SVEUČILIŠTE U ZAGREBU
FILOZOFSKI FAKULTET U ZAGREBU

Tomislav Knežević

MOBILNA APLIKACIJA ZA DIJELJENJE I ARHIVIRANJE MULTI-MEDIJSKOG SADRŽAJA

Diplomski rad

Mentor: Dr. sc. Vedran Juričić

Zagreb, Listopad 2016.

Sadržaj

Mobilna aplikacija za dijeljenje i arhiviranje multimedijskog sadržaja

Sažetak

Fokus ovog diplomskog rada uz to što će biti kao što se iz naslova može iščitati na mobilnoj aplikaciji za dijeljenje i arhiviranje multimedijskog sadržaja biti će i na platformi za koju se razvijala aplikacija te bibliotekama korištenim za razvoj.

Android operacijski sustav se koristio kao platforma te će se prvo proći kroz njegovu povijest verzija nakon kojeg ide analiza njegove arhitekture i samog životnog vijeka svih aplikacija gdje će se dobiti uvid na koji način se aplikacije stvaraju, drže u memoriji i ispuštaju iz memorije.

Analizom popularnih biblioteka Retrofit, GSON, Butterknife, Picasso, Otto koje se za vrijeme pisanja ovog rada koriste u svijetu od strane razvojnih programera te koje su se također koristile u aplikaciji dobiti će se uvid u sve pogodnosti koje one donose u razvoju.

Rad će biti zaključen predstavljanjem prototipne aplikaciju Photo Arhive i njezine strukture.

Ključne riječi: android, aplikacija, arhiviranje, dalvik, art, biblioteke, retrofit, povijest verzija androida

Mobile application for sharing and archiving multimedia content

Abstract

The focus of this diploma thesis will be as the title says on the mobile application for sharing and archiving multimedia content and will also be on the platform and libraries which are both used to develop that application.

Android operating system is used as a platform and we will first go through the version history and then will come analysis of its architecture and the life cycle of applications where you will get a glimpse of how applications are created, kept in memory and discharged from memory.

The analysis of popular libraries Retrofit, GSON, Butterknife, Picasso, Otto that are during the writing of this paper used worldwide by developers and are also used in the application will give insight to all the benefits that they bring to development.

The work will be completed by presenting application prototype Photo Archive used and its structure.

Keywords: android, application, archiving, dalvik, art, library, retrofit, history version of the android

1. Uvod

Tema ovog diplomskog rada je aplikacija za dijeljenje i spremanje multimedijskog sadržaja naziva Photo Archive. Ona i njezin servis trenutno podržavaju samo spremanje i dijeljenje slika i ona je više prototip nego završni proizvod. Sama aplikacija će doći kao prilog pismenom diplomskom radu. U radu će se pisati o aplikaciji i njenoj implementaciji te o Android operacijskom sustavu koji je služio kao platforma za razvoj aplikacije.

Ideja aplikacije je došla od strane Dr. sc. Hrvoja Stančića koji je ujedno šef katedre na Arhivistici Filozofskog Fakulteta Zagreb. S njom se naknadno može ići u kompletno ostvarivanje ideje da ljudi na svojim mobilnim uređajima imaju jedinstvenu aplikaciju za privatno digitalno arhiviranje i dijeljenje svojih multimedijskih datoteka.

Android je izabran kao platforma zbog svoje otvorenosti i pristupačnosti. Njegovi alati su besplatni i vrlo lako ih se može instalirati i krenuti raditi na projektu. Uz tu svoju pristupačnost i besplatnost Android je najzastupljeniji operacijski sustav na mobilnim uređajima. Androida to sve zajedno čini odličnim kandidatom za ovakav projekt. Količina same dokumentacije na internetu, te pomoć u vidu raznih foruma i raznih biblioteka koji olakšavaju stalno ponavljajuće procese kroz koje svi programeri moraju prolaziti olakšava sam početak razvoja.

Bitnost same platforme u radu i razvoju aplikacija općenito je velika. Svaka platforma je drukčija i ima svoja pravila, implementaciju i drugačije pogodnosti unatoč svojim sličnostima. Android je intrigantan operacijski sustav i kao ideja sistema koji će se moći pogoniti na raznim mobilnim uređajima je vrlo zanimljiv ali i u isto vrijeme vrijedan istraživanja. Cijeli razvoj aplikacije kroz koji sam prolazio radeći aplikaciju Photo Archive je bio uvjetovan nad pravilima i implementaciji Androida te je on sam kao platforma zaslužan za mnoge njezine funkcije. Veći dio ovog diplomskog rada pripada Androidu a manji dio aplikaciji jer bez platforme nad kojoj aplikacija može ispravno i u svoj svojoj cijelosti funkcionirati ona sama kao takva ne bi mogla postojati.

Kroz ovaj diplomski rad prvo ćemo proći kroz te temelje Androida kao platforme. Proći ćemo kroz njegovu povijest verzija, kroz njegovu arhitekturu, nastanak i softverska rješenja. Također ćemo nakratko proći kroz najpopularnije biblioteke koje se koriste za sam Android koje su ujedno korištene i u samoj Photo Archive aplikaciji. Na kraju ćemo opisati aplikaciju, njezinu strukturu te njezine važne dijelove i klase.

2. Android

Android je mobilni operacijsku sustav razvijen od strane Googlea i temeljen na Linux jezgri. On je kao takav razvijan poglavito za uređaje sa dodirnim mogućnostima koji sadrže korisničko sučelje sukladno dodirnim gestama zajedno s virtualnom tipkovnicom. Kasnije se proširio na TV, Android Auto te pametne satove Android Wear.[1]

Android Inc. kao firma koja je započela razvoj Androida mobilnog sustava je osnovana 2003. godine te im je cilj bio razviti pametnije mobilne uređaje koji su svjesni lokacije uređaja i preferansa korisnika te se htjelo biti rival sistemima tipa Symbian i Microsoft Windows Mobile.

2005 Google je kupio firmu Android Inc. za 50 miliona dolara te su svi ključni ljudi Android Inc. firme ostali u firmi. Tim potezom i Googlovim marketingom i financijama Android je postao najzastupljeniji operacijski sustav na mobilnim uređajima.[1]

Android podržava mnoge mogućnosti koje su postale već standardne u svijetu operacijskih sustava pametnih telefona poput kamere, gledanja videozapisa, email klijenta, raznih senzora poput gps-a koji određuje lokaciju te žiroskopa, mogućnost skidanja novih aplikacija itd. Ono što krasi Android nad konkurencijom je njegova otvorenost, mogućnost modificiranja i mogućnost ulaženja u samu srž operacijskog sustava.

Android izvorni kod je izdan pod licencom otvorenog koda. Licenca omogućava raznim razvojnim programerima da unaprjeđuju, dodaju i predstavljaju mnoge inačice i verzije Androida. Također je moguće i stvaranje nadograđene i unaprijeđene verzije sustava za starije mobilne uređaje koji više ne dobivaju službena unaprijeđena što je inače praksa među proizvođačima. Stvorila se takva situacija da zbog te svoje otvorenosti svaki proizvođač mobilnih uređaja koji dobije verziju Android operacijskog sustava u nju ugrađuje svoje izmjene te takvu izmijenjenu verziju šalje na tržište. Korisnici još dobiju nekoliko unaprjeđenja koje nakon godinu ili dvije staju jer proizvođač već na tržištu ima nekoliko novih uređaja koji preuzimaju primat.

Povijest verzija Android mobilnog operacijskog sustava počela je izbacivanjem Android alpha verzije 2007. godine dok je prva komercijalna verzija Android 1.0 izbačena 2008. godine u rujnu.[2]

Beta je također bila dostupna u 2007. godini zajedno sa softverskim razvojnim alatima te se Studeni 5. često slavi kao rođendan Androida. Android je kontinuirano razvijan od strane

[1] Meier Reto. Professional Android 4 application development. 2012
[2] Android Pre-History. Android Central. http://www.androidcentral.com/android-pre-history

Googlea i OpenHandsetAllieance-a te je vrlo učestalo nakon svog prvog izbacivanja na tržište bio nadograđivan s novim verzijama.

Prve verzije 1.0 i 1.1 se nisu razvijale i izbacivale na tržište pod specifičnim imenima ali nakon 2009. godine i mjeseca travnja Android 1.5 je bio nazvan "Cupcake" te se od tada svaka njihova nova verzija koja bi se izbacila na tržište zvala po specifičnom kodnom imenu s naglašenom tematikom na komercijalne proizvode. Imena idu po alfabetskom redu s trenutno zadnjoj dolazećoj verziji imena "Nougat".[2]

Kroz sljedeće natuknice ćemo prikazati sve verzije Android sustava te ukratko ispisati njihove specifičnosti i nadogradnje nad starijim verzijama.

