****

**Univerzitet Singidunum**

**Tehnički fakultet**

**Razvoj mobilne aplikacije - ImgExplore**

**- Projektna dokumentacija -**

Predmet: **Razvoj mobilnih aplikacija**

|  |  |  |
| --- | --- | --- |
| **Profesor**: | **Radila**: | |
| Miodrag Živković  **Asistent**:  Uroš Arnaut | | 1. Anđela Jovanović 2017200827 SII |
|  |
|  |

Beograd, 2020. godine

# Uvod

Kroz ovu dokumentaciju dat je kratak opis ImgExplore mobilne aplikacije I platforme na kojoj je rađena. ImgExplore je besplatna aplikacija koja svojim korisnicima omogućava pretragu i prikaz različitih fotografija, kako od profesionalnih fotografa, tako i od običnih amatera koji žele da zabeleže neko svoje životno iskustvo u digitalnom formatu ili ispričaju neku priču. Koristeći podatke sa zvaničnog Flickr sajta, aplikacija stvara mali community, gde razni korisnici mogu pokazati svoje talente, stilove fotografisanja ili napraviti svoj lični portfolio. Svaki korisnik prilikom registracije dobija svoj profil, na kome se prikazuju lični podaci, koje može promeniti u bilo kom trenutku jednostavnim dodirom na dugme. Svoj profil može dodatno personalizovati dodavanjem svojih ličnih fotografija, od kojih kasnije može napraviti neke svoje kolekcije omiljenih radova. Jednostavan i uniforman dizajn omogućava lako skrolovanje kroz newsfeed, lajkovanje zanimljivih slika ili pretragu u posebnoj sekciji explore. Navigacija kroz aplikaciju je takođe izuzetno jednostavna i dinamična, dovljno je da korisnik swipe-uje kroz određene sekcije i vrlo lako može doći do delova koji ga interesuju. Vrlo često se dešava da neke spontane i brze fotografije budu one najlepše, tako da ukoliko korisnik poželi da odmah uslika neki trenutak, jednostavno može pokrenuti direktno kameru preko aplikacije i zabeležiti šta god poželi. Dovoljno je samo dopustiti joj da prisupi kameri i lokalnim fajlovima, a ostalo je sve u rukama fotografa.

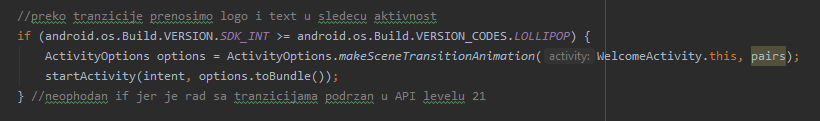
Aplikacija je generalno namenjena korisnicima svih uzrasta koji žele da na lak i brz način uslikaju neki trenutak i sačuvaju ga.

# ****Opis platforme i kratak istorijat****

Celokupna funkcionalnost rađena je u Java programskom jeziku, a aplikacija je prilagođena Android telefonima, koji imaju minimalni API nivo 16 (Android JellyBean). Android je operativni sistem za ekrane osetljive na dodir koji se u svojoj osnovi bazira na Linux jezgru. Njegov razvoj počeo je 2003. godine od strane kompanije Android Inc. koju nešto kasnije, 2005. godine, preuzima Google. Nakon svog uspeha, Android postaje vodeći član Open Headset Alliance-a, a Google proširuje podršku i za druge uređaje, poput pametnih televizora, Android Wear-a itd. Sam Android se zasniva na licenci otvorenog koda i može biti kompatabilan sa različitim uređajima. Ovo se postiže time što se softver razdvaja od hardvera na kome se izvršava, pa je samim tim omogućeno velikom broju različitih uređaja da izvršavaju istu aplikaciju.

