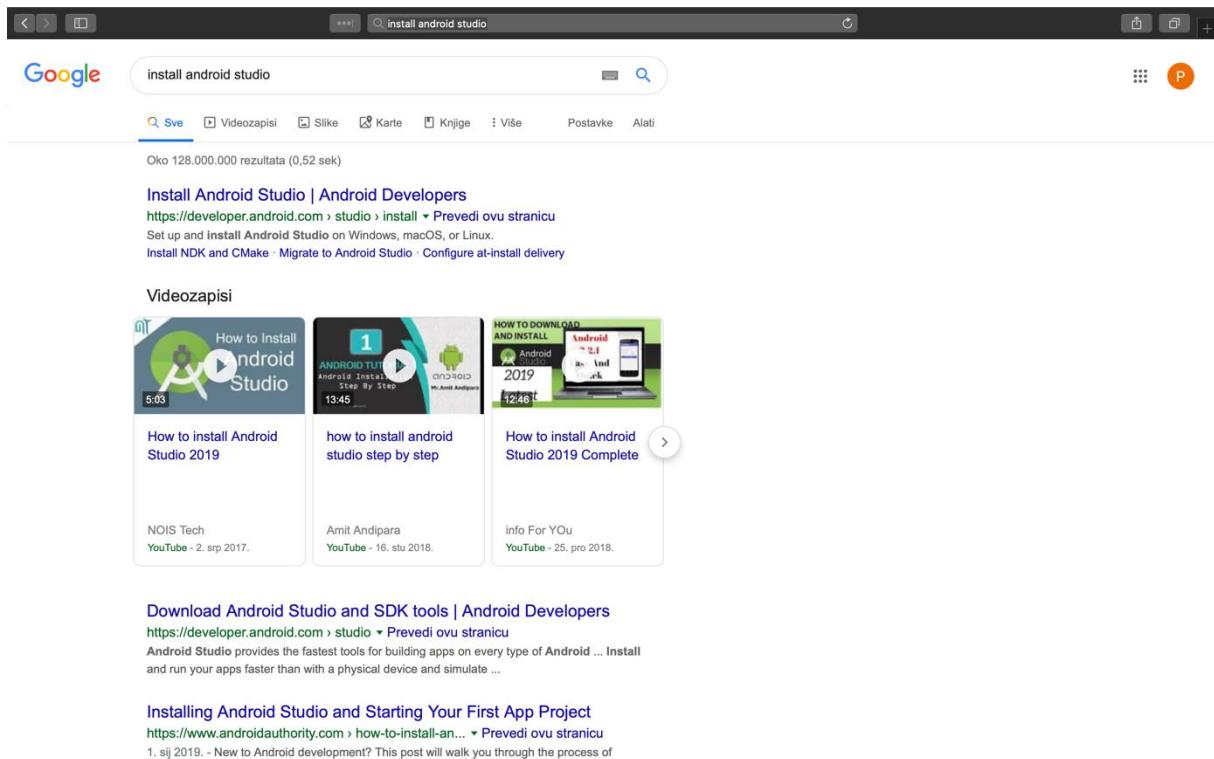
Discount – Tablica predstavlja kolicišnu i trajanje popusta određenog proizvoda

Chart – Tablica chart jest košarica svih proizvoda koje je korisnik odabrao za kupnju

4. Razvoj aplikacije

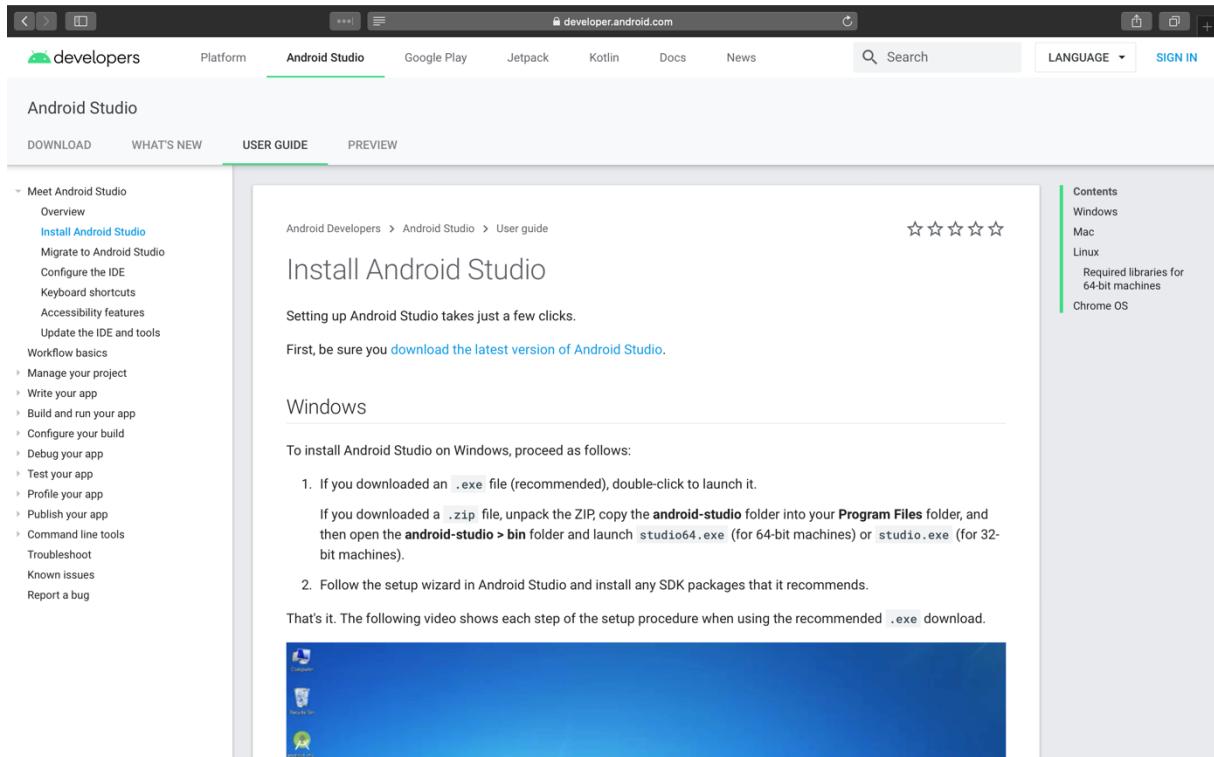
4.1. Instalacija razvojnog okruženja

Prije samog početka rada u Kotlin programskom jeziku potrebno je instalirati razvojno okruženje uz pomoć kojeg ćemo kasnije kreirati naš projekt i isprogramirati potrebne funkcionalnosti. Za potrebe ovog rada koristit ćemo Android Studio. Kako bi instalirali Android Studio potrebno je u bilo kojem internet pregledniku otvoriti Google i utipkati „install android studio“.



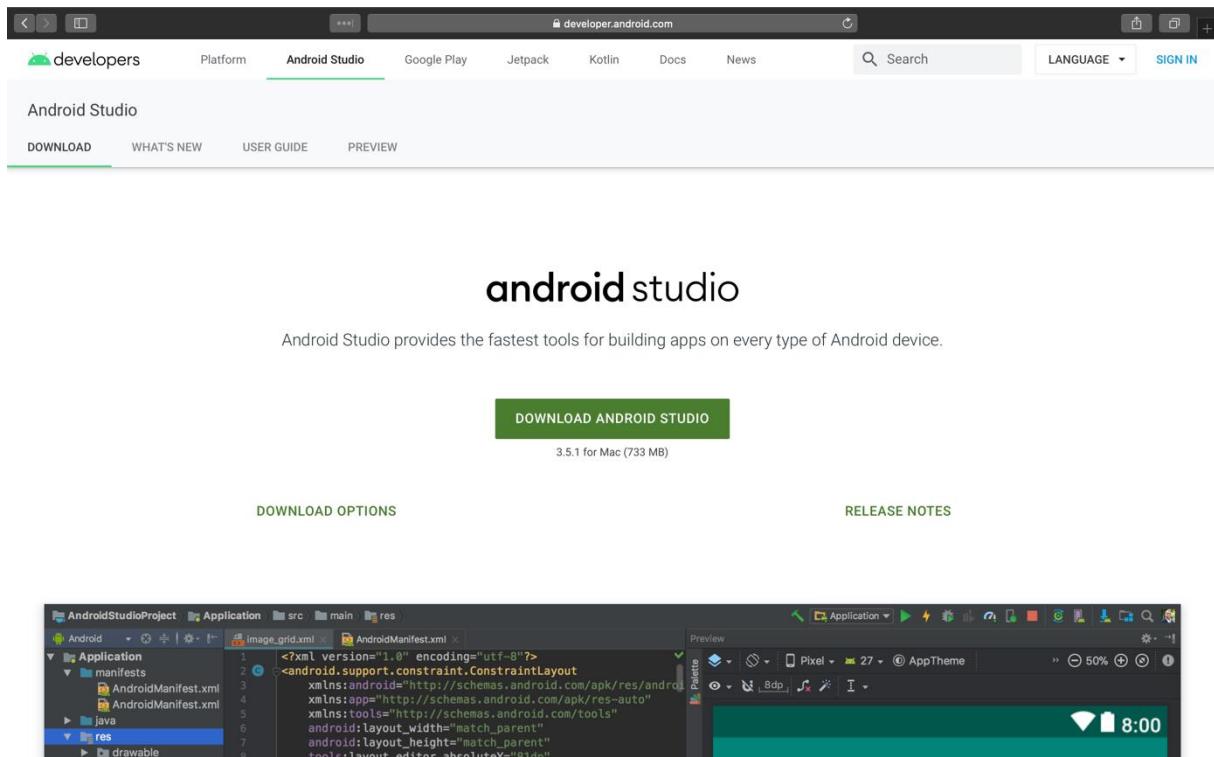
Slika 1. Pronalazak Android Studio razvojnog okruženja

Klikom na prvi link nam se otvara službena developer.android stranica na kojoj klikom na link „download the latest version of Android Studio“ odlazimo na stranicu za preuzimanje zadnje Android Studio verzije.



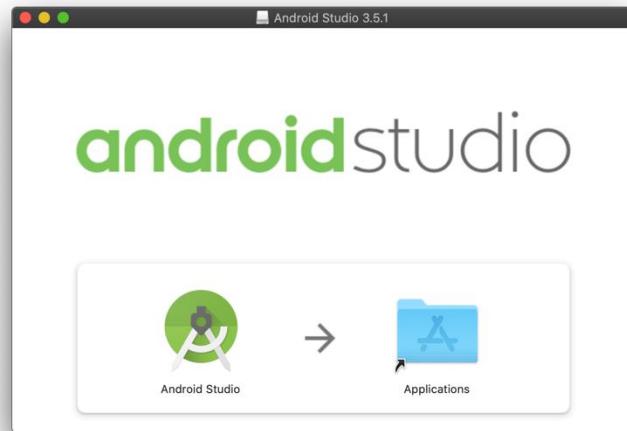
Slika 2. Pronalazak zadnje verzije Android Studio razvojnog okruženja

Nakon ovog klika ćemo imati prikazanu zadnju verziju Android Studio razvojnog okruženja za operativni sustav računala na kojemu se nalazimo, u našem slučaju je to verzija 3.5.1 za Mac, kao što je prikazano na sljedećoj slici.



Slika 3. Zadnja verzija Android Studio razvojnog okruženja

Sljedeći korak je preuzimanje Android Studia klikom na zeleni gumb „DOWNLOAD ANDROID STUDIO“, nakon čega ćemo morati potvrditi uvjete korištenja i naše preuzimanje će započeti. Kada završi preuzimanje instalacijske datoteke u cijelosti potrebno je otvoriti istu. U našem slučaju je potrebno „drag&drop“ akcijom potrebno potvrditi početak instalacije.

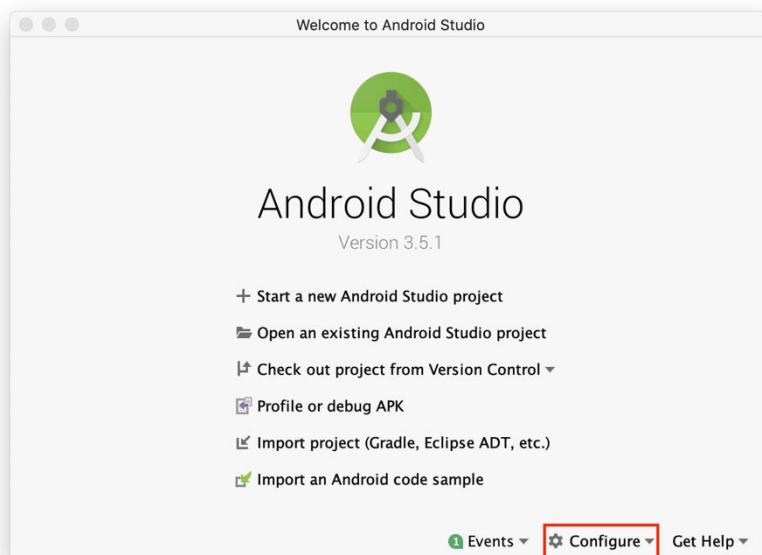


Slika 4. Početak instalacijskog postupka

Nakon toga unutra Applications foldera pokrećemo Android Studio i na taj način pokrećemo proces instalacije. S obzirom da se radi o prvoj instalaciji Android Studio razvojnog okruženja, nećemo uvoziti nikakve prethodne postavke. Odabiremo Custom oblik instalacije, po želji se odabire tema razvojnog okruženja. Nakon toga je potrebno označiti Android Virtual Device kako bi instalirali, virtualni Android uređaj, kako mu i samo ime govori. Iduće je potrebno definirati količinu memorije koje ćemo dodijeliti takozvanom „Intel Hardware Accelerated Execution Manageru, skraćenica HAXM. Radi se o programu razvijenom od strane Intel-a i ubrzava rad x86 emulatora. Ako nemate Intel CPU, nećete imati ovu mogućnost. Ako imate 8 GB RAM-a ili više dovoljno je ostaviti na predloženoj vrijednosti(2 GB), ukoliko imate manje, primjerice 4GB, bilo bi dobro razmisliti o dodijeljivanju manjeg dijela memorije. Nakon toga klikom na gumb „Finish“ završavamo postavljanje instalacije i kreće preuzimanje SDK-a (software development kit), odnosno softwarea koji će nam omogućiti pisanje Android aplikacija i on je odvojen od onoga što smo preuzeli u početku, a to je samo razvojno okruženje. Nakon što se preuzimanje završilo klikom na „Finish“ potvrđujemo završetak instalacijskog postupka i prikazuje nam se prozor dobrodošlice Android Studio razvojnog okruženja.

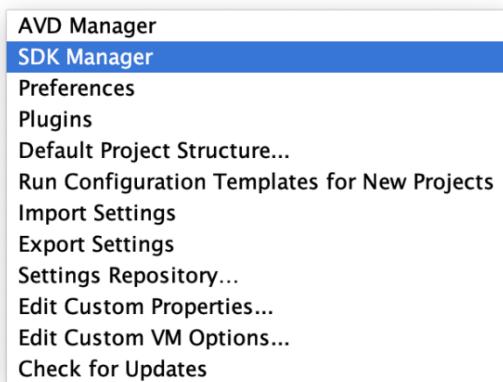
4.2. Postavljanje Android SDK-a

U ovom poglavlju ćemo napraviti određene konfiguracijske promjene i instalirati određene opcije koje mogu biti korisne. Ukoliko već nije pokrenut, potrebno je pokrenuti Android Studio. Na početnom ekranu potrebno je odabratи opciju „Configure“.



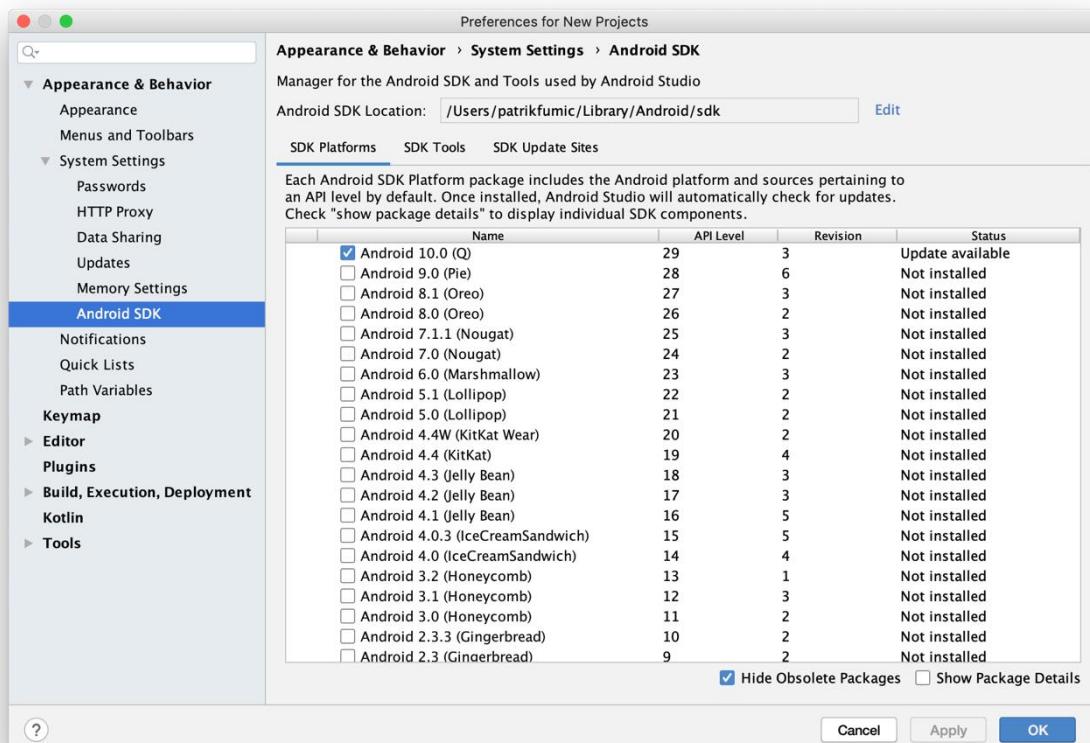
Slika 5. Početni prikaz Android Studio razvojnog okruženja

I nakon toga odabiremo SDK Manager opciju.



