

TEHNIČKO VELEUČILIŠTE U ZAGREBU

STRUČNI STUDIJ RAČUNARSTVA

Tomislav Brabec

**ANDROID APLIKACIJA ZA PRAĆENJE OSOBNIH
FINANCIJA**

ZAVRŠNI RAD br. 745

Zagreb, srpanj 2016.

TEHNIČKO VELEUČILIŠTE U ZAGREBU

STRUČNI STUDIJ RAČUNARSTVA

Tomislav Brabec

**ANDROID APLIKACIJA ZA PRAĆENJE OSOBNIH
FINANCIJA**

JMBAG: 0246053630

ZAVRŠNI RAD br. 745

Zagreb, srpanj 2016.

TEHNIČKO VELEUČILIŠTE U ZAGREBU
POVJERENSTVO ZA ZAVRŠNI RAD

Zagreb, 4. srpnja 2016.

Veleučilište - **Tehničko veleučilište u Zagrebu**
odjelno
organizirano:
Predmet: **Programiranje u jeziku Java**

ZAVRŠNI ZADATAK br. 745

Pristupnik: **Tomislav Brabec (0246053630)**
Studij: **Stručni studij računarstva**
Smjer: **Programsko inženjerstvo**

Zadatak: **Android aplikacija za praćenje osobnih financija**

Opis zadatka:

Implementirati Android aplikaciju koja će korisnicima omogućavati unos troškova te pregled različitih osobnih financijskih izvještaja. Aplikacija mora podržavati i unos udjela u dionicama, štednje, razne vrste fondova i porezna davanja. Opisati ključne značajke Android sustava i prikazati ključne dijelove programskog koda. Na primjerima ekrana prikazati scenarije korištenja aplikacije.

Zadatak uručen pristupniku: 15. ožujka 2016.
Rok za predaju rada: 8. srpnja 2016.

Mentor:

Predsjednik povjerenstva za
završni rad:

Aleksander Radovan, pred.

Sažetak

U ovom završnom radu bit će detaljno objašnjena izrada Android aplikacije za praćenje osobnih financija te karakteristike Android operacijskog sustava za mobilne uređaje. Također, bit će objašnjeno trenutno stanje Androida na svjetskom mobilnom tržištu i što se može od njega očekivati u bliskoj budućnosti.

Sadržaj

1. Uvod.....	10
2. Android operacijski sustav.....	11
2.1. Povijest Androida.....	12
2.2. Linux.....	14
2.3. Android zajednica.....	15
2.4. Sigurnost.....	16
2.5. Android tržište.....	17
3. Alati i programski jezici korišteni za izradu aplikacije.....	18
3.1. Android Studio.....	18
3.2. Java.....	19
3.3. XML.....	20
3.4. Online server.....	23
3.5. Parse komunikacija sa serverom.....	24
4. Izrada aplikacije.....	26
4.1. Prijavljivanje i registracija korisnika ("LoginActivity" klasa).....	27
4.1.1. Metoda <i>findViews()</i>	28
4.1.2. Metoda <i>onCreate()</i>	29
4.1.3. Komunikacija <i>LoginActivity</i> klase s Parse serverom.....	30
4.1.4. Gumb za prijavu korisnika.....	31
4.1.5. "Alert Dialog" prozorčić.....	33
4.2. Upravljanje računima.....	35
4.2.1. "Invoice" klasa.....	35
4.2.1.1. Metoda <i>getCardExpiryYear()</i>	35
4.2.2. "MainActivity" klasa.....	36
4.2.2.1. XML pogled <i>MainActivity</i> klase.....	37
4.2.2.2. "FloatingActionButton" gumb.....	38
4.2.2.3. <i>DrawerLayout</i> kontejner.....	39
4.2.2.4. Metoda <i>onNavigationItemSelected()</i>	40
4.2.2.5. Metoda <i>parseFetchInvoices()</i>	41
4.2.2.6. Metoda <i>getInvoices()</i>	42
4.2.2.7. Metoda <i>dataSetChanged()</i>	43

4.2.2.8. Metode <i>onCreateOptionsMenu()</i> i <i>onOptionsItemSelected()</i>	43
4.2.3. "InvoiceDetailsActivity" klasa.....	44
4.2.4. <i>AddEditInvoiceActivity</i> klasa.....	46
4.2.4.1. Polje za unos broja kreditne kartice.....	48
4.2.4.2 Gumb za pohranu računa.....	49
4.2.5. SQLiteHelper klasa (Lokalna SQLite baza podataka).....	51
4.2.5.1. Metoda <i>onCreate()</i>	51
4.2.5.2. Metode <i>addInvoice()</i> i <i>updateInvoice()</i>	52
4.2.5.3. Metoda <i>updateOrInsertInvoice()</i>	53
4.2.5.3. Metoda <i>getInvoices()</i>	54
4.2.6. Postavke u aplikaciji ("SettingsActivity" klasa).....	55
4.2.6.1. XML datoteka postavki (pref_general.xml).....	55
5. Zaključak.....	56
6. Literatura.....	57
7. Prilozi.....	58
1. Prvi prilog.....	58

Popis slika

Slika 1. Eric Schmidt, Andy Rubin and Hugo Barra[1].....	10
Slika 2. Slojevi Linux jezgre[1].....	12
Slika 3. Primjer lažnog virusa[9].....	14
Slika 4. Grafički prikaz rasta popularnosti Androida od 2008. godine[12].....	15
Slika 5. Izgled Android Studio sučelja[11].....	16
Slika 6. Izgled OpenShift online aplikacije.....	21
Slika 7. Izgled Parse dashboard sučelja[10].....	22
Slika 8. Pogled za prijavu.....	25
Slika 9. Pogled za prikaz svih računa.....	34
Slika 10. Bočni izbornik.....	37
Slika 11. Pogled detalja računa.....	42
Slika 12. Pogled za dodavanje i uređivanje računa.....	44

Popis koda

Kod 1. Jednostavan primjer XML pogleda.....	21
Kod 2. Primjer povezivanja s Parse serverom.....	25
Kod 3. Primjer spremanja Parse objekta.....	25
Kod 4. Definicija LoginActivity klase.....	28
Kod 5. Deklaracija varijabli unutar LoginActivity klase.....	28
Kod 6. Metoda za povezivanje Java kontrola s XML elementima.....	28
Kod 7. Primjer onCreate() metode.....	29
Kod 8. Provjera prijavljenoga korisnika.....	30
Kod 9. Postavljanje XML pogleda na aktivnost.....	30
Kod 10. Izvršavanje akcije prilikom pritiska gumba za prijavu.....	31
Kod 11. Provjera internetske veze.....	32
Kod 12. Prikazivanje prozorčića s upitom o registraciji korisnika.....	33
Kod 13. XML za prikaz Dialog prozorčića.....	34
Kod 14. Definicija varijabli računa.....	35
Kod 15. Dohvaćanje godine isteka računa.....	35
Kod 16. XML za MainActivity pogled.....	37
Kod 17. Primjer definiranja FloatingActionButton objekta.....	38
Kod 18. Definiranje DrawerLayout objekta te postavljanje NavigationView izbornika.....	39
Kod 19. Primjer funkcije koja započinje novu akciju ovisno o odabranoj stavci bočnog izbornika.....	40
Kod 20. Primjer vraćanja Intent objekta u slučaju odabira pregleda svih računa u bočnom izborniku.....	40
Kod 21. Dohvaćanje računa s Parse servera.....	41
Kod 22. Dohvaćanje računa iz lokalne baze.....	42
Kod 23. Osvježavanje liste računa na dretvi korisničkog sučelja.....	43
Kod 24. Instanciranje izbornika na naslovnoj traci.....	43
Kod 25. Izvršavanje akcije prilikom odabira postavki u izborniku naslovne trake.....	43
Kod 26. Primjer dohvaćanja računa iz Intent objekta.....	45
Kod 27. Dohvaćanje postavki zadane valute te preračunavanje iste.....	45
Kod 28. Dio XML koda sa Spinner elementom.....	47
Kod 29. Popunjavanje polja ako se radi o uređivanju računa.....	48

Kod 30. Slučaj izmjene vrijednosti Spinnera ako se radi o Visa kartici.....	48
Kod 31. Poziv validacije kod pritiska gumba za pohranu računa.....	49
Kod 32. Pokušaj dohvaćanja id vrijednosti računa.....	49
Kod 33. Pohrana vrijednosti u ParseObject.....	50
Kod 34. Deklaracija konstanti SQLiteOpenHelper klase.....	51
Kod 35. SQL upit za izradu tablice računa unutar "onCreate()" metode.....	51
Kod 36. Isječak koda iz addInvoice() i updateInvoice() metoda koji služi za dodavanje vrijednosti u redak tablice.....	52
Kod 37. Naredba koja služi za dodavanje novog računa u tablicu.....	52
Kod 38. Naredba koja služi za pohranu postojećeg računa u tablicu.....	52
Kod 39. Provjera postojanja računa unutar baze.....	53
Kod 40. Dohvaćanje svih računa iz lokalne baze.....	54
Kod 41. XML datoteka postavki.....	55
Kod 42. Definicija zadane XML datoteke postavki.....	55
Kod 43. Poziv DatePickerDialog prozorčića.....	58

Popis tablica

Tablica 1: Udio mobilnih operacijskih sustava na tržištu[13].....	9
Tablica 2: Verzije Androida zajedno s njihovim trenutnim udjelom na tržištu[1].....	11

Popis kratica

AOSP - Android Open Source Project

AOT - Ahead Of Time

API - Application Programming Interface

ART - Android Runtime

EE - Enterprise Edition

JDK - Java Development Kit

JRE - Java Runtime Environment

JSON - Javascript Object Notation

JVM - Java Virtual Machine

ME - Micro Edition

SDK - Software Development Kit

SE - Standard Edition

1. Uvod

U sljedećem poglavlju ovoga rada je objašnjena povijest, napredak i trenutno stanje Androida na tržištu kao i rad same Android platforme, prednosti i mane te sigurnost u odnosu na ostale mobilne operacijske sustave. Ukratko je opisan sam način rada i građe Android operacijskog sustava te kakve promjene se mogu očekivati. Također je opisano kakav utjecaj i doprinos ima Android u današnjem društvu i tehnologiji. Android Studio, jedan od najpopularnijih programskih okruženja za izradu Android aplikacija, također je opisan u ovome dijelu završnog rada.

U trećem poglavlju završnog rada opisani su alati i programski jezici pomoću kojih je razvijena aplikacija, a u četvrtom poglavlju završnog rada detaljno je objašnjena izrada Android aplikacije za praćenje osobnih financija gdje su opisani programski kodovi funkcija i klasa u aplikaciji, povezivanje s bazom podataka te *Parse* biblioteka koja to omogućuje kao i ostale biblioteke koje su koristile za izradu ove aplikacije.

2. Android operacijski sustav

Android je najpopularniji operacijski sustav za mobilne uređaje utemeljen na Linuxu te se trenutno nalazi na 65,5% svih „pametnih“ mobilnih uređaja (mobilni telefoni, notebook računala, tableti, pametni satovi, Google TV) diljem svijeta. Za razliku od ostalih popularnih operacijskih sustava za mobilne uređaje kao što su Apple iOS i Windows Phone, Android je otvoren sustav što je privuklo veliki broj razvijatelja mobilnih aplikacija zbog velike kontrole nad samom jezgrom Android operacijskog sustava.[1]

Tablica 1. Udio mobilnih operacijskih sustava na tržištu[13]

Period	Android	iOS	Windows Phone	BlackBerry OS	Others
2015Q2	82.8%	13.9%	2.6%	0.3%	0.4%
2014Q2	84.8%	11.6%	2.5%	0.5%	0.7%
2013Q2	79.8%	12.9%	3.4%	2.8%	1.2%
2012Q2	69.3%	16.6%	3.1%	4.9%	6.1%

Trenutno je za Android dostupno više od milijun aplikacija koje se mogu skinuti s “Google Play Store”, dok broj mjesečnih aktivnih korisnika premašuje milijardu. Većina razvijatelja mobilnih aplikacija razvija aplikacije upravo za Android, odnosno njih nešto više od 70%, dok je oko 40% profesionalnih razvijatelja zaposleno u razvoju Android aplikacija, što je za 3% više od razvijatelja iOS aplikacija.[4][3]

2.1. Povijest Androida

Riječ Android prvi puta se spominje sredinom 19. stoljeća u Americi te predstavlja robota koji je izgledom i osobinama identičan čovjeku. Sam Android trebao je zamijeniti čovjeka u raznim poslovima te mu tako olakšati svakodnevni život. Istu ideju za naziv svoje nove tvrtke, odnosno mobilnog uređaja imali su i Andy Rubin, Rich Miner, Nick Sears i Chris White kada su 2003. imali zamisao da naprave “pametniji” mobilni uređaj koji će čovjeku pomoći u orijentaciji te biti prilagodljiv svakome. Jedan od prvih ciljeva kompanije bio je operacijski sustav za digitalne kamere. Nakon što je utvrđena neisplativost razvoja operacijskog sustava za digitalne kamere zbog malog tržišta, kompanija se okreće razvoju mobilnog operacijskog sustava koji će parirati tadašnjim velikim tvrtkama poput „Symbiana“ i „Windows Phone“. [1]