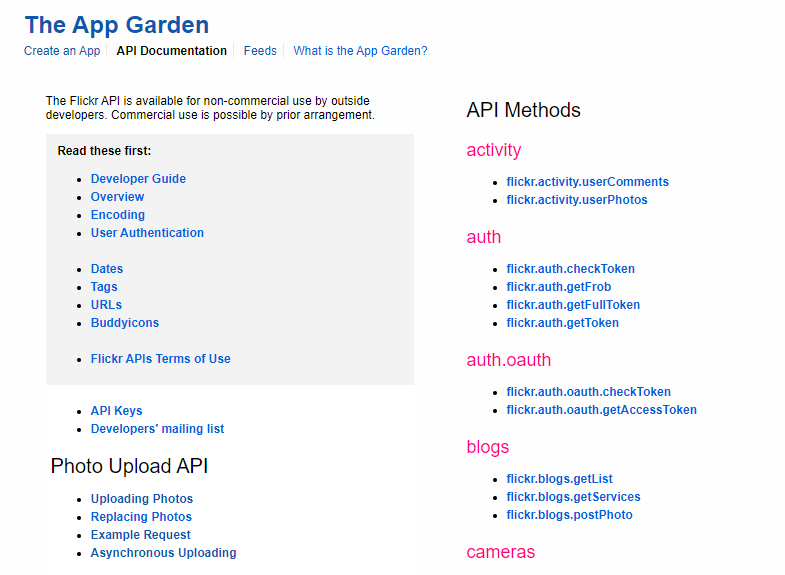
2008. godine pojavljuje se prvi komercijalno dostupni model telefona sa Android OS-om, HTC Dream, koji je bio poznat i poda nazivom T-Mobile G1, dok su prve verzije Androida bile interno poznate kao AstroBoy i Bender. Svakoj verziji dodeljuje se određeni API nivo, koji vrlo koristan indikator programerima, jer pokazuje koje su verzije brojnih klasa i biblioteka dostupne za razvoj. Do verzije 10, svaki Androidov *release* je imao svoje kodno ime, poput JellyBean-a, Kit-Kat-a, Oreo itd.

Za svakog programera, veoma bitna informacija jeste poznavanje situacije na tržištu, odnosno rasprostranjenost verzija. Za ovaj projekat konkretno, kao što je napomenuto, izabran je API nivo 16, koji se poklapa sa standardnom verzijom Androida 4.1 (Jelly Bean), čime se praktično dobija veliki udeo na tržištu, s obzirom da potencijalno gađa veliki broj uređaja na kojima će moći nesmetano da funkcioniše. Može se desiti da određene animacije ne funkcionišu na starijim verzijama Androida, što je u samom kodu obrađeno kao izuzetak na adekvatan način:

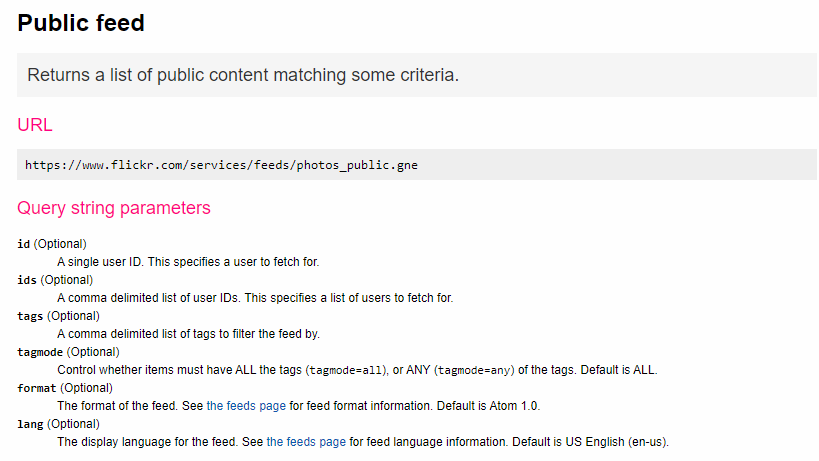


Radno okruženje koje je korišćeno za razvoj aplikacije je Android Studio, koji predstavlja zvanično integrisano okruženje, koji je dostupan za sve operativne sisteme (Windows, MAC, Linux). Izgrađen je na JetBrains-ovom Intellij IDEA softveru i specijalizovan za razvoj mobilnih aplikacija. Developerima je na raspolaganju i build-in emulator, na kome mogu pokrenuti i testirati aplikaciju. Jedino ograničenje jeste što on ne funkcioniše na AMD-ovim procesorima, ali je moguće skinuti novu verziju koja može pokrenuti AVD na novijim AMD procesorima. Emulator takođe ne predstavlja fizički uređaj, tako da se može desiti da testiranje i debagovanje aplikacije bude otežano, međutim dobra stvar je ta što je moguće pokrenuti aplikaciju na starijoj verziji Androida i testirati njene performanse. S obzirom na to da autor koristi AMD procesor, aplikacija je testirana na ličnom telefonu, koji je pripremljen otključavanjem *developer options-a* u podešavanjima.