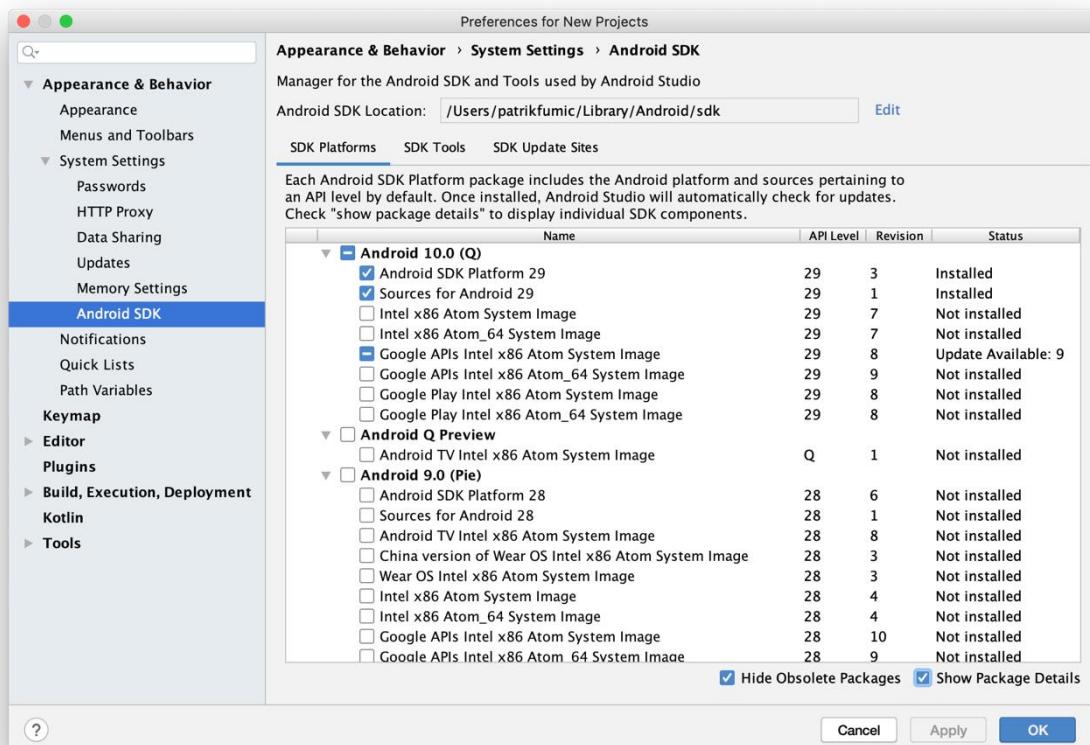
Slika 6. Izbornik mogućih postavki

Nakon što smo izvršili prethodne upute, pojavit će nam se sljedeći prozor na kojem vidimo Android SDK postavke.



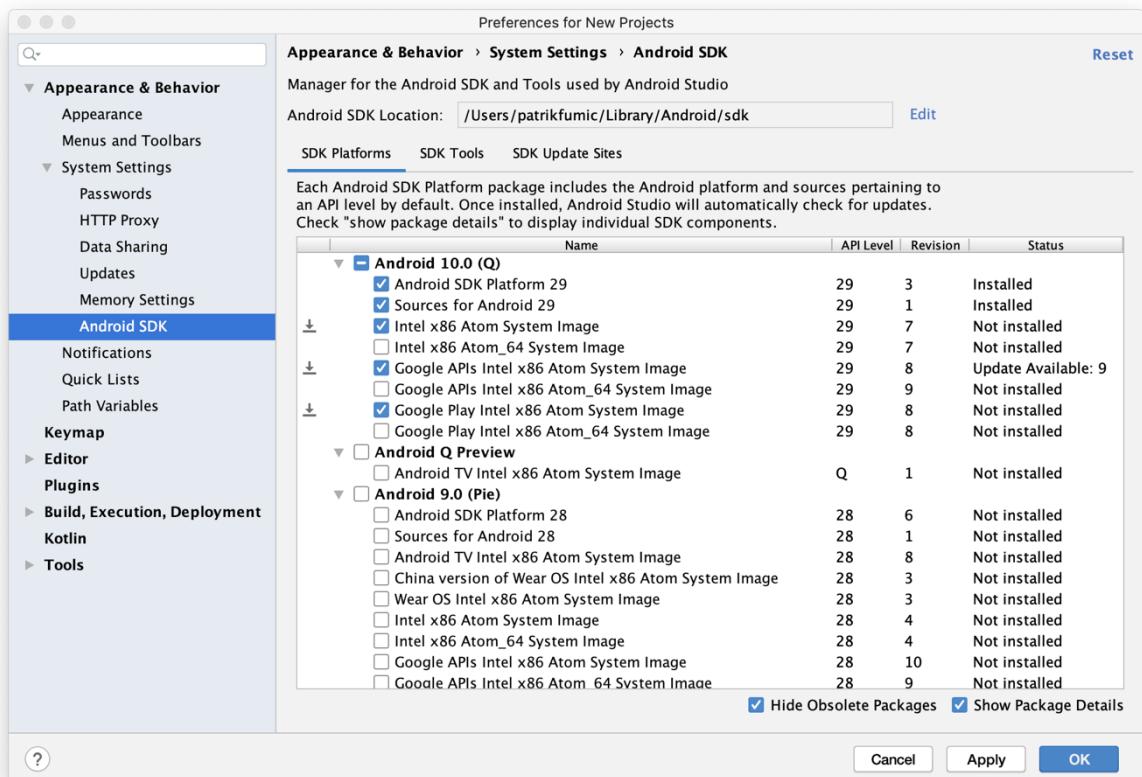
Slika 7. Android SDK postavke

Prvo što je potrebno provjeriti, posebice na Windows operativnim sustavima, je putanju Android SDK-a. Putanju možemo vidjeti pri vrhu prikazanog prozora na prethodnoj slici nakon natpisa „Android SDK Location“. Problem može uzrokovati razmak u samoj putanji koja je navedena. Uvjerite se postoji li negdje razmak u putanji. Ukoliko postoji morat ćeće prebaciti lokaciju Android SDK-a. To možete napraviti na način da ručno kopirate prikazanu mapu negdje na istom disku čija putanja ne sadrži razmak. Nakon što smo se osigurali da nam putanja ne sadrži razmake možemo nastaviti dalje. Trenutno se nalazimo u Android SDK upravitelju i ovdje možemo instalirati dodatne značajke ili možemo ažurirati pojedine elemente ukoliko su ažuriranja dostupna. Ukoliko označimo „Show Package Details“ dobit ćećemo dodatan prikaz pojedinih Android verzija.



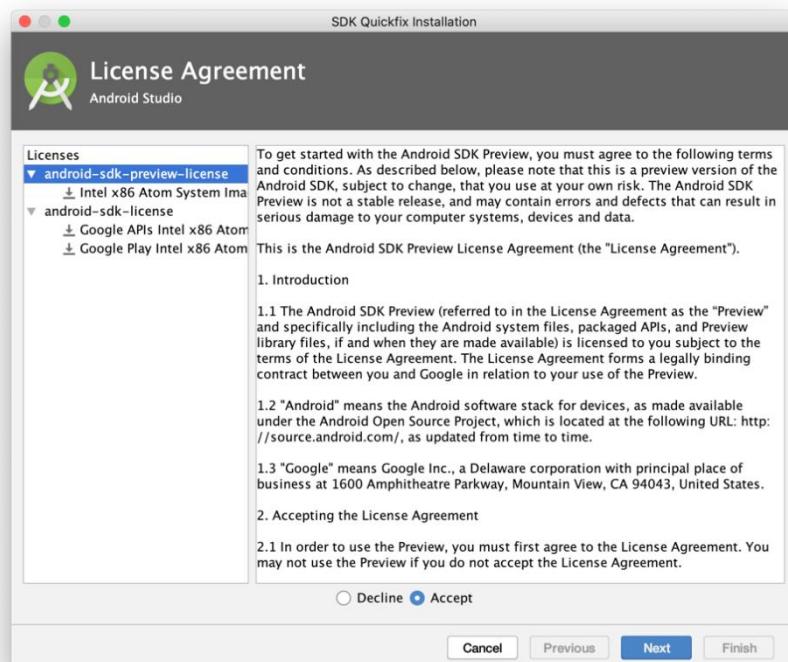
Slika 8. Prikaz detalja paketa u pojedinim verzijama Androida

Važno je naglasiti da nije potrebno instalirati većinu prethodnih Android verzija, zadnja verzija SDK-a će nam omogućiti razvoj aplikacija koje se mogu izvršavati na ranijim verzijama. Druga stvar koju ćemo naglasiti da SDK folder može jako brzo narasti u veličini ukoliko preuzimamo mnogo opcija. Jednostavno je popuniti približno 40 GB čak i bez da smo instalirali sve opcije. Međutim ukoliko želimo prikazati aplikacije razvijene u prethodnim verzijama bit će potrebno preuzeti danu verziju. Ono što se može uočiti da se mogu pojaviti takozvane „Preview“ verzije SDK i tu se radi o verzijama koje su samo za prezentaciju nadolazećih verzija i ne preporuča ih se koristiti za isporuku aplikacija. Mi ćemo svoj fokus usmjeriti Android 10.0 (Q) verziji i želimo imati instalirano sve označene komponente plus Intel x86 Atom System Image i Google Play Intel x86 Atom System Image. Nakon što smo označili sve što je navedeno u prethodnoj rečenici prozor bi trebao izgledati kao što je prikazano na sljedećoj slici i možemo kliknuti na gumb „Apply“.



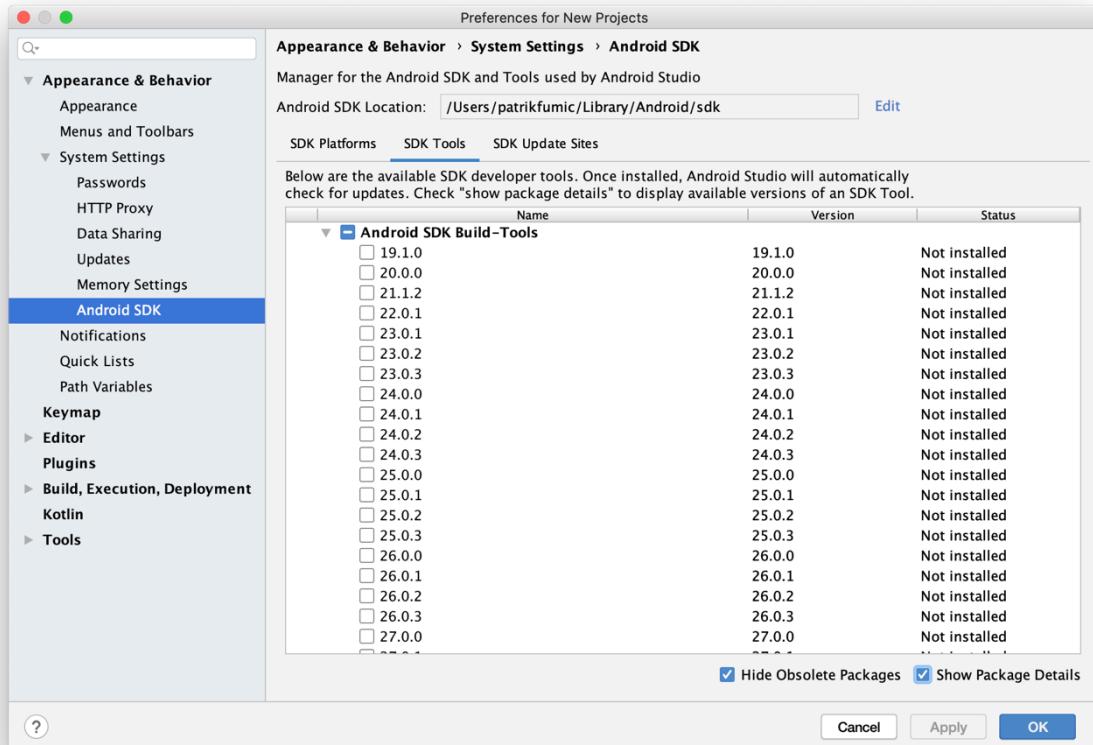
Slika 9. Prikaz označenih potrebnih paketa

Nakon što smo kliknuli na gumb „Apply“ potrebno je prihvatići uvjete korištenja svih licenca.



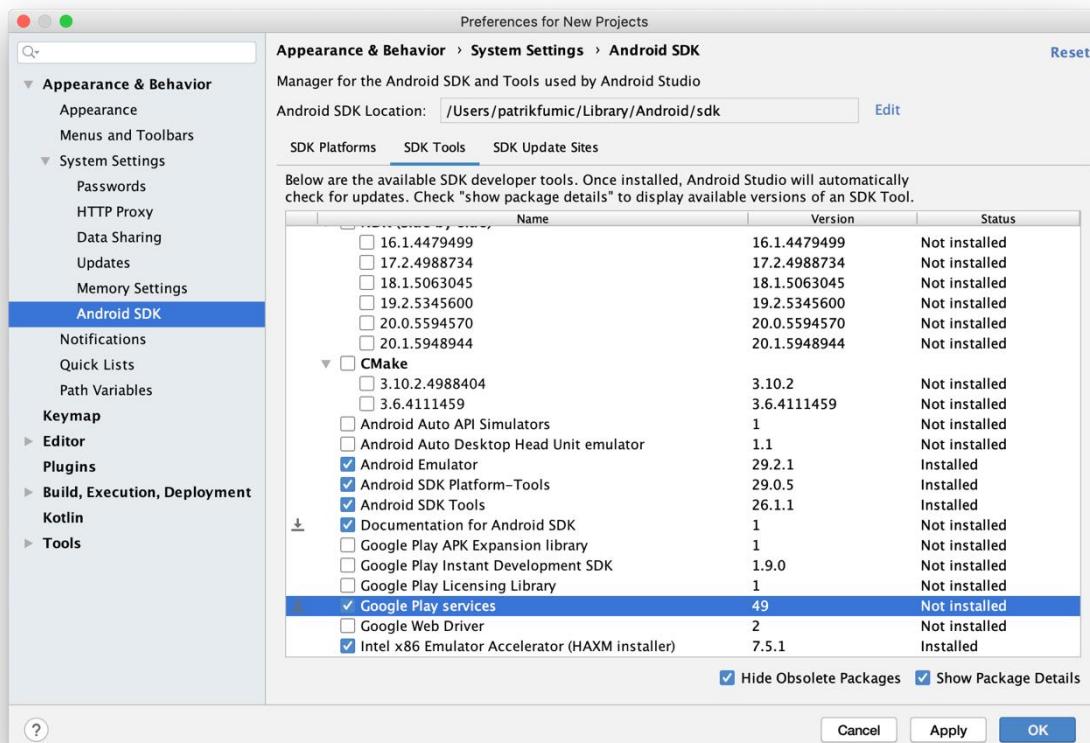
Slika 10. Prihvatanje uvjeta korištenja paketa

Nakon toga će započeti preuzimanje zatraženih paketa. Na kraju preuzimanja, samo potvrdimo proces klikom na „Finish“ gumb. Nakon toga na prikazu SDK upravitelja odabiremo tab SDK Tools i pri dnu označimo „Show Package Details“.



Slika 11. Prikaz SDK alata

Prvo što je potrebno napraviti je provjeriti postoji li dostupno ažuriranje za neki od instaliranih paketa. Ukoliko postoji potrebno je označiti isti i pustiti ažuriranje. Drugo što je potrebno napraviti je preuzeti zadnju verziju Android SDK Build – Tools. Povlačenjem prikaza dostupnih verzija prema dolje doći ćemo do kraja popisa i želimo imati preuzetu zadnju moguću verziju bez „RC“ kratice, navedena kratica označava pojam „Release Candidate“ što znači da se ne radi o dovršenoj verziji. Sljedeći korak je provjeriti jesu li označeni Android Emulator, Android SDK Platform-Tools, Android SDK Tools, Documentation for Android SDK, Google Play services i Google USB Driver. Svi alati osim zadnja tri bi trebali biti prethodno označeni, dok će zadnja tri biti potrebno ručno preuzeti. Google Play services je iznimno važan zato što nam omogućuje funkcionalnosti kao što su Google Maps, a Google USB Driver nam omogućava stabilnije spajanje s fizičkim Android uređajem.

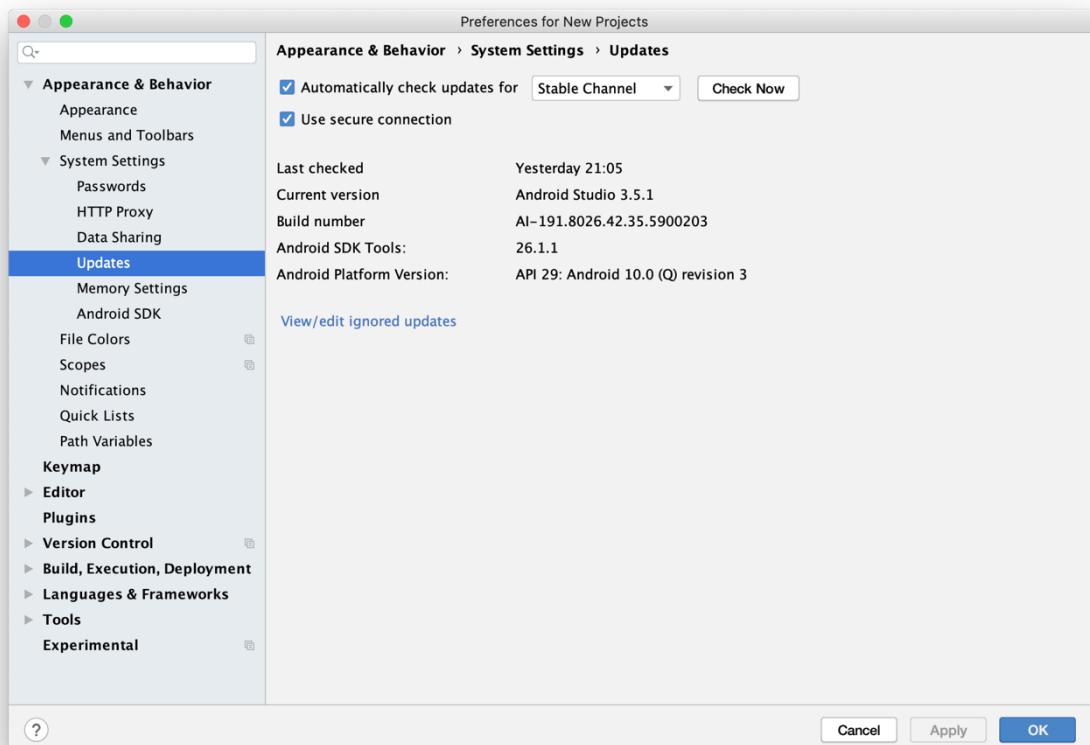


Slika 12. Prikaz potrebnih SDK alata

Važno je naglasiti da se u bilo kojem trenutku možemo vratiti na ovaj prikaz i naknadno instalirati potrebne alate, ukoliko se uvidi potreba za istima.