Slika 1. Eric Schmidt, Andy Rubin and Hugo Barra[1]

Osim financijskih poteškoća, u razdoblju od 2003. do 2005. godine o Androidu se ne zna previše. Google otkupljuje kompaniju 2005. godine za više od 50 milijuna dolara te svi osnivači osim Searsa ostaju u tvrtci. U početku Google nije puno otkrivao o novo otkupljenoj kompaniji, ali se nagađalo da Google planira ući u mobilno tržište. Tim Googleovih stručnjaka predvođeni s Rubinom razvili su novi operacijski sustav za mobilne uređaje za kojeg su tvrdili da je siguran, fleksibilan i nadogradiv. U početku Google je razvijao projekt pod nazivom “Sooner” koji je trebao biti namijenjen uređajima sličnim BlackBerry-u s fizičkom tipkovnicom, no kasnije je dobio podršku i za uređaje na dodirni zaslon.[1]

Krajem 2007. godine velike kompanije poput Google-a, HTC-a, Samsunga i T-Mobile izrazile su ideju o stvaranju otvorenih standarda za mobilne uređaje te iste godine Android postaje platforma otvorenoga koda. Prvi mobilni uređaj s Android operacijskim sustavom bio je HTC Dream koji je na tržište izašao 22. listopada 2008. godine. Od 2008. godine Android se kontinuirano nadograđuje i popravljaju sve propuste. Iste te godine izlazi SDK (Software Development Kit) verzije 1.0 i kasnije svaka velika nadogradnja na Androidu nosi zanimljiv naziv neke poslastice po abecednom redoslijedu. Tako je 2009. godine izašla verzija 1.5 koja je nosila naziv „Cupcake”, odnosno „Donut” (1.6), „Eclair” (2.0 i 2.1). U 2010. Google je izbacio i vlastite modele mobilnih uređaja pod nazivom Nexus koje i danas proizvodi, a razvoj samog Androida je donio i novu verziju 2.2 pod nazivom „Froyo” te nešto poslije i „Gingerbread” (2.3). U to vrijeme postojalo je već više od 60 različitih uređaja koji su koristili Android kao operacijski sustav. U 2012. godini dolaze „Honeycomb” (3.0 – 2.2.6) i „Ice Cream Sandwich” (4.0 – 4.3.1). 2012. donosi „Jelly Bean” (4.1 – 4.3.1), dok u 2013. dolazi „KitKat” (4.4). Najnovije verzije su „Lollipop” (5.0 – 5.1) iz 2014. te „Marshmallow” (6.0) iz listopada 2015. godine.[1][4][3]

Tablica 2. Verzije Androida zajedno s njihovim trenutnim udjelom na tržištu[1]

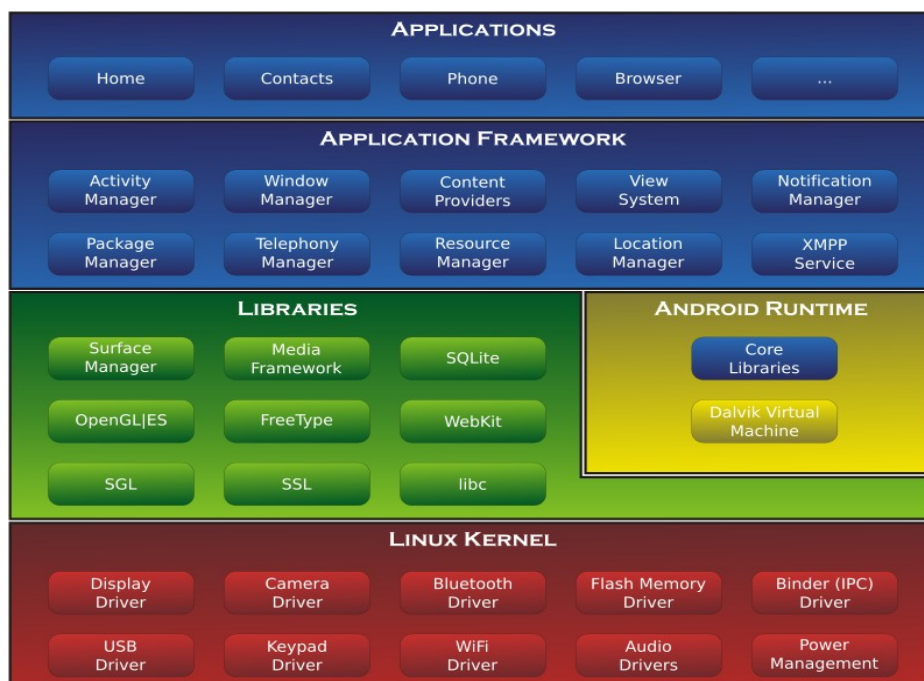
Verzija	Naziv	API	Distribucija
2.2	Froyo	8	0.2%
2.3.3 - 2.3.7	Gingerbread	10	3.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	2.7%
4.1.x	Jelly Bean	16	9.0%
4.2.x		17	12.2%
4.3		18	3.5%
4.4	KitKat	19	36.1%
5.0	Lollipop	21	16.9%
5.1		22	15.7%
6.0	Marshmallow	23	0.7%

2.2. Linux

Jezgra Android sustava temelji se na Linux jezgri te uglavnom koriste verzije 3.4 ili 3.10 Linux jezgre. Razlozi odabiranja Linux jezgre su portabilnost, sigurnost i nadogradivost. Prilikom samog razvoja Androida Google je napravio razne preinake na jezgri, a najveća je „wakelocks” koja služi za upravljanje korištenjem energije, ali je uz mnoge druge Google-ove preinake odbijena od strane glavnih razvijatelja same jezgre, iako su s kasnijim razvojem jezgre implementirane.[1][4]

Flash spremnik podataka na Androidu je podijeljen na više particija kao npr. „/system” za sam operacijski sustav i „/data” za korisničke podatke. Za razliku od desktop verzija operacijskih sustava Linux, Android korisnici nemaju „root” ovlasti nad samim sustavom i neke particije kao npr. „/system” moguće je samo čitati. Unatoč zaštitama, ipak je moguće dobiti „root” ovlasti sa zaobilaženjem nekih sigurnosnih postavki koje najviše koriste sami razvijatelji da bi uspjeli poboljšati razne mogućnosti samih uređaja, ali također to koriste i razni zločinitelji za instalaciju virusa i ostalih zloćudnih softvera.

Na samome vrhu Linuxove jezgre postoje programske biblioteke i API koji su pisani u C-u te aplikacijski softver koji se pokreće preko aplikacijskog *framework-a* koji sadrže Java-kompatibilne biblioteke.



Slika 2. Slojevi Linux jezgre[1]

Sve do verzije 5.0 Android je koristio virtualni stroj pod nazivom „Dalvik” s pravovremenim prevođenjem koda. Android 4.4 predstavio je „Android Runtime” (ART) koji je koristio „ahead-of-time” (AOT) prevođenje te je cijelu aplikaciju preveo u strojni kod prilikom instalacije za razliku od Dalvika. Iako u verziji 4.4 AOT nije bio postavljen kao zadani način izvođenja aplikacije, nego samo kao eksperiment, u verziji 5.0 postaje jedini i zadani način izvođenja.[1]

Android kao vlastite Java biblioteke koristi podskup već zastarjelog projekta „Apache Harmony” te je stoga Google najavio krajem 2015. da će se sljedeća verzija Androida prebaciti na Java implementaciju temeljenu na „OpenJDK” projektu.

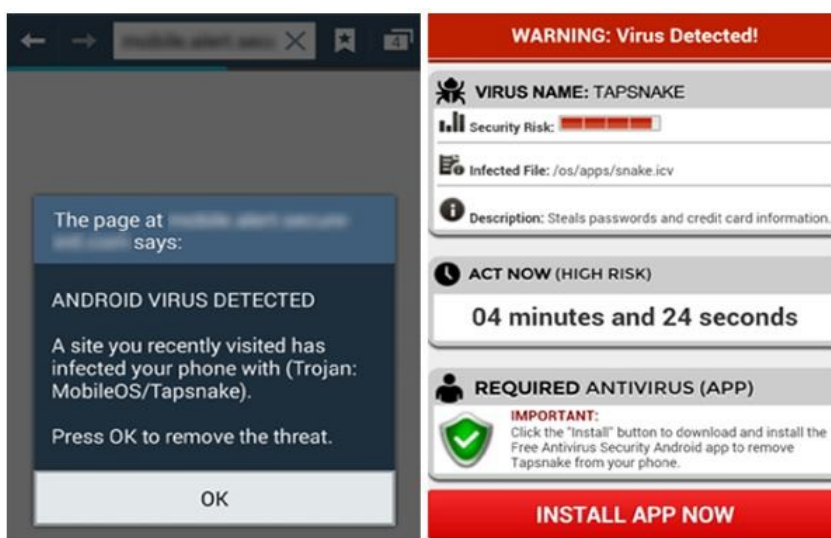
2.3. Android zajednica

Android ima veliku zajednicu razvijatelja i entuzijasta koji koriste „Android Open Source Project” ili skraćeno „AOSP” otvoreni kod za razvoj i distribuciju vlastitih i preinačenih verzija Androida. Takvi sustavi često donesu nove mogućnosti i nadogradnje uređajima brže nego oni razvijeni od strane službenih tvrtki. Osim toga, zajednice održavaju podršku za starije uređaje koji više ne dobivaju nadogradnje ili instaliraju Android na uređaje koji nemaju Android kao službeni operacijski sustav. Primjer za to je HP-ov „Touch Pad”. Također zajednica se bavi i poboljšavanjem specifikacija uređaja s podizanjem brzine rada procesora uređaja.[1]

Službeni proizvođači uređaja su oduvijek bili sumnjičavi prema aplikacijama razvijenima od strane običnih programera unatoč tome što su mnogo doprinijeli samom razvoju. U nekim zemljama je dozvoljen „jailbreaking”, odnosno otključavanje pametnih uređaja za neke kompanije poput HTC-a, Motorole, Samsunga i Sonya. To je doprinijelo proizvodnji već otključanih uređaja poput Google-ovog Nexus-a, no dosta dobavljača uređaja još uvijek zahtjeva da oni ostanu zaključani.[1]

2.4. Sigurnost

Sve Android aplikacije pokrenute su u tzv. „Sandboxu” tj. okružju gdje ne ovise niti utječu jedna na drugu te nemaju mogućnost pristupa resursima samoga sustava, osim ako im to nije dozvoljeno. Prilikom instalacije neke aplikacije, Google Play Store prikazuje sve potrebne dozvole da bi se aplikacija mogla ispravno pokrenuti i koristiti. Npr. za neku igru su potrebne dozvole za vibraciju uređaja ili za spremanje podataka na vanjsku memoriju, dok će neka aplikacija za slikanje tražiti dozvolu za pristup kameri uređaja itd.



Slika 3. Primjer lažnog virusa[9]

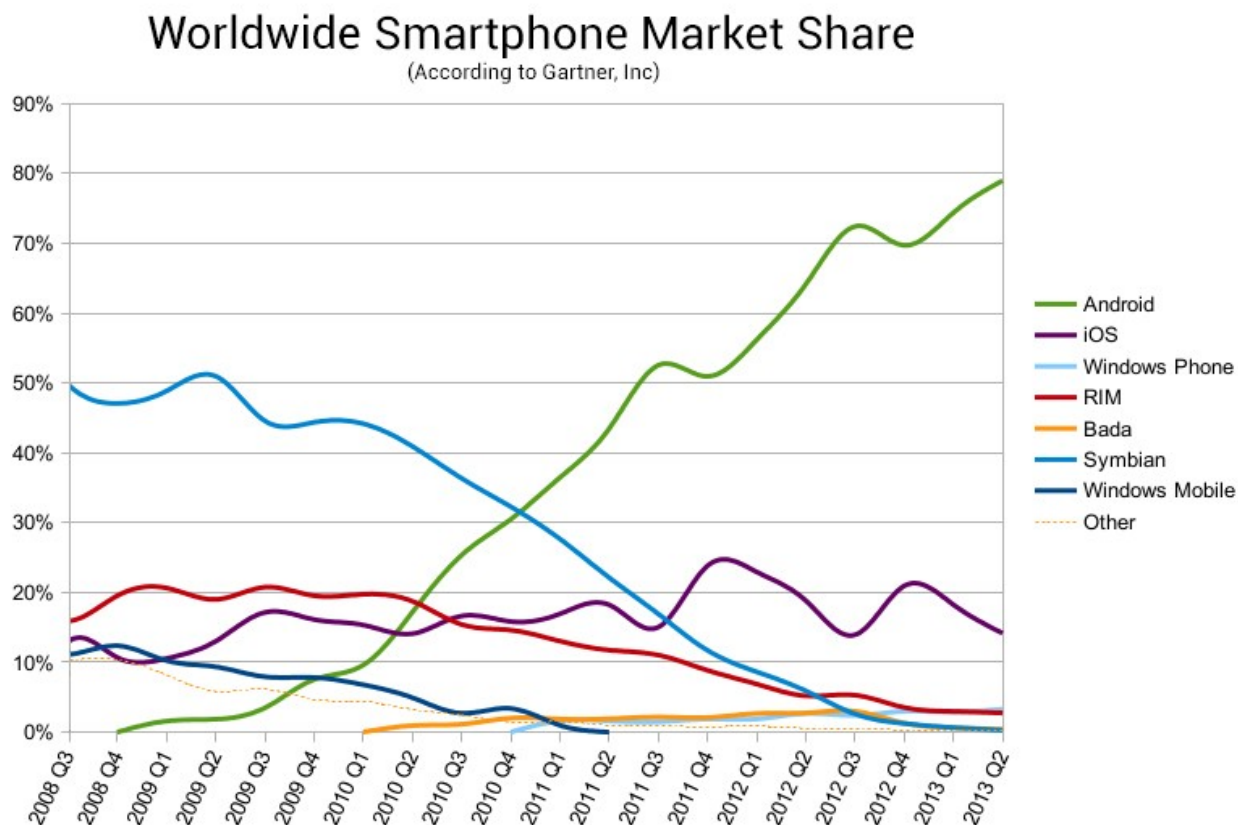
Na posljednjoj verziji Androida, 6.0 „Marshmallow”, izmijenjen je sustav dozvola tako da sam korisnik može kontrolirati samostalno sve dozvole za neku aplikaciju. Kontrolu aplikacija sa svim dozvolama ima samo „root” korisnik.