Dizaj korisničkog interfejsa je odrađen podešavanjem globalne teme i korišćenjem Material Design komponenata. Material Design je vizuelni jezik koji sintetiše klasične principe dobrog dizajna sa integracijom tehnologije i nauke. Omogućava kreiranje jedinstveneog sistema koji objedinjuje korisničko iskustvo na svim platformama, uređajima i u zavisnosti od metoda unosa. Podaci koje korisnici mogu da pretražuju se povlače sa zvaničnog sajta Flickr.



Flickr predstavlja platformu za deljenje slika i pravljenje vizuelnih portfolija, a pored toga što je *user-friendly*, ima takođe i podrušku za developere. U sekciji *developer options* mogu se naći čitave kolekcije besplatnih API-ja, koje programeri mogu iskoristi za potrebe svojih aplikacija koje razvijaju. Određeni API-ji se mogu koristiti isključivo uz *API key*, koji se može lako dobiti, međutim radi smanjenja kompleksnosti same aplikacije, iskorišćen je njihov besplatni API za povlačenje različitih fotografija sa svojim internim podacima. Format podataka je u vidu JSON-a, kako bi bilo lakše i brže manipulisati njima, a takođe lako se mogu skinuti sa API-ja preko AyncTask-a u pozadinskoj niti, pri čemu se ne blokira korisnički interfejs i ne kvari doživljaj. Još jedan razlog zbog koga je korišćen ovaj API jeste i mogućnost menjanja samog linka sa koga dohvatamo podatke, tako da ga je moguće praktično napraviti shodno potrebama aplikacije i ličnim preferencama.

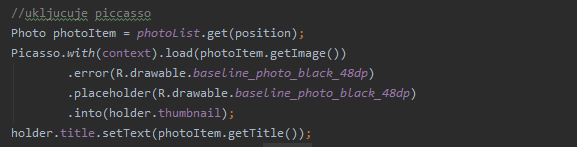


Za dinamičko učitavanje slika, korišćenja je Piccasso biblioteka. Ovo je *third-party* biblioteka napravljena od strane Square-a, koja preuzima i kešira remote slike i omogućava njihovo jednostavno korišćenje i prikazivanje. Posebno je pogodna za rad sa RecyclerView-om. Kako bi se biblioteka uključila u projekat, potrebno je dodati odgovarajući *dependency* u build.gradle fajlu:

dependencies

implementation 'com.squareup.picasso:picasso:2.5.2'

Primer korišćenja u kodu:

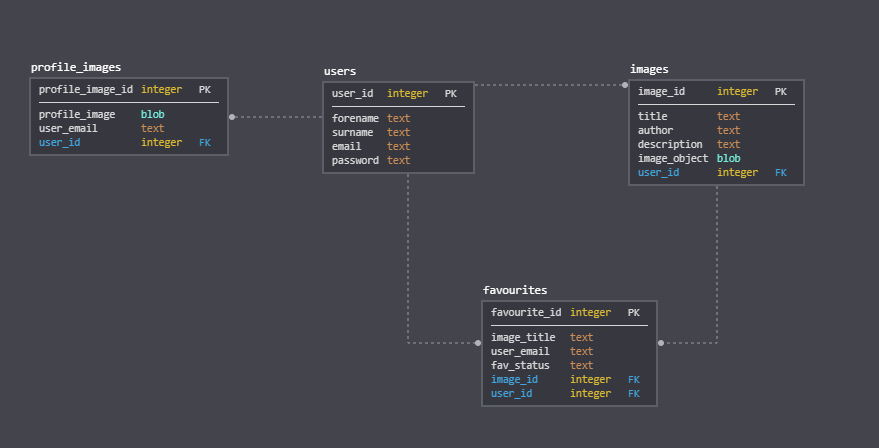


# Implementacija baze i rad sa podacima

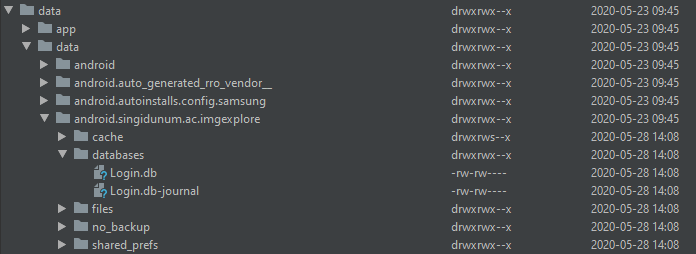
Sama aplikacija radi sa značajnom količinom podataka, neke čuva interno, neke u ekternom storage-u, a neke unutar same baze. Kada govorimo o čuvanju podataka, Android sistem, kao i svaki drugi, koristi sistem fajlova. Postoji nekoliko načina kako možemo sačuvati podatke sa kojima naša aplikacija radi:

* Interni prostor za smeštanje vidljiv samoj aplikaciji: služi za smeštanje fajlova koje može koristiti samo aplikacija, ili je u okviru nekog *shared* direktorijuma u internom prostoru
* Deljeni prostor: smeštaju se fajlovi koje aplikacija deli sa drugim, tipa multimedijalnog sadržaja ili dokumenata
* Shared preferences: smeštanje privatnih primitivnih podataka u obliku parova *key-value*
* Baze podataka: čuvanje struktuiranih podataka u privatnoj bazi na uređaju

Za rad sa bazom podataka možemo koristit SQLite bazu ili Room biblioteku. U ovom projektu je konkretno korišćen SQLite API, preko kojeg je razvijena jednostavna baza koja čuva podatke o korisnicima, profilnim slikama, slikama koje ubacuju u samu aplikaciju kao personalizovani feed na profilu i kolekcijama koje mogu praviti, jednostavnim favorizovanjem željenih slika.



U kodu je napravljena klasa DatabaseHelper koja sadrži metode neophodne za rad sa bazom, gde se izvršavaju osnovni CRUD ciklusi. Klasa ekstenduje SQLiteOpenHelper klasu, koja služi kao pomoćna klasa za pravljenje baze i menadžment podataka. Sam Android bazu podataka smešta u okviru privatnog direktorijuma aplikacije u okviru internal storage-a. Time su podaci zaštićeni, pošto ovom direktorijumu druge aplikacije ne mogu pristupiti. Ukoliko se aplikacija testira na fizičkom uređaju, moguće je u okviru Android Studia proveriti gde je tačno sačuvana baza u sekciji View-Tool Windows-Device File Explorer:

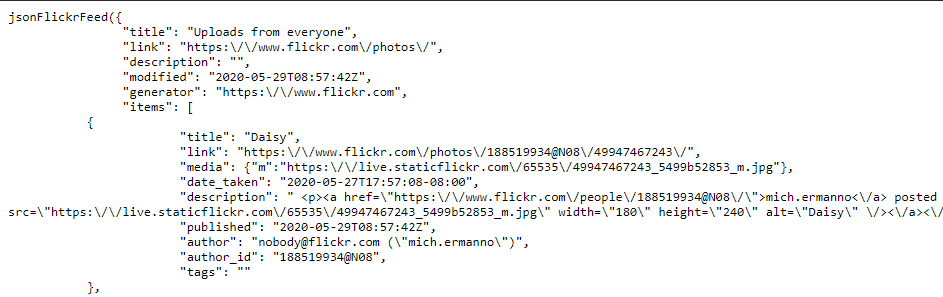


# Struktura projekta i logika

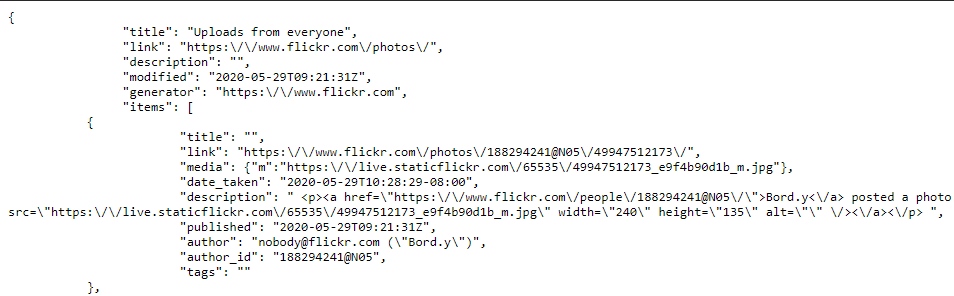
Kao što je ranije napomenuto, aplikacija radi sa standardnim setom podataka koji se može naći u sekciji Feeds – Public photos and videos na Flickr zvaničnom sajtu. Podaci su dati u JSON formatu kako bi bila lakša i brža manipulacija njima, i kasnije parsiranje. Ovaj format koristi tekst koji je čitljiv za ljude i prilično efikasan kada govorimo o čuvanju i prenosu podataka, koji su predstavljeni u obliku parova *key:value*. Može se lako integrisati sa drugim tehnologijama, te je tako našao široku primenu. Konkretno u ovoj aplikacji JSON format podrazumeva niz objekata, gde svaki objekat predstavlja jednu konkretnu sliku, koja ima dodatne informacije, kao što je naziv, ime autora, tagovi (koji su korisni za pretragu) itd. Slika se čuva zapravo u vidu linka, koji direktno vodi do nje. Ukoliko bismo samo link našeg API-ja kopirali u browser, videli bismo da su podaci u svom izvornom obliku predstavljeni kao xml.



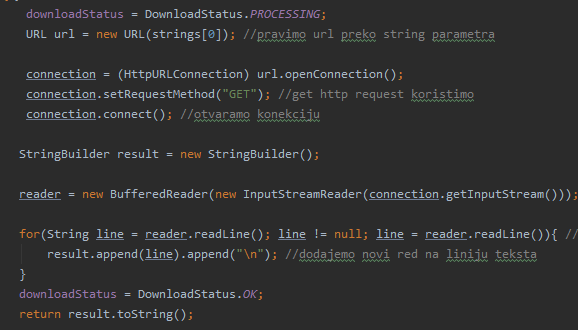
Da bismo došli do JSON formata, potrebno je da nakon linka dodamo parametar format=json: <https://www.flickr.com/services/feeds/photos_public.gne?format=json>



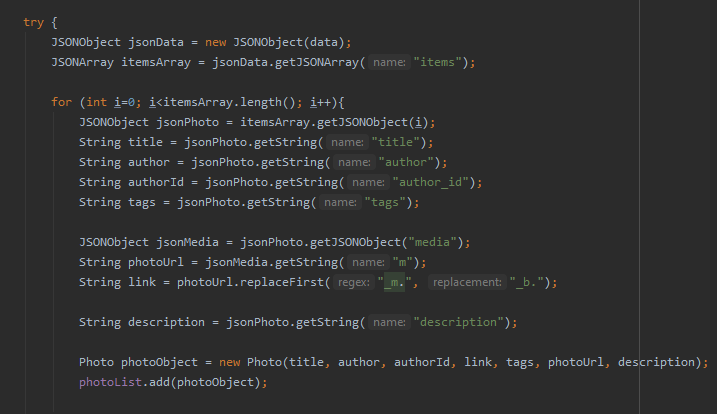
Dobra praksa je da kada radimo sa public API-jima imamo neki način kako možemo validirati da li je format podataka koji se na njemu nalazi 100% ispravan, kako ne bismo kasnije prilikom razvoja aplikacije imali problema. Za potrebe ovog projekta korišćen je JSONLint validator, gde je potrebno samo kopirati celu json strukturu i on će automatski proći kroz kod i proveriti ispravnost formata. U ovom slučaju, validator je prijavljivao grešku na samom početku json-a, tj ukazivao je na: jsonFlickrFeed, koji je praktično nepotreban parametar. Kada pošaljemo request ka datom linku, mi zapravo dobijamo tzv. JSONP response, gde se pored podataka može videti i callback funkcija koja te podatke obmotava. Kako bi se dobio čist json niz objekata, bez naziva funkcije, potrebno je u url-u nakon format=json dodati još jedan parametar nocallbackjson=1: <https://www.flickr.com/services/feeds/photos_public.gne?format=json&nojsoncallback=1>