4.3. Preporučene postavke Android Studio razvojnog okruženja

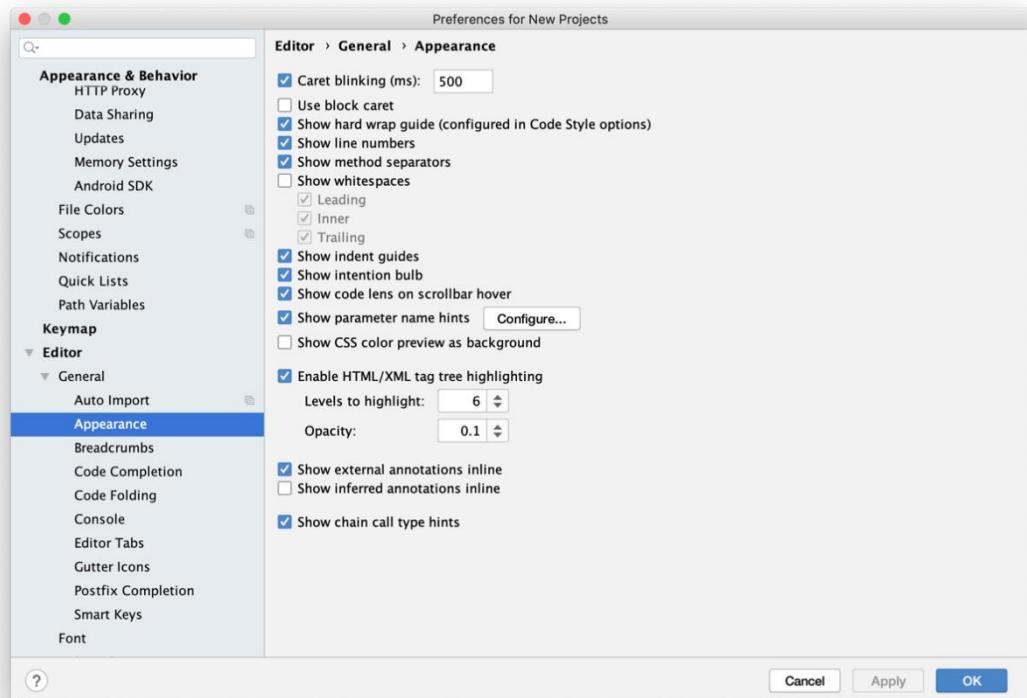
Na početnom prikazu Android Studio razvojnog okruženja odabiremo opciju pri dnu „Configure“ i nakon toga odabiremo „Settings“ na Windows računalima ili „Preferences“ na Mac računalima. Nakon što nam se otvorio traženi prozor potrebno se pozicionirati kao što je prikazano na sljedećoj slici.



Slika 13. Android Studio Updates

Potrebno je odabrati iz padajućeg izbornika, ukoliko već nije, „Stable channel“ opciju. Nakon toga idemo u Editor odjeljak i potrebno je konfigurirati sve kako je prikazano u nastavku. [1]

Slika 14. Auto Import



Slika 15. Appearance

„Show method separators“ u prethodnoj slici je u potpunosti opcionalna postavka, ono što nam omogućava je lakše uočavanje različitih metoda i na taj način potencijalno ubrzava rad. U idućem odjeljku Code Folding je potrebno ukloniti neke opcije, za razliku od prethodnih odjeljaka gdje smo ih dodatno označavali.

Slika 16. Code Folding

Opcije na prethodnoj slici definiramo na ovaj način zato što se radi o opcijama koje su više usmjerene naprednim Android razvojnim programerima. Za nekoga tko tek počinje s razvojem u Android studio razvojnom okruženju ovo je dobra praksa zato što navedene opcije skrivaju kod i mogu biti zbumujuće.

4.4. Kreiranje novog projekta

Prvi korak kod kreiranja novog programa je pokretanje Android Studio razvojnog okruženja i na početnom prozoru odabiranje opcije „Start a new Android Studio project“.

Slika 17. Kreiranje novog projekta

Nakon toga se pojavljuje prozor s prethodno definiranim Android predlošcima. U našem slučaju ćemo odabratи opciju „Empty Activity“.

Slika 18. Odabir Android predloška

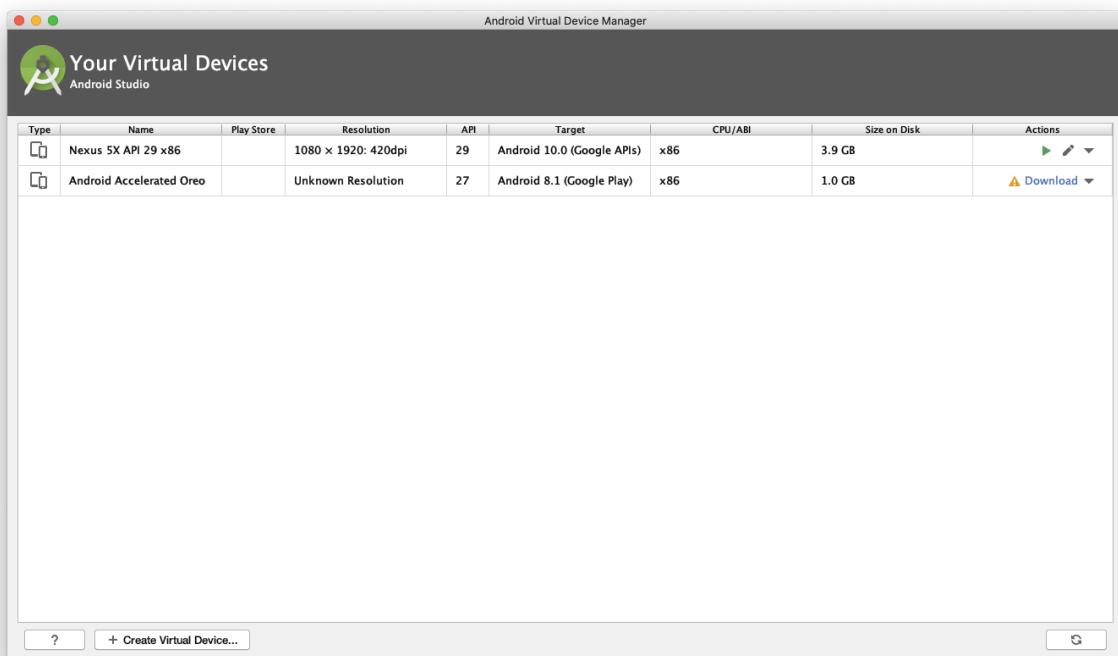
U idućem koraku je potrebno navesti ime aplikacije i ime paketa. Package name bi trebao biti jedinstven kako bi referenca na aplikaciju bila jedinstvena u svijetu. Razlog je taj da jednom kad želimo uploadati aplikaciju na Google Play Store, aplikacija mora imati jedinstveni identifikator koji nitko drugi ne koristi. Lokaciju na koju želimo spremiti naš projekt proizvoljno postavljamo, u našem slučaju to je lokacija na koju smo klonirali repozitorij koji smo dobili od strane AIR kolegija. Nakon toga odaberemo jezik u kojem želimo razvijati projekt, a to je Kotlin. Zadnja postavka koju odabiremo je Minimalna razina API-a. Ovdje dolazimo do trenutka u kojem moramo odabirati između većeg broja značajki kod novijih verzija ili većeg podržanog broja korisnika koje nam omogućuju starije verzije. Mi smo se odlučili na API 19: Android 4.4 (KitKat). Klikom na gumb „Finish“ potvrđujemo postupak postavljanja projekta.

Slika 19. Početno postavljanje projekta

4.5. Postavljanje virtualnog uređaja

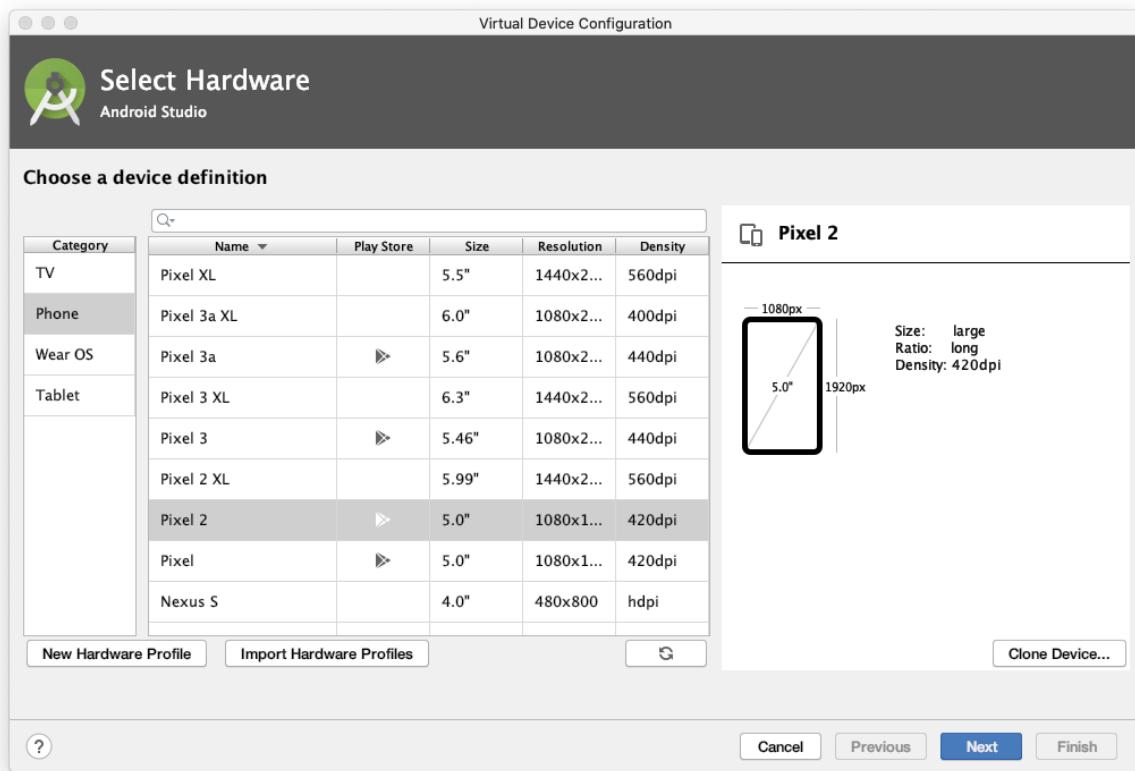
Za pokretanje aplikacija koje razvijamo potrebno je postaviti virtualan uređaj ili koristiti fizički uređaj kao domaćina za pokretanje aplikacije. U ovom poglavlju ćemo opisati postupak kreiranja i postavljanja virtualnog uređaja.

Prvi korak je, unutar Android Studio alata odabratи iz izborne trake Tools i zatim AVD Manager nakon čega dobivamo sljedeći prikaz.



Slika 20. AVD Manager

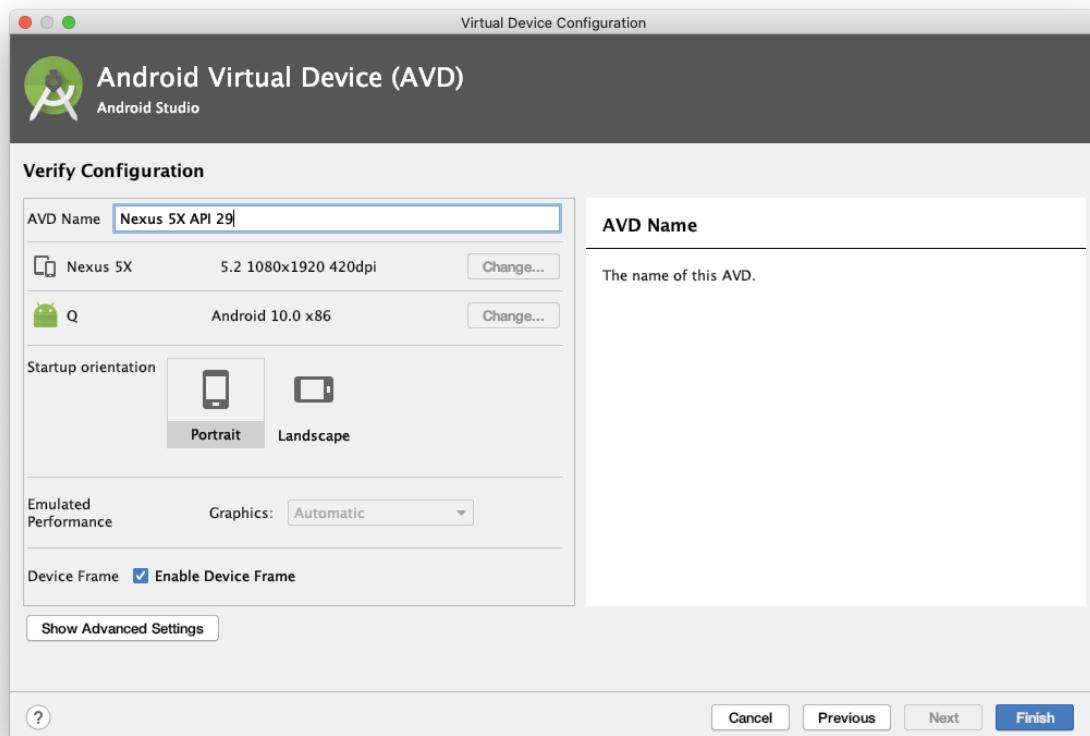
U prikazanom prozoru odabiremo opciju pri dnu „Create Virtual Device“. I nakon toga nam se otvara prozor sa svim mogućim uređajima koje možemo virtualizirati.



Slika 21. Odabir virtualnog uređaja

Mi ćemo u ovom slučaju odabrati Nexus 5X. Također je važno obratiti pozornost koji uređaj podržava Google Play Store, ukoliko nam je potreban. U slučaju da imate nešto slabije računalo dobro je proučiti uređaje i odabrati onaj koji je manje zahtjevan za pokretanje. Nakon što smo odabrali uređaj, prikazuje se prozor u kojem je potrebno odabrati System Image. Ukoliko se nalazite na linuxu najbolji odabir je x86 verzija određenog System Imagea. U slučaju da se nalazite na Windows ili Mac računalu i uspjeli ste aktivirati HAXM kao što je navedeno u prethodnim poglavljima onda je također x86 verzija optimalna. Ako HAXM nije aktivan morat ćete odabrati neku od drugih opcija kao što je armeabi-v7a.

Zadnji korak je dodijeljivanje naziva uređaju i odabiranje početne orijentacije. Ime proizvoljno dodijelite, a za orijentaciju je preporučeno ostaviti u portret prikazu. Postupak završavamo s klikom na gumb „Finish“.



Slika 22. Zaključivanje konfiguracije virtualnog uređaja

4.6. Kreiranje modula baze podataka

4.6.1. Firebase

Za kreiranje klase koje predstavljaju entitete baze podataka koristimo "data class" tip klase koju Kotlin koristi za klase koje služe samo za pohranu podataka. Kotlin za ovakve klase automatski generira equals() / hashCode(), toString(), copy() funkcije za svaki od atributa deklariranih u primarnom konstruktoru klase. Također, automatski generira i tzv. componentN() funkcije za dekonstruktiranje objekta u određeni broj varijabli. U nastavku ćemo objasniti kreiranje novog paketa (eng. package) i kreiranje data class objekata za sve entitete baze podataka.

Prvi korak jest kreiranje modula naziva "database". Desnim klikom odabiremo "app", te odabiremo New -> Modul, odabiremo "Android library". Nakon toga unosimo ime modula kojeg želimo kreirati, u našem slučaju "database", te odabiremo Ok. Nakon toga unutar kreiranog modula, kreiramo pakete "converters" "entities" i "views". U "entities" se nalaze klase entiteta

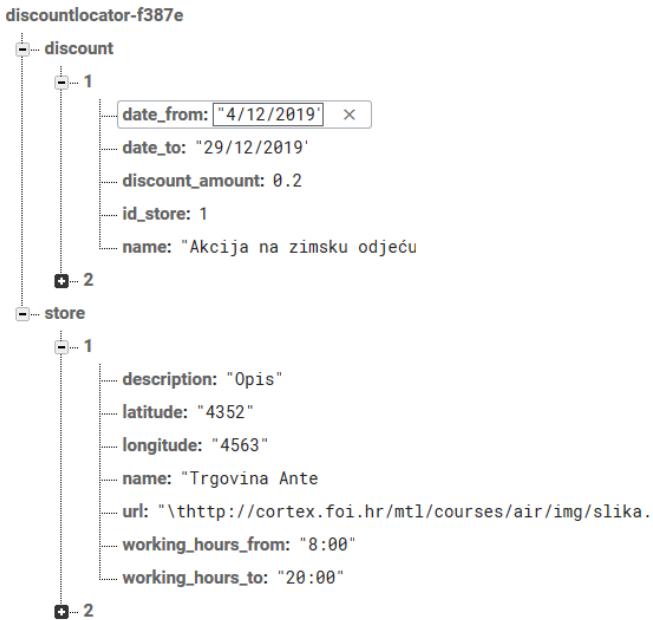
baze, dok se u "views" paketu nalaze razni pogledi koje dohvaćamo iz baze podataka. Isto tako, unutar "converters" paketa nalaze se konverteri podataka. Novi paket kreiramo desnim klikom na "database" i odabiremo New -> Package. Nakon kreiranja potrebnih paketa struktura projekta treba izgledati kao na slici ispod.

Slika 23. Struktura projekta

Nakon toga, potrebno je kreirati Firebase projekt na kojem ćemo kreirati bazu podataka. Na [službenoj stranici](#) Firebase-a prijavljujemo se Google računom, te kreiramo novi projekt. Nakon kreiranja novog projekta odabiremo "Add app" kao na slici 12. I dodajemo id aplikacije ili u našem slučaju modula, npr. com.companyname.firebaseio. Firebase tada kreira datoteku google-services.json koju je potrebno staviti u root direktorij modula unutar "Project" pogleda.

Slika 12. Dodavanje aplikacije u Firebase

Kako Firebase koristi bazu podataka koja se na bazira na tablicama već na strukturi stabla, korijen stabla predstavlja našu bazu podataka. Svako dijete korijena stabla je jedan entitet baze podataka, dok je svako dijete entiteta jedan unos tog entiteta u bazu podataka. Kreirana baza sa nekoliko unosa treba izgledati kao na slici ispod.