2.1. Povijest verzija operacijskog sustava Android

2.1.1 Verzija 1.0

Prvi uređaj s Android sustavom koji je bio komercijalno dostupan bio je HTC Dream

- web preglednik koji je mogao učitavati pune HTML stranice te je podržavao kontrole približavanja
- podršku za kameru, iako nije bilo opcija za kameru poput kvalitete slike i sl.
- pregledavanje foldera
- podrška za POP3, IMAP4 te SMTP
- sinkronizacija s Gmail sustavom
- Google kontakte
- Google kalendar
- Google mape
- Google Sync (sinkronizacija s kontaktima, email-om te kalendarom)
- Google Search (pretraživanje po internetu, aplikacijama i kontaktima)
- Google talk (instant poruke)
- Media pokretač (bez video i Bluetooth podrške)
- Notifikacije s mogućnostima upozorenja korisnika s LED svijetlom po primitku
- Mogućnost biranja kontakta za zvanje glasom
- YouTube pokretač video zapisa
- Wi-Fi i Bluetooth podrška[3]

[3] Android's Early Days. Android Central. http://www.androidcentral.com/androids-early-days

2.1.2. Verzija 1.1

- detalje i recenzije kada korisnik pretražuje poslovne objekte na mapama
- mogućnost skrivanja tipkovnice s brojevima za zvanje te duža stanka kada se koristi zvučnik
- mogućnost spremanja dodataka u porukama[3]

2.1.3. Verzija 1.5 Cupcake

- podrška virtualnim tipkovnicama treće strane
- podrška za Widget
- video snimanje u MPEG-4 I 3GP formatima
- automatsko sparivanje i stereo podrška za Bluetooth
- mogućnosti kopiranja i lijepljenja u web pregledniku
- korisničke slike koje se prikazuju u kontaktima
- specifična vremena koja se prikazuju u prikazu prošlih poziva
- animirane tranzicije između ekrana
- auto rotacija
- mogućnost slanja videozapisa na YouTube i slika na Picasa servise[4]

2.1.4. Verzija 1.6 Donut

- veće mogućnosti kod pretraživanja poput uključivanja povijesti knjižnih oznaka, kontakata i weba te mogućnost da razvojni programeri dodaju svoj sadržaj u pretraživanje
- glasovna sinteza kako bi iz glasa sustav mogao generirati tekst
- galerija, kamera i video snimanje su bolje integrirani u sustav
- mogućnost višestrukog selektiranja
- podrška za WVGA rezolucije
- poboljšanje u brzinama u aplikacijama za pretraživanje i snimanje[5]

2.1.5. Verzije 2.0, 2.0.1 Eclair

- bolja sinkronizacija za korisničke profile
- podrška za Microsoft Exchange

[4] Android 1.5 Platform. Android Developers. https://developer.android.com/about/versions/android-1.5.html
[5] Android 1.6 Platform. Android Developers. https://developer.android.com/about/versions/android-1.6-highlights.html

- Bluetooth 2.1
- bolji pristup kontaktima i jednostavniji izbornik na dodir
- pretraživanja starih poruka
- nove mogućnosti za kameru
- novi osvježeni web preglednik s podrškom za HTML5
- optimizirani hardware
- Google mape 3.1.2
- mogućnosti višestrukog dodira
- animirane pozadine na glavnom ekranu[6]

2.1.6. Verzija 2.2 Froyo

- poboljšanja u brzini, memoriji i performansama
- dodatna poboljšanja implementirana kroz JIT kompilaciju
- integracija Chrome V8 JavaScript pogona u aplikaciju preglednika
- podrška za Andorid Cloud to Device Messaging koja omogućuje gurnute notifikacije
- USB i Wi-Fi funkcionalnost javne pristupne točke
- mogućnost isključivanja mobilnih podataka
- dodana automatska nadogradnja aplikacija
- dodane numeričke i alfabetske lozinke
- slanje podatkovnih datoteka kroz preglednik
- preglednik prikazuje sve GIF sličice
- podrška za ekrane s visokom gustoćom piksela poput 4 inčnih 720p ekrana[7]

2.1.7. Verzija 2.3, 2..3.2 Gingerbread

- poboljšano iskustvo korisničkog sučelja
- podrška za ekstra velike ekrane
- rodna podrška za internetsku telefoniju
- bolja virtualna tipkovnica
- poboljšana funkcionalnost kopiranja i lijepljenje

[6] Android 2.0 Platform Highlights. Android Developers. https://developer.android.com/about/versions/android-2.0-highlights.html
[7] Android 2.2 Platform Highlights. Android Developers. https://developer.android.com/about/versions/android-2.2-highlights.html

- podrška za NFC
- dodani novi efekti za audio
- nova aplikacija naziva Download Manager koji daje korisnicima lagan pristup podatcima skinutim s interneta
- podrška za više kamera
- dodana podrška za upravljanje energijom
- dodana konkurentni kolekcija smeća za bolje performanse
- dodana podrška za senzore poput žiroskopa i barometra
- Google Talk aplikacija
- animacije sjene za liste[8]

2.1.8. Verzija 3.0 , 3.1, 3.2 Honeycomb

- bolje podrška za UI tablete
- sistem bar za brzi pristup notifikacijama i navigacijskim gumbovima na dnu ekrana
- dodan Action bar kontrola za kontekstualne opcije navigacije i druge tipove sadržaja na vrhu ekrana
- olakšano upravljanje s višestruko aktivnim zadaćama tako da korisnici na dodir Sistemskog bara mogu vidjeli aplikacije koje su aktivne i brzo mijenjati iz jednu u drugu
- dodane tab kontrole u web preglednik te incognito mod
- mogućnost pregleda slika u galeriji u prikazu punog ekrana
- podrška za višejezgrene procesore
- mogućnost enkripcije svih korisničkih podataka
- zabrana aplikacijama pisanje u sekundarnu memoriju, odnosno izvan dodijeljenih aplikaciji specifičnih direktorija. Puni pristup primarnoj internoj memoriji je i dalje dozvoljen uz zatražene dozvole korisnika.
- USB On-The-Go
- podrška za FLAC
- podrška za upravljače te eksterne tipkovnice i miševe
- bolja hardverske podrška posebice za tablete.[9]

[8] Gingerbread. Android Developers. https://developer.android.com/about/versions/android-2.3-highlights.html
[9] Honeycom. Android Developers.https://developer.android.com/about/versions/android-3.0-highlights.html

2.1.9. Verzija 4.0, 4.0.3 Ice Cream Sandwich

Bila je jedna od većih nadogradnji, donijela je puno novih dodataka i puno rafiniraniji i moderniji izgled Androidu

- tema Holo je dobila novu nadogradnju s novim fontom Roboto
- Widget kontrole sada dolaze u zasebnom tabu
- stvaranje foldera dobiva mogućnost drag-n-drop
- kontrole približavanja za kalendar
- mogućnost slikanja ekrana
- pristup aplikacija s zaključanog ekrana
- bolja integracija za glas te glas u tekst mogućnost
- poboljšana statistika i mogućnosti za mobilne podatke
- gašenje aplikacija potezom prsta
- Android Beam, NFC mogućnost za razmjenu između uređaja
- 1080p snimanje videa[10]

2.1.9. Verzija 4.1, 4.2, 4.3 Jelly Bean

Naglasak u ovoj nadogradnji je bio na brzinu i performanse s projektom naziva "Project Butter" s kojim je došlo dodirno iščekivanje za brže dodire, trostruko poliranje, poboljšan Vsync i popravljeno osvježavanje ekrana na 60 sličica po sekundi.

- tekst koji sadrži mogućnost da bude bi-direkcijski za internacionalnu podršku i specifične jezike npr. desna na lijevo
- notifikacije koje se mogu rastegnuti
- automatsko postavljanje veličina Widget kontrola ako je potrebno da bi sve uspjelo stati na ekran
- audio s više kanala
- brze postavke
- bolje mogućnosti u pristupu
- OpenGl ES 3.0 podrška
- bolje performanse podatkovnog sistema korištenjem fstrim komande dok uređaj miruje.[11]

[10] Ice Cream Sandwich. Android Developers. https://developer.android.com/about/versions/android-4.0-highlights.html
[11] Jelly Bean. Android Developers. https://developer.android.com/about/versions/jelly-bean.html

- podrška za 4k rezolucije

2.1.10. Verzija 4.4, 4.4W KitKat

Optimizacija u pokretanju operacijskog sustava na većem rasponu uređaja nego prije imajući 512mb radne memorije kao preporučujući minimum te ova verzija prva uvodi tad još eksperimentalni i po izboru okoliš vremena izvršavanja zvan ART kao zamjenu za tad Dalvik.

- elementi korisničkog sučelja više prelaze na bijelu boju
- Immersive Mode (postavka u kojoj se otvara puni ekran za pregledavanje aplikacije)
- aplikativna restrikcija za pristup vanjske memorije osim posebnih direktorija
- optimizacija za uređaje s malo memorije
- ispisivanje preko WiFi mreže
- kontrole preglednika za korištenje unutar aplikacije se temelje na Chromiumu
- javni API za razvijanje i održavanje klijenata za slanje tekstualnih poruka
- novi okvir za tranzicije među ekranima
- snimanje zaslona
- 4.4W
    - izbacivanje na tržište posebne verzija API zaslužne za android pametne satove[12]

2.1.11. Verzija 5.0, 5.1 Lollipop

Glavna značajka ove verzije je uvođenje novog dizajna zvanog *"material design"* te službena zamjena Dalvik-a za ART (Android Runtime) sa svojom kompilacijom prije vremena i poboljšanoj kolekciji smeća.