Najčešći zloćudni *software* na Androidu je *malware*. Android uređaj zaražen *malware-om* najčešće nekontrolirano šalje SMS poruke na druge uređaje, odnosno brojeve iz imenika, a da to sam vlasnik uređaja ne zna. Ostale vrste *malware-a* prikazuju neželjeni reklamni sadržaj na uređaju ili šalje osobne podatke na nepouzdana odredišta. Sigurnosne prijetnje na Android uređajima su iz dana u dan sve veće, iako su Googleovi inženjeri poprilično sigurni kako je riječ o marketinškom triku kako bi veće kompanije prodale svoje antivirusne aplikacije.[1]

Google koristi vlastiti „Google Bouncer“ s kojim skenira sve aplikacije na Google Play Storeu i označava sve sumnjive aplikacije kao opasnim kako bi se kasnije moglo obavijestiti korisnike o mogućim posljedicama prilikom instalacije. Kao i za računala, za Androide također postoje antivirusni programi, a neki od najpoznatijih su „Lookout Mobile Security“, „AVG“ i „McAfee“.

2.5. Android tržište

Krajem 2009. godine Android je imao udio od 2.8% svih pametnih uređaja na svijetu, a već u zadnjem kvartalu 2010. godine taj broj je narastao na 33% te tako postao najprodavanija platforma pametnih uređaja. Do 3. kvartala 2012. godine Android je preuzeo čak 75% ukupnog svjetskog tržišta pametnih uređaja te se ta brojka do danas nije previše promijenila. U 2014. godini Google je oborio rekord s do tada više od milijardu prodanih Android uređaja koji je na kraju 2014. još porastao na 1,16 milijardi, što je 4 puta više od iOS-a te 3 puta više od Windows mobilnih uređaja.[1][4]



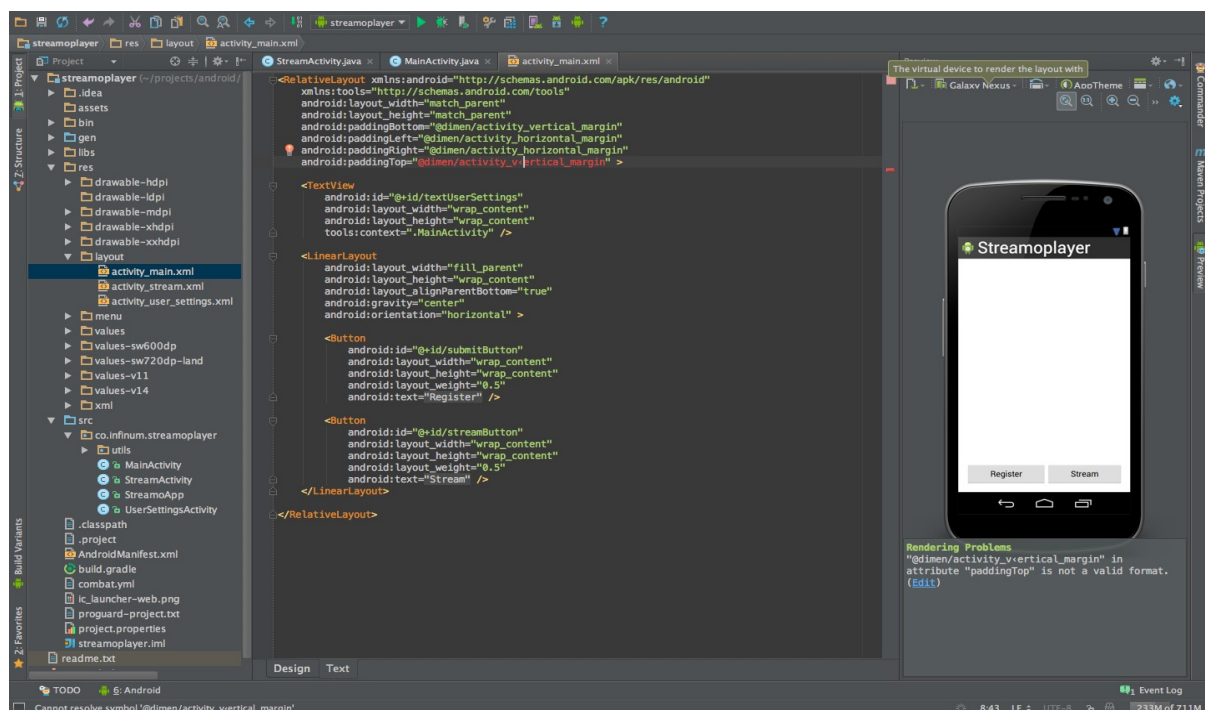
Slika 4. Grafički prikaz rasta popularnosti Androida od 2008. godine[12]

3. Alati i programski jezici korišteni za izradu aplikacije

3.1. Android Studio

„Android Studio“ je službeno integrirano razvojno okruženje za razvoj na Android platformi. Prva verzija Android Studia bila je najavljena još u svibnju 2013. godine, a prva stabilna verzija izašla je krajem 2014. godine.

Sam Android Studio osnovan je na „JetBrainsovom IntelliJ IDEA“ razvojnom okruženju i omogućen je na Windows, Linux i Mac OS X operacijskim sustavima te je zamijenio Eclipse u razvoju Android aplikacija. Android Studio koristi „Gradle“ sustav za izgradnju aplikacija koji je otvorenog koda i koristi koncepte „Apache Mavena“ i „Apache Anta“ te uz programski jezik „Groovy“ omogućuje konfiguraciju cijelog projekta.[7]



Slika 5. Izgled Android Studio sučelja[11]

Velika moć Android Studia nalazi se u njegovom emulatoru koji omogućava postavljanje specifikacija virtualnog uređaja na kojem je moguće testirati aplikaciju koja se razvija. Također postoje još neki emulatori koji se mogu koristiti uz Android Studio kao npr. Genymotion.

3.2. Java

Java je visoki objektno orijentiran programski jezik razvijen od strane kompanije Sun Microsystems 1995. godine s ciljem da se bilo koja aplikacija napisana u Javi može izvršiti na svim operacijskim sustavima koji je podržavaju.[2][4]

Java platforma se sastoji od prevedenog Java koda („Java bytecode“), odnosno strojnog jezika od Java virtualnog uređaja („Java Virtual Machine“) koji će izvršiti Java kod neovisno o operacijskom sustavu na kojem se nalazi. Iako je zbog tog dodatnog sloja u prevođenju koda izvršavanje Java aplikacija nešto sporije nego kod npr. programskih jezika C i C++, u današnje vrijeme je ta razlika puno manja jer postoje „real-time“ prevoditelji koda, odnosno „just-in-time“ kompajleri.[2]

Trenutno postoje tri Java verzije, a to su „Java Standard Edition“ (Java SE) koja služi uglavnom za razvoj desktop aplikacija, „Java Enterprise Edition“ (Java EE) koja služi za razvoj web aplikacija te „Java Micro Edition“ (Java ME) koja služi za razvoj aplikacija na manjim i resursima ograničenim uređajima.

Današnji vlasnik Java SE verzije je Oracle te su napravili dvije vlastite distribucije:

- JRE („Java Runtime Environment“) sadrži JVM i namijenjen je običnim korisnicima za pokretanje Java aplikacija.
- JDK („Java Development Kit“) sadrži sve što JRE i još dodatne alate za razvoj Java aplikacija kao što su Java kompajler, Javadoc i debugger te je namijenjen razvijateljima.

Trenutna verzija Jave je 8 i jedan je od najpopularnijih programskih jezika u svijetu upravo zbog kompatibilnosti i ogromne zajednice. Trenutna statistika pokazuje da u svijetu postoji 9 milijuna Java programera.[2]

3.3. XML

Iako je moguće sam izgled Android aplikacije definirati programski u Javi, danas se najčešće izgled Android aplikacije definira u jeziku za označavanje podataka, odnosno u jeziku XML (*Extensible Markup Language*). XML se u računalnom svijetu koristi u razne svrhe kao npr. razmjenu podataka, odvajanje podataka od prezentacije te kao temelj za razvoj novih jezika za označavanje. Sam jezik je vrlo sličan HTML (*HyperText Markup Language*) jeziku koji služi za označavanje izgleda web stranica. Sličnu funkciju kod Android aplikacija ima i XML u kojem njegove oznake zapravo označavaju Android elemente tj. kontrole na koje se dodaju različiti atributi kao npr. visina, širina, id, boja itd. Potrebno je naglasiti da je jedinica vrijednosti za attribute kao što su npr. „layout_width“ (*širina*) i „layout_height“ (*visina*) „Density-independent pixel“ ili skraćeno „dp“ koji je podržan na svim uređajima i neovisan o rezoluciji ekrana. Unutar Android aplikacije XML, osim za prikazivanje izgleda aktivnosti, se koristi još i za spremanje različitih vrijednosti kao stringova, boja, postavki itd., te se razvrstavaju u različite direktorije ovisno o namjeni (npr. XML datoteke za izgled aplikacije se postavljaju unutar *layout* direktorija). Svaka XML datoteka uglavnom predstavlja jedan pogled, odnosno izgled neke aktivnosti unutar aplikacije te mora sadržavati korijenski element unutar kojega se dodaju ostali elementi. Jako je bitno da se prilikom dodavanja Android atributa koristi imenski prostor "android".[8]

Neke od najosnovnijih Android kontrola su:

- TextView - Android XML element koji predstavlja labelu unutar aplikacije
- EditText - predstavlja element za unos teksta
- ImageView - predstavlja element za prikazivanje slikanje
- Spinner - predstavlja element za odabir vrijednosti iz zadane liste
- Button - predstavlja gumb na čiji se klik/dodir obično izvršava neka funkcija
- LinearLayout - kontejner element unutar kojega se elementi mogu pozicionirati u stupcu ili retku. Ukoliko nije potrebno, ne bi se trebao koristiti unutar RelativeLayout elementa za dodatno pozicioniranje zbog dodatnog učitavanja što usporava aplikaciju.
- RelativeLayout - jedan od najčešće korištenih elemenata koji predstavlja kontejner unutar kojega se ostali elementi mogu pozicionirati u međusobnoj relaciji ili u relaciji s elementom koji im je roditelj.