Slika 13. Firebase baza podataka

Poslijednji korak dodavanja Firebase-a u naš projekt jest dodavanje tzv. “dependencies” u gradle datoteku database modula, te u globalnu (Project: Discount Locator) gradle datoteku kako bi mogli koristiti Google Play Services. Unutar globalne gradle datoteke, pod “dependencies” dodajemo liniju koda:

```
classpath 'com.google.gms:google-services:4.3.3'
```

Nakon toga u gradle datoteku našeg modula dodajemo:

```
implementation 'com.google.firebaseio:firebase-auth:19.1.0'
implementation 'com.google.firebaseio:firebase-firebase:21.3.0'
implementation 'com.google.firebaseio:firebase-database:19.2.0'
apply plugin: 'com.google.gms.google-services'
```

Idući korak je kreiranje klase entiteta baze unutar naše Android aplikacije. Desnim klikom odabiremo paket “entities”, te odabiremo New -> Kotlin File/Class. Nakon toga unosimo ime klase, odnosno entiteta kojeg kreiramo i odabiremo Ok. U nastavku ćemo prikazati kod svake klase entiteta. Kako smo ranije napomenuli koristimo “Data Class” vrstu klase u Kotlinu, stoga će naše klase entiteta sadržavati samo atribute entiteta kojeg prestavljaju.

```
data class Store (
    var id_store: Int,
    var name: String,
    var address: String,
    var email: String,
    var phone_number: String,
    var latitude: String,
    var longitude: String,
    var working_hours_from: String,
    var working_hours_to: String,
    var description: String
)

data class Discount (
    var id_discount: Int,
    var discount_amount: Double,
```

```
        var date_from: String,  
        var date_to: String,  
        var id_offer: Int  
    )
```

Nakon što smo kreirali potrebne entitete, potrebno je kreirati klasu koja će izvoditi sve potrebne operacije sa bazom podataka. Unutar direktorija u kojem se nalaze naši paketi kreiramo klasu naziva “*MyDb*”, desni klik na paket com.companyname.database i odabiremo New -> Kotlin File/Class, dodajemo ime i odabiremo Ok. Prva stvar koju moramo napraviti jest konekcija na bazu. Firebase to izvršava pomoću reference (*eng. reference*) na trenutno stanje baze. Referencu na bazu spremamo u varijablu “*database*”, preko koje dohvaćamo sve elemente baze podataka. Referencu dohvaćamo na sljedeći način:

```
val database = FirebaseDatabase.getInstance().reference
```

Sada preko reference izvodimo sve operacije nad bazom, insert funkcije izvršavaju se preko "insertValue()" naredbe. U nastavku slijedi kod svih insert funkcija.

```
fun insertStore(store: Store){
    database.child("store").child(store.id_store.toString()).setValue(store)
}

fun insertDiscount(discount: Discount){
    database.child("discount").child(discount.id_discount.toString()).setValue(discount)
}
```

Brisanje elemenata izvodi se naredbom "removeValue()", kod svi funkcija brisanja elemenata nalazi se u nastavku.

```
fun deleteStore(store_id: Int){
    database.child("store").child(store_id.toString()).removeValue()
}

fun deleteDiscount(discount_id: Int){
    database.child("discount").child(discount_id.toString()).removeValue()
}
```

Ispis svih elemenata entiteta odvija se dodavanjem tzv. "ValueEventListener" objekta na entitet ili dijete stabla čije zapise želimo dohvatiti. Svakim dohvaćanjem zapisa potrebno je kreirati ValueEventListener, te ga dodati na željeno dijete stabla (entitet baze). Listener override-a dvije funkcije "onDataChange()", u kojem dohvaća podatke, te "onCancelled()" koju poziva u slučaju greške. U nastavku su prikazane funkcije dohvaćanja svih elemenata entiteta.

```
fun selectStore(){
    val listener = object : ValueEventListener {
        override fun onDataChange(dataSnapshot: DataSnapshot) {
            val post = dataSnapshot.value
            Log.i("querry", post.toString())
        }

        override fun onCancelled(databaseError: DatabaseError) {
            Log.e("error", "Getting data failed.")
        }
    }
    database.child("store").addValueEventListener(listener)
}

fun selectDiscount(){
    val listener = object : ValueEventListener {
        override fun onDataChange(dataSnapshot: DataSnapshot) {
            val post = dataSnapshot.value
            Log.i("querry", post.toString())
        }

        override fun onCancelled(databaseError: DatabaseError) {
            Log.e("error", "Getting data failed.")
        }
    }
}
```

```

        }
        database.child("discount").addValueEventListener(listener)
    }
}

```

4.6.2. SQLLite (Room)

Kako bismo spremali podatke sa webservisa u lokalnu bazu podataka na mobilnom uređaju koristit ćemo SQLLite bazu podataka, te biblioteku Room. Sada ćemo modificirati klase entiteta kako bi radile sa Room bibliotekom koja se zasniva na anotacijama. Međutim, prije rada sa Room bibliotekom prvo ju moramo uključiti u gradle našeg modula za bazu podataka. Popis svih potrebnih dependencija se nalazi u nastavku:

```

api           "androidx.room:room-runtime:$room_version"
kapt          "androidx.room:room-compiler:$room_version"
implementation "androidx.legacy:legacy-support-v4:1.0.0"
implementation 'androidx.lifecycle:lifecycle-extensions:2.2.0'
implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.2.0'

```

Anotacije koje će nam biti potrebne za prilagođavanje klasa entiteta za rad s bazom su `@Entity`, koji označava da neka klasa predstavlja jedan entitet baze, `@PrimaryKey` i `@ForeignKey` koji označavaju atribute koji predstavljaju primarni i strane ključeve entiteta, te koristimo još i `@ColumnInfo` kako bi dali dodatne informacije o atributu entiteta. U nastavku slijede sve modificirane klase entiteta.

```

//class Store
@Entity(tableName = "store")
data class Store (
    @PrimaryKey var id: Int,
    @ColumnInfo(name = "name") var name: String,
    @ColumnInfo(name = "description") var description: String,
    @ColumnInfo(name = "url") var url: String,
    @ColumnInfo(name = "latitude") var latitude: String,
    @ColumnInfo(name = "longitude") var longitude: String,
    @ColumnInfo(name = "working_hours_from") var working_hours_from: String,
    @ColumnInfo(name = "working_hours_to") var working_hours_to: String
)

//class Discount
@Entity(tableName = "discount")
data class Discount (
    @PrimaryKey var id: Int,
    ...
)

```

```

    @ColumnInfo(name      = "name")      var      name:      String,
    @ColumnInfo(name      = "start_date")  var      start_date:  String,
    @ColumnInfo(name      = "end_date")   var      end_date:   String,
    @ColumnInfo(name      = "discount")   var      discount:   Int,

    @ForeignKey(entity  = Store::class, parentColumns = ["id"],
    childColumns           = ["id_store"])
    @ColumnInfo(index          = true)
    var                  id_store:      Int
)

```

Sada kreiramo klasu “*DAO.kt*” (eng. “*Data Access Object*”) koja služi kao sučelje za pristup bazi podataka. Korištene su anotacije `@Dao`, te `@Querry`, `@Insert`, `@Update` i `@Delete` za definiranje operacija nad bazom podataka. U nastavku slijedi kod klase DAO.

```

@Dao
interface DAO {
    @Query("SELECT * FROM store")
    fun getStores(): List<Store>

    @Query("SELECT * FROM discount")
    fun getDiscounts(): List<Discount>

    @Insert
    fun insertStore(vararg new_store: Store)

    @Insert
    fun insertDiscount(vararg new_discount: Discount)

    @Delete
    fun deleteStore(store: Store)

    @Delete
    fun deleteDiscount(discount: Discount)

    @Update
    fun updateStore(vararg store: Store)

    @Update
    fun updateDiscount(vararg discount: Discount)
}

```

Sada kreiramo klasu “*MyDatabase.kt*” koja ima sličnu ulogu kao i “*MyDb.kt*” klasa. Pomoću klase MyDatabase kreiramo lokalnu bazu podataka, te dohvaćamo DAO. Pomoću anotacije @Database podešavamo podatke o bazi podataka, npr. koje entitete ona sadrži. U nastavku slijedi kod klase MyDatabase.

```
@Database(entities = [Store::class, Discount::class], version =
1,                                     exportSchema          =           false)
abstract      class      MyDatabase:      RoomDatabase()      {

    companion                           object{
        @Volatile  private  var  instance:  MyDatabase?  =  null
        private      val      LOCK       =           Any()

            operator fun  invoke(context:  Context)  =  instance  ?:

synchronized(LOCK){
            instance  ?:  buildDatabase(context).also{  instance  =
it
        }

            private  fun  buildDatabase(context:  Context)  =
Room.databaseBuilder(context,                               MyDatabase::class.java,
"localDatabase.db").build()
        }

    abstract      fun      getDAO():  DAO
}
```

4.7. Osnovni UI

U ovome poglavlju reći ću nešto više o radu s UI elementima. Neki od elemenata koje ću prikazati su TextView, Button, ListView i slično. Počnimo s MainActivity.kt datotekom, čija .xml datoteka sadrži jedan TextView i jedan Button kako je vidljivo na slici ispod.

```
<LinearLayout 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:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingStart="16dp"
    android:paddingTop="8dp"
    android:paddingEnd="16dp"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/txtListOfStores"
        style="@style/NameStyle"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="0"
        android:text="Show available stores"
        android:textAlignment="center"
        android:textSize="40sp" />

    <Button
        android:id="@+id/btnShow"
        style="@style/Widget.AppCompat.Button.Colored"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_weight="0"
        android:fontFamily="@font/roboto"
        android:onClick="showListView"
        android:text="Show" />
</LinearLayout>
```

Slika 24. activity_main.xml

Pritikom na Button koji smo definirali pokreće se funkcija *showListView*, koja pokreće novu aktivnost pod nazivom *DisplayListActivity*. [2] Ta aktivnost implementira klasu koju sam napravio pod nazivom *MyCustomAdapter*. Ta klasa je kreirana kako bi implementirali

ViewHolder uzorak koji preporučuje Google. [3], [4], [5] Kod klase MyCustomAdapter vidljiv je na slici 25.

```
class MyCustomAdapter : BaseAdapter() {
    private val storeNames : ArrayList<String> = arrayListOf("Varteks", "Super Nova", "Lidl")

    public fun getStoreNames(): ArrayList<String> {
        return storeNames
    }

    override fun getView(position: Int, convertView: View?, viewGroup: ViewGroup): View {
        val rowMain: View

        if (convertView == null) {
            val layoutInflater: LayoutInflater = LayoutInflater.from(viewGroup.context)
            rowMain = layoutInflater.inflate(R.layout.row_main, viewGroup, attachToRoot: false)

            val viewHolder = ViewHolder(rowMain.nameTextView, rowMain.positionTextView)
            rowMain.tag = viewHolder
        } else {
            rowMain = convertView
        }

        val viewHolder: ViewHolder = rowMain.tag as ViewHolder
        viewHolder.nameTextView.text = storeNames.get(position)
        val rowPosition: Int = position.inc()
        viewHolder.positionTextView.text = "Row number: $rowPosition"
        return rowMain
    }

    override fun getItem(position: Int): Any {
        return "TEST STRING"
    }

    override fun getItemId(position: Int): Long {
        return position.toLong()
    }

    override fun getCount(): Int {
        return storeNames.size
    }

    private class ViewHolder(val nameTextView: TextView, val positionTextView: TextView)
}
```

Slika 25. MyCustomAdapter.kt

Unutar te klase definirao sam privatnu varijablu *storeNames*, koja sadrži listu nekoliko trgovina za primjer podataka. Na idućoj slici prikazan je kod aktivnosti *DisplayListActivity*.

```
package com.air.fumic.maodus.persic.poljak.discountlocator

import android.os.Bundle
import android.widget.ListView
import androidx.appcompat.app.AppCompatActivity
import com.air.fumic.maodus.persic.poljak.discountlocator.adapters.MyCustomAdapter

class DisplayListActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_display_list)

        val listView: ListView! = findViewById<ListView>(R.id.list_view_Stores)
        listView.adapter = MyCustomAdapter()
    }
}
```

Slika 26. *DisplayListActivity.kt*

Funkcija *showListView*, koja se aktivira pritiskom na gumb, prikazuje sadržaj nove aktivnosti. Kod funkcije unutar *MainActivity* prikazan je na idućoj slici.

```

package com.air.fumic.maodus.persic.poljak.discountlocator

import android.content.Intent
import android.os.Bundle
import android.view.View
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }

    fun showListView(view: View) {
        val intent : Intent = Intent(packageContext: this, DisplayListActivity::class.java).also { it: Intent {
            startActivity(it)
        } }
    }
}

```

Slika 27. MainActivity.kt

Layout koji koristi aktivnost za prikaz ListView definiran je u *row_main.xml* datoteci.

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout 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:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="12dp">

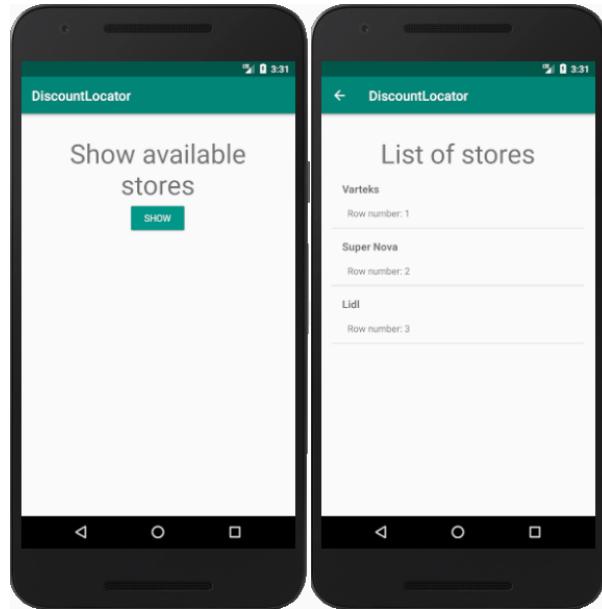
    <TextView
        android:id="@+id/stores_textview"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:paddingLeft="16dp"
        android:paddingTop="16dp"
        android:paddingRight="16dp"
        android:textSize="16sp"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:text="Trgovina" />

    <TextView
        android:id="@+id/position_textview"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="24dp"
        android:layout_marginTop="16dp"
        android:text="Row number:"
        android:textSize="14sp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/stores_textview" />
</androidx.constraintlayout.widget.ConstraintLayout>

```

Slika 28. row_main.xml

Izgled aktivnosti *MainActivity* i *DisplayListActivity* vidljivi su na slikama ispod. Kao što sam već rekao prije, pritiskom na gumb 'SHOW' otvara se *DisplayListActivity*.



Slika 29. Prikaz osnovnog UI

4.8. Navigacija

U ovome poglavlju prikazat ću kreiranje navigacijskih elemenata unutar Android-a, kao što su navigacijska ladica (eng. *Navigation Drawer*) i alatna traka (eng. *Toolbar*). Isto tako prikazat ću i rad s fragmentima, pošto ih koristimo unutar aplikacije. Prvo ću pokazati sve .xml datoteke koje koristimo za dizajniranje navigacijskih elemenata. Same elemente navigacijske ladice definirao sam unutar *drawer_menu.xml* datoteke.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    tools:showIn="navigation_view">

    <group android:checkableBehavior="single">
        <item
            android:id="@+id/nav_listView"
            android:icon="@drawable/ic_format_list_bulleted_black_24dp"
            android:title="List view" />
        <item
            android:id="@+id/nav_mapView"
            android:icon="@drawable/ic_map_black_24dp"
            android:title="Map view" />
    </group>

    <item android:title="Additional">
        <menu>
            <item
                android:id="@+id/nav_aboutApp"
                android:icon="@drawable/ic_info_outline_black_24dp"
                android:title="About app" />
        </menu>
    </item>
</menu>
```

Slika 30. drawer_menu.xml

Sadašnji prikaz glavne aktivnosti, definiran je u izmijenjenom *activity_main.xml* čiji kod možemo vidjeti na slici 31.

```
<?xml version="1.0" encoding="utf-8"?>

<layout 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">

    <androidx.drawerlayout.widget.DrawerLayout
        android:id="@+id/drawer_layout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:fitsSystemWindows="true"
        android:orientation="vertical"
        tools:context=".MainActivity"
        tools:openDrawer="start">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="vertical">

            <androidx.appcompat.widget.Toolbar
                android:id="@+id/toolbar"
                android:layout_width="match_parent"
                android:layout_height="?attr/actionBarSize"
                android:background="@color/colorPrimary"
                android:elevation="4dp"
                android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
                app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />

            <FrameLayout
                android:id="@+id/fragment_container"
                android:layout_width="match_parent"
                android:layout_height="match_parent" />

        </LinearLayout>

        <com.google.android.material.navigation.NavigationView
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_gravity="start"
            android:id="@+id/nav_view"
            app:headerLayout="@layout/nav_header"
            app:menu="@menu/drawer_menu" />
    </androidx.drawerlayout.widget.DrawerLayout>
</layout>
```

Slika 31. activity_main.xml

Dizajn alatne trake vrlo je jednostavan i definiran je u *options_menu.xml* datoteci.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:title="Settings"
        android:id="@+id/settings">

        </item>

    </menu>
```

Slika 32. options_menu.xml

Trenutno alatna traka sadrži opciju *Settings* koja otvara aktivnost za prikaz postavki aplikacije. Definiranje izgled postavki vidljivo je na slici ispod.