- podrška 64bit procesorima
- OpenGL ES 3.1
- ekran s nedavnim aktivnostima koje se pamte i nakon ponovnog pokretanja dolaze kao zamjena ekranu s nedavnim aplikacijama
- vektorske slikovne podatkovne datoteke (Vector Drawables)
- nadograđeni ekran zaključavanja bez Widget kontrola
- Project Volta za dugotrajnu bateriju

[12] Android KitKat. Android Developers. https://developer.android.com/about/versions/kitkat.html

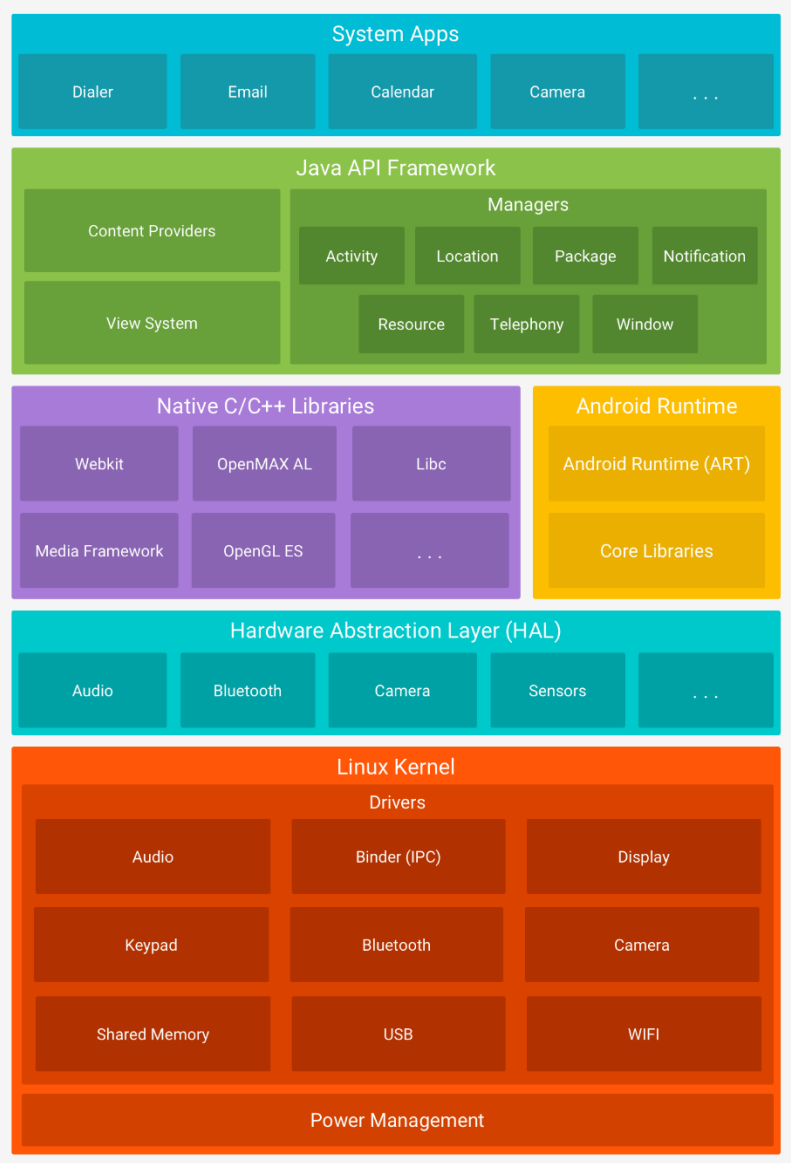
- mogućnost prijavljivanja u sustav kao gost
- aplikacijama se vraća mogućnost čitanja eksterne memorije
- promjene kako alarmi budu procesor i kako se alarmi natječu u borbi za resursima uređaja
- mogućnost pridruživanja WiFi mrežama putem brzih postavki
- pozivi visoke razlučivosti
- nadogradnje nad prioritetima za notifikacije[13]

2.1.11. Verzija 6.0 Marshmallow

- Now on Tap mogućnost
    - dobivanje informacija što je na ekranu držeći home gumb
- Doze mode koji smanjuje procesorsku brzinu dok je ekran isključen
- Alfabetski dostupna ladica za aplikacije
- rodna podrška čitaća za prste
- App linking za brže otvaranje linkove pomoću njihovih zaduženih aplikacija
- USB Type-C podrška
- automatska rezervna pohrana i vraćanje za aplikacije
- MIDI podrška za glazbene instrumente
- eksperimentalna mogućnost za višestruke prozore
- dozvole za aplikacije se dodjeljuju za vrijeme korištenja same aplikacije naspram prijašnjeg sustava u kojem se iste prezentiraju prije instalacije[14]

[13] Android Lolipop. Android Developers. https://developer.android.com/about/versions/lollipop.html
[14] Android 6.0 Marshmallow. Android Developers. https://developer.android.com/about/versions/marshmallow/index.html

2.2 Arhitektura



Slika 1 Arhitektura sustava.

Izvor: https://developer.android.com/guide/platform/index.html

2.2.1 Linux

Android jezgra eng. *"krenel"* je utemeljena na Linux verziji jezgre 2.6. Počevši od travnja 2014 Android uređaji uglavnom koriste verzije 3.10, 3.18 ili 3.4. Specifična verzija jezgre ovisi o Android uređaju i njegovoj hardverskoj platformi. Jezgra je centralni dio bilo kojeg operacijskog sustava. Linux kao otvoreni sustav je nudio Android razvojnim programerima odličan početak te temelje na koje oni mogu nadograditi te dodatno specificirati verziju sustava kako bi bila pogodna za mobilne uređaje.

Linux je operacijski sustav koji je razvijen 1991. godine kao operacijski sustav otvorenog koda za desktop kompjutere od strane Linus Torvaldsa. Kada se pokrene Android Linux

jezgra, učitavanje se odvija isto kao i na nekoj od Linux distribucija. Iako je razvoj Android operacijskog sustava temeljen na Linuxu, operacijski sustav ne koristi kompletne mogućnosti standardne Linux jezgre tako da često možemo vidjeti debatu da li je Android spada u Linux distribuciju. [15]

Nije moguće pogoniti Android aplikacije na tipičnim Linux distribucijama i isto tako nije moguće pokretati Linux programe na Androidu. Android arhitektura za sad podržava samo dva tipa arhitekture; x86 i ARM. Linux podržava različite tipove arhitektura uključujući x86 arhitekturu koja je najčešće korištena arhitektura u osobnim računalima i serverima. Android koristi tu arhitekturu za Mobilne Internet Uređaje i ARM platformu za mobitele.[16]

Linux jezgra se postavlja na dno Android softverske arhitekture te ona omogućuje apstrakciju između hardvera samog uređaja i gornjih dijelova Android arhitekture.

Za vrijeme razvoja Android sustava nekoliko dodatnih mogućnosti je dodano u Linux jezgru uključujući: pokretački program za alarme, centralni tragač za pogreškama eng. *kernel debugger*, program za zapisivanje aktivnosti eng. *logger*, menadžment za energiju te Android dijeljeni driver. Ove mogućnosti su razvijane poviše standardne Linux jezgre. Svakom novom verzijom je unaprjeđivan operacijski sistem od svog samog početka.

Linux ima puno različitih vrsta svojih distribucija poput: Edubuntu, Gobuntu, Kubuntu, Lubuntu, Xubuntu, Ubuntu Netbook, Ubuntu mobile, Ubuntu Server edition itd.[15] Većina ovih distribucija koristi GNU C biblioteku koja se brine o potrebnim rutinama unutar sistema biblioteka. Android ima svoju posebnu C biblioteku znanu kao Bionic koja je dizajnirana da omogućuje brzu egzekuciju. Također Android je razvio svoju virtualnu mašinu naziva Dalvik o kojoj ćemo kasnije pisati za razliku od standardne Java virtualne mašine (JVM).

Media koja se koristi za pohranu ide pod nazivom Yet Another Flash File System (YAFFS). Ta flash memorija je uglavnom korištena radi manjih veličina sveukupnog prostora za pohranu na mobilnim uređajima. Ona također ima veću otpornost na udarce za razliku od tradicionalnih tvrdih diskova. Flash memorija je razdvojena u nekoliko patricija poput /system za operacijski sustav i /data za korisničke podatke i prostor za instalaciju aplikacija. Za razliku od desktop korisnika Android korisnici nemaju pristup glavnom dijelu operacijskog sistema i osjetljive patricije su samo za čitanje.[16]

Android dopušta aplikacijama njihovu memoriju za korištenje te zbog toga aplikacije koje se pogone u virtualnoj mašini ne dopuštaju sistemu da uđe u stanje spavanja. To je dosta

[15] What is Linux. Linux. https://www.linux.com/what-is-linux
[16] Stefan Brahler. Analysis of the Android Architecture. 2010. https://os.itec.kit.edu/downloads/sa_2010_braehler-stefan_android-architecture.pdf

bitno jer po standardnom načinu, Android pokušava staviti sistem u stanje spavanja ili još bolje u stanje obustavljanja što je prije moguće. Aplikacije se mogu pobrinuti da ekran stoji upaljen ili procesor stoji u stanju pripravnosti kako bi primio dodatne upite. Način kako se Android brine za ove stvari se zove Wake Lock.[17]

One se mogu dobiti od strane jezgrovitih komponenti ili putem procesa korisničkog prostora. Taj korisnički prostor je sadržan u datoteci /sys/power/wakelock u kojemu je ime Wake lock-a zapisano. Wake lock se može automatski obustaviti nakon nekog određenog vremena. Sve trenutno aktivne Wake Lock instance su ispisane u n /proc/wakelocks.[17]

Taj koncept je duboko integriran u Android putem drivera i mnoge aplikacije ga koriste. To je jedan od važnijih blokova Android koda jezgre koji je dodan unutar Linux glavnog koda.

Power manager je klasa u aplikacijskom okviru koja daje virtualnoj mašini pristup Wake Lock mogućnostima:

- PARTIAL WAKE LOCK - procesor je budan čak ako je gumb gašenja/paljenja stisnut na uređaju
- SCREEN DIM WAKE LOCK - ekran je uključen ali je zamračen
- SCREEN BRIGHT WAKE LOCK - ekran je uključen sa normalnim osvjetljenjem
- FULL WAKE LOCK – tipkovnica i ekran su u normalnom stanju[17]

Poviše Linux jezgre možemo naći cijeli softverski sustav, imamo srednji sloj, biblioteke i API-e napisane u C programskom jeziku te aplikacijski softver koji se pokreće na aplikacijskom okviru koji uključuje Java kompatibilne biblioteke.