Primjer jednostavnog XML Android pogleda:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
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"
    tools:context=".activity.invoice.InvoiceDetailsActivity"
    >

<!-- Name -->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/name"
    android:id="@+id/label_name" />

<EditText
    android:id="@+id/edit_invoice_name"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@id/label_name"
    android:hint="@string/edit_invoice_name" />

<Button

    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/btn_save_invoice"
    android:text="@string/button_save"
    android:textSize="20dp"
    android:layout_below="@id/edit_invoice_name"
    android:layout_centerHorizontal="true"
    android:paddingLeft="30dp"
    android:paddingRight="30dp" />

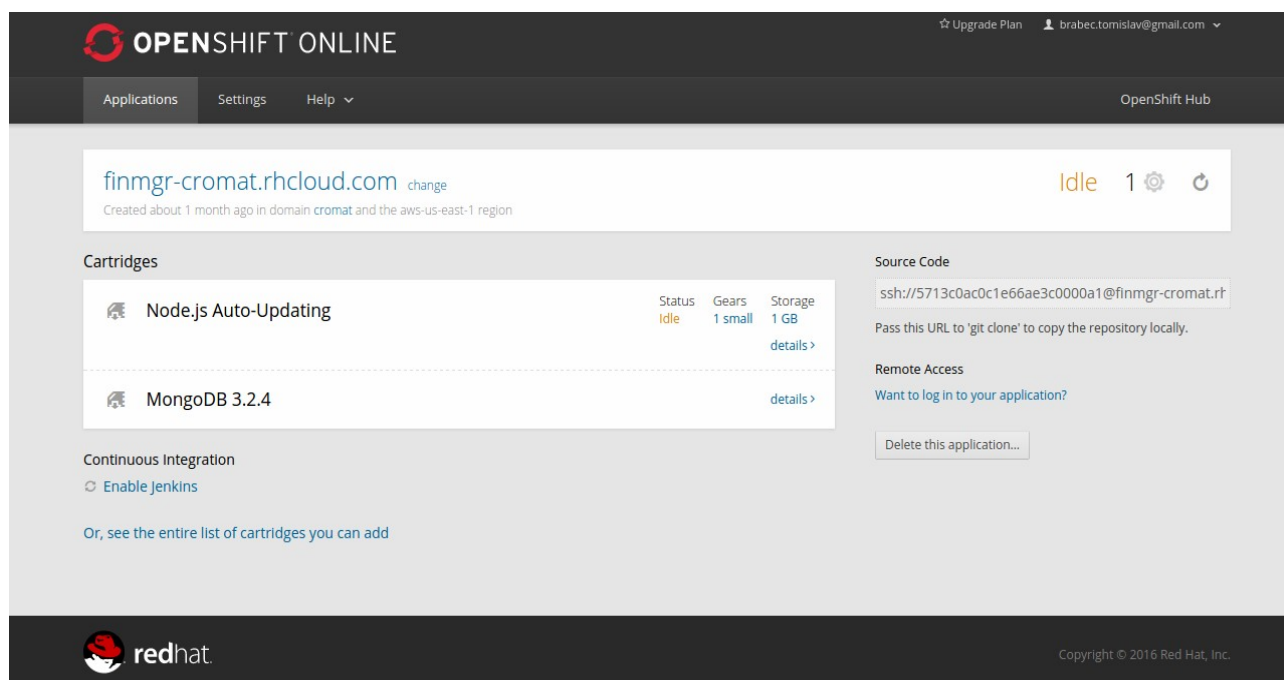
</RelativeLayout>
```

Kod 1. Jednostavan primjer XML pogleda

U ovom jednostavnom XML pogledu prva linija `<?xml version="1.0" encoding="utf-8"?>` označava verziju i vrstu enkodiranja XML dokumenta. Korijenski element je *RelativeLayout* te se njegovi atributi mogu postavljati redom odmah iza imena oznake, no radi preglednosti obično se svaki atribut postavlja u novi redak. Prvi atribut unutar *RelativeLayouta* `xmlns:android="http://schemas.android.com/apk/res/android"` postavlja putanju imenskog prostora „android“ na zadanu lokaciju. Isto vrijedi i za sljedeće dvije linije koda koje postavljaju imenske prostore „app“ i „tools“ na njihove lokacije da se kasnije mogu koristiti bez pisanja cijele putanje. Linije `android:layout_width="match_parent"` i `android:layout_height="match_parent"` označavaju visinu i širinu korijenskog elementa koje su u ovom slučaju postavljene na „match_parent“ koji označava maksimalnu vrijednost od roditelja elementa. S obzirom da je u ovom slučaju *RelativeLayout* korijenski element, on će poprimiti maksimalnu visinu i širinu samog uređaja. Posljednji atribut unutar *RelativeLayout* elementa `tools:context` postavlja kontekst pogleda tako da bi se pogled mogao povezati s pravilnom temom ukoliko ih više postoji. Unutar ostalih elemenata također se mogu uočiti već spomenuti atributi „layout_width“ i „layout_height“, ali s vrijednosti „wrap_content“ koja zapravo govori da će se element po visini „omotati“ oko sadržaja elementa, odnosno poprimiti će visinu sadržaja unutar elementa. Jedan od najvažnijih atributa je `android:id` preko kojeg se elementi mogu vezati s Java kodom za daljnju programsku uporabu i manipulaciju. Elementi *EditText* i *TextView* u ovom slučaju sadrže još attribute `android:text` i `android:hint` gdje „text“ označava unaprijed zadanu tekstualnu vrijednost na elementu, dok „hint“ označava tekst u pozadini elementa ukoliko je prazan te daje do znanja što bi trebalo biti uneseno u polje. Unutar *Button* elementa nalaze se atributi `android:textSize` koji označava veličinu teksta na gumbu, `android:layout_centerHorizontal="true"` koji označava poziciju gumba unutar roditeljskog elementa koja je u ovom slučaju na sredini, `android:paddingLeft` i `android:paddingRight` koji označavaju proširenje unutar gumba s lijeve i desne strane te atribut `android:layout_below` koji označava pozicioniranje ispod nekog elementa koji se također nalazi unutar *RelativeLayout* elementa. Postoji još atributa za pozicioniranje unutar *RelativeLayout* elementa kao i svih ostalih atributa različite namjene koji će biti obrađeni u sljedećem dijelu ovog završnog rada.

3.4. Online server

Prilikom izrade aplikacije za pohranu podataka je korišten Parse *backend* servis koji je zasnovan na Node.js *frameworku* i MongoDB bazi podataka. Da bi se Parse uspješno izvršavao, potrebno je imati osposobljen Node.js server. Za tu svrhu u ovoj aplikaciji korišten je besplatni online servis za razvijatelje pod nazivom OpenShift kojeg je razvila američka kompanija Red Hat. OpenShift je besplatan servis, zasnovan na Docker i Kubernetes kontejnerima. Veoma je fleksibilan i služi kao hosting uz ponudu raznih programskih jezika, baza podataka i *frameworka* te je ponajviše korišten od strane razvijatelja za razna testiranja, ali i ozbiljnije projekte. Također sami programeri mogu dodati vlastite jezike i dodatke.[6]



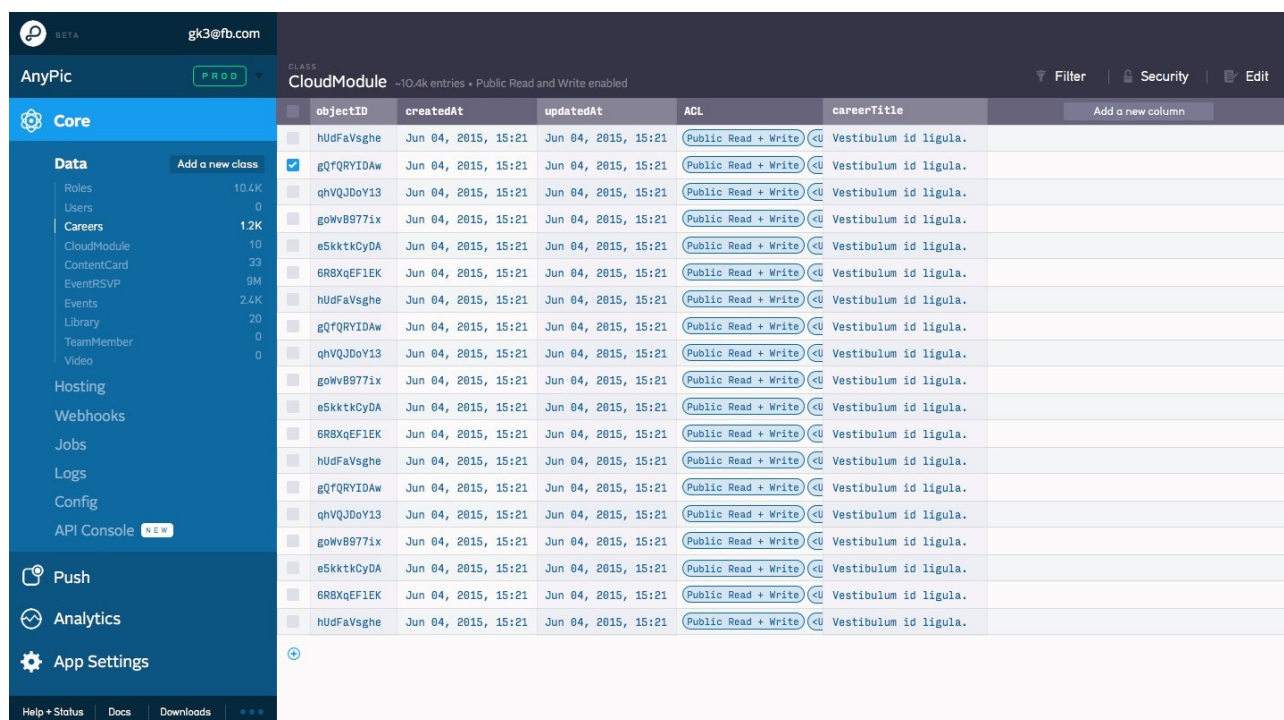
Slika 6. Izgled OpenShift online aplikacije

Sučelje Openshift online aplikacije nudi jednostavan prikaz instaliranih dodataka na projektu, omogućava *start*, *stop* i *restart* aplikacije te također nudi pristup projektu preko *ssh* klijenta.

3.5. Parse komunikacija sa serverom

Kao što je već ranije spomenuto, za komunikaciju sa serverom aplikacija koristi Parse. Parse je mobilni *backend* servis koji omogućuje jednostavnu komunikaciju između mobilnog uređaja i MongoDB baze podataka na serveru. MongoDB spada u nerelacijske „NoSQL“ baze podataka te podatke drži spremljene u JSON (*JavaScript Object Notation*) obliku. Takav način omogućuje jednostavniju i bržu migraciju podataka, što ovu bazu čini pogodnijom za komunikaciju s mobilnim uređajima.[5]

Parse je nedavno prestao sa svojim radom na službenim serverima te je postao otvorenog koda, što razvijateljima omogućava instalaciju Parse servisa na vlastitom računalu ili serveru. Također, uz Parse server instalaciju, moguće je instalirati i Parse *dashboard* koji omogućuje grafički prikaz baza i podataka te njihov unos.



The screenshot shows the Parse dashboard interface. On the left is a sidebar with navigation options: Core, Data, Hosting, Webhooks, Jobs, Logs, Config, API Console, Push, Analytics, and App Settings. The main area displays a table for the 'CloudModule' class, which has 10.4k entries. The table columns are objectID, createdAt, updatedAt, ACL, and careerTitle. The data rows show various object IDs and their corresponding career titles, all with 'Public Read + Write' ACL permissions.

objectID	createdAt	updatedAt	ACL	careerTitle
hUdFaVsghe	Jun 04, 2015, 15:21	Jun 04, 2015, 15:21	Public Read + Write	Vestibulum id ligula.
gQfQRYIDAw	Jun 04, 2015, 15:21	Jun 04, 2015, 15:21	Public Read + Write	Vestibulum id ligula.
qhVQJDoY13	Jun 04, 2015, 15:21	Jun 04, 2015, 15:21	Public Read + Write	Vestibulum id ligula.
goWvB977ix	Jun 04, 2015, 15:21	Jun 04, 2015, 15:21	Public Read + Write	Vestibulum id ligula.
e5kktkCyDA	Jun 04, 2015, 15:21	Jun 04, 2015, 15:21	Public Read + Write	Vestibulum id ligula.
6R8XqEF1EK	Jun 04, 2015, 15:21	Jun 04, 2015, 15:21	Public Read + Write	Vestibulum id ligula.
hUdFaVsghe	Jun 04, 2015, 15:21	Jun 04, 2015, 15:21	Public Read + Write	Vestibulum id ligula.
gQfQRYIDAw	Jun 04, 2015, 15:21	Jun 04, 2015, 15:21	Public Read + Write	Vestibulum id ligula.
qhVQJDoY13	Jun 04, 2015, 15:21	Jun 04, 2015, 15:21	Public Read + Write	Vestibulum id ligula.
goWvB977ix	Jun 04, 2015, 15:21	Jun 04, 2015, 15:21	Public Read + Write	Vestibulum id ligula.
e5kktkCyDA	Jun 04, 2015, 15:21	Jun 04, 2015, 15:21	Public Read + Write	Vestibulum id ligula.
6R8XqEF1EK	Jun 04, 2015, 15:21	Jun 04, 2015, 15:21	Public Read + Write	Vestibulum id ligula.
hUdFaVsghe	Jun 04, 2015, 15:21	Jun 04, 2015, 15:21	Public Read + Write	Vestibulum id ligula.
gQfQRYIDAw	Jun 04, 2015, 15:21	Jun 04, 2015, 15:21	Public Read + Write	Vestibulum id ligula.
qhVQJDoY13	Jun 04, 2015, 15:21	Jun 04, 2015, 15:21	Public Read + Write	Vestibulum id ligula.
goWvB977ix	Jun 04, 2015, 15:21	Jun 04, 2015, 15:21	Public Read + Write	Vestibulum id ligula.
e5kktkCyDA	Jun 04, 2015, 15:21	Jun 04, 2015, 15:21	Public Read + Write	Vestibulum id ligula.
6R8XqEF1EK	Jun 04, 2015, 15:21	Jun 04, 2015, 15:21	Public Read + Write	Vestibulum id ligula.
hUdFaVsghe	Jun 04, 2015, 15:21	Jun 04, 2015, 15:21	Public Read + Write	Vestibulum id ligula.

Slika 7. Izgled Parse dashboard sučelja[10]

Parse koristi više od 500 000 Android aplikacija zbog jednostavne instalacije i integracije u Java kodu. Za instalaciju Parse SDK za Android dovoljno je u *gradle* datoteku pod *dependencies* dodati compile 'com.parse:parse-android:1.13.0'.