```
<PreferenceScreen xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <PreferenceCategory app:title="User settings">

        <EditTextPreference
            app:key="username"
            app:title="Username"
            app:useSimpleSummaryProvider="true" />

    </PreferenceCategory>

    <PreferenceCategory app:title="Application settings">

        <ListPreference
            app:defaultValue="system"
            app:entries="@array/language_entries"
            app:entryValues="@array/language_values"
            app:key="language"
            app:title="Choose language"
            app:useSimpleSummaryProvider="true" />

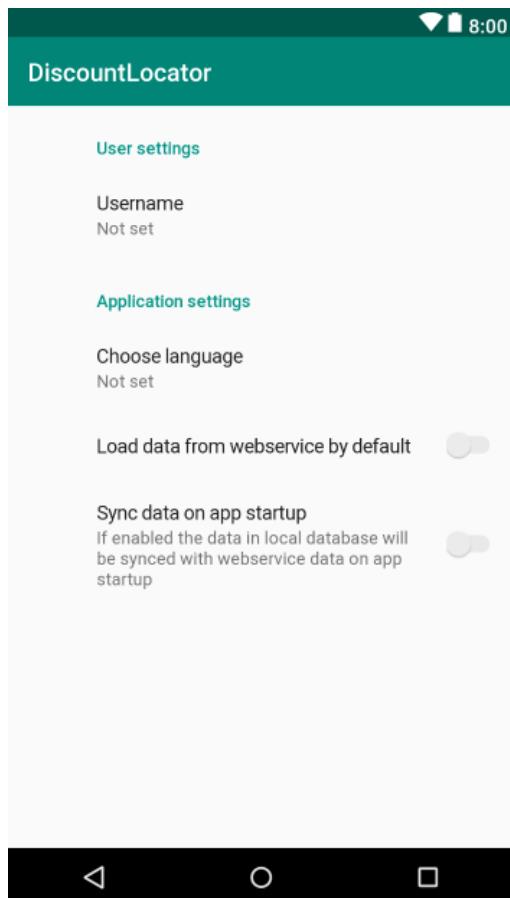
        <SwitchPreferenceCompat
            app:key="webservice"
            android:disableDependentsState="true"
            app:title="Load data from webservice by default" />

        <SwitchPreferenceCompat
            app:key="sync"
            app:dependency="webservice"
            app:title="Sync data on app startup"
            app:summary="If enabled the data in local database will be synced wi..." />

    </PreferenceCategory>
```

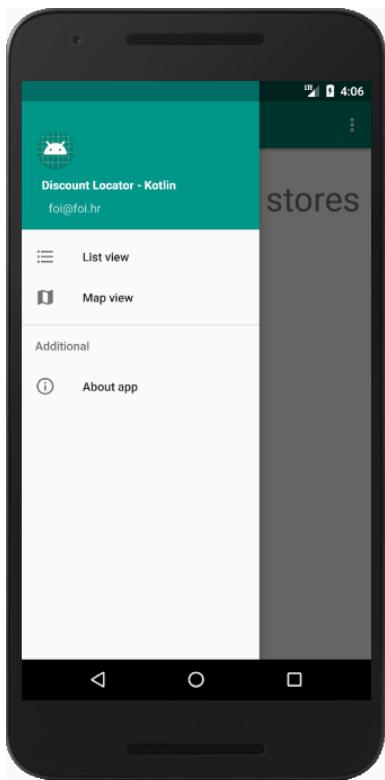
Slika 33. root_preference.xml

Izgled postavki možete vidjeti na slici 34.



Slika 34. Postavke aplikacije

Kada se aplikacija upali prvo se učitava fragment *ListViewFragment*, pritiskom na elemente navigacijske ladice pokreću se drugi odgovarajući fragmenti. [6] Fragmenti koje sam kreirao, osim već spomenutog, su *MapViewFragment* i *AboutAppFragment*.



Slika 35. Prikaz navigacijske ladice

```
 MainActivity.kt x
 27
 28 override fun onCreate(savedInstanceState: Bundle?) {
 29     super.onCreate(savedInstanceState)
 30     setContentView(R.layout.activity_main)
 31
 32     toolbar = findViewById(R.id.toolbar)
 33     setSupportActionBar(toolbar)
 34
 35     drawerLayout = findViewById(R.id.drawer_layout)
 36     navView = findViewById(R.id.nav_view)
 37
 38     val toogle = ActionBarDrawerToggle(
 39         activity: this, drawerLayout, toolbar,
 40         openDrawerContentDescRes: 0, closeDrawerContentDescRes: 0)
 41
 42     toogle.isDrawerIndicatorEnabled = true
 43     drawerLayout.addDrawerListener(toogle)
 44     toogle.syncState()
 45     nav_view.setNavigationItemSelectedListener(this)
 46
 47     notificationSetup()
 48
 49     //fragment code - ListViewFragment is the main fragment
 50     listViewFragment = ListViewFragment()
 51     supportFragmentManager.beginTransaction().replace(R.id.fragment_container, listViewFragment)
 52         .setTransition(FragmentTransaction.TRANSIT_FRAGMENT_OPEN).commit()
```

Slika 36. Kod navigacijske ladice unutar MainActivity

Kako bi pokrenuli određeni fragment pritiskom na elemente navigacijske ladice potrebno je nadglasati (eng. *override*) funkciju *onNavigationItemSelected(item: MenuItem)*. [7], [8]

```
override fun onNavigationItemSelected(item: MenuItem): Boolean {
    when (item.itemId) {
        R.id.nav_listView -> {
            listViewFragment = ListViewFragment()
            supportFragmentManager.beginTransaction()
                .replace(R.id.fragment_container, listViewFragment)
                .setTransition(FragmentTransaction.TRANSIT_FRAGMENT_OPEN).commit()
        }
        R.id.nav_mapView -> {
            mapViewFragment = MapViewFragment()
            supportFragmentManager.beginTransaction()
                .replace(R.id.fragment_container, mapViewFragment)
                .setTransition(FragmentTransaction.TRANSIT_FRAGMENT_OPEN).commit()
        }
        R.id.nav_aboutApp -> {
            aboutAppFragment = AboutAppFragment()
            supportFragmentManager.beginTransaction()
                .replace(R.id.fragment_container, aboutAppFragment)
                .setTransition(FragmentTransaction.TRANSIT_FRAGMENT_OPEN).commit()
        }
    }
    drawerLayout.closeDrawer(GravityCompat.START)
    return true
}
```

Slika 37. Funkcija za odabir elemenata navigacijske ladice

Pritiskom na *About app* prikazuje se fragment *AboutAppFragment*, a izgled možete vidjeti na slici ispod.



Slika 38. About app opcija

4.9. Dohvaćanje podataka s webservisa

Kao izvor podataka u ovom tutorialu koristit ćemo već gotov webservis koji se nalazi na sljedećem linku: <http://cortex.foi.hr/mlt/courses/air/>. Radi se o webservisu pruženom od strane Fakulteta organizacije i informatike koji će nam putem dvije stores.php i discounts.php skripti vraćati potrebne podatke za prikaz. Potrebno je uspostaviti komunikaciju s webservisom i preuzeti aktualne podatke koji se nalaze na istom.

U ovom primjeru ćemo koristiti retrofit REST klijent library za lakše dohvaćanje podatka s webservisa. Prvo što je potrebno je uključiti retrofit u naš projekt kako bi ga mogli koristiti u nastavku. Potrebno je pozicionirati se u build.gradle datoteku webservice modula i unutar dependencies zagrada unijeti linije označene na sljedećoj slici.

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
    implementation 'androidx.appcompat:appcompat:1.0.2'
    implementation 'androidx.core:core-ktx:1.0.2'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.0'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.1'

    implementation project(":core") //core module

    //Retrofit
    implementation 'com.squareup.retrofit2:retrofit:2.3.0'
    implementation 'com.squareup.retrofit2:converter-gson:2.3.0'
}
```

Slika 39. Uključivanje retrofita u projekt

S obzirom da podatke dohvaćamo s interneta potrebno je našem webservice modulu u manifestu dodijeliti prava na Internet.

```
1 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
2   package="com.air.fumic.maodus.persic.poljak.webservice">
3
4   <uses-permission android:name="android.permission.INTERNET"/>
5   <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
6
7 </manifest>
```

Slika 40. AndroidManifest.xml webservice modula

Idući korak je saznati kako izgleda odgovor webservisa u pitanju i prema tom odgovoru dodati klasu koja je model već navedenog odgovora i koja će se puniti podacima s webservisa. Klasa modelirana prema odgovoru webservisa se nalazi prikazana na sljedećoj slici.



```
1 package com.air.fumic.maodus.persic.poljak.webservice.responses
2
3 import com.google.gson.annotations.SerializedName
4
5 class MyWebserviceResponse {
6     @SerializedName( value: "responseId")
7     var responseId: Int = 0
8     @SerializedName( value: "responseText")
9     var responseText: String? = null
10    @SerializedName( value: "timeStamp")
11    var timeStamp: Int = 0
12    @SerializedName( value: "items")
13    var items: String? = null
14 }
```

Slika 41. MyWebserviceResponse klasa

Dodajemo gson tag @SerializedName zbog dobre prakse kako bi osigurali da znamo povezati podatke kod serijalizacije istih između JSON objekta i objekta koji će biti kreiran na temelju klase prikazane na prethodnoj slici.

Nakon toga je potrebno definirati sučelje koje će sadržavati informacije o tome kako trebaju izgledati naša dva POST zahtjeva i ciljane skripte koje će pružati povratne podatke. Također dodajemo retrofit tag @FormUrlEncode koji označava da će tijelo u zahtjevu koristiti kodiranje url forme. Također dodajemo @Field tag koji označava form-dana parametre koji će biti proslijedjeni unutar tijela zahtjeva u ovom slučaju se radi o metodi koju želimo proslijediti, odnosno onu metodu koju želimo da php skripta izvrši. Na sljedećoj slici se nalazi prikaz sučelja u pitanju.

```

1 package com.air.fumic.maodus.persic.poljak.webservice
2
3 import com.air.fumic.maodus.persic.poljak.webservice.responses.MyWebserviceResponse
4 import retrofit2.Call
5 import retrofit2.http.Field
6 import retrofit2.http.FormUrlEncoded
7 import retrofit2.http.POST
8
9
10 interface MyWebservice {
11     @FormUrlEncoded
12     @POST( value: "stores.php" )
13     fun getStores(@Field( value: "method" ) method: String): Call<MyWebserviceResponse>
14
15     @FormUrlEncoded
16     @POST( value: "discounts.php" )
17     fun getDiscounts(@Field( value: "method" ) method: String): Call<MyWebserviceResponse>
18 }

```

Slika 42. MyWebservice sučelje

Posljednja stvar koju ćemo napraviti u sklopu ovog webservice modula je klasa koja će upućivati zahtjeve prema webservisu i procesirati odgovore istog. Kreiramo novu klasu pod imenom MyWebserviceCaller i u prvo koraku kreiramo companion object koji će sadržavati temeljni url našeg webservisa i zadanu metodu. Companion object je singleton koji se koristi u Kotlinu kada želimo vezati funkciju ili svojstvo direktno za klasu a ne za pojedine instance dane klase. Funkcije i metode unutar companion objekta je moguće izravno pozvati preko naziva klase koja sadržava dani companion objekt.

```

86 companion object{
87     var BaseUrl = "http://cortex.foi.hr/mlt/courses/air/"
88     var Method = "getAll"
89 }
90

```

Slika 43. Companion object MyWebserviceCaller klase

Sljedeće dvije metode koje ćemo implementirati zadužene su za procesiranje vraćenog zahtjeva od strane webservisa i kao rezultat se dobiva polje trgovina ili popusta ovisno o tome koju metodu pozivamo. Ovdje koristimo Gson objekt koji unutar sebe sadrži potrebnu metodu za procesiranje JSON-a i izdvajanje zasebnih vrijednosti u njihove odgovarajuće varijable.

```

70     fun processStoresResponse(response: Response<MyWebserviceResponse>): Array<Store>{
71         var gson: Gson = Gson()
72         var allStores: Array<Store> = gson.fromJson(
73             response.body()!!.items, Array<Store>::class.java
74         )
75         return allStores
76     }
77
78     fun processDiscountsResponse(response: Response<MyWebserviceResponse>): Array<Discount>{
79         var gson: Gson = Gson()
80         var allDiscounts: Array<Discount> = gson.fromJson(
81             response.body()!!.items, Array<Discount>::class.java
82         )
83         return allDiscounts
84     }

```

Slika 44. Metode za procesiranje odgovora webservisa

Nakon uspješnih metoda za procesiranje, implementiramo metode koje su zadužene za dohvaćanje trgovina odnosno popusta. Unutar svake metode uz pomoć Retrofit buildera kreiramo retrofit objekte uz pomoć kojih ćemo uputiti zahtjev prema webservisu. Nakon toga kreiramo service na temelju MyWebservice sučelja. I na kraju call koji će u sebi sadržavati poziv koji želimo izvršiti prema webservisu. Na objektu call izvršavamo enqueue metodu koja asinkrono dohvaća podatke i na zaprimanju odgovora se poziva prethodno kreirana metoda za procesiranje odgovora. Zasada metode ne proslijeduju odgovor do sučelja.

```

42     public fun getDiscountsData(){
43         val retrofit = Retrofit.Builder()
44             .baseUrl(BaseUrl)
45             .addConverterFactory(GsonConverterFactory.create())
46             .build()
47
48         val service = retrofit.create(MyWebservice::class.java)
49         val call = service.getDiscounts(Method)
50
51         call.enqueue(object: Callback<MyWebserviceResponse>{
52             override fun onResponse(
53                 call: Call<MyWebserviceResponse>?,
54                 response: Response<MyWebserviceResponse>
55             ) {
56                 processDiscountsResponse(response)
57                 TODO( reason: "data needs to be passed on to UI")
58             }
59
60             override fun onFailure(call: Call<MyWebserviceResponse>?, t: Throwable?) {
61                 TODO( reason: "not implemented") //To change body of created functions use File | Settings | File Templates.
62             }
63         })
64     }

```

Slika 45. getDiscounts metoda unutar MyWebserviceCallera

4.10. Kreiranje core modula

Sada moramo kreirati novi modul na isti način kao i u prethodnom modulu za bazu podataka. Nazovimo modul “core”. Pomoću ovog modula spajamo i sinkroniziramo module kako bismo postigli potpunu modularnost. Modul se sastoji od dvije klase Store i Discount koje služe za lakše parsiranje podataka sa webservisa. Klase su atributima jednake kao i prošle Store i Discount klase. Nadalje, imamo DataPresenter koji služi kao sučelje za ostale module. Kod DataPresenter modula nalazi se u nastavku.

```
interface DataPresenter {  
    fun setData(stores: List<Store>, discounts: List<Discount>)  
}
```

Nadalje, sučelje DataLoader sadrži funkcije loadData() i isDataLoaded() koje služe za modularnost izvora podataka aplikacije. U nastavku se nalazi kod DataLoader sučelja.

```
interface DataLoader {  
    fun loadData(listener: DataLoadedListener)  
    fun isDataLoaded(): Boolean  
}
```

Na kraju, “core” modul se sastoji od DataLoadedListener sučelja koje kada se podaci učitaju inicijalizira potrebne liste podataka. Kod sučelja nalazi se u nastavku.

```
interface DataLoadedListener {  
    fun stores:  
        discounts:  
    )  
}
```

4.11. Kreiranje map view modula

Sada je vrijeme da kreiramo modul koji će, uz pomoć Google Maps API-a, prikazivati na karti markere za sve trgovine koje se nalaze u bazi podataka. Prvo kreiramo modul na način na koji smo to radili i do sada, te ga nazovemo “*map_view*”. Kako bismo mogli koristiti Google Maps API u našem novokreiranom modulu trebamo u njegovu gradle datoteku uključiti sljedeće dependencije.

```
implementation 'com.google.android.gms:play-services-maps:11.8.0'  
implementation 'com.google.android.gms:play-services-location:11.8.0'
```

Sada kreiramo aktivnost koja će prikazivati mapu, te markere na njoj. Unutar *map_view* paketa, u “java” folderu desnim klikom odabiremo paket te odabiremo New -> Activity -> Gallery. Sada nam se otvorila galerija svih vrsta aktivnosti koje možemo kreirati. Odabiremo Google Maps Activity, te je imenujemo “*MapsModule*”. Android Studio sada sam generira osnovne funkcije *OnCreate()*, što je osnovna funkcija svake aktivnosti i *OnMapReady()* koja upravlja mapom nakon što ona postane dostupna. Također, aktivnost implementira DataPresenter, te stoga nadjačava funkciju *setData* i pridružuje primljene vrijednosti privatnim varijablama.. Kod funkcije *OnMapReady()* i funkcije *setData()* nalazi se u nastavku.

```
override fun onMapReady(googleMap: GoogleMap) {  
    mMap = googleMap  
  
    // Add a markers for Mock Data and move the camera  
    val kapucinski = LatLng(46.305173, 16.336326)  
    mMap.addMarker(MarkerOptions().position(kapucinski).title("Kapucinski  
trg"))  
  
    val angelus = LatLng(46.307349, 16.339824)  
    mMap.addMarker(MarkerOptions().position(angelus).title("Angelus"))  
  
    setMarkers()  
    mMap.moveCamera(CameraUpdateFactory.newLatLng(kapucinski))  
    mMap.setMinZoomPreference(12.0f)  
}  
  
override fun setData(  
    stores: List<Store>,  
    discounts: List<Discount>  
) {
```

```

        this.stores = stores
        this.discounts = discounts
    }
}

```

Vidimo kako unutar OnMapReady() imamo i poziv setMarkers() funkcije. Ona dohvaća sve trgovine sa webservisa, te kreira za njih markere na karti. Nakon poziva setMarkers() funkcije vidimo podešavanje početne pozicije kamere, te podešavanje količine zoom-a. Kod setMarkers() funkcije nalazi se u nastavku.

```

fun setMarkers(){
    if (stores != null){
        for (store: Store in stores!!){
            var marker = LatLng(store.latitude.toDouble(),
                store.longitude.toDouble())
            mMap.addMarker(MarkerOptions().position(marker).title(store.name))
        }
    }else Log.i("stores", "Stores is null!")
}

```

Završeni map view modul bi trebao izgledati kao na slici ispod.



Slika 46. Map view

4.12. RecyclerView

RecyclerView je naprednija i fleksibilnija verzija ListView i GridView elemenata. Koristi se kod prikaza velikog broja podataka na način da će prikaz podataka biti skrolabilan a sami podaci neće zatrpati memoriju zato što se koristi takozvana kružna dodjela memorije. Ovaj način dodijele memorije koristi unaprijed zadan broj predmeta liste u memoriji koji će se dodjeljivati na način da će se u memoriji spremati samo oni elementi liste koji su trenutno prikazani na zaslonu i nekoliko elemenata ispred i nekoliko elemenata iza prikazanih. Kada korisnik pokuša skrolati tako će se prva zauzeta mjesta prazniti i dodjeljivati novim elementima koji će puniti listu.

Prije nego što počnemo raditi potrebno je uključiti potrebne pakete u naš projekt. Kao što je prikazano na sljedećoj slici. Dodajemo cardview koji će nam služiti za takozvani kartični prikaz, recyclerview je naravno za sam.recyclerview prikaz i na kraju glide koji će nam služiti za prikaz slika.

```
40 // Card View
41 def cardview_version = "1.0.0"
42 implementation "androidx.cardview:cardview:$cardview_version"
43
44 // Recyclerview
45 def recyclerview_version = "1.1.0"
46 implementation "androidx.recyclerview:recyclerview:$recyclerview_version"
47
48 //glide
49 def glide_version = "4.8.0"
50 implementation "com.github.bumptech.glide:glide:$glide_version"
51 annotationProcessor "com.github.bumptech.glide:compiler:$glide_version"
```

Slika 47. Potrebni paketi za kompletan RecyclerView

Nakon što smo uključili potrebne pakete, prvo što je potrebno je definirati layout za pojedini element liste. To ćemo napraviti na način da ćemo dodati Constraint layout koji će unutar sebe sadržavati jedan ImageView i jedan LinearLayout koji u sebi sadrži dva TextView elementa. ImageView će sadržavati sliku trgovine, a dva TextView elementa će sadržavati naziv trgovine i njen opis. Nakon toga u activity_main layout dodajemo RecyclerView element.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res-auto"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <androidx.recyclerview.widget.RecyclerView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/recycler_view"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Slika 48. RecyclerView widget

Unutar core modula nam se već nalaze podatkovne klase za trgovine i popuste prema kojima će se modelirati naši podaci unutar RecyclerViewa. Sljedeći korak je napraviti prilagođeni adapter za naš RecyclerView.