2.2.2 C/C++ biblioteke

Biblioteke koje su uključene da popune široki i raznolik raspon funkcija uključujući 2D i 3D grafičko crtanje, Secure Sockets Layer (SSL) komunikaciju, SQLite menadžment baza podataka, audio i video pokretanje, renderiranje slikovnih datoteka i vektorskih fontova, prikaz podsistema, grafički sloj menadžmenta i implementaciju standardne C sistemske biblioteke (libc).

U praksi tipični razvojni programer Android aplikacija će pristupati ovim bibliotekama kroz Java utemeljenje Android jezgrovite aplikacijske API-e. U slučaju da je potreban direktni

[17] Keeping The Device Awake. Android Developers. https://developer.android.com/training/scheduling/wakelock.html

pristup, može se ostvariti putem Android Native Development Kit (NDK), čija je svrha da poziva rodne metode u drugim programskim jezicima različitim od Jave kroz njezin kod koristeći se tehnologijom naziva Java Native Interface.

2.2.3 Dalvik

Pošto Android okruženje za programere primarno koristi Javu programski jezik za pisanje aplikacija koja je uvijek bila prezentirana sa sloganom zapiši jednom, pokreći svuda eng. *'write once, run anywhere''* za što je zaslužna Java Platforma čiji je ključni dio Java Virtual Machine (JVM). Iznenađujuće je bilo što je Google odlučio napustiti JVM te odlučiti se za alternativno rješenje u vidi Dalvik-a. Google se također odlučio za alternativnu i limitiranu implementaciju Java biblioteka. [18]

Ta odluka je uglavnom bila donesena razmišljanjem u smjeru toga da su mobilni uređaji ipak limitirani radnom memorijom, procesorskom snagom i podatkovnim prostorom za pohranu. Dalvik je virtualna mašina optimizirana za mobilne uređaje unutar Android operacijskog sustava koju pokreče aplikacije napisane za Android. [18]

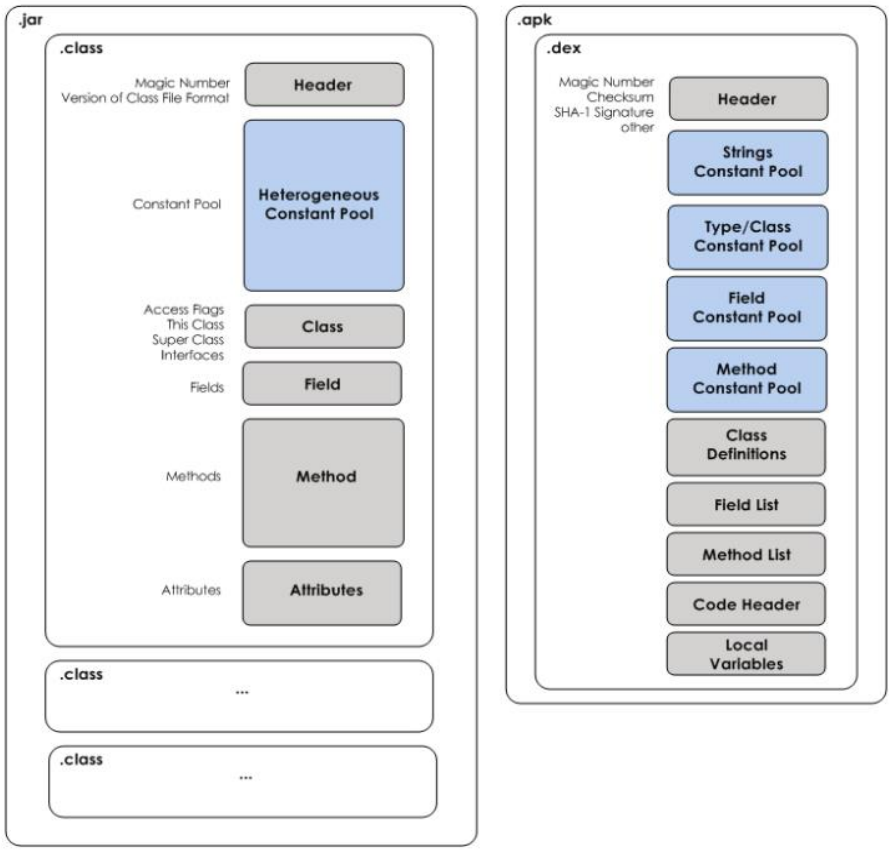
Dalvik je ime grada u Islandu te je napisan od strane Dan Bornsteina. On je nastao iz projekta otvorenog koda, nazvanog Harmony kojemu je bio cilj prenijeti Java virtualnu mašinu na mobilne uređaje. Što je dovelo do raznih sudskih procesa između Google-a i Oracle-a koji je tvrdio da je Google samo kopirao Java JVM. Google se uspio obraniti te je dokazano da Dalvik i JVM nisu jedno te isto.

Dalvik koristi upravo u vrijeme kompilaciju u kojoj se kod kompilira u trenutku kada je potreban za aplikaciju. Svaka Android aplikacija izvršava se u svom procesu, zajedno sa svojom instancom Dalvik virtualne mašine. Dalvik je napisan na takav način da omogućuje uređaju da pokreće nekoliko virtualnih mašina od jednom. On izvršava datoteke u Dalvik Executable (.dex) formatu koji je optimiziran za minimalni memorijski trag. Virtualna mašina je temeljena na registrijima i ona pokreće klase .class kompilirane od strane Java kompajlera koje se transformiraju u .dex format. [19]

Što je malo drukčije od standardnog Java okruženja u kojem je Java izvorni kod kompilira direktno u bitni kod pohranjen u .class datotekama. Takve datoteke se kasnije čitaju od

[18]David Ehringer, March, 2010. The Dalvik Virtual Machine Arhitecture. http://davidehringer.com/software/android/The_Dalvik_Virtual_Machine.pdf
[19]David Ehringer, March, 2010. The Dalvik Virtual Machine Arhitecture. http://davidehringer.com/software/android/The_Dalvik_Virtual_Machine.pdf

strane JVM-a za vrijeme trajanja procesa. Svaka klasa u Java kodu će rezultirati s jednim .class datotekom. Dex datoteke sadržavaju nekoliko različitih klasa od jednom.



Slika 2 Dalvik dijagram.

Izvor: http://techiezjunkyard-android.blogspot.hr/2011/12/dalvik-virtual-machine-dvm_2269.html

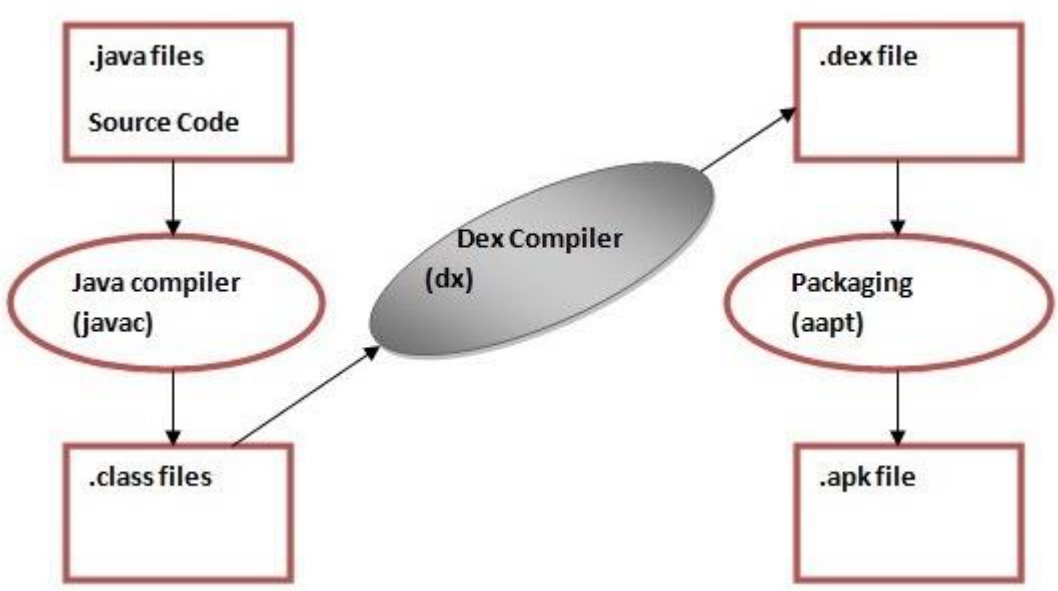
Virtualna mašina Dalvik obavlja transformaciju aplikacijskog Dalvik bitnog koda u nativne instrukcije. Postoje specifične za platformu Dalvik virtualne mašine za svaki hardver koji pokreće Android bio to Intel, ARM i sl. Kompajler stvara Dalvik bitni kod te Dalvik virtualna mašina raspoznaje i raščlanjiva taj bitni kod.

Dex datoteke:

- koriste zajedničke specifične bazene konstanti poput metoda, klasa/tipova, nizova teksta kao primarni mehanizam za čuvanje memorije. Umjesto da se ove vrijednosti čuvaju zasebno kroz klasu, one su uvijek referirane po svom Indexu u bazenima konstanti na taj način se sprječava repetitivnost u zapisivanju u memoriju.

    - pomoću zasebnih bazena konstanti .dex format je smanjio veličinu memorije na pola u zajedničkim sistemskim bibliotekama i aplikacijama koji dolaze sa Androidom
- Android programi se kompiliraju u .dex podatke koju su svi zajedno spremljeni u jednu .apk datoteku na uređaju. Oni se mogu stvarati automatski prevođenjem kompiliranih aplikacija napisanih u Java programskom jeziku[20]

Dalvik koristi svoj Dalvik Debug Monitor Service (DDMS) kao ključni program za pronalaženje grešaka.