```
Parse.initialize(new  
Parse.Configuration.Builder(getApplicationContext())  
    .applicationId(AppConfig.APP_ID)  
    .server(AppConfig.PARSE_LINK)  
    .build())
```

Kod 2. Primjer povezivanja s Parse serverom

Nakon sinkronizacije projekta postaju dostupne Parse klase. Primjer inicijalizacije konekcije s Parse serverom prilikom pokretanja aplikacije:

Jednostavan primjer spremanja testnog objekta:

```
ParseObject testObject = new ParseObject("TestObject");  
testObject.put("foo", "bar");  
testObject.saveInBackground();
```

Kod 3. Primjer spremanja Parse objekta

Neki ostali primjeri korištenja Parse SDK bit će detaljno objašnjeni prilikom opisa izrade aplikacije.

4. Izrada aplikacije

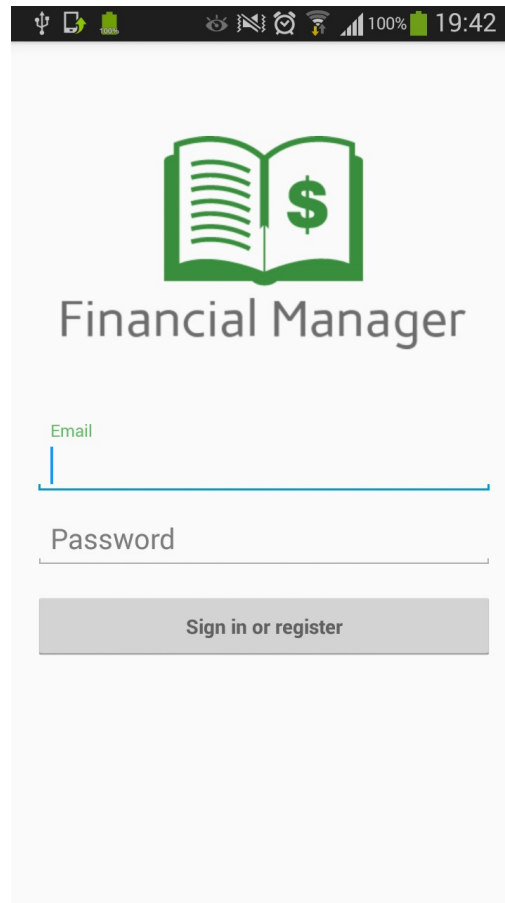
U ovom dijelu završnog rada detaljno će bit objašnjena izrada Android aplikacije za praćenje osobnih financija.

Smisao aplikacije je da pomogne korisnicima Android uređaja da lako nadgledaju svoje troškove, poreze, dionice, ali i fondove u kojima ostvaruju udio.

Android aplikacija za praćenje osobnih financija sastoji se od nekoliko cjelina, a to su računi, fondovi, dionice i porezi. Svaka od tih cjelina sadrži pogled za pregled njihovih lista, pogled za dodavanje i uređivanje te pogled za prikaz detalja pojedinog objekta. Svaka cjelina ima ostvarenu komunikaciju s online bazom podataka, ali sadrži i vlastitu lokalnu *SQLite* bazu koja se brine da podaci unatoč lošijoj internetskoj vezi uvijek budu pristupni korisniku.

Većina klasa ima slične (obavezne) metode te sličnu strukturu, npr. u aplikaciji postoji pogled za prikaz računa i pogled za prikaz fondova te su oni gotovo identični pa će kod biti opisan samo na jednom primjeru, tj. računima.

4.1. Prijavljivanje i registracija korisnika ("LoginActivity" klasa)



Slika 8. Pogled za prijavu

Ukoliko korisnik prvi puta pokreće aplikaciju ili nije prije bio prijavljen, prikazuje mu se pogled za prijavu. Ukoliko je korisnik već prijavljen, aplikacija ga preusmjerava na *MainActivity*, odnosno pogled za prikaz liste svih računa. Na istom pogledu za prijavu korisnik se može i registrirati tako da u polja unese željene informacije. Prilikom pritiska gumba za prijavu ili registraciju, korisniku se javlja prozorčić s upitom želi li on stvoriti novoga korisnika s unesenim informacijama. Ukoliko pritisne "Yes", bit će registriran novi korisnik i preusmjeren na *MainActivity*, a ukoliko pritisne "No", prozorčić će se maknuti i korisnik će ostati na istom pogledu.

Prilikom razvoja Android aplikacija, *activity* klase nasljeđuju klasu *Activity* ili neku od njenih izvedenica, što ovisi o želji samog korisnika.

```
public class LoginActivity extends Activity implements Validator.ValidationListener
```

Kod 4. Definicija LoginActivity klase

Gore napisana linija koda označava definiciju *LoginActivity* klase koja nasljeđuje *Activity* klasu i implementira sučelje *ValidatorListener* koji služi za provjeru unesenih vrijednosti u određenim poljima, a detaljnije će biti opisan u nastavku. Sve funkcije i varijable *LoginActivity* klase nalaze se unutar vitičastih zagrada iza deklaracije iste klase.

Primjer deklaracije varijabli u *LoginActivity* klasi:

```
@NotEmpty
@email
private autoCompleteTextView email;
@NotEmpty
@Password
private EditText password;
private Button signInButton;
String usernameTxt;
...
```

Kod 5. Deklaracija varijabli unutar LoginActivity klase

Anotacije *@NotEmpty*, *@Email* i *@Password* dolaze iz paketa za validaciju polja te označavaju zahtjeve koje određena polja unutar na nekoj formi u pogledu moraju imati da bi validacija prošla.

4.1.1. Metoda *findViews()*

```
private void findViews() {
    email = (AutoCompleteTextView)findViewById( R.id.email );
    password = (EditText)findViewById( R.id.password );
    signInButton = (Button)findViewById( R.id.sign_in_button );
}
```

Kod 6. Metoda za povezivanje Java kontrola s XML elementima

Metoda `findViews()` služi za povezivanje varijabli iz Java koda s elementima unutar XML-a preko njihovih *id* vrijednosti. Iako se povezivanje često vrši direktno u `onCreate()` metodi, radi lakšeg snalaženja i boljeg pregleda, unutar aplikacije se uvijek koristila ova funkcija koja se tada poziva u `onCreate()` metodi.

4.1.2. Metoda `onCreate()`

Svaka klasa koja nasljeđuje *Activity* klasu, odnosno koja i sama predstavlja aktivnost, mora implementirati metodu `onCreate()`. Metoda `onCreate()` se poziva prilikom prvog pokretanja neke aktivnosti i u njoj se obično vrši inicijalizacija varijabli, postavljanje `onClickListener()` funkcija na gumbе koje označavaju radnju koja će se izvršiti ukoliko korisnik klikne na neki gumb i slično.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);}
```

Kod 7. Primjer `onCreate()` metode

Metoda `onCreate()` kao parametar prima *Bundle* objekt koji je uvijek prilikom prvog poziva `onCreate()` metode jednak *null*, a ukoliko se `onCreate()` metoda programski pozove, može se proslijediti *Bundle* objekt u kojem mogu biti pohranjene neke postavke pogleda te se iskoristiti. Zbog veličine `onCreate()` metode unutar *LoginActivity* klase, biti će redom opisani dijelovi koda unutar nje.

4.1.3. Komunikacija *LoginActivity* klase s Parse serverom

Za povezivanje s Parse serverom koristi se *initialize()* statička metoda koja kao argument prima *Configuration* objekt koji sadrži identifikacijsku oznaku aplikacije te link za spajanje na Parse server. Nakon povezivanja s Parse serverom vrši se provjera prijavljenog korisnika.

```
ParseUser user = ParseUser.getCurrentUser();  
if (user != null)  
    startActivity(new Intent(getApplicationContext(), MainActivity.class));
```

Kod 8. Provjera prijavljenoga korisnika

S funkcijom *ParseUser.getCurrentUser()* dohvaća se trenutno prijavljeni korisnik te ukoliko niti jedan korisnik nije prijavljen u aplikaciju na uređaju, ova funkcija će vratiti *null* vrijednost te će stoga biti potrebno izvršiti prijavu, a u suprotnom, ako je korisnik prijavljen, biti će preusmjeren na glavnu aktivnost odnosno *MainActivity*.

Unutar *onCreate()* metode obavezno je postaviti pogled na koji se aktivnost veže, a to se postiže metodom *setContentView()* koja kao argument prima *id* XML pogleda.

```
setContentView(R.layout.activity_login);
```

Kod 9. Postavljanje XML pogleda na aktivnost

4.1.4. Gumb za prijavu korisnika

```
signInButton.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        validator.validate();
        if (AppConfig.isNetworkAvailable(getApplicationContext())) {
            if (validated) {
                usernametxt = email.getText().toString();
                passwordtxt = password.getText().toString();
                ParseUser.logInInBackground(usernametxt, passwordtxt,
                    new LogInCallback() {
                        public void done(ParseUser user, ParseException e) {
                            if (user != null) {
                                Intent intent = new Intent(
                                    LoginActivity.this,
                                    MainActivity.class);
                                startActivity(intent);
                                Toast.makeText(getApplicationContext(),
                                    "Successfully Logged in",
                                    Toast.LENGTH_LONG).show();
                                finish();
                            } else {
                                Toast.makeText(
                                    getApplicationContext(),
                                    "No such user exist!",
                                    Toast.LENGTH_LONG).show();
                                builder.show();
                            }
                        }
                    }
                ));
            }
        } else {
            Toast.makeText(getApplicationContext(), "No internet
connection!", Toast.LENGTH_LONG).show();
        }
    }
});
```

Kod 10. Izvršavanje akcije prilikom pritiska gumba za prijavu

Prilikom pritiska gumba za prijavu moraju se izvršiti neke provjere prije nego što se korisniku dozvoli pristup aplikaciji. Prva je provjera povezanosti s internetom jer ukoliko nema povezanosti, nije moguće izvršiti komunikaciju sa serverom i obaviti provjeru korisnika. Provjera povezanosti obavljena je funkcijom *isNetworkAvailable()* koja je definirana kao statička metoda u klasi *AppConfig*. Kao parametar u funkciju se proslijeđuje *Context*, a definicija funkcije izgleda ovako:

```
public static boolean isNetworkAvailable(final Context context) {  
    final ConnectivityManager connectivityManager = ((ConnectivityManager)  
context.getSystemService(Context.CONNECTIVITY_SERVICE));  
    return connectivityManager.getActiveNetworkInfo() != null &&  
connectivityManager.getActiveNetworkInfo().isConnected();  
}
```

Kod 11. Provjera internetske veze

Ako provjera povezanosti s internetom nije prošla, korisnik će biti obaviješten o tome u kratkotrajnom oblačiću koji se dobiva statičkom metodom *makeText()* iz klase *Toast*. Nakon provjere povezanosti s internetom, provjerava se je li je validacija svih polja prošla ispravno. To se postiže provjerom *bool* varijable *validated* koja je postavljena na „true“ unutar *onValidationSucceded()* metode koja se nasljeđuje iz prije spomenutog sučelja *Validator.ValidatorListener*, a poziva se samo kada validacija nad poljima s *validator* anotacijama prođe bez greške. U suprotnom se poziva metoda *onValidationFailed()* unutar koje se može obavijestiti korisnika o eventualnim pogreškama prilikom unosa.

Ukoliko i validacija polja prođe, potrebno je pokušati prijaviti korisnika tako da se u metodu *loginInBackground()* proslijede uneseno korisničko ime (*e-mail*) i lozinka te koja vraća *ParseUser* objekt koji, ako je uspješno dohvaćen, omogućava prijavu i preusmjerenje na *MainActivity* unutar aplikacije. U suprotnom će korisniku opet biti ispisana kratkotrajna poruka na ekranu, no ovoga puta da korisnik ne postoji, te se pojavljuje prozorčić s upitom želi li se korisnik registrirati u aplikaciju s unesenim korisničkim imenom i lozinkom. Ukoliko korisnik pritisne „Yes“, bit će pozvana metoda *signupInBackground()* i registrirat će se novi korisnik.