Pravimo adapter koji će raditi s podacima trgovina. Nazvat ćemo ga StoreRecyclerAdapter i on će nasljeđivati RecyclerView.Adapter<RecyclerView.ViewHolder>. Potrebne metode su onCreateViewHolder, onBindViewHolder i getItemCount. Prvo ćemo implementirati najlakšu metodu a to je getItemCount gdje ćemo samo vratiti broj elemenata u listi.

```
override fun getItemCount(): Int {
    return items.size
}
```

Slika 49. Metoda getItemCount u adapteru za RecycleView

Nakon toga mora kreirati prilagođenu ViewHolder klasu koja će opisivati kako će pogledi izgledati u našem RecyclerView elementu. Nazvat ćemo navedenu klasu StoreViewHolder i ona će primati unutar svog konstruktora View, a nasljeđivati će RecyclerView.ViewHolder klasu. Vrijednosti primljenog View elementa ćemo povezati s lokalnim varijablama i nakon toga povezati pojedinačne trgovine s View elementom. I na kraju koristimo glide paket kako bi postavili sliku. Kreiramo osnovne opcije zahtjeva i spremamo ih unutar requestOptions objekta. Nakon toga dodjeljujemo kreirane opcije zahtjeva glide objektu i učitavamo sliku. Prilagođena ViewHolder klasa je prikazana u nastavku.

```

40     class StoreViewHolder constructor(
41         itemView: View
42     ): RecyclerView.ViewHolder(itemView){
43         //itemView.ID se odnosi na layout_recycler_list_item
44         val storeImage: ImageView = itemView.store_image
45         val storeName: TextView = itemView.store_name
46         val storeDescription: TextView = itemView.store_description
47
48         fun bind(store: Store){
49             storeName.setText(store.name)
50             storeDescription.setText(store.description)
51
52             val requestOptions = RequestOptions()
53                 .placeholder(R.drawable.ic_launcher_background)
54                 .error(R.drawable.ic_launcher_background)
55
56             Glide.with(itemView.context)
57                 .applyDefaultRequestOptions(requestOptions)
58                 .load(store.imgUrl)
59                 .into(storeImage)
60
61     }

```

Slika 50. Prilagođen ViewHolder

Sada kada smo gotovi s prilagođenim ViewHolder elementom sve što je potrebno napraviti je kratko implementirati preostale metode, a to su `onCreateViewHolder`, `onBindViewHolder` i `submitList`. Metoda `submitList` je zadužena za prihvaćanje liste trgovina koje će se prikazati u `RecyclerView` elementu.

```

15     private var items: List<Store> = ArrayList()
16
17     override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecyclerView.ViewHolder {
18         return StoreViewHolder(
19             LayoutInflater.from(parent.context).inflate(R.layout.layout_recycler_list_item, parent,
20             false)
21         )
22
23     override fun onBindViewHolder(holder: RecyclerView.ViewHolder, position: Int) {
24         when(holder){
25             is StoreViewHolder ->
26                 holder.bind(items.get(position))
27         }
28     }
29
30
31     fun submitList(storeList: List<Store>){
32         items = storeList
33     }

```

Slika 51. Preostale metode `RecyclerView` elementa

Zadnja stvar koju je potrebno napraviti je inicijalizirati RecyclerView unutar MainActivity klase i napuniti ga podacima s webservisa.

```
48     private fun initializeRecyclerView(dataList: List<Store>){  
49         recycler_view.apply { this: RecyclerView!  
50             layoutManager = LinearLayoutManager( context: this@MainActivity)  
51             val topSpacingItemDecoration = TopSpacingItemDecoration( padding: 30)  
52             addItemDecoration(topSpacingItemDecoration)  
53             storeAdapter = StoreRecyclerAdapter()  
54             storeAdapter.submitList(dataList)  
55             recycler_view.adapter = storeAdapter  
56         }  
57     }
```

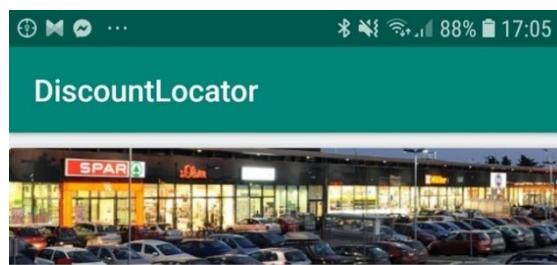
Slika 52. Inicijalizacija RecyclerView elementa

Ovdje ćemo još spomenuti da koristimo dodatan prilagođen TopSpacingItemDecoration kako bi pojedinačne kartice bile međusobno odvojene u RecyclerView prikazu.

```
7     class TopSpacingItemDecoration (private val padding: Int): RecyclerView.ItemDecoration(){  
8         override fun getItemOffsets(  
9             outRect: Rect,  
10            view: View,  
11            parent: RecyclerView,  
12            state: RecyclerView.State  
13        ) {  
14             super.getItemOffsets(outRect, view, parent, state)  
15             outRect.top = padding  
16         }  
17     }
```

Slika 53. TopSpacingItemDecoration klasa

Na kraju je samo potrebno poslati listu trgovina dohvaćenih s webservisa kao argument metode initializeRecyclerView jednom kada su podaci dohvaćeni i dobit ćemo uspješan RecycleView prikaz trgovina.



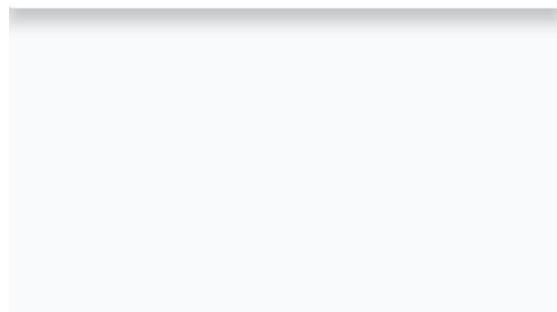
Super Nova

Veliko blagdansko sniženje u odabranim dućanima
u centru Super Nova.



Varteks

Popust na sve kolekcije ljetne odjeće u Varteks
Outlet dućanu.



Slika 54. Dovršen RecycleView prikaz

4.13. Push notifikacije

Push notifikacije su obavijesti koje korisnik dobiva od administratora aplikacije kada se određeni proizvod nalazi na sniženju ili kada određena trgovina ima velika sniženja. Korisniku na smartphone dolazi poruka o obavijesti s naslovom i kratkim opisom povodom čega mu je obavijest poslana. Za implementaciju ovih obavijesti koristi se Firebase-ova platforma za slanje besplatnih obavijesti, a zove se FCM(Firebase Cloud Messaging).

Kako bi ove obavijesti normalno radile potrebno je postaviti određene postavke u Android Studiu. U gradle skripti app modula potrebno je dodati ovisnost(eng. *Dependency*) za korištenje FCM-a te ovisnost za korištenje google-play servisa u gradle skripti projekt modula, što je prikazano na slikama ispod.

```
dependencies {  
    implementation fileTree(dir: 'libs', include: ['*.jar'])  
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"  
    implementation 'androidx.appcompat:appcompat:1.1.0'  
    implementation 'androidx.core:core-ktx:1.1.0'  
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'  
    testImplementation 'junit:junit:4.12'  
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'  
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'  
  
    implementation 'com.google.firebaseio:firebase-messaging:20.1.0'  
    implementation 'com.google.android.gms:play-services-ads:18.3.0'  
  
    implementation project(":database") //database module  
    implementation project(":core")  
    implementation 'androidx.recyclerview:recyclerview:1.1.0'  
    implementation 'androidx.cardview:cardview:1.0.0'  
    implementation 'androidx.navigation:navigation-fragment-ktx:2.2.0'  
    implementation 'androidx.navigation:navigation-ui-ktx:2.2.0' //core module
```

Slika 55. Ovisnosti potrebne za korištenje Firebase servisa za poruke

```

}buildscript {
    ext {
        ext.kotlin_version = '1.3.61'
    }
    repositories {
        google()
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.5.3'
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
        classpath 'com.google.gms:google-services:4.3.3'
        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        google()
        jcenter()
    }
}

```

Slika 56. Ovisnosti potrebne za korištenje Google Play servisa

Zatim je potrebno dodati u AndroidManifest.xml servis koji omogućuje sve radnje s porukama koristeći Firebase, a to je prikazano na slici ispod.

```

<service
    android:name=".notification.MessagingService"
    android:exported="false">
    <intent-filter>
        <action android:name="com.google.firebaseio.MESSAGING_EVENT" />
    </intent-filter>
</service>

```

Slika 57. Servis u AndroidManifest.xml koji omogućuje sve radnje s porukama

Nakon toga je potrebno dohvatiti trenutni registracijski token aplikacije koji se stvara kod slanja poruke, a to se izvede tako da se napravi funkcija koja će dohvatiti taj token. Funkcija je prikazana na slici ispod.

```
fun notificationSetup() {
    FirebaseMessaging.getInstance().isAutoInitEnabled = true
    FirebaseInstanceId.getInstance().instanceId
        .addOnCompleteListener(OnCompleteListener { task ->
            if (!task.isSuccessful) {
                return@OnCompleteListener
            }
            val token = task.result?.token
            //Toast.makeText(baseContext, token, Toast.LENGTH_SHORT).show()
        })
}
```

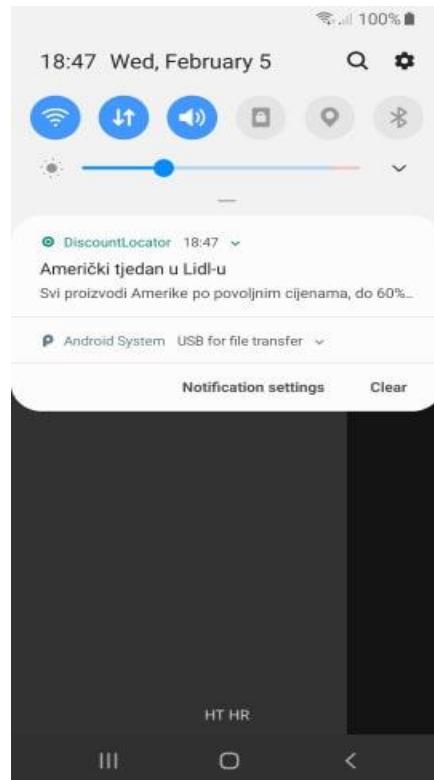
Slika 58. Funkcija koja dohvaća token aplikacije za slanje poruka

Poruke se kreiraju u Firebase-ovom kontrolnom sučelju za slanje poruka. Aplikacija je spojena s Firebaseom te se nakon sastavljanja poruke ona šalje svim korisnicima aplikacije. Izgled sučelja prikazan je na sljedećoj slici.

							Create experiment	New notification
	Notification	Status	Platform	Start / Send	End	Sends	Opens	
▶	Američki tjedan u Lidl-u Svi proizvodi Amerike po povoljn... <small>...</small>	✓ Completed	tv	Feb 5, 2020 6:47 PM	—	<1000	0%	
▶	Veliki popusti u Lidl-u	✓ Completed	tv	Feb 3, 2020 5:56 PM	—	<1000	0%	
▶	Vrući popusti u Zari -30% na sve	✓ Completed	tv	Feb 3, 2020 5:51 PM	—	<1000	0%	
▶	Test	✓ Completed	tv	Feb 3, 2020 2:01 PM	—	<1000	0%	
▶	TEST TEST	✓ Completed	tv	Feb 3, 2020 1:48 PM	—	<1000	0%	
▶	DiscountLocator Vruci popusti u Zari - 30%	✓ Completed	tv	Feb 3, 2020 1:42 PM	—	<1000	0%	
▶	DiscountLocator Vruci popusti u Zari - 30%	✓ Completed	tv	Feb 3, 2020 1:40 PM	—	<1000	0%	
▶	TEST TEST	✓ Completed	tv	Jan 30, 2020 7:27 PM	—	<1000	0%	
▶	1 1	✓ Completed	tv	Jan 30, 2020 7:07 PM	—	<1000	0%	⋮
▶	TEST TEST	✓ Completed	tv	Jan 30, 2020 6:28 PM	—	<1000	0%	
⌚ 0/10 Recurring notifications <small>?</small>								

Slika 59. Prikaz Firebase sučelja za slanje poruka

Konkretan izgled obavijesti je prikazan na slici ispod.



Slika 60. Prikaz push notifikacija
na smartphone-u

4.14. Google Ads

Google Ads su reklame koje se prikazuju korisnicima aplikacije te im „smetaju“. Oglašavanjem je moguće zaraditi od strane vlasnika oglasa jer svaki put kada korisnik pritisne na reklamu određeni dio novca vlasnik oglasa isplaćuje vlasniku aplikacije.

Za implementaciju oglasa koristio se Google AdMob. Prvo je potrebno postaviti ovisnost o Google servisima za reklame u gradle skriptu aplikacijskog modula.

```
|dependencies {  
    implementation fileTree(dir: 'libs', include: ['*.jar'])  
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"  
    implementation 'androidx.appcompat:appcompat:1.1.0'  
    implementation 'androidx.core:core-ktx:1.1.0'  
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'  
    testImplementation 'junit:junit:4.12'  
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'  
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'  
  
    implementation 'com.google.firebaseio:firebase-messaging:20.1.0'  
    implementation 'com.google.android.gms:play-services-ads:18.3.0'  
  
    implementation project(":database") //database module  
    implementation project(":core")  
    implementation 'androidx.recyclerview:recyclerview:1.1.0'|  
    implementation 'androidx.cardview:cardview:1.0.0'  
    implementation 'androidx.navigation:navigation-fragment-ktx:2.2.0'  
    implementation 'androidx.navigation:navigation-ui-ktx:2.2.0'//core module
```

Slika 61. Dodavanje ovisnosti za Google reklame

Zatim je potrebno dodati ID aplikacije na kojoj će se prikazivati reklame u AndroidManifest.xml. To je prikazano na slici ispod.

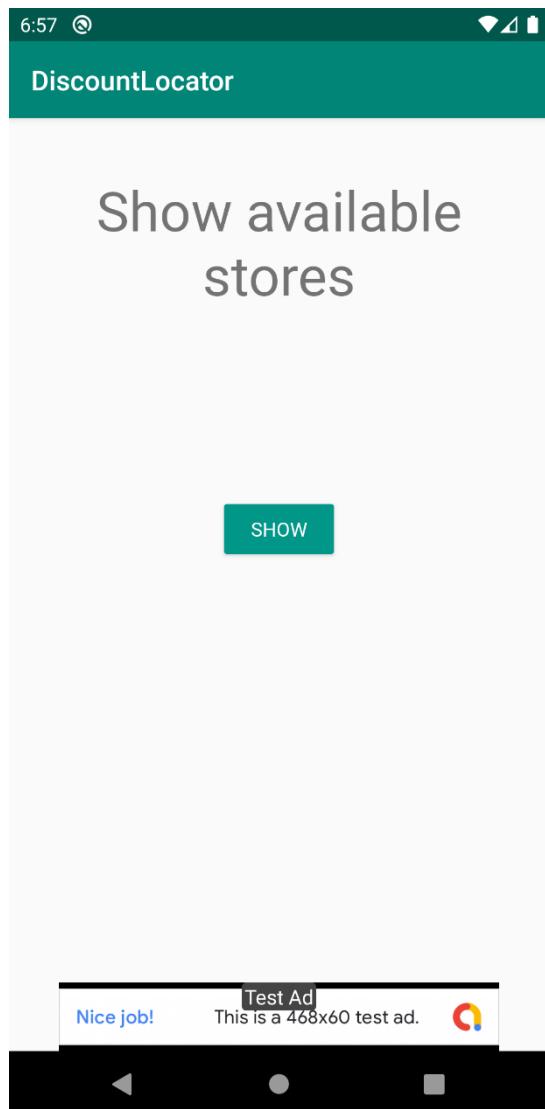
```
<meta-data  
    android:name="com.google.android.gms.ads.APPLICATION_ID"  
    android:value="ca-app-pub-1953989624902260~4998986487" />
```

Slika 62. Dodavanje ID-a aplikacije u AndroidManifest.xml

Nakon što je ID postavljen u aktivnost je potrebno dodati metodu za inicijalizaciju reklama, u ovom slučaju je dodana na događaj onCreate u glavnoj aktivnosti(MainActivity.kt) pa se prilikom pokretanja aplikacije odmah postavlja reklama na ekranu korisnika. Slike s metodom te izgledom reklame na ekranu korisnika se nalaze u nastavku.

```
MobileAds.initialize(this, "ca-app-pub-1953989624902260~4998986487")
val adRequest = AdRequest.Builder().build()
adView.loadAd(adRequest)
```