Slika 3 Dex kompajler.

Izvor: http://www.javatpoint.com/dalvik-virtual-machine

- **javac tool** kompilira java izvorni kod u jednu datoteku klase
- **dx tool** uzima sve datoteke klasa aplikacije i generira jedan .dex datoteka. To je platformi specifičan alat
- **Android Assets Packaging Tool** se brine o procesu pakiranja

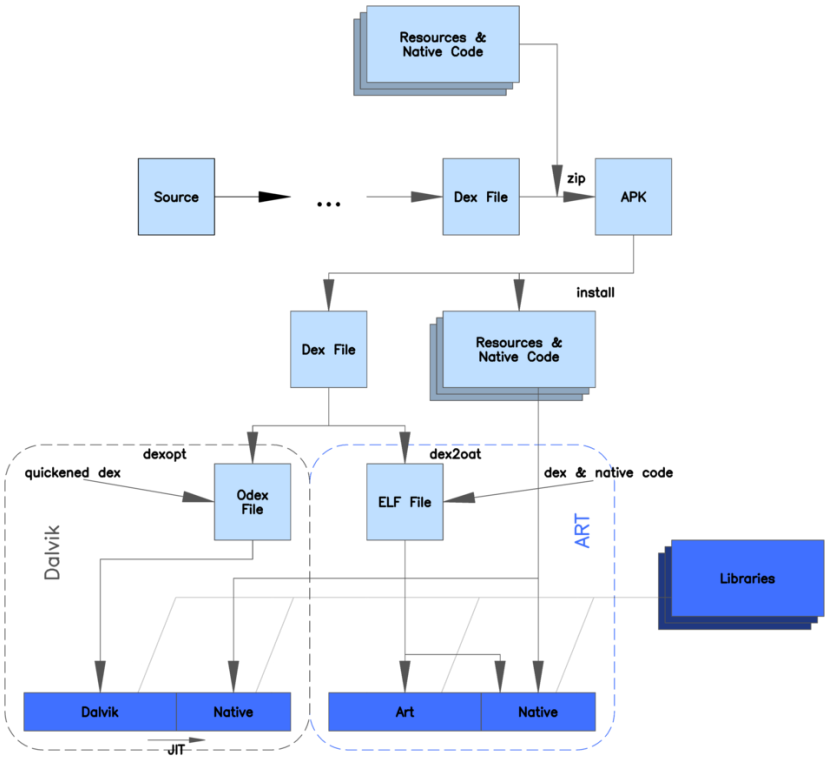
Pošto se svaka aplikacija pokreće u svojoj instanci virtualne mašine, instance se moraju pokretati brzo kada se nova aplikacija pali, te memorijski odraz mora biti što manji. Zygote koncept tu ulazi u priču. On omogućuje dijeljenje koda između instanci virtualnih mašina i omogućuje brzo pokretanje novih instanci. Koncept pretpostavlja da postoji bitni broj jezgrenih

[20] Dalvik bytecode. Android Developers. https://source.android.com/devices/tech/dalvik/dalvik-bytecode.html

klasa biblioteka i odgovarajućih struktura hrpe eng. heap kroz mnogo aplikacija. Ukratko podatci i klase koje mnoge aplikacije koriste ali nikad ne modificiraju. [21]

Zygote je proces unutar virtualne mašine koji se pokreće u samom paljenju sistema te automatski učitava i inicijalizira jezgrovite klase biblioteka, prije samog korištenja koje se kasnije dijele među procesima.

2.2.4 ART



Slika 4 ART dijagram.

Izvor: https://commons.wikimedia.org/wiki/File:ART_view.png

ART (Andoird Runtime) je aplikacijski okoliš za pokretanje korišten od strane Android operacijskog sustava koji zamjenjuje Dalvik počevši od 4.4 verzije Andorida Kitkat kao dodatna opcija u postavkama. Verzija Androida 5.0 Lolipop je donijela službenu zamjenu Dalvika za ART kao primarno rješenje na svojem operacijskom sustavu.

[21] David Ehringer, March, 2010. The Dalvik Virtual Machine Arhitecture. http://davidehringer.com/software/android/The_Dalvik_Virtual_Machine.pdf

U početku Dalvik je počeo kao relativno jednostavna Virtualna mašina nemjerena mobilnim uređajima s malo memorije i procesorske snage. Nakon nekog vremena usavršavanja hardverskih specifikacija mobilnih uređaja i napredovanju tehnologije Google je osjetio potrebu da reagira na brige vezane za performanse i jačanje hardverske snage.

ART je dizajniran da bude potpuno kompatibilan s Dalvikovim formatom bitnog koda znanog dex. Kao takav, s perspektive programera, on ne sadrži promjene koje bi mogle utjecati na samo pisanje aplikacija i brigu vezanu za kompatibilnost s starim kodom aplikacija. Glavna stvar koju ART donosi je umjesto upravo u vrijeme eng. *Just-in.time* (JIT) kompilacije on sada kompilira aplikacijski kod metodom prije vremena eng. *Ahead-of-Time* (AOT).[22]

Značajke AOT kompilacije:

- kompiliranje se događa prije vremena, odnosno sama kompilacije se izvršava za vrijeme instalacije aplikacije
- dobivaju se mnoge prednosti naspram JIT kompilacije pogotovo u smjeru efikasnosti i trošenja energije
- dobivanjem na performansama dobiva se i veličina samih .apk datoteka te dugotrajnije vrijeme instalacije što je sada pogodno radi sve većih memorijskih prostora
- kompajler ima pogled na sav kod za razliku od JIT kompajlera koji radi optimizaciju samo lokalnih i vidljivih dijelova[23]

dex2oat alat se koristi za kompilaciju. On prihvaća DEX datoteke kao input i generira kompiliranu izvršnu datoteku za ciljani uređaj.

Tehnologija kolekcije smeća može uvelike usporiti i imati utjecaj na performanse aplikacije često dovodeći do trzaja i gubljenju sličica u sekundi u trenutcima svog rada i otpuštanju i pisanju u memoriju. Ako takvi procesi dovoljno dugo potraju korisnik to može primijetiti i to može rezultirati sporim korisničkim sučeljem unutar aplikacije.

ART donosi mnoge prednosti u tom pogledu naspram Dalvik-a

- jedna GC pauza umjesto dvije
- paralelno procesiranje tijekom preostalih GC pauza
    - primjer: puni GC će se samo pokretati kada je uređaj zaključan i korisnička interakcija više nije bitna.

[22] Configuring ART. Android. https://source.android.com/devices/tech/dalvik/configure.html#how_art_works
[23] Art and Dalvik. Android. https://source.android.com/devices/tech/dalvik/index.html

- sakupljač s manjim totalnim GC vremenom za specijalne slučajeve nedavnog alociranja kratko živućih objekata
- bolja GC ergonomija, s istovremenim kolekcijama s boljim pogađanjem vremena, što uzrokuje puno manje učestalim GC_FOR_ALOC eventima
- smanjuje se pozadinska memorijska korištenost i fragmentacija[24]

Prednosti kod dijagnostike i podataka vezanih za rušenje programa

- puno veći kontekst u pogreškama
- ART omogućuje proširene detalje o iznimkama
    - java.lang.ClassCastException
    - java.lang.ClassNotFoundException
    - java.lang.NullPointerException
- Na primjer java.lang.NullPointerException sada nudi informacije o tome što je aplikacija pokušala napraviti s null pokazivačem npr. polje aplikacije u koje se htjelo pisati ili metoda koju se pokušavalo pozvati. Npr.
    - java.lang.NullPointerException: Attempt to write to field 'int android.accessibilityservice.AccessibilityServiceInfo.flags' on a null object reference
    - java.lang.NullPointerException: Attempt to invoke virtual method 'java.lang.String java.lang.Object.toString()' on a null object reference[25]

2.2.5 Java interoperabilne biblioteke

Standardni Java razvojni okoliš uključuje različiti raspon klasa koje sadrže Java jezgrovite biblioteke. Te biblioteke omogućuju potporu za zadaće poput npr. upravljanje nizom znakova, mrežne zadaće i manipulaciju datotekama. Java Inter operabilne biblioteke su poznate i širom korištene svim Java razvojnim programerima bez obzira na platformu.

Java interoperabilne biblioteke su implementacija otvorenog koda utemeljena na Apache Harmony projektu i dio su Standardnih Java biblioteka koje su prilagođene za korištenje aplikacija koje koriste Dalvik virtualnu mašinu.