4.1.5. "Alert Dialog" prozorčić

```
dialogClickListener = new DialogInterface.OnClickListener() {

    @Override
    public void onClick(DialogInterface dialog, int which) {
        switch (which){
            case DialogInterface.BUTTON_POSITIVE:
                //Yes button clicked
                dialogUser = new ParseUser();
                dialogUser.setUsername(usernameetxt);
                dialogUser.setPassword(passwordetxt);
                dialogUser.signUpInBackground(new SignUpCallback() {
                    public void done(ParseException e) {
                        if (e == null) {
                            Toast.makeText(getApplicationContext(),
                                "Successfully Signed up.",
                                Toast.LENGTH_LONG).show();
                        } else {
                            if (dialogUser != null)
                                Toast.makeText(getApplicationContext(),
                                    "User already exists!", Toast.LENGTH_LONG)
                                    .show();
                            else
                                Toast.makeText(getApplicationContext(),
                                    "Sign up Error", Toast.LENGTH_LONG)
                                    .show();
                        }
                    }
                });
                break;
            case DialogInterface.BUTTON_NEGATIVE:
                break;
        }
    }
};

builder = new AlertDialog.Builder(new ContextThemeWrapper(this, R.style.myDialog));
builder.setMessage("Do you want to register with email " + email.getText().toString() +
    " and given password?").setPositiveButton("Yes", dialogClickListener)
    .setNegativeButton("No", dialogClickListener);
```

Kod 12. Prikazivanje prozorčića s upitom o registraciji korisnika

AlertDialog klasa služi za prikazivanje raznih prozorčića u Android aplikacijama. U ovom slučaju pojavljuje se prozorčić s mogućnošću izbora „Yes“ ili „No“. *DialogInterface.OnClickListener()* omogućuje izradu funkcionalnosti prilikom pritiska na jedan od ponuđenih izbora unutar prozorčića. „BUTTON_POSITIVE“ označava pritisak na gumb „Yes“ nakon kojega se izvršava registracija korisnika ukoliko on već ne postoji, a „BUTTON_NEGATIVE“ označava pritisak na gumb „No“ te u ovom slučaju označava samo izlaz iz prozorčića.

Prije poziva prozorčića potrebno je instancirati *AlertDialog.Builder* objekt s kojim se poziva prozorčić. *AlertDialog.Builder* objekt kao parametar prima XML *style*, odnosno izgled prozorčića.

```
<style name="myDialog" parent="Theme.AppCompat.Dialog">
    <item name="android:windowNoTitle">true</item>
</style>
```

Kod 13. XML za prikaz Dialog prozorčića

Naknadno se na *AlertDialog.Builder* objekt može postaviti naslov, poruka te gumbi koji će biti prikazani na prozorčiću i koji su vezani na *dialogClickListener* objekt.

4.2. Upravljanje računima

4.2.1. "Invoice" klasa

Model računa se sastoji od proizvoljnog naziva, broja, broja kartice, datuma isteka, vrste kartice, svote, valute i banke u kojoj je račun izrađen. Uz te varijable, na svaki model dodani su još „id“ i „userId“ koji označavaju jedinstveni identifikator svakog objekta te identifikator korisnika u čijem je vlasništvu taj objekt.

```
private String id;

private String userId;
private String name;
private String invoiceNumber;
private String cardNumber;
private Date cardExpiry;
private String cardType;
private double balance;
private String currency;
private String bank;
```

Kod 14. Definicija varijabli računa

Uz konstruktore (jedan prazni i jedan koji prima sve vrijednosti računa), *getter* i *setter* metode za svaku varijablu dodane su još metode za parsiranje datuma u *String* i obrnuto.

4.2.1.1. Metoda *getCardExpiryYear()*

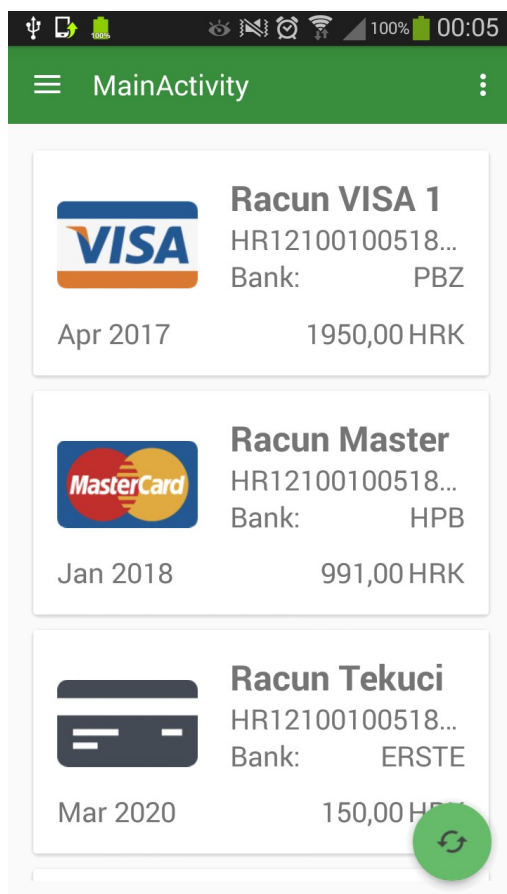
```
public String getCardExpiryYear() {

    SimpleDateFormat sdf = new SimpleDateFormat("EEE MMM dd HH:mm:ss z yyyy", Locale.US);
    Calendar c = Calendar.getInstance();
    try {
        c.setTime(sdf.parse(cardExpiry.toString()));
        return Integer.toString(c.get(Calendar.YEAR));
    }
    ...
}
```

Kod 15. Dohvaćanje godine isteka računa

Metoda *getCardExpiryYear()* služi za dohvaćanje godine u obliku *Stringa* od navedenog datuma isteka računa. Pomoću *SimpleDateFormat* objekta zadaju se lokalne postavke i format datuma koji će se parsirati u *String*. Objekt klase *Calendar*, koja služi za lakše rukovanje datumima, kao argument prima parsirani datum te na kraju vraća samo godinu.

4.2.2. "MainActivity" klasa



Slika 9. Pogled za prikaz svih računa

MainActivity glavna je aktivnost aplikacije i služi za prikazivanje liste s računima. Sadrži bočni izbornik iz kojega je moguće otvoriti aktivnosti za prikaz fondova, poreza i dionica, aktivnosti za dodavanje istih, aktivnost postavki ili se odjaviti. Također, sadrži izbornik i na naslovnoj traci iz kojega je moguće otvoriti aktivnost za dodavanje novog računa ili se odjaviti.

4.2.2.1. XML pogled *MainActivity* klase

```
<android.support.v7.widget.RecyclerView  
  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:id="@+id/recycler_main"  
    >  
</android.support.v7.widget.RecyclerView>  
<TextView android:id="@+id/text_no_invoices"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:textAlignment="center"  
    android:gravity="center_vertical"  
    android:visibility="gone"  
    android:text="@string/no_invoices"  
    android:textSize="16dp"  
    />
```

Kod 16. XML za MainActivity pogled

Main XML pogled sastoji se od dva elementa (*RecyclerView* i *TextView*) koji se nalaze unutar korijenskog *LinearLayout* elementa. *RecyclerView* služi za prikaz liste računa te je za razliku od *ListView* elementa optimiziraniji i sadrži efekte koji korisniku ostavljaju ugodniji osjećaj prilikom listanja. *TextView* element u ovom slučaju prikazuje se samo ukoliko nema niti jednog računa te o tome obavještava korisnika.

MainActivity klasa nasljeđuje *AppCompatActivity*, što omogućuje prikaz gornje trake s izbornikom na aplikaciji. Također, *MainActivity* implementira sučelje *NavigationView.OnNavigationItemSelectedListener* koje omogućuje implementaciju bočnog izbornika. Osim varijabli koje označavaju *RecyclerView* i *TextView* kontrole, *MainActivity* klasa sadrži još *LinearLayoutManager* objekt, *Adapter* za *RecyclerView*, listu računa te lokalnu *SQLite* bazu podataka koju u ovom slučaju predstavlja naslijeđena klasa *SQLiteHelper*.

Na početku *onCreate()* metode inicijalizira se objekt *SQLite* baze podataka računa, a zatim se odmah nakon toga pozivom metode *getWritableDatabase()* otvara konekcija s istom. Slično kao i kod *LoginActivity* klase, aplikacija se pokušava povezati s Parse serverom ukoliko već nije povezana.

4.2.2.2. "FloatingActionButton" gumb

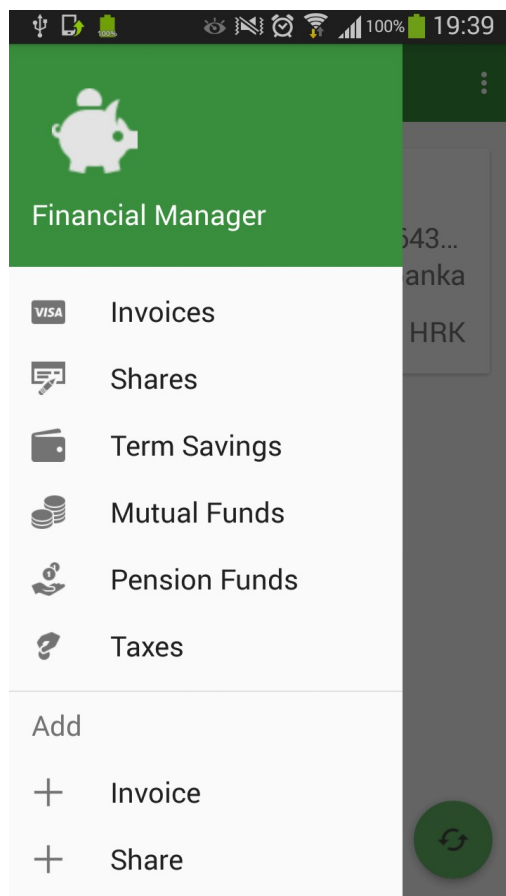
```
FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);

fab.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        dataSetChanged();
    }
});
```

Kod 17. Primjer definiranja FloatingActionButton objekta

Od Android API verzije 21 preferira se *Material Design* te uz njega i *FloatingActionButton* objekt koji predstavlja lebdeći gumb koji se nalazi iznad samog pogleda te bi uvijek trebao imati funkciju najvažnije i najčešće korištene radnje unutar aplikacije.

4.2.2.3. *DrawerLayout* kontejner



Slika 10. Bočni izbornik

```
DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);

ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
    this, drawer, toolbar, R.string.navigation_drawer_open,
    R.string.navigation_drawer_close);
drawer.setDrawerListener(toggle);
toggle.syncState();
NavigationView navigationView = (NavigationView) findViewById(R.id.nav_view);
navigationView.setNavigationItemSelectedListener(this);
```

Kod 18. Definiranje *DrawerLayout* objekta te postavljanje *NavigationView* izbornika

DrawerLayout objekt predstavlja kontejner za *NavigationView* koji je ujedno i bočni izbornik na kojeg se naknadno vežu akcije za pojedinu stavku izbornika.

4.2.2.4. Metoda `onNavigationItemSelectedListener()`

Za povezivanje stavki izbornika s akcijama koristi se metoda `onNavigationItemSelectedListener()`.

```
@Override
public boolean onNavigationItemSelectedListener(MenuItem item) {
    int id = item.getItemId();
    Intent intent = AppConfig.getStartingActivity(id, getApplicationContext());
    String name = (String) intent.getSerializableExtra("name");
    if(name.equals(LoginActivity.class.getName())) {
        ParseUser.logout();
        startActivity(intent);
    }
    if(name.equals(this.getClass().getName()))
        dataSetChanged();
    else
        startActivity(intent);
    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
    drawer.closeDrawer(GravityCompat.START);
    return true;
}
```

Kod 19. Primjer funkcije koja započinje novu akciju ovisno o odabranoj stavci bočnog izbornika

U ovom slučaju zbog velike količine stavki unutar izbornika i ponavljanja koda, u `AppConfig` je postavljena statička metoda `getStartingActivity()` kojoj se proslijeđuje `id` odabrane stavke i kontekst te koja vraća odabranu stavku izbornika. Primjer povratne vrijednosti iz `getStartingActivity()` metode ukoliko je na izborniku odabran prikaz svih računa:

```
if (id == R.id.nav_invoices) {
    Intent intent = new Intent(context, MainActivity.class);
    intent.putExtra("name", MainActivity.class.getName());
    intent.addFlags(Intent.FLAG_ACTIVITY_NO_HISTORY);
    return intent;
}
```

Kod 20. Primjer vraćanja `Intent` objekta u slučaju odabira pregleda svih računa u bočnom izborniku

Ukoliko je korisnik odabrao u izborniku stavku za odjavu, tada se izvršava odjava preko *ParseUser* klase koristeći metodu *logout()*, a ukoliko je korisnik odabrao istu aktivnost u izborniku i u kojoj se trenutno nalazi, tada će se samo osvježiti podaci na ekranu.