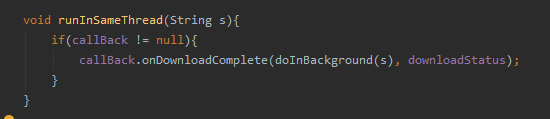
Kao što je ranije napomenuto, ovi podaci se povlače u AsyncTask klasi, koja u principu kreira pozadinsku nit. Generalno svaka Android aplikacija se izvršava u okviru svog Linux procesa, a pošto se u principu bazira na stalnoj interakciji sa korisnikom, svaka glavna funkcionalnost se izvršava u okviru glavne niti ili UI thread. Takođe, svaka callback metoda se okida i izvršava u svojoj niti. Ono što može biti problem jeste da kada recimo stavimo neku kompleksnu operaciju da se izvršava u glavnoj niti, kao što je u ovom slučaju skidanje podataka preko Interneta, može doći do kočenja UI. Ukoliko interfejs ostane zabagovan duže od 5s može iskočiti ARN dijalog, ili da praktično dođe do pucanja cele aplikacije, što naravno može ostvariti nezadovoljstvo krajnjih korisnika. Iz tog razloga, korišćena je AsyncTask klasa, koja objedinjuje dva glavna procesa, skidanje podataka u pozadinskoj niti i prikazivanje istih na korisničkom interfejsu. S obzirom da se radi sa nešto složenijim podacima, za potrebe ovog projekta napravljene su dve klase koje nasleđuju AsyncTask klasu, GetRawData.java i GetJsonData.java. Cilj je napraviti što labaviju spregu između komponenata, kako bi one mogle po potrebi da se recikliraju i izmene. GetRawData je generička klasa, čiji je jedini zadatak da u pozadini skine sirove podatke sa bilo kog linka koji joj se prosledi u bilo kom formatu. U njoj je otvorena konekcija i pušten GET zahtev ka resursima na prethodno pomenutom linku.



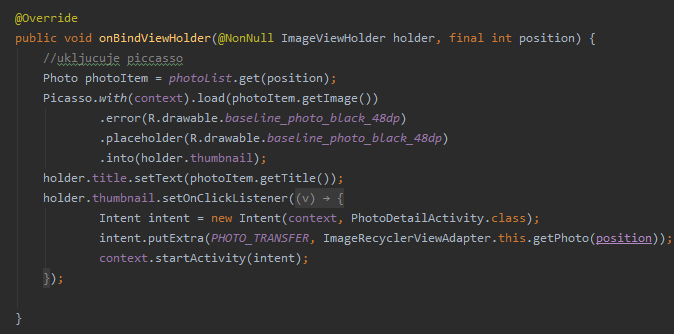
Klasa takođe sadrži jedno polje enum tipa, koji predstavlja indikaciju trenutnog statusa našeg zahteva, tj. podataka, kako bi bilo olakšano praćene kroz kod i konzolu. U URL konstruktoru je data indikacija da se zahteva samo jedan parametar, tj. prvi u nizu strings. Iščitavanje podataka se izvršava preko metode readLine(), koja iščitava red po red, a zatim se indikator DownloadStatus postavi na OK i rezultat se doda u StringBuilder. S obzirom da radimo sa URL konekcijom, može doći do raznih izuzetaka, koji su adekvatno obrađeni u catch blokovima. Ukoliko je sve prošlo u redu, i ako su podaci uspešno skinuti, GetRawData preko svoje callback metode obaveštava GetJsonData klasu da su podaci skinuti i spremni za parsiranje. GetJsonData zatim preuzima podatke, parsira ih,a zatim preko svoje callback metode, obaveštava svoju caller klasu, koja je u ovom slučaju ImageRecyclerViewAdapter klasa, da su podaci spremni za prikaz na UI.