[24] Debugging ART Garbage Collection. Source Android. https://source.android.com/devices/tech/dalvik/gc-debug.html
[25] Art and Dalvik. Android. https://source.android.com/devices/tech/dalvik/index.html

2.2.6 Aplikacijski okvir

Aplikacijski okvir je set servisa koji kolektivno stvaraju okoliš u kojem se Android aplikacije mogu pogoniti i razvijati. Taj okvir implementira koncept za samo izradu aplikaciju i njihovih komponenti koje mogu biti za višekratnu upotrebu i razmjenljive.
Aplikacijski okvir uključuje:

- Activity Manager – kontrolira sve aspekte aplikacijskog životnog vijeka i stog samih aktivnosti unutar aplikaciji
- Content Providers – omogućuje aplikacijama da objavljuju i dijelu podatke sa drugim aplikacijama
- Resource Manager – omogućuje pristup ugrađenim resursima poput nizova znakova eng. *string*, postavki za boju i korisničkih planova
- Notifications Manager – dopušta aplikacijama da prikazuju notifikacije korisniku
- View System – set za stvaranje korisničkih sučelja
- Package Manager – menadžer u kojem aplikacije mogu naći informacije o drugim aplikacija instaliranim na uređaju
- Telephony Manager – daje informacije aplikaciji o telefonskim servisima dostupnim na uređaju
- Location Manager – omogućuje pristup lokacijskim servisima koji dopuštaju aplikaciji da prima dojave o promjenama lokacije uređaja.[26]

2.2.7 Android biblioteke

Primjeri nekih od jezgrovitih Android biblioteka dostupnih Android razvojnom programeru:

- android.app – je pristup aplikacijskom modelu i glavni dio je svih Android aplikacija
- android.content – omogućuje pristup sadržaju, objavljivanje i slanje poruka između aplikacija i dijelova aplikacije
- android.database – uključuje SQLite menadžment za base podataka i pružatelje sadržaja
- android.graphics –nisko slojeviti 2D grafički API za crtanje

[26] Platform Architecture. Android Developers. https://developer.android.com/guide/platform/index.html

- android.hardware – API koji omogućuje pristup hardveru poput svjetiljke i žiroskopa
- android.opengl – Java sučelje za OpenGL ES 3D grafiči API
- android.os – omogućuje aplikacija pristup standardnim servisima operacijskog sustava poput poruka, sistem servisa i inter-procesne komunikacije
- android.media – pristup klasama koji omogućuju pokretanje audio i video podataka
- android.net – set API-a koji omogućuju pristup mreži
- android.provider – set klasa koji omogućuju pristup standardnim bazama Android pružatelja sadržaja poput onih za kontakte i kalendare iz kojih se mogu vući korisnički podatci
- android.text – renderiranje i manipulacija teksta
- android.util – razne klase za raznovrsne svrhe poput XML upravljanja, konverziju brojeva itd.
- android.view – fundamentalni dijelovi za aplikacijsko korisničko sučelje
- android.widget – kolekcija već prije razvijenih UI komponenata poput gumbova, lista, tabela itd.
- android.webkit – set klasa koji služe za ostvarenje mogućnosti poput web preglednika koji se može gurnuti unutar aplikacije[27]

2.2.8 Aplikacije

Aplikacije predstavljaju najgornji sloj Android okvira i on se sastoji od aplikaciji koje su ugrađene u operacijski sustav i uglavnom razvijene od strane Android tima ili od neke treće strane. Tipične aplikacije su: Camera, Alarm, Clock, Calculator, Contacts, Calendar, Media Player itd. Aplikacije koje razvojni programeri naprave se isto nalaze na ovom sloju.

Niti jedna aplikacija po standardnom načinu rada nema dozvolu da obavlja bilo kakve operacije koje bi negativno utjecale na druge aplikacije, operacijski sistem ili korisnika. To uključe čitanje ili pisanje korisničkih privatnih podataka. čitanje ili pisanje podataka tuđih aplikacija osim svoje, obnašanje mrežnog pristupa, držanje uređaja budnim itd.

[27] An Overview of the Android Architecture. Techotopia. http://www.techotopia.com/index.php/An_Overview_of_the_Android_Architecture

Jedne od glavnih mogućnosti koje pomažu u razvijanju sigurnih aplikacija na android stranicama se tvrdi:

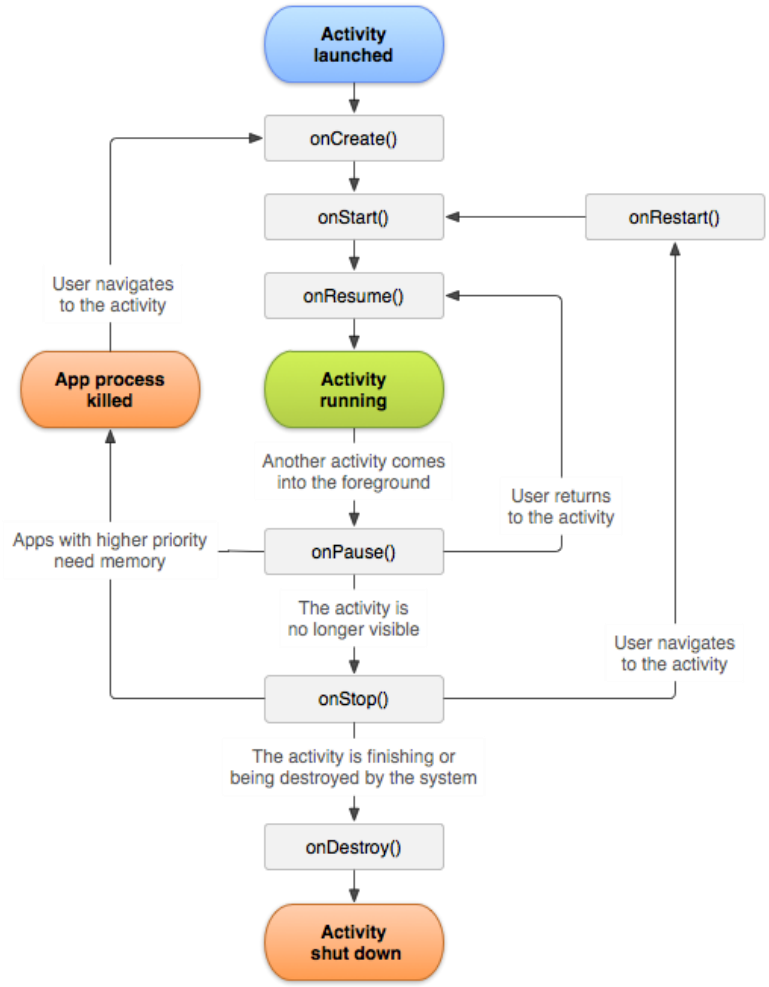
- Android Application Sandbox tehnologija izolira podatke aplikacije i programski kod od drugih aplikacija
- Aplikacijski okvir sa robusnom implementaciju standardnih sigurnosnih funkcija poput kriptografije, dozvola i sigurnog IPC-a.
- Tehnologije tipa ASLR, NX, ProPolice, safe_iop, OpenBSD dlmalloc, OpenBSD calloc, i Linux mmap_min_addr se koriste za migraciju rizika asociranih sa standardnim greškama u memorijskom menadžmentu
- Kriptirani sistem datoteka koji se može omogućiti za zaštitu podataka i ukradenih uređaja
- Korisnički generirane dozvole za zabranu pristupa sistemskim mogućnostima i korisničkim podatcima
- Aplikacijski definirane dozvole za kontrolu aplikacijskih podataka na bazi svake aplikacije[28]

Ove mogućnosti su implementirane na razini samog operacijskog sustava i na bazi samog aplikacijskog okvira a ne unutar virtualne mašine. Svaka aplikacija se pokreće u svojoj instanci virtualne mašine i samim time ona ne može upadati u druge aplikacije. Svaki proces ima svoj korisnički ID s kojim se može pristupiti samo limitiranim mogućnostima unutar sistema. U slučaju dodatnih dozvola, razvojni programer mora uključiti dodatne dozvole u Manifest datoteku. Dozvole se prikazuju korisniku ako je na uređaju ispod verzije 5.0 prije same instalacije odnosno u samom potrebnom momentu u kojem su one potrebne ako je na verzijama poviše verzije 5.0 Androida.

Iako virtualna mašina nije zaslužna za sigurnost većina Java standardnih klasa zaduženih za sigurnost je uključena u Android poput klase SecurityManager i neke klase unutar java.security paketa.

[28] Security Tips. Android Developers. https://developer.android.com/training/articles/security-tips.html

2.2.9. Životni vijek Aktivnosti



Slika 5Aplikacijski životni vijek dijagram.

Izvor: https://developer.android.com/reference/android/app/Activity.html

Dijagram pokazuje važne događaje i statuse kroz koje jedna Aktivnost prolazi u svom vijeku trajanja. Pravokutni oblici predstavljaju metode povratnog poziva eng. *callback* koje se mogu implementirati kako bi se obavljale operacije u trenutnu odvijanja tih procesa. U njima se programer sam može brinuti o ponašanju svog koda koji prolazi kroz te događaje.

Aktivnosti možemo shvatiti kao jednu, fokusiranu stvar koji korisnik može napraviti. Greška je nju smatrati kao trenutnim ekranom. Ona je cjelina koja sadrži mnoge stvari. Skoro sve aktivnosti vrše interakciju sa korisnikom, pa se Activity klasa brine o stvaranju samog prozora u kojem programer može programirati korisnička sučelja za aplikaciju započevši pozivanjem prve metode u prvom Activity eventu onCreate `setContentView(View)` kojoj se prosljeđuje View objekt. One se ne moraju prezentirati korisniku kao prozori punog ekrana, one se mogu koristiti i u druge svrhe: kao leteći prozori eng. *floating windows* uključeni unutar neke

druge aktivnosti, kao skladište Fragmenata itd. Eventi unutar Aktivnosti tipa onCreate koji se brinu o inicijalizaciji korisničkog sučelja i xml datoteka te onPause (kada Aktivnost ulazi u pauzirao stanje često okidajući se stiskanjem navigacijskog gumba nazad i/ili navigacijom izvan Aktivnosti) su samo neki od evenata koji se pozivaju tokom jednog životnog vijeka jedne Aktivnosti.

Aktivnosti su vrlo bitan dio aplikacijskog sveukupnog životnog vijeka te samog programiranja aplikacija za platformu Android.

Sve aktivnosti počinju pozivanjem metode `Context.startActivity()` te sve Aktivnosti moraju imati svoju xml deklaraciju u datoteci naziva AndroidManifest.xml koja uz to što sadrži generirane informacije o samoj aplikaciji, aktivnostima i servisima sadrži i dozvole koje zahtjeva aplikacija.