4.2.2.5. Metoda *parseFetchInvoices()*

```
public void parseFetchInvoices(){

    searching = true;
    ParseQuery<ParseObject> query = new ParseQuery<ParseObject>("Invoices");
    query.whereEqualTo("user_id",
ParseUser.getCurrentUser().getObjectId()).findInBackground(new FindCallback<ParseObject>()
{
    public void done(List<ParseObject> invoicesParse, ParseException e) {
        if (e == null) {
            for (ParseObject invoiceObject : invoicesParse) {
                Invoice invoice = new Invoice();
                invoice.setId(invoiceObject.getObjectId());
                invoice.setUserId(invoiceObject.getString("user_id"));
                invoice.setName(invoiceObject.getString("name"));
                invoice.setInvoiceNumber(invoiceObject.getString("invoice_number"));
                invoice.setCardNumber(invoiceObject.getString("card_number"));
                invoice.setCardExpiry(invoiceObject.getString("card_expiry"));
                invoice.setCardType(invoiceObject.getString("card_type"));
                String balanceTmp = invoiceObject.getString("balance");
                if (balanceTmp.equals(""))
                    balanceTmp = "0";
                invoice.setBalance(Double.parseDouble(balanceTmp));
                invoice.setCurrency(invoiceObject.getString("currency"));
                invoice.setBank(invoiceObject.getString("bank"));
                db.updateOrInsertInvoice(invoice);
                dataSetChanged();
            }
        }
        else {
            e.printStackTrace();
        }
    }
});}
```

Kod 21. Dohvaćanje računa s Parse servera

Unutar *parseFetchInvoices()* metode dohvaćaju se svi računi iz Parse baze podataka te se spremaju u lokalnu *SQLite* bazu podataka za daljnje korištenje čak i kada aplikacija nije povezana s internetom.

ParseQuery klasa omogućava stvaranje upita prema Parse serveru. U ovom slučaju dohvaća se *ParseObject* pod nazivom „Invoices“, odnosno dohvaćaju se svi računi kojima je „user_id“ jednak onome od trenutno prijavljenog korisnika. Zatim se iterira kroz sve dohvaćene objekte, pretvaraju se u objekte računa te ih se sprema u lokalnu bazu. Dohvaćanje *Stringa* iz *ParseObject* objekta omogućuje metoda *getString()*, za dohvaćanje *integer* vrijednosti koristi se *getInt()* itd.

4.2.2.6. Metoda *getInvoices()*

```
public void getInvoices(){

    try {
        invoices = db.getInvoices();
    }
    catch (NullPointerException e){
        e.printStackTrace();
    }
    dataSetChanged();
    if (invoices.size() == 0){
        textNoInvoices.setVisibility(View.VISIBLE);
        recyclerInvoices.setVisibility(View.GONE);
    }
    else if (invoices.size() > 0){
        textNoInvoices.setVisibility(View.GONE);
        recyclerInvoices.setVisibility(View.VISIBLE);
    }
}
```

Kod 22. Dohvaćanje računa iz lokalne baze

Metoda *getInvoices()* poziva se odmah nakon *parseFetchInvoices()* metode i služi za dohvaćanje svih računa iz lokalne baze podataka. Unutar nje poziva se metoda *dataSetChanged()* koja osvježava listu s računima te ukoliko ne postoji niti jedan račun, *TextView* objekt *textNoInvoices* bit će prikazan korisniku te ga obavijestiti da ne postoji niti jedan račun.

4.2.2.7. Metoda `dataSetChanged()`

```
@UiThread

protected void dataSetChanged() {
    try {
        invoices.clear();
        getInvoices();
        adapterInvoices = new InvoiceAdapter(invoices);
        adapterInvoices.notifyItemInserted(invoices.size());
        adapterInvoices.notifyDataSetChanged();
        recyclerInvoices.setAdapter(adapterInvoices);
    }
    catch (NullPointerException ex) {
        ex.printStackTrace();
    }
}
```

Kod 23. Osvježavanje liste računa na dretvi korisničkog sučelja

Metoda `dataSetChanged()` vrši osvježavanje `Adaptora` računa i tako prikazuje promijenjenu aktivnost liste računa.

4.2.2.8. Metode `onCreateOptionsMenu()` i `onOptionsItemSelected()`

```
@Override

public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main, menu);
    return true;}
}
```

Kod 24. Instanciranje izbornika na naslovnoj traci

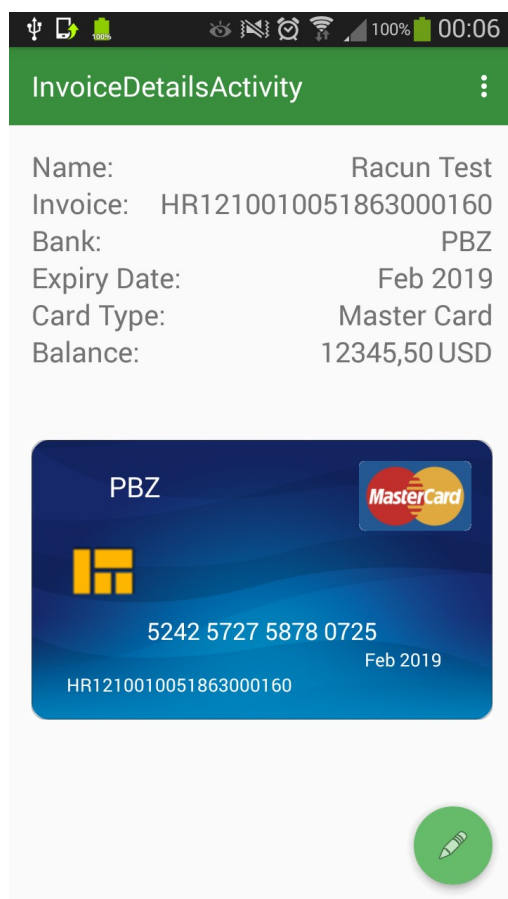
```
@Override

public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
    if (id == R.id.action_settings) {
        startActivity(new Intent(getApplicationContext(), SettingsActivity.class));}
    ...
}
```

Kod 25. Izvršavanje akcije prilikom odabira postavki u izborniku naslovne trake

Metoda `onCreateOptionsMenu()` služi za instanciranje izbornika na naslovnoj traci tako da postavlja XML pogled za izbornik, a `onOptionsItemSelected()` metoda služi za postavljanje akcija na pritisak određene stavke iz izbornika.

4.2.3. "InvoiceDetailsActivity" klasa



Slika 11. Pogled detalja računa

XML pogled detalja računa namijenjen je prikazu naziva, broja, banke, datuma isteka, vrste kartice i svote računa. Radi boljeg korisničkog iskustva dodana je kreditna kartica koja se dinamički mijenja s obzirom na podatke računa. Uz donji desni rub nalazi se lebdeći gumb te se pritiskom na njega otvara aktivnost za uređivanje istoga računa. S lijeve strane nalaze se labela koje pojašnjavaju značenje pojedine vrijednosti s desne strane. Kod prikaza svote računa koristi se valuta koja, ukoliko se u postavkama postavi na neku zadanu vrijednost, dinamički u cijeloj aplikaciji preračunava svotu u zadanu valutu. Svi elementi su međusobno relativno poslagani pa je tako njihov korijenski element *RelativeLayout*.

Kao i kod prijašnjih klasa potrebno je deklarirati sve objekte kontrola koje se povezuju s elementima u pogledu. Nakon toga, iz *Intent* objekta dohvaća se objekt računa kojega je potrebno prethodno serijalizirati i poslati iz prethodne aktivnosti. Sljedeći kod prikazuje dohvaćanje objekta računa preko *Intent* objekta:

```
Intent i = getIntent();

invoice = (Invoice) i.getSerializableExtra("invoice");
```

Kod 26. Primjer dohvaćanja računa iz Intent objekta

Nakon toga, na kontrole je potrebno postaviti odgovarajuće vrijednosti iz dohvaćenog računa.

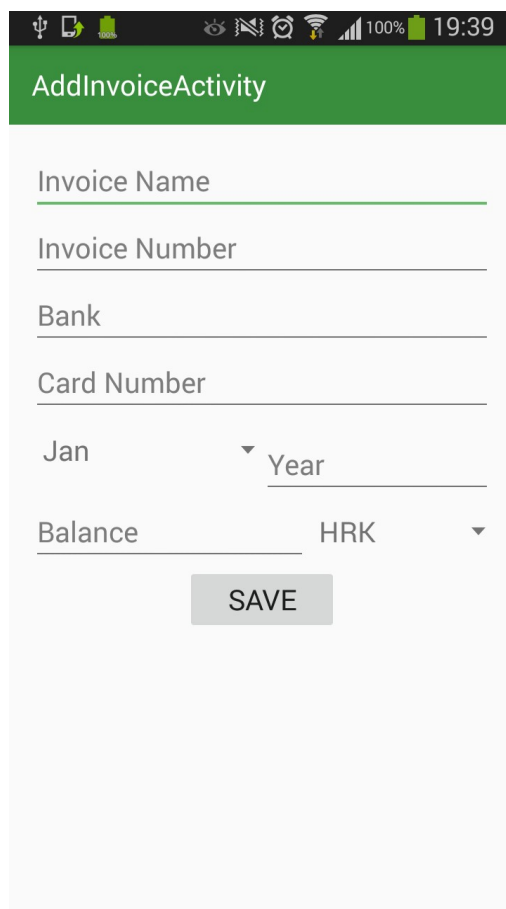
Prilikom dohvaćanja svote i valute potrebno je provjeriti je li u postavkama postavljena zadana valuta za cijelu aplikaciju te ukoliko je, mora se izvršiti preračunavanje svote.

```
SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);

boolean useDefaultCurrency = prefs.getBoolean(AppConfig.PREF_DEFAULT_CURRENCY, false);
textBalance.setFilters(new InputFilter[] {new DecimalDigitsInputFilter(2)});
if (useDefaultCurrency){
    String defaultCurr = prefs.getString(AppConfig.PREF_CURRENCY,
Currencies.HRK.toString());
    SQLiteCurrencies dbCurr = new SQLiteCurrencies(getApplicationContext());
    double balanceRecalc = dbCurr.getFromTo(invoice.getCurrency(), defaultCurr,
invoice.getBalance());
    textBalance.setText(Double.toString(balanceRecalc));
    textCurrency.setText(defaultCurr);
}
else {
    textCurrency.setText(invoice.getCurrency());
    textBalance.setText(Double.toString(invoice.getBalance()));
}
```

Kod 27. Dohvaćanje postavki zadane valute te preračunavanje iste

4.2.4. *AddEditInvoiceActivity* klasa



Slika 12. Pogled za dodavanje i uređivanje računa

Ovaj pogled sadrži elemente za unos, odnosno *EditView* elemente putem kojih korisnik unosi vrijednosti novog ili već postojećeg računa. Osim *EditView* elemenata, ovaj pogled sadrži još i dva *Spinner* elementa koji služe za odabir jedne vrijednosti iz liste.

```

<LinearLayout

    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/linear_expiry"
    android:weightSum="2"
    android:layout_below="@id/edit_card_number"
    android:orientation="horizontal"
>
    <Spinner
        android:id="@+id/spinner_month_expiry"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
    />

...

```

Kod 28. Dio XML koda sa Spinner elementom

Ovdje se može vidjeti primjer *Spinner* elementa unutar *LinearLayout* elementa koji ima postavljen „layout_width“ na „0dp“ odnosno nema širinu. Budući da *LinearLayout* ima atribut „weightSum“ postavljen na 2, elementi unutar njega mogu poprimati širinu s postavljanjem vrijednosti na atribut „layout_weight“ pri čemu bi maksimalna širina u ovom slučaju iznosila „2“.

AddEditInvoiceActivity klasa služi za uređivanje ili kreiranje novog računa te za razliku od prikaza detalja računa, potrebno je provjeriti uređuje li se postojeći račun tj. proslijeđuje li se objekt računa iz neke prijašnje klase. Ukoliko da, dinamički se popunjavaju polja s postojećim vrijednostima iz objekta računa.

Ukoliko je dohvaćen objekt tj. ukoliko nije „null“, popunjavaju se polja s postojećim vrijednostima.


```

if (invoice != null){

    editInvoiceName.setText(invoice.getName());
    editInvoiceNumber.setText(invoice.getInvoiceNumber());
    editBank.setText(invoice.getBank());

    ...

```

Kod 29. Popunjavanje polja ako se radi o uređivanju računa

4.2.4.1. Polje za unos broja kreditne kartice

Broj kreditne kartice označava i sam tip iste. Većinu tipova kreditnih kartica označavaju prve dvije znamenke pa se tako prilikom unosa broja kartice dinamički mijenja i tip kartice na *Spinner* elementu te se prema njemu i postavlja određena slika kartice. Da bi se takva provjera omogućila, potrebno je na *EditText* objekt postaviti *addTextChangedListener()* metodu s *TextWatcher* objektom te izvršiti željene promjene nakon unosa teksta.

```

editCardNumber.addTextChangedListener(new TextWatcher() {

    @Override
    public void afterTextChanged(Editable s) {
        if (s.length() >= 2){
            if (s.toString().substring(0,1).equals("4"))

                spinnerCardType.setSelection(adapterCardTypes.getPosition
                (CardTypes.VISA.toString()));

            ...