Kada izvršavamo kod klasa koje nasleđuju AsyncTask klasu, sva logika će se izvršavati u posebnim nitima. Međutim, ovde je cilj samo razvojiti logiku na dve klase, koje će biti usko specijalizovane za izvršavanje određene logike, tako da kako bi smo obezbedili da se oba procesa, skidanje i parsiranje podataka izvršava u istoj niti napravjen je metod runInSameThread u GetRawData klasi.

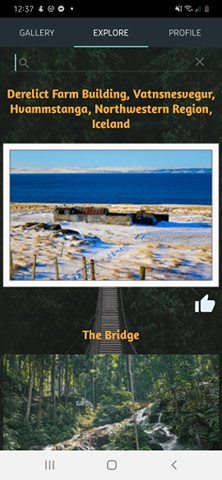


Što se tiče samog interfejsa, kao što je napomenuto, aplikacija je organizovana kroz fragmente i aktivnosti. Aktivnosti su realizovane preko intenta koji komuniciraju razmenom podataka. Kako je na više sekcija aplikacije neophodno dobaviti korisnički email kako bi se prikazao adekvatan sadržaj za trenutno ulogovanog korisnika, on se doprema do dela gde je potreban preko Bundle objekta. Aplikacija u sušitni radi sa jednom vrstom korisnika, a to su registrovani korisnici, koji nakon pokretanja aplikacije moraju uneti svoje kredincijale, tj. email i šifru, a zatim će biti redirektovani na home page, gde se nalaze podaci koji su na prethodno opisani način povučeni sa Interneta. Podaci u prikazani u RecyclerView, za koji je neophodan adapter koji dohvata obrađene podatke iz GetJsonData klase i setuje na odgovarajuća polja. RecyclerView je zapravo naslednik ListView-a, i vrlo je koristan kada radimo sa podacima koji se dinamički ubacuju na view i kao takvi se prikazuju korisniku prilikom skrolovanja. RecyclerView ne kreira svaki put poseban view za svaki podatak koji se pojavi, već reciklira isti view svaki put kada dobije naznaku da su mu dopremljeni novi podaci.



Kao što je ranije napomenuto, za automasko učitavanje slika koristi se Piccasso biblioteka. Kada korisnik dodirne („klikne“) na neku sliku, otvara se nova aktivnost koja prikazuje detalje te određene slike. Još jedna dobra stvar kod RecyclerView-a, pored njegove dinamičnosti, jeste i to što je vrlo koristan u situacijama kada treba odratiti neku logiku nad nekim spec. View-om kao što je ovo slučaj, jer RecyclerView pamti poziciju svakog elementa prilikom skrolovanja kroz njega.

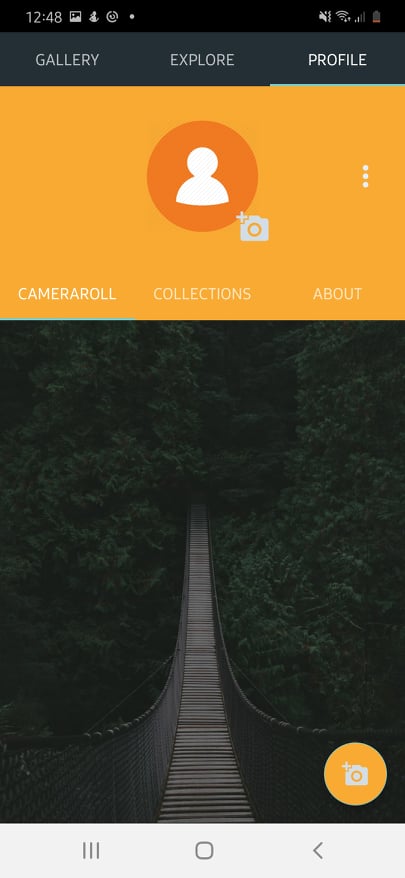
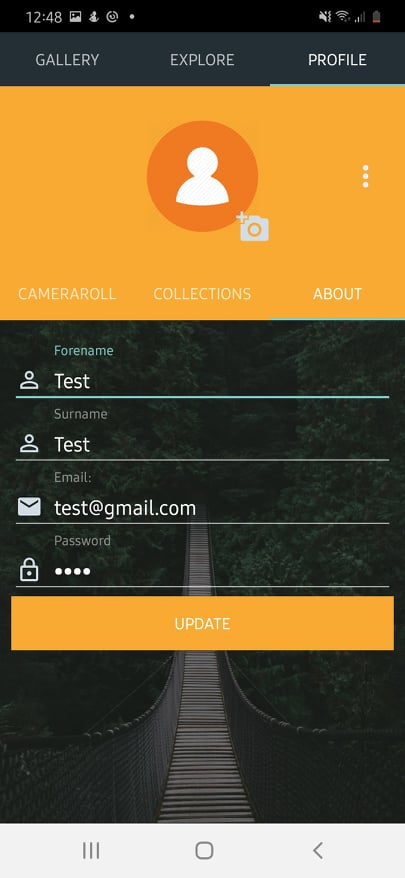
Što se tiče Explore sekcije, logika je slična kao u prethodno opisanoj komponenti, s jedinom razlikom da je na UI postavljem SearchView koji dinamički reaguje na ključne reči koje korisnik unosi i obrađuje querry u pozadini i izbacuje filtrirane rezultate.