Slika 63. Metoda za incijalizaciju reklama

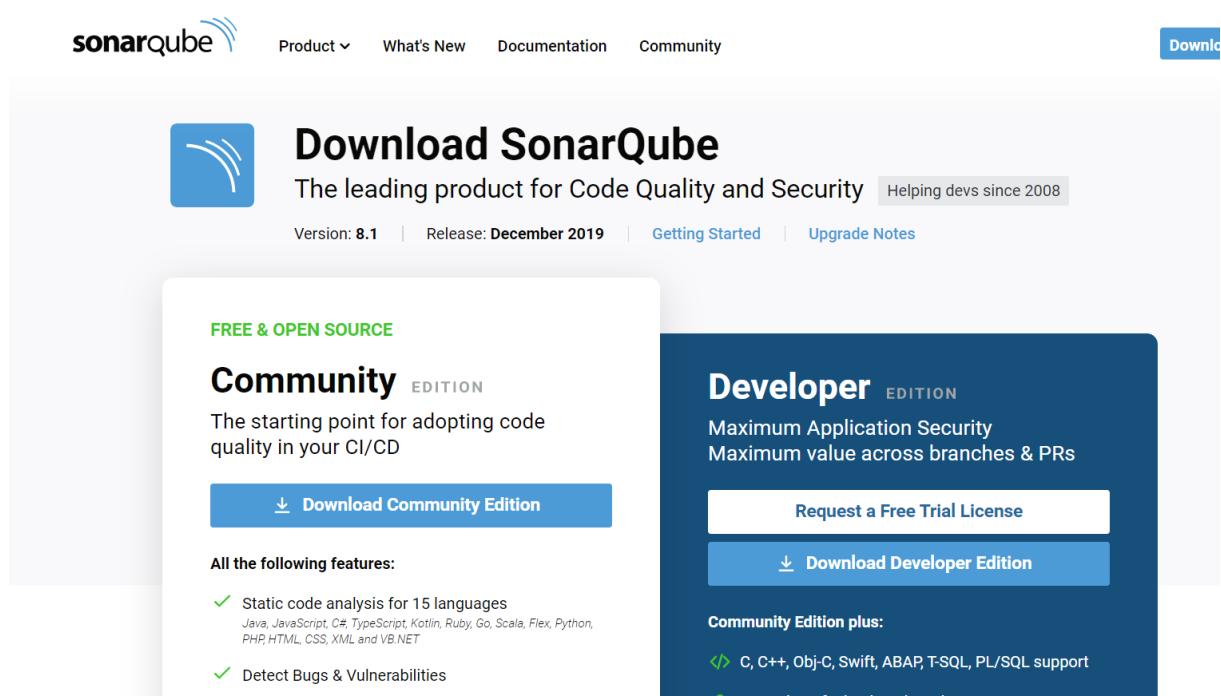


Slika 64. Prikaz ekrana s reklamom

4.15. Testiranje i analiza koda

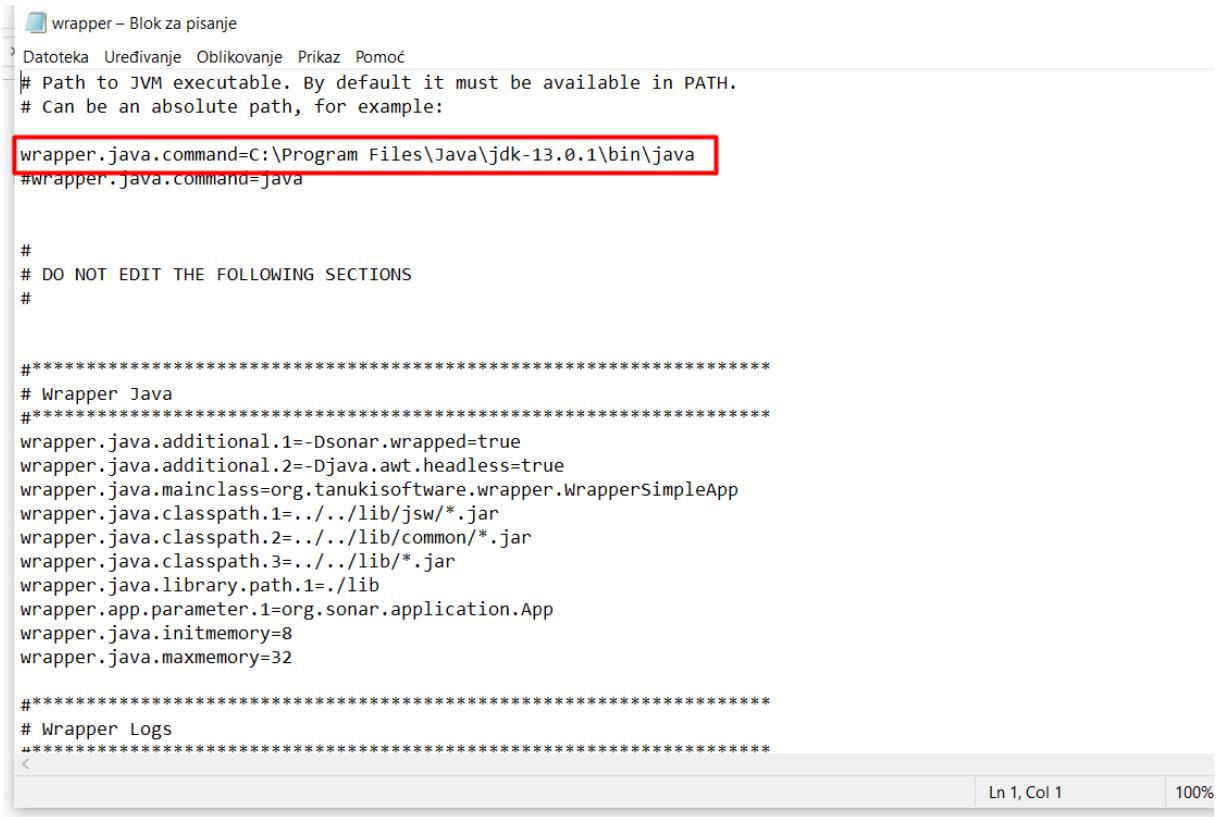
Prilikom razvoja aplikacije potrebno je s vremena na vrijeme testirati ko te provesti analizu koda. Time se može utvrditi kvaliteta koda, odnosno, koliko je kod dobro napisan, ima li u kodu dijelova koji su nebitni ili se ne koriste, jesu li možda krivo napisani ili se ponavljaju, ima li možda nekih dijelova koji nedostaju i slično. To je vrlo važno provoditi, naročito prije nego što se sama aplikacija pusti na korištenje korisnicima jer je važno da svaki test dobro prođe kako se ne bi događale neke neobične stvari s aplikacijom kasnije. Ovdje je korišten alat SonarQube za provođenje analize i testiranja koda. To je besplatan alat koji se može podešiti da podatke sprema lokalno ili na nekom serveru te se uz njega koristi najčešće sonar-scanner alat koji omogućuje skeniranje koda, odnosno pronalaženje loših i neobičnih stvari u kodu. U nastavku se nalaze slike s opisima kako i što treba podešiti za analizu kodova, a te slike zapravo prate tutorial dokumentacije sa službene stranice SonarQube-a.

Prvo je potrebno preuzeti i spremiti sami SonarQube client. To se napravi na službenoj stranici SonarQube-a kao što je prikazano na slici ispod.



Slika 65. Preuzimanje SonarQube alata

Nakon što je alat preuzet potrebno ga je otpakirati(unzip) bilo gdje na računalu te podešiti konfiguracijske datoteke koje se nalaze u datoteci „conf“. Potrebno je u konfiguracijskoj datoteci „wrapper“ postaviti wrapper.java.command na putanju do mape gdje je instaliran Java Developement Kit (JDK). To se može vidjeti na slici ispod.



```

wrapper - Blok za pisanje
Datoteka Uređivanje Oblikovanje Prikaz Pomoć
# Path to JVM executable. By default it must be available in PATH.
# Can be an absolute path, for example:
wrapper.java.command=C:\Program Files\Java\jdk-13.0.1\bin\java
#wrapper.java.command=java

#
# DO NOT EDIT THE FOLLOWING SECTIONS
#


*****#
# Wrapper Java
*****#
wrapper.java.additional.1=-Dsonar.wrapped=true
wrapper.java.additional.2=-Djava.awt.headless=true
wrapper.java.mainclass=org.sonar.application.App
wrapper.java.classpath.1=../../lib/jsw/*.jar
wrapper.java.classpath.2=../../lib/common/*.jar
wrapper.java.classpath.3=../../lib/*.jar
wrapper.java.library.path.1=/lib
wrapper.app.parameter.1=org.sonar.application.App
wrapper.java.initmemory=8
wrapper.java.maxmemory=32

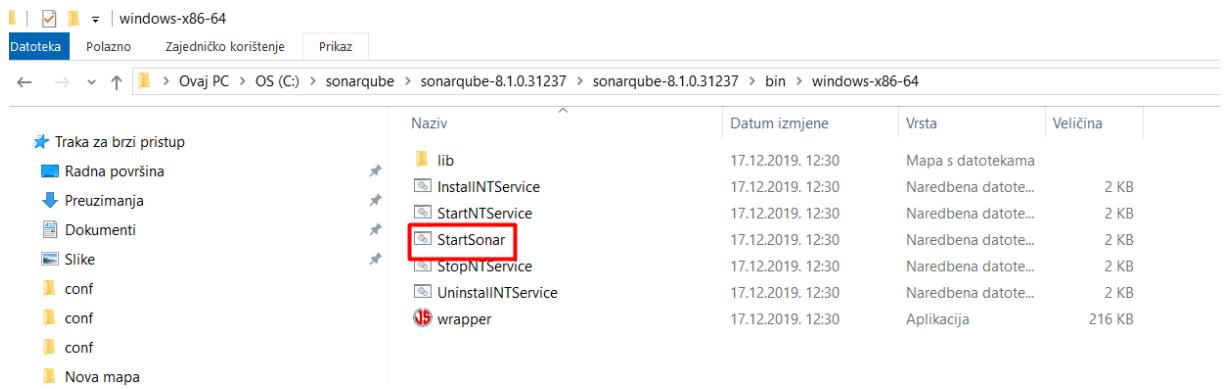
*****#
# Wrapper Logs
*****#
<

```

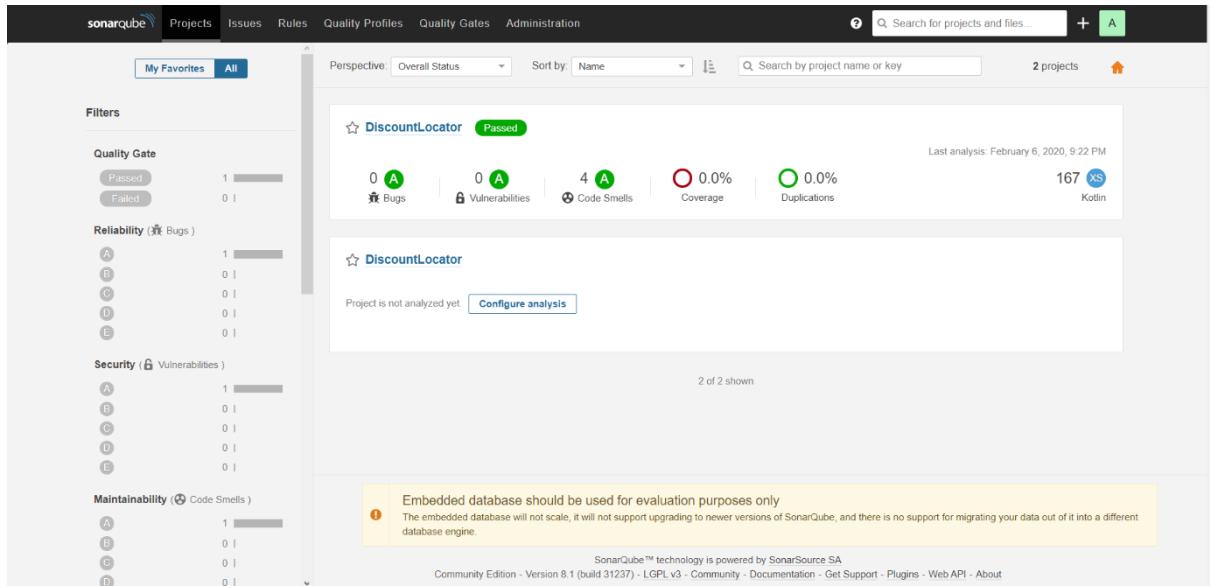
Ln 1, Col 1 100%

Slika 66. Uređivanje konfiguracijske datoteke wrapper

Nakon toga se može pokrenuti SonarQube tako da se iz bin datoteke pokrene StartSonar.bat datoteka te se pričeka dok se ne izvrše sve naredbe i nakon toga se otvori <http://localhost:9000> pa se otvorí lokalni klijent gdje će se moći vidjeti sve analize i testovi kodova. Na slici ispod je prikazano pokretanje te izgled SonarQube-a na web-u.

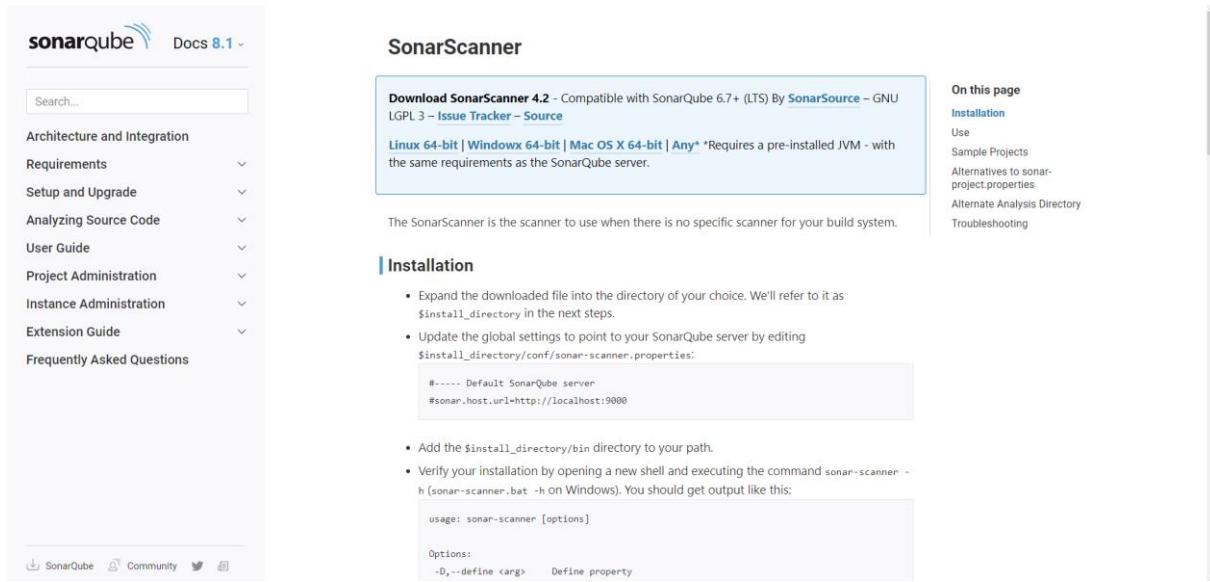


Slika 67. Pokretanje SonarQube-a

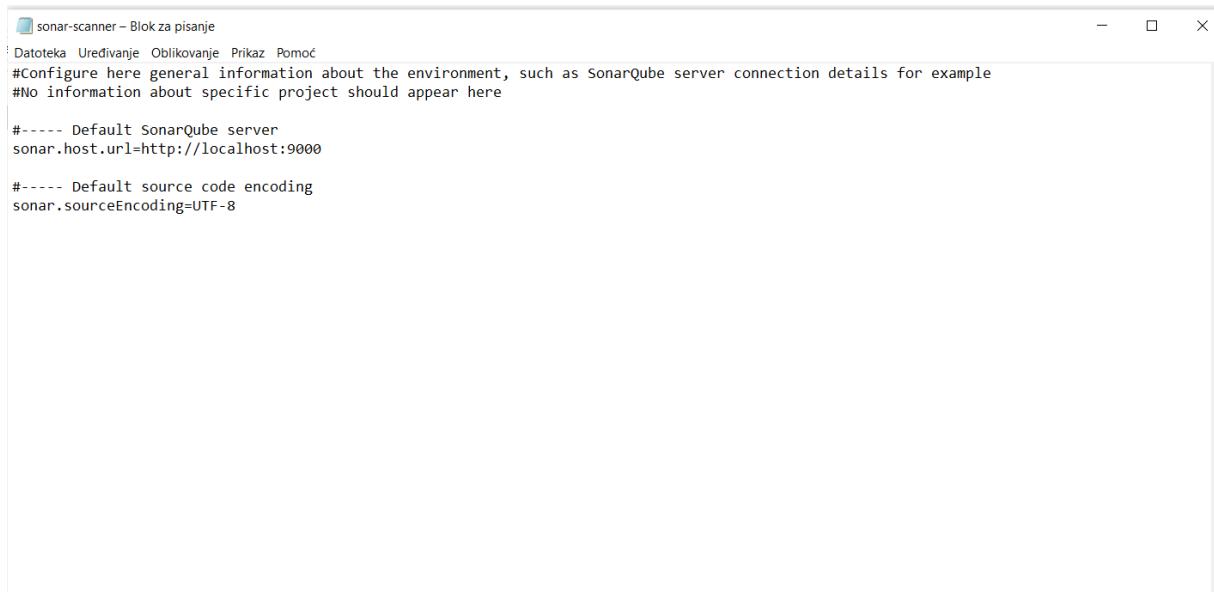


Slika 68. Izgled SonarQube-a na web-u

Sada je potrebno preuzeti i podesiti Sonar-scanner s kojim ćemo pokrenuti sami proces analize, odnosno testiranja koda. Preuzima se na službenim stranicama SonarQube-a te se također dobije .zip datoteka koju je potrebno negdje na računalu otpakirati. Kada se to učini potrebno je ući u konfiguracijsku datoteku i urediti ju, odnosno, samo maknuti znak „#“ ispred dvije vrijednosti koje su već upisane u konfiguracijskoj datoteci. Na slikama u nastavku je to prikazano.



Slika 69. Stranica za preuzimanje SonaScanner alata



```
sonar-scanner - Blok za pisanje
Dototeka Uređivanje Oblikovanje Prikaz Pomoć
#Configure here general information about the environment, such as SonarQube server connection details for example
#No information about specific project should appear here

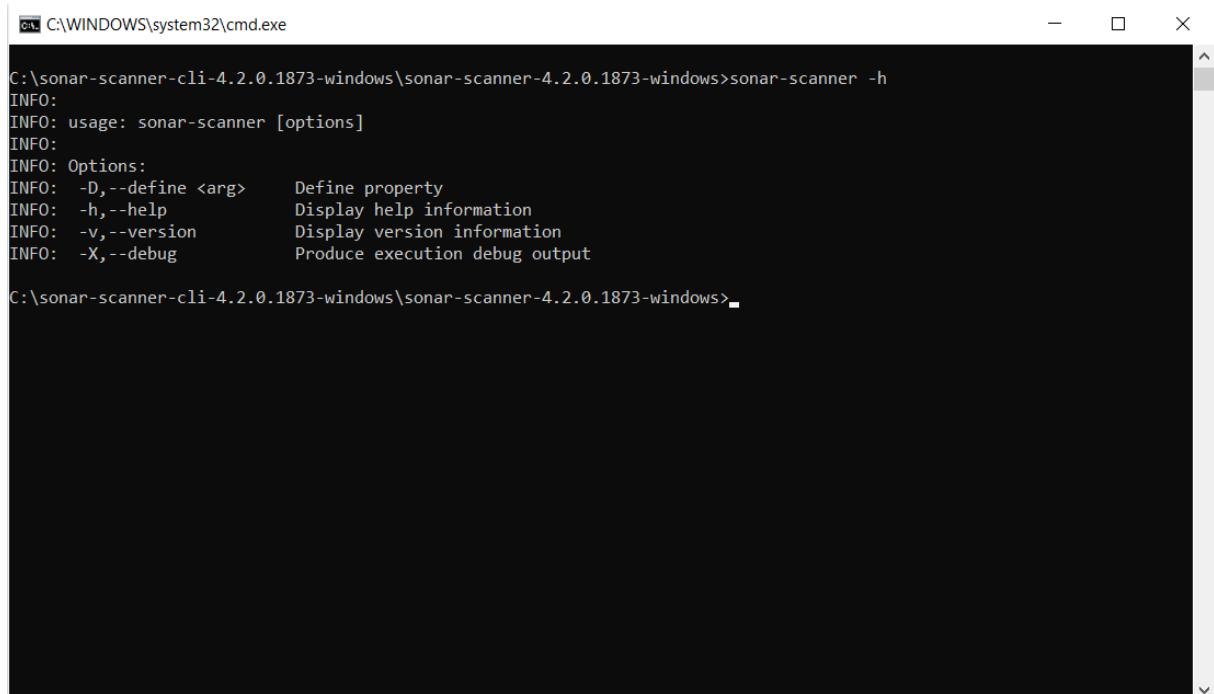
#----- Default SonarQube server
sonar.host.url=http://localhost:9000

#----- Default source code encoding
sonar.sourceEncoding=UTF-8
```

Slika 70. Prikaz potrebnog izgleda konfiguracijske datoteke SonarScanner alata

Kada je konfiguracijska datoteka podešena, potrebno je u Windows sustavu dodati path, upute kako se path dodaje se nalaze na sljedećem linku: <https://docs.telerik.com/teststudio/features/test-runners/add-path-environment-variables>.