Tipičan primjer AndroidManifest.xml datoteke:

AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example"
    android:versionCode="1"
    android:versionName="1.0">

    <uses-sdk android:minSdkVersion="15" />

    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:label="@string/app_name"
        android:icon="@drawable/ic_launcher">
        <activity
            android:name="MyActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Slika 6 AndroidManifes

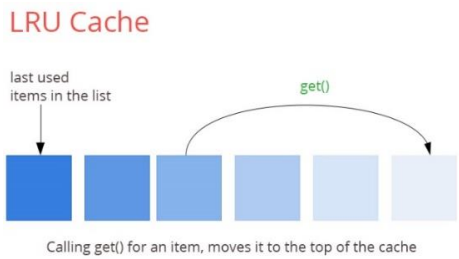
Svaka aplikacija mora imati jednu glavnu i početnu Aktivnost koja se prva okida na pokretanju same aplikacije. Android integrirani okoliši uglavnom automatski uvijek generiraju jednu takvu aktivnost naziva MainActivity.java.

U idealnom slučaju sve Android aplikacije započete od strane korisnika ostaju u memoriji što čini samo ponovno pokretanje aplikacija puno brže nego što bi bilo bez toga iako je sama memorija na Android uređajima limitirana. Kako bi se rješavali ti problemi sa manjkom memorije Android operacijski sustav ima dozvolu da sam ubije pojedine aplikacijske procese ako to poželi ali mora pratiti određeni set pravila odnosno prioriteta po kojim se on vodi kada

izvodi takve akcije ubijanja aplikacija. Ubijanje aplikacijskih procesa oslobađa memoriju za nove aplikacijske procese koje korisnik u tom trenutku zatraži. Aktivnosti u sistemu se vode kao stog Aktivnosti. Kada je nova aktivnost započeta ona se smješta na vrh stoga i postaje aktivnost koja se pogoni. Preostale Aktivnosti uvijek ostaju ispod u samom stogu i neće doći u stanje prvog plana eng. *Foreground* dok je trenutna aktivnost aktivna.

Sistem prioriteta:

- stanje prvog plana: događa se između prvog poziva onResume() sve do zadnjeg poziva onPause().
    - aplikacija s kojom korisnik vrši interakciju putem Aktivnosti ili koja ima servis koji je vezan za Aktivnost
    - servis koji izvršava neku od svojih metoda životnog vijeka
- vidljivo stanje: događa se između onStart() i onStop() gdje korisnik može još vidjeti Aktivnost ali ne može s njom može vršiti interakciju.
    - korisnik ne vrši interakciju sa Aktivnosti ali je aktivnost još i dalje parcijalno vidljiva
    - aplikacija ima servis koji je korišten od strane neaktivne ali vidljive Aktivnosti
- pozadinsko stanje
    - aplikacija koje samo zaustavila aktivnosti i bez servisa je
    - android ih drži u pred memoriji naziva LRU (Least Recently Used) koji sadrži listu svih nedavno korištenih aplikacija te terminira onaj koji je prvi ušao u samu listu
- prazno stanje
    - aplikacija bez ikakve aktivne komponente svi procesi u ovoj grupi se dodaju u LRU pred memorija[29]

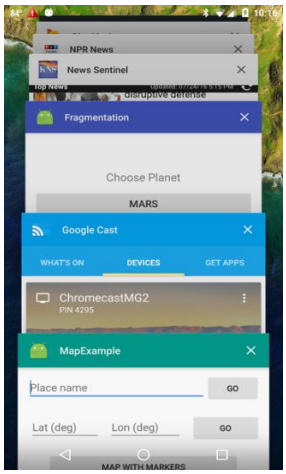


Slika 7 LRU Cache..

Izvor: http://cqmsjt.com/files6/lru-cache.html

[29] Activity. Android Developers. https://developer.android.com/reference/android/app/Activity.html

Android uređaji imaju ugrađeni mehanizam za pronalaženje aplikacija koje su nedavno bile izvršavane u stanju prvog plana. Počevši od Androida 4.0 ova funkcionalnost je zamijenjena iz akcije držanja Home gumba na stiskanje krajnjeg desnog gumba koji otvara ekran koji prikazuje aplikacije koje su se koristile te koje se sortirane na samoj listi po vremenu zadnjeg korištenja.



Slika 8 Aplikacijski stog.

Izvor: http://www.anandtech.com/show/8628/the-android-50-lollipop-review/4

Cijeli vijek jedne aktivnosti se događa između prvog poziva na onCreate sve do zadnjeg njezinog poziva onDestroy:

onCreate(),

- poziva se u trenutku stvaranja same Aktivnosti. Koristi se za inicijalizaciju aktivnosti npr. za izradu korisničkog sučelja

onResume(),

- poziva se kada aktivnost postaje opet vidljiva i korisnik započinje interaktivnost. Koristi se za inicijalizaciju polja, slušača itd.

onPause(),

- zove se kada neka druga Aktivnost dolazi u stanje prvog plana. Uvijek se zove prije nego što Aktivnost postaje nevidljiva samom korisniku. U ovom stanju se ispuštaju svi resursi ili se samo stanje varijabli i drugih podataka sprema za ponovnu upotrebu.
- preporuča se u slučaju da su podatci u Aktivnosti bitni da se u ovom eventu uvijek spašavaju podatci jer Android uvijek može ubiti aktivnost.

onStop().

- poziva se kada je sama aktivnost postala nevidljiva korisniku. Procesi koji su dosta zahtjevni za procesor se često obavljaju u ovom stanju tipa zapisivanje u bazu podataka.
- ova metoda ima garanciju za svoj poziv započevši od API-a 11

Uz dodatak na ove evente koji se garantirano pozivaju za razliku od npr. onDestroy() eventa koji nema garanciju da će se pozvati pa ga se i ne koristi baš previše u samom procesu razvijanja aplikacija imamo još evente same konfiguracije. Eventi konfiguracije se događaju kada se dogodi promjena u konfiguraciji. Oni se događaju kada korisnik promjeni samu orijentaciju ekrana. U tom slučaju Android ponovo pokriće aktivnu Aktivnost jer postoji šansa da će se koristiti drugačiji resursi za drugačiju orijentaciju ekrana te je to jedini sigurni način za rukovanje promjenama u konfiguraciji. Pošto aktivnosti već znaju kako spasiti svoje stanje i izvršiti rekreaciju istoga ovo je zgodan način za to. [30]

`onConfigurationChanged(Configuration)` - Za bilo kakve promjene u konfiguraciji uvijek se poziva događaj koji prima samu konfiguraciju prema kojoj programer može odlučivati što uraditi sa svojim podatcima. Programer ako to želi i to mu je u cilju može onesposobiti rotiranje same Aktivnosti te se ovaj poziv i sama promjena konfiguracija nikad neće pozvati. To ostvarimo zapisivanjem u sam AndroidManifes.xml datoteka – npr. `android:screenOrientation="landscape"`[30]

`onSaveInstanceState(Bundle)` - Ovaj event se poziva prije stavljanja Aktivnosti u pozadinsko stanje te on dopušta da se spremaju bilo kakvi dinamički podatci u Aktivnosti u odgovarajući Bundle objekt. Često su to jednostavni podatci npr. korisnikovo zapisivanje nekakvog teksta, kako se ne bi izgubili radi promjene same konfiguracije.

`onRestoreInstanceState()` – u ovom pozivu se rekreira samo stanje aplikacija

2.2.10. XML

XML (Extensible Markup Language) je jezik za označavanje koji koristi set pravila za kodiranje dokumenata u formatu koji je čitljiv ljudima i mašinama.[31]

XML je razvijen od strane XML Working Group. Ciljevi dizajna bili su:

- xml će se jednostavno i iskreno koristiti putem Interneta

[30] Handling Runtime Changes. Android Developers. https://developer.android.com/guide/topics/resources/runtime-changes.html
[31] XML. Extensible Markup Language (XML) 1.0 (Fifth Edition). https://www.w3.org/TR/REC-xml/#sec-origin-goals

- xml će podržavati razne aplikacije
- xml će biti podržan od strane SGML
- biti će jednostavno pisati programe koji procesiraju XML dokumente
- broj opcionalnih mogućnosti će se svesti na minimum
- xml dokumenti moraju biti čitljivi od strane ljudi i čisti
- xml dizajn se mora moći pripremati brzo
- xml dizajn će biti formalan i precizan
- xml dokumenti se moraju moći lagano stvarati i razvijati
- sažetost je od minimalne važnosti[31]

XML u okviru razvijanja aplikacija za Android se koristi u deklariranju elemenata korisničkih sučelja. Android daje jednostavan XML vokabular pomoću kojeg se deklariraju View klase i pod klase. Fokus se usmjerava nad grupaciji elemenata i pod elemenata. Svaka xml. datoteka koja deklarira određeni tlocrt ili plan korisničkog sučelja mora imati jedan korjeniti element koji mora biti View ili ViewGroup objekt. Nakon korjenitog elemente se mogu dodavati njegovi pod elementi kroz hijerarhiju. Svakom elementi je moguće dodavati njegova svojstva koja izravno utječi na korisničkog sučelje pojedinog elementa. Java kod može referencirati pojedini element korisničkog sučelja te nakon povezivanja i inicijalizacije elementa u java kodu nad njim može raditi razne operacije. Za inicijalizaciju svakog elementa potrebno je znati i dodijeliti mu ID atribut.[32]

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

Programski isječak 1 xml layout primjer

[32] Layouts. Android Developers. https://developer.android.com/guide/topics/ui/declaring-layout.html

3. Analiza biblioteka

U projektu razvijanja aplikacije kao aplikativnog primjera primjene Androida kao platforme za izvedbu aplikacija korištene su mnoge biblioteke.