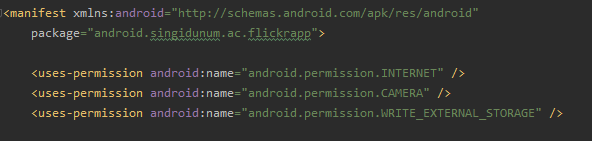
Sve glavne sekcije aplikacije su formirane kao fragmetni u okviru Home aktivnosti, koja praktično služi kao placeholder za njih. Svaka sekcija predstavlja jedan tab kroz koje korisnici mogu da *swipe-uju* i relativno brzo dođu do dela koji ih interesuje. Za rad sa fragmentima koji su organizovani u vidu tabova potrebno je takođe koristiti adapter, ViewPagerAdapter. U okviru HomeActivity klase se formira jedan niz u koji se dodaju pojedinačni tabovi u obliku *fragment:title*, tj. dodajemo konkretnu klasu koja se treba prikazati i naziv fragmenta koji će biti vidljiv na samom tabu.



Poslednja glavna sekcija jeste Profile odeljak, gde će korisnici moći da ubacuju neke svoje slike, koje će biti izlistane u cameraroll fragmentu, zatim moćiće da ih favorizuju i one će se automatski pojaviti u sekciji collections, ili mogu da vide i izmene svoje podatke u sekciji about. Korisnicima je takođe omogućeno da odaberu neku sliku iz telefona koju mogu dodati kao profilnu i koju mogu takođe u bilo kom trenutku promeniti.

Još jedan dodatak aplikaciji jeste mogućnost korišćenja kamere. Korisnici mogu nakon što odobre aplikaciji da pristupi određenim segmentima uređaja, da u bilo kom trenutku fotografišu šta god požele direktno iz same aplikacije. Time se eliminiše potreba da korisnik prvo pokreće *build-in* kameru kako bi slikao, jer je aplikacija pokreće automatski umesto korisnika, i fotografiju čuva u fajlovima same aplikacije, tako da su i slike koje korisnici prave zaštićene, jer im ne mogu pristupiti druge aplikacije. Naravno, da bi sve ove funkcionalnosti mogle da se implementiraju, neophodo je bilo u Manifest fajlu definisati dozvole, prvenstveno za Internet, kako bi mogli da se dopreme podaci, a zatim i CAMERA i WRITE\_EXTERNAL\_STORAGE za direktan rad sa android kamerom i čuvanje slika u fajlovima aplikacije.



# Zaključak

Ovaj projekat je rađen na slobodnu temu iz predmeta Razvoj aplikacija, tako da je isključivo namenjen za edukativnu upotrebu. Ideja za aplikaciju je proizašla iz samog sajta Flickr čiji su podaci iskorišćeni. Sama aplikacija je napravljena tako da pokrije sve minimalne zahteve koji su navedeni u specifikaciji, ali ima i dodatne funkcionalnosti, koje su obrađene na predavanjima, kako bi i samim korisnicima bilo zanimljivo korišćenje aplikacije, kao što je samom autoru bilo zanimljivo implementirajući je.