Nakon toga, potrebno je pokrenuti u Command Prompt-u naredbu „sonnar-scanner.bat -h“.



```
C:\WINDOWS\system32\cmd.exe
C:\sonar-scanner-cli-4.2.0.1873-windows\sonar-scanner-4.2.0.1873-windows>sonar-scanner -h
INFO:
INFO: usage: sonar-scanner [options]
INFO:
INFO: Options:
INFO: -D,--define <arg>      Define property
INFO: -h,--help                 Display help information
INFO: -v,--version              Display version information
INFO: -X,--debug                Produce execution debug output
C:\sonar-scanner-cli-4.2.0.1873-windows\sonar-scanner-4.2.0.1873-windows>
```

Slika 71. Prikaz naredbe sonar-scanner.bat -h

Sada se potrebno pozicionirati u direktorij od projekta nad kojim želimo napraviti analizu koda i to u mapu gdje se nalazi MainActivity. Zatim u njega kopiramo iz mape SonarScanner-a

konfiguracijsku datoteku i preimenujemo ju u „sonar-project.properties“ i u nju stavimo naredbe sa sljedeće slike.



```
sonar-project - Blok za pisanje
Datoteka Uređivanje Oblikovanje Prikaz Pomoć
sonar-project.properties
# must be unique in a given SonarQube instance
sonar.projectKey=904a47a0ce89aee2676af98eda7a37a8679c1ca2
|
# --- optional properties ---

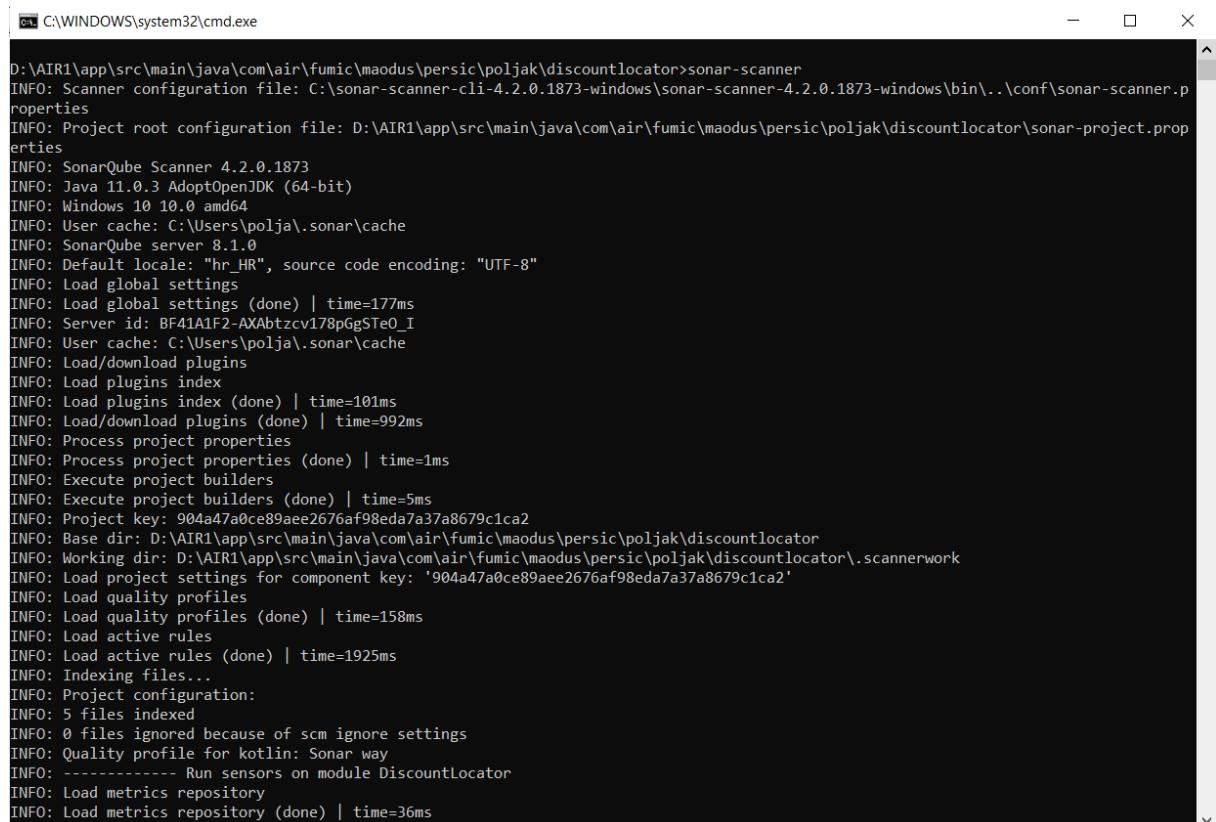
# defaults to project key
sonar.projectName=DiscountLocator
# defaults to 'not provided'
#sonar.projectVersion=1.0

# Path is relative to the sonar-project.properties file. Defaults to .
#sonar.sources=.

# Encoding of the source code. Default is default system encoding
sonar.sourceEncoding=UTF-8
```

Slika 72. Prikaz kreirane konfiguracijske datoteke za SonarScanner u direktoriju projekta

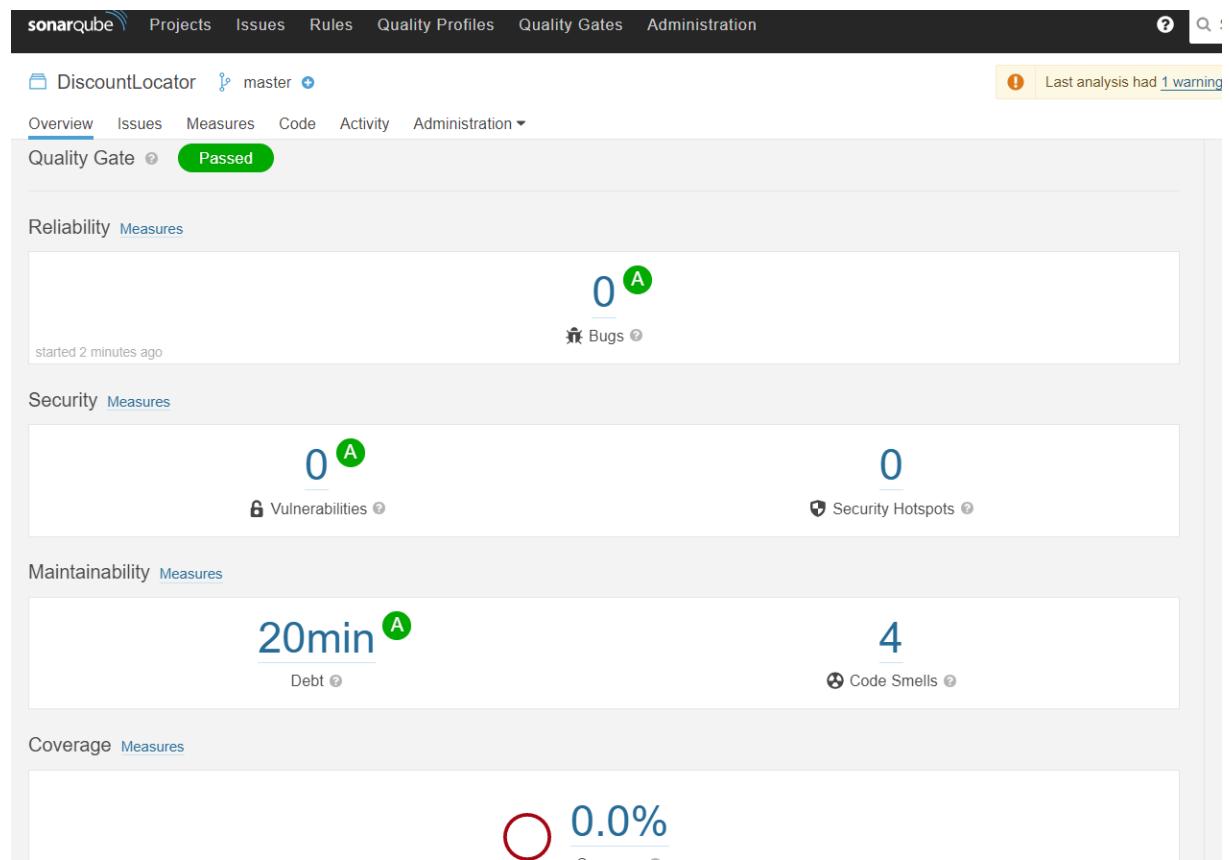
Sada je moguće pokrenuti analizu i test aplikacije na način da se unutar direktorija aplikacije pokrene Command Prompt te se izvrši naredba sonnar-scanner.



```
C:\WINDOWS\system32\cmd.exe
D:\AIR1\app\src\main\java\com\air\fumic\maodus\persic\poljak\discountlocator>sonnar-scanner
INFO: Scanner configuration file: C:\sonar-scanner-cli-4.2.0.1873-windows\sonar-scanner-4.2.0.1873-windows\bin\..\conf\sonar-scanner.properties
INFO: Project root configuration file: D:\AIR1\app\src\main\java\com\air\fumic\maodus\persic\poljak\discountlocator\sonar-project.properties
INFO: SonarQube Scanner 4.2.0.1873
INFO: Java 11.0.3 AdoptOpenJDK (64-bit)
INFO: Windows 10 10.0 amd64
INFO: User cache: C:\Users\polja\.sonar\cache
INFO: SonarQube server 8.1.0
INFO: Default locale: "hr_HR", source code encoding: "UTF-8"
INFO: Load global settings
INFO: Load global settings (done) | time=177ms
INFO: Server id: BF41A1F2-AXAbtzcv178p0gSt0_I
INFO: User cache: C:\Users\polja\.sonar\cache
INFO: Load/download plugins
INFO: Load plugins index
INFO: Load plugins index (done) | time=101ms
INFO: Load/download plugins (done) | time=992ms
INFO: Process project properties
INFO: Process project properties (done) | time=1ms
INFO: Execute project builders
INFO: Execute project builders (done) | time=5ms
INFO: Project key: 904a47a0ce89aee2676af98eda7a37a8679c1ca2
INFO: Base dir: D:\AIR1\app\src\main\java\com\air\fumic\maodus\persic\poljak\discountlocator
INFO: Working dir: D:\AIR1\app\src\main\java\com\air\fumic\maodus\persic\poljak\discountlocator\scannerwork
INFO: Load project settings for component key: '904a47a0ce89aee2676af98eda7a37a8679c1ca2'
INFO: Load quality profiles
INFO: Load quality profiles (done) | time=158ms
INFO: Load active rules
INFO: Load active rules (done) | time=1925ms
INFO: Indexing files...
INFO: Project configuration:
INFO: 5 files indexed
INFO: 0 files ignored because of scm ignore settings
INFO: Quality profile for kotlin: Sonar way
INFO: ----- Run sensors on module DiscountLocator
INFO: Load metrics repository
INFO: Load metrics repository (done) | time=36ms
```

Slika 73. Izvršavanje analize i testa koda

Sada se mogu vidjeti rezultati testa, u ovom slučaju test je prošao, nije bilo problema, postoje samo četiri zamjerke koje se mogu poraviti, ne postoji ponavljajući ili kopirani kod, ne postoje bug-ovi niti ranjivosti koje mogu dovesti do rušenja programa. Sve je to vidljivo na slici ispod.



Slika 74. Izgled izvršenog testa

Popis literature

- [1] „Update the IDE and SDK Tools | Android Developers“. [Na internetu]. Dostupno na: <https://developer.android.com/studio/intro/update.html>. [Pristupljeno: 30-sij-2020].
- [2] „Start another activity | Android Developers“. [Na internetu]. Dostupno na: <https://developer.android.com/training/basics/firstapp/starting-activity?hl=en#kotlin>. [Pristupljeno: 04-velj-2020].
- [3] „Using lists in Android wth ListView - Tutorial“. [Na internetu]. Dostupno na: <https://www.vogella.com/tutorials/AndroidListView/article.html>. [Pristupljeno: 30-sij-2020].
- [4] „Android Kotlin: Essentials to Creating a ListView (Ep 1) - YouTube“. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=EwwdQt3_fFU&t=. [Pristupljeno: 31-sij-2020].
- [5] „Android ListView Tutorial with Kotlin | raywenderlich.com“. [Na internetu]. Dostupno na: <https://www.raywenderlich.com/155-android-listview-tutorial-with-kotlin>. [Pristupljeno: 30-sij-2020].
- [6] „Activities or Fragments? A little sharing... - Elye - Medium“. [Na internetu]. Dostupno na: <https://medium.com/@elye.project/activities-or-fragments-a-little-sharing-c1ddc1041f79>. [Pristupljeno: 03-velj-2020].
- [7] „Implementing Navigation Drawer in Android App using Kotlin“. [Na internetu]. Dostupno na: <https://www.androdocs.com/kotlin/implementing-navigation-drawer-in-android-app-using-kotlin.html>. [Pristupljeno: 05-velj-2020].
- [8] „Get started with the Navigation component | Android Developers“. [Na internetu]. Dostupno na: <https://developer.android.com/guide/navigation/navigation-getting-started>. [Pristupljeno: 04-velj-2020].

Popis slika

Slika 1. Pronalazak Android Studio razvojnog okruženja	10
Slika 2. Pronalazak zadnje verzije Android Studio razvojnog okruženja	11
Slika 3. Zadnja verzija Android Studio razvojnog okruženja.....	11
Slika 4. Početak instalacijskog postupka	12
Slika 5. Početni prikaz Android Studio razvojnog okruženja.....	13
Slika 6. Izbornik mogućih postavki.....	13
Slika 7. Android SDK postavke.....	14
Slika 8. Prikaz detalja paketa u pojedinim verzijama Androida	15
Slika 9. Prikaz označenih potrebnih paketa	16
Slika 10. Prihvatanje uvjeta korištenja paketa.....	16
Slika 11. Prikaz SDK alata.....	17
Slika 12. Prikaz potrebnih SDK alata.....	18
Slika 13. Android Studio Updates	19
Slika 14. Auto Import	20
Slika 15. Appearance	20
Slika 16. Code Folding	21
Slika 17. Kreiranje novog projekta	22
Slika 18. Odabir Android predloška	22
Slika 19. Početno postavljanje projekta	23
Slika 20. AVD Manager	24
Slika 21. Odabir virtualnog uređaja.....	25
Slika 22. Zaključivanje konfiguracije virtualnog uređaja	26
Slika 23. Struktura projekta.....	27
Slika 24. activity_main.xml	34
Slika 25. MyCustomAdapter.kt	35
Slika 26. DisplayListActivity.kt	36

Slika 27. MainActivity.kt.....	37
Slika 28. row_main.xml.....	37
Slika 29. Prikaz osnovnog UI.....	38
Slika 30. drawer_menu.xml	2
Slika 31. activity_main.xml	3
Slika 32. options_menu.xml.....	4
Slika 33. root_preference.xml.....	4
Slika 34. Postavke aplikacije	5
Slika 35. Prikaz navigacijske ladice	6
Slika 36. Kod navigacijske ladice unutar MainActivity	6
Slika 37. Funkcija za odabir elemenata navigacijske ladice	7
Slika 38. About app opcija	8
Slika 39. Uključivanje retrofita u projekt	9
Slika 40. AndroidManifest.xml webservice modula	9
Slika 41. MyWebserviceResponse klasa	10
Slika 42. MyWebservice sučelje	11
Slika 43. Companion object MyWebserviceCaller klase	11
Slika 44. Metode za procesiranje odgovora webservisa	12
Slika 45. getDiscounts metoda unutar MyWebserviceCallera	12
Slika 46. Potrebni paketi za kompletan RecyclerView.....	16
Slika 47. RecyclerView widget.....	17
Slika 48. Metoda getItemCount u adapteru za RecycleView.....	17
Slika 49. Prilagođen ViewHolder	18
Slika 50. Preostale metode RecyclerView elementa.....	18
Slika 51. Inicijalizacija RecyclerView elementa	19
Slika 52. TopSpacingItemDecoration klasa	19
Slika 53. Dovršen RecycleView prikaz.....	20
Slika 54. Ovisnosti potrebne za korištenje Firebase servisa za poruke	21

Slika 55. Ovisnosti potrebne za korištenje Google Play servisa.....	22
Slika 56. Servis u AndroidManifest.xml koji omogućuje sve radnje s porukama.....	22
Slika 57. Funkcija koja dohvata token aplikacije za slanje poruka.....	23
Slika 58. Prikaz Firebase sučelja za slanje poruka	24
Slika 59. Prikaz push notifikacija na smartphone-u.....	24
Slika 60. Dodavanje ovisnosti za Google reklame	25
Slika 61. Dodavanje ID-a aplikacije u AndroidManifest.xml	25
Slika 62. Metoda za inicijalizaciju reklama.....	26
Slika 63. Prikaz ekrana s reklamom.....	26
Slika 64. Preuzimanje SonarQube alata	27
Slika 65. Uređivanje konfiguracijske datoteke wrapper	28
Slika 66. Pokretanje SonarQube-a	28
Slika 67. Izgled SonarQube-a na web-u	29
Slika 68. Stranica za preuzimanje SonaScanner alata.....	29
Slika 69. Prikaz potrebnog izgleda konfiguracijske datoteke SonarScanner alata.....	30
Slika 70. Prikaz naredbe sonar-scanner.bat -h	30
Slika 71. Prikaz kreirane konfiguracijske datoteke za SonarScanner u direktoriju projekta ...	31
Slika 72. Izvršavanje analize i testa koda	31
Slika 73. Izgled izvršenog testa	32

Popis tablica

Tablica 1. Popis funkcionalnosti 2