```

Kod 30. Slučaj izmjene vrijednosti Spinnera ako se radi o Visa kartici

4.2.4.2 Gumb za pohranu računa

Prilikom pritiska na gumb za pohranu računa vrši se validacija svih polja za unos te dodatna provjera broja računa odnosno IBAN-a koja je dodana u aplikaciju iz *Apache Commons Validator* biblioteke.

```
buttonSave.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        if (!isInvoiceNumValid(editInvoiceNumber.getText().toString())){
            String message = "Wrong IBAN number!";
            editInvoiceNumber.setError(message);
        }
        validator.validate();
    }
    ...
}
```

Kod 31. Poziv validacije kod pritiska gumba za pohranu računa

Ukoliko je validacija prošla te ukoliko se radi o uređivanju postojećeg računa, vrši se pokušaj dohvaćanja *id*-a postojećeg računa.

```
ParseObject invoiceObj = new ParseObject("Invoices");

try {

    if (invoice != null) {
        ParseQuery query = new ParseQuery("Invoices");
        invoiceObj = query.get(invoice.getId());
    }
} catch (ParseException e) {
    e.printStackTrace();
}
```

Kod 32. Pokušaj dohvaćanja id vrijednosti računa

Zatim se vrši pohrana podataka u objekt klase *ParseObject*.

```
invoiceObj.put("user_id", ParseUser.getCurrentUser().getObjectId());

invoiceObj.put("name", editInvoiceName.getText().toString());
invoiceObj.put("bank", editBank.getText().toString());
invoiceObj.put("invoice_number", editInvoiceNumber.getText().toString());

...
```

Kod 33. Pohrana vrijednosti u ParseObject

Prije poziva spremanja objekta na Parse server, potrebno je provjeriti postoji li povezivost s internetom. Ukoliko postoji, nad *ParseObject* objektom se poziva metoda *saveInBackground()* koja objekt računa direktno sprema na Parse server. Ukoliko nema internet konekcije, koristi se metoda *pinInBackground()* koja će objekt spremiti lokalno te će se prilikom prvog sljedećeg povezivanja s internetom objekt spremiti na Parse server.

4.2.5. SQLiteHelper klasa (Lokalna SQLite baza podataka)

```
private static final int DATABASE_VERSION = 1;
private static final String DATABASE_NAME = "finmgr_invoices";
private static final String TABLE_INVOICES = "Invoices";
private static final String COLUMN_ID = "id";
private static final String COLUMN_USER_ID = "user_id";
private static final String COLUMN_NAME = "name";

...
```

Kod 34. Deklaracija konstanti SQLiteOpenHelper klase

Za omogućavanje podataka korisniku kada nema internetske veze najčešće se koristi *SQLite* lokalna baza podataka. Prilikom implementiranja *SQLite* baze podataka za određen model, potrebno je naslijediti *SQLiteOpenHelper* klasu. Prilikom definiranja pomoćne *SQLite* klase obično se prvo definiraju konstante s imenima stupaca, baze, tablice i verzije baze.

4.2.5.1. Metoda onCreate()

U *onCreate()* metodi se izvršava tipična SQL naredba za izradu tablice.

```
public void onCreate(SQLiteDatabase db) {
    String CREATE_INVOICE_TABLE = "CREATE TABLE " + TABLE_INVOICES + "("
        + COLUMN_ID + " TEXT PRIMARY KEY,"
        + COLUMN_USER_ID + " TEXT,"
        + COLUMN_NAME + " TEXT,"
        + COLUMN_INVOICE_NUMBER + " TEXT,"
        + COLUMN_CARD_NUMBER + " TEXT,"
        + COLUMN_CARD_EXPIRY + " TEXT,"
        + COLUMN_CARD_TYPE + " TEXT,"
        + COLUMN_BALANCE + " REAL,"
        + COLUMN_CURRENCY + " TEXT,"
        + COLUMN_BANK + " TEXT"
        + ")";
    db.execSQL(CREATE_INVOICE_TABLE);
}
```

Kod 35. SQL upit za izradu tablice računa unutar "onCreate()" metode

4.2.5.2. Metode *addInvoice()* i *updateInvoice()*

...

```
SQLiteDatabase db = this.getWritableDatabase();
ContentValues values = new ContentValues();
values.put(COLUMN_ID, invoice.getId());
values.put(COLUMN_USER_ID, invoice.getUserId());
values.put(COLUMN_NAME, invoice.getName());
values.put(COLUMN_INVOICE_NUMBER, invoice.getInvoiceNumber());
```

...

*Kod 36. Isječak koda iz *addInvoice()* i *updateInvoice()* metoda koji služi za dodavanje vrijednosti u redak tablice*

```
long id = db.insert(TABLE_INVOICES, null, values);
```

Kod 37. Naredba koja služi za dodavanje novog računa u tablicu

```
db.update(TABLE_INVOICES, values, COLUMN_ID + "=\''" + invoice.getId() + "\'", null);
```

Kod 38. Naredba koja služi za pohranu postojećeg računa u tablicu

Ove dvije metode kao argument primaju *Invoice* objekt i gotovo su identične osim što se prilikom izvršavanja upita kod dodavanja računa poziva *insert()* metoda, a kod uređivanja postojećeg računa *update()* metoda.

4.2.5.3. Metoda *updateOrInsertInvoice()*

```
public void updateOrInsertInvoice(Invoice invoice) {

    SQLiteDatabase db = this.getWritableDatabase();
    String query = "SELECT COUNT(*) FROM " + TABLE_INVOICES + " WHERE " + COLUMN_ID + "
= \'" + invoice.getId() + "\'";
    Cursor mCount = db.rawQuery(query, null);
    mCount.moveToFirst();
    int exists = mCount.getInt(0);
    if (exists == 0)
        addInvoice(invoice);
    else
        updateInvoice(invoice);
    db.close();
}
```

Kod 39. Provjera postojanja računa unutar baze

Iz aktivnosti unutar aplikacije se prilikom dodavanja ili uređivanja računa u lokalnu bazu uvijek poziva metoda *updateOrInsertInvoice()* koja tada poziva *addInvoice()* ili *updateInvoice()* metodu ovisno o potrebi.

4.2.5.3. Metoda *getInvoices()*

```
public ArrayList<Invoice> getInvoices() {

    ArrayList<Invoice> invoices = new ArrayList<>();
    String selectQuery = "SELECT * FROM " + TABLE_INVOICES + ";";
    SQLiteDatabase db = this.getReadableDatabase();
    try {
        Cursor cursor = db.rawQuery(selectQuery, null);
        try {
            if (cursor.moveToFirst()) {
                do {
                    Invoice inv = new Invoice();
                    inv.setId(cursor.getString(0));
                    inv.setUserId(cursor.getString(1));
                    inv.setName(cursor.getString(2));
                    inv.setInvoiceNumber(cursor.getString(3));
                    inv.setCardNumber(cursor.getString(4));

                    ...
                } while (cursor.moveToNext());
            }
        }
    }
}
```

Kod 40. Dohvaćanje svih računa iz lokalne baze

Metoda *getInvoices()* služi za dohvaćanje svih računa iz lokalne baze. Dohvaćanje podataka iz tablice se vrši pomoću kursora koji iz svakog retka SQL naredbe za dohvaćanje svih računa uzima vrijednosti iz svakog stupca i pretvara ih u odgovarajući tip podataka te ih se pohranjuje u privremeni objekt računa koji se dodaje u listu.

4.2.6. Postavke u aplikaciji ("SettingsActivity" klasa

4.2.6.1. XML datoteka postavki (pref_general.xml)

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

    <ListPreference
        android:defaultValue="1"
        android:entries="@array/currency_array"
        android:entryValues="@array/currency_array"
        android:key="pref_currency"
        android:negativeButtonText="@null"
        android:positiveButtonText="@null"
        android:title="@string/pref_default_currency" />

    <CheckBoxPreference
        android:defaultValue="false"
        android:title="@string/pref_use_default_currency"
        android:key="pref_check_default_currency"
    />
</PreferenceScreen>
```

Kod 41. XML datoteka postavki

XML datoteka postavki mora imati korijensku oznaku *PreferenceScreen* unutar koje se definiraju sve postavke. U aplikaciji za praćenje financija postoje dvije postavke, lista za odabir zadane valute i *CheckBoxPreference* koji označava hoće li se izabrana valuta koristiti na svim mjestima ili ne.

```
public class SettingsActivity extends AppCompatActivity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.pref_general);
    }
}
```

Kod 42. Definicija zadane XML datoteke postavki

SettingsActivity klasa služi za pohranu svih postavki neke aplikacije. Postavke aplikacije definiraju se unutar XML datoteke koju se unutar *SettingsActivity* klase postavlja kao zadanu.

5. Zaključak

U ovom radu objašnjen je rad Android operacijskog sustava, njegova povijest, budućnost i trenutno stanje na tržištu. Vrijeme je pametnih mobilnih uređaja i razvoj mobilnih aplikacija itekako ima potencijala na svjetskom tržištu. Iako je Java još uvijek najpopularniji programski jezik za razvoj Android aplikacija, sve je više alternativa kao Xamarin i Kotlin, što pokazuje da razvoj Android aplikacija itekako napreduje.

Većina Android aplikacija se spaja putem API poziva na servere te tako dohvaća podatke koji su pohranjeni na internetu. U ovom radu opisana je Parse tehnologija koja također radi na principu API poziva, ali je modernija i jednostavnija za korištenje, ali može i znatno uštedjeti vrijeme samom programeru.

Za razvoj aplikacije koristio se Android Studio koji je danas najpoznatije programsko okruženje za razvoj Android aplikacija. Zbog mogućnosti i pomoći u pisanju koda te mnogih drugih naprednih funkcija preporuka je za svakoga tko se bavi ili se planira baviti programiranjem Android aplikacija.

U sklopu rada objašnjen je razvoj Android aplikacije za praćenje financija koja je prvenstveno namijenjena kao primjer i edukacijski materijal te zbog sigurnosnih propusta nije namijenjena za svakodnevnu uporabu, što ne znači da u bliskoj budućnosti neće biti. Prilikom razvoja aplikacije objašnjeni su klasični pristupi razvoja pomoću programskog jezika Java u kombinaciji s XML jezikom za označavanje. U aplikaciji je moguće pregledavati, uređivati i dodavati račune, fondove, poreze i dionice koje su u našem vlasništvu. Aplikacija omogućava registraciju i prijavu korisnika i podešavanje postavki zadane valute kroz cijelu aplikaciju.

6. Literatura

- [1] Wikipedia: Android (operating system)
[https://en.wikipedia.org/wiki/Android_\(operating_system\)](https://en.wikipedia.org/wiki/Android_(operating_system)) (25.5.2016.)
- [2] Wikipedia: Java (software platform)
[https://en.wikipedia.org/wiki/Java_\(software_platform\)](https://en.wikipedia.org/wiki/Java_(software_platform)) (25.5.2016.)
- [3] Wikipedia: Usage share of operating systems
https://en.wikipedia.org/wiki/Usage_share_of_operating_systems (25.5.2016.)
- [4] Marko Gargenta: Learnin Android - O'Reilly (2011.)
- [5] Wikipedia: Parse (company) [https://en.wikipedia.org/wiki/Parse_\(company\)](https://en.wikipedia.org/wiki/Parse_(company))
(12.6.2016.)
- [6] Wikipedia: OpenShift <https://en.wikipedia.org/wiki/OpenShift> (12.6.2016.)
- [7] Wikipedia: Android Studio https://en.wikipedia.org/wiki/Android_Studio (10.6.2016.)
- [8] Wikipedia: XML <https://en.wikipedia.org/wiki/XML> (26.5.2016.)
- [9] Graham Cluley <https://www.grahamcluley.com/2013/12/android-scware/>
(25.5.2016.)
- [10] Parse Blog <http://blog.parse.com/announcements/the-new-parse-developer-experience/> (4.6.2016.)
- [11] Infinum.co <https://infinum.co/the-capsized-eight/articles/android-studio-vs-eclipse-1-0>
(4.6.2016.)
- [12] Androidauthority <http://www.androidauthority.com/why-developers-choose-android-285774/> (25.5.2016)
- [13] IDC <http://www.idc.com/prodserv/smartphone-os-market-share.jsp> (25.5.2016.)

7. Prilozi

1. Prvi prilog

```
bttFundDateTo.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        myCalendar = Calendar.getInstance();
        DatePickerDialog.OnDateSetListener date = new
        DatePickerDialog.OnDateSetListener() {
            @Override
            public void onDateSet(DatePicker view, int year, int monthOfYear,
                                int dayOfMonth) {
                myCalendar.set(Calendar.YEAR, year);
                myCalendar.set(Calendar.MONTH, monthOfYear);
                myCalendar.set(Calendar.DAY_OF_MONTH, dayOfMonth);
                String myFormat = "dd.MM.yyyy";
                SimpleDateFormat sdf = new SimpleDateFormat(myFormat, Locale.UK);
                bttFundDateTo.setText(sdf.format(myCalendar.getTime()));
            }
        };
        new DatePickerDialog(AddEditFundActivity.this, date, myCalendar
            .get(Calendar.YEAR), myCalendar.get(Calendar.MONTH),
            myCalendar.get(Calendar.DAY_OF_MONTH)).show();
    }
});
```

Kod 43. Poziv DatePickerDialog prozorčica

Prilikom poziva prozorčica za odabir datuma, odnosno *DatePickerDialog* elementa, potrebno je na pritisak gumba instancirati novi *DatePickerDialog* objekt na koji se postavlja mogućnost odabira godine, mjeseca i dana. Nakon što korisnik odabere željeni datum, on se vraća u zadanom formatu i postavlja kao tekst gumba.