U ovom poglavlju ćemo ukratko o njima pisati te se osvrnuti na razlog zašto je pojedina biblioteka korisna za razvoj aplikacija te koje su razlike implementacije pojedinih mogućnosti između njih i rodnog koda kojeg pruža Android SDK. Proći ćemo kroz one najkorištenije među razvojnim programerima.

3.1 Retrofit i GSON

Službena stranica Retrofit biblioteke u jednoj rečenici opisuje Retrofit kao za tipove sigurnog REST klijenta za Android i Javu.

Koriste se anotacije koje služe za opis HTTP upita, URL zamjenski parametri te upitni parametri za potporu koji su svi zajedno integrirani po standardu. Ona također omogućuje višedijelne eng. *multipart* upite u tijelu te slanje datoteka.

Retrofit koji je razvijen od strane Squarea je jedan od najpopularnijih HTTP klijenata za Android zbog svoje jednostavnosti i performansa.[33]

Sljedeći kod je uzet sa Retrofit službene stranice:
U koracima ćemo ukratko opisati neke od primjera metoda:

- @GET("users/list")
    - svaka metoda mora imati HTTP anotaciju koja omogućuje upit i URL
    - postoji pet već dodanih anotacija : GET, POST, PUT, DELETE
- @GET("users/list?sort=desc")
    - mogu se specificirati upitni eng. *query* parametri
- @GET("group/{id}/users")
    - Call<List<User>> groupList(@Path("id") int groupId);
        - URL upit se može nadograđivati dinamično sa blokovima koji mijenjaju put samog URL-a i parametrima u metodi putem @Path anotacije

[33] Retrofit. https://square.github.io/retrofit/

        - zamjenski blok se okružuje zagradama{ }

- @POST("users/new")
    - Call<User> createUser(@Body User user);
        - objekt se može specificirati za upotrebu kao HTTP upit unutar Body elementa
    - @Headers("Cache-Control: max-age=640000")
        - moguća je upotreba statičkih zaglavlja

Upiti se mogu izvršavati sinkrono i asinkrono.

Po standardnom načinu Retrofit može obraditi HTTP tijela u OkHttp ResponseBody tip i može samo primati RequestBody tip za @Body parametar. Zbog toga je moguće dodati razne pretvarače kako bi se podržavalo i druge tipove.

Ovo su neki od najpopularnijih:

- Gson: com.squareup.retrofit:converter-gson
    - Gson biblioteka se koristila u svrhu izrade Photo Archive aplikacije te ćemo u narednom primjeru uz Retrofit vidjeti i njezinu implementaciju
- Jackson: com.squareup.retrofit:converter-jackson
- Moshi: com.squareup.retrofit:converter-moshi
- Protobuf: com.squareup.retrofit:converter-protobuf
- Wire: com.squareup.retrofit:converter-wire
- Simple XML: com.squareup.retrofit:converter-simplexml

Za rad u Retrofitu potrebno je stvoriti najmanje 3 klase:

1. model klasu koja služi kako bi se podatci postavili u mapu iz JSON formata
2. sučelje koji definira sve moguće HTTP operacije
3. Retrofit.Builder klasu – instancu koja koristi interfejs. Builder API koji dopušta definiranje URL završnih točaka za HTTP operacije.

U sljedećem kodu ćemo vidjeti primjer ovih klasa.

```
import java.util.List;

public class ResponseItems {
    List<Item> items;

private class Item {
    String title;
    String link;

    @Override
    public String toString() {
        return(title);
    }
}
}
```

Programski isječak 2 Klasa modela

```
import retrofit2.Callback;
import retrofit2.http.GET;
import retrofit2.http.Query;
import retrofit2.Call;

public interface RetrofitApi {
    @GET("/2.2/items?order=desc&sort=creation")
    Call<ResponseItems> loadItems (@Query("tagged") String tags);
}
```

Programski isječak 3 REST API sučelje

```
Gson gson = new GsonBuilder()
                .setDateFormat("yyyy-MM-dd'T'HH:mm:ssZ")
                .create();
        Retrofit retrofitClient = new Retrofit.Builder()
                .baseUrl("https://api.stackexchange.com")

.addConverterFactory(GsonConverterFactory.create(gson))
                .build();

        RetrofitApi retrofitApi =
retrofitClient.create(RetrofitApi.class);

        Call<ResponseItems> call = stackOverflowAPI.loadItems
("android");
        call.enqueue(this);
```

Programski isječak 4 Inicijalizacija Retrofit klase

Implementacijom gornjeg sučelja dobivamo dvije metode `onResponse` te onFailure

```
@Override
public void onResponse(Call<ResponseItems> call,
Response<ResponseItems> response) {
        ArrayAdapter<Item> adapter = (ArrayAdapter<Item>)
getListAdapter();
        adapter.clear();
        adapter.addAll(response.body().items);
    }

    @Override
    public void onFailure(Call<StackOverflowQuestions> call,
Throwable t) {
        Toast.makeText(MainActivity.this, t.getLocalizedMessage(),
Toast.LENGTH_SHORT).show();
    }
}
```

Programski isječak 5 Implementacija Retrofit Call metoda

Za implementaciju sličnog koda koristeći klase iz Andorid SDK paketa nam prvo treba stvaranje AsyncTask klase koja služi za izvršavanje mrežnog upita u pozadinskom zadatku te puno ostalih pomoćnih klasa za prolaženje kroz rezultate i kasnije ručno obrađivanje JSON formata u zasebne objekte.

Primjer:

```
public class MyAsyncTask extends AsyncTask<URL, Void, String> {

        private HttpURLConnection urlConnection;
        private Context mContext;
        private ProgressDialog mDialog;
        private TaskListener mListener;

        public MyAsyncTask(Context context, TaskListener listener) {
            this.mContext = context;
            this.mListener = listener;
        }

        @Override
        protected void onPreExecute() {
            super.onPreExecute();
        }

        @Override
        protected String doInBackground(URL... params) {
```

```
            StringBuilder result = new StringBuilder();

            try {
                URL url = params[0];
                urlConnection = (HttpURLConnection)
url.openConnection(/*proxy*/);
                urlConnection.setDoInput(true);
                urlConnection.setConnectTimeout(20 * 1000);
                urlConnection.setReadTimeout(20 * 1000);

                if (urlConnection.getResponseCode() ==
HttpURLConnection.HTTP_OK) {
                    InputStream in = new
BufferedInputStream(urlConnection.getInputStream());
                    BufferedReader reader = new BufferedReader(new
InputStreamReader(in));
                    String line;
                    while ((line = reader.readLine()) != null) {
                        result.append(line);
                    }
                }
            } catch (Exception e) {
                e.printStackTrace();
            } finally {
                urlConnection.disconnect();
            }
            return result.toString();
        }

        @Override
        protected void onPostExecute(String s) {
            super.onPostExecute(s);
            mListener.onTaskComplete(s);
        }
    }
```

Programski isječak 6 AsyncTask

Nakon što smo dobili niz znakova u metodi onPostExecute taj niz znakova bi tada ručno obrađivao na ovakav sličan način što u slučaju da se JSON sastoji od puno objekata i lista može kod pretvoriti u teško čitljivo stanje

```
JSONObject mainObject = new JSONObject(Your_Sring_data);
JSONObject uniObject = mainObject.getJSONObject("university");
String  uniName = uniObject.getJSONObject("name");
String uniURL = uniObject.getJSONObject("url");
```

```
JSONObject oneObject = mainObject.getJSONObject("1");
String id = oneObject.getJSONObject("id");
```

Programski isječak 7 Ručna obrada JSON objekata

Zaključak je vrlo lagan. Implementacija mrežnih upita putem Retrofit i GSON biblioteka nam puno olakšava stvari. Ostvaruje se automatska implementacija pozadinskog djelovanja mrežnog upita bez napornog stvaranja nove AsyncTask klase kako bi se izbjegli upiti na dretvi korisničkog sučelja. Dobivaju se ogromne mogućnosti širenja same klase sučelja u kojoj jednostavno jednu ispod drugih možemo dodavati nove upite. Skoro pa automatsko obrađivanje JSON formata u naše objekte stvorene od strane Model klase uz već nabrojene mogućnosti dobivamo jednu solidnu platformu za mrežne upite koja se vrlo lagano može nadograđivati i lagana je i čista u svojoj implementaciji.

3.2 Butterknife

Butterknife biblioteka je proizašla iz svakodnevne potrebne android programera za referenciranjem potrebnih objekata View klase. [34]

Standardan način po kojem bi se takva operacija radila je:

```
- TextView textView = (TextView)
  findViewById(R.id.someTextViewid);
- ImageView imageView = (ImageView)
  findViewById(R.id.someImageViewid);
- RelativeLayout relativeLayout = (RelativeLayout)
  findViewById(R.id.someRelLayoutid);
```

Butterknife uvelike taj proces olakšava te je moguće implementacijom Butterknife priključka u Android Studio izbjeći samo pisanje koda te desnim klikom na glavni tlocrt generirati Butterknife injekcije:

```
- @BindView(R.id.someTextViewId) TextView textView;
- @BindView(R.id.someImageViewId) ImageView imageView;
- @BindView(R.id.someRelLayoutId) RelativeLayout relativeLayout;
```

Butterknife biblioteka također može spojiti resurse za korištenje unutar projekta:

```
- @BindString(R.string.title) String title;
- @BindDrawable(R.drawable.graphic) Drawable graphic;
- @BindColor(R.color.red) int red; // int or ColorStateList field
- @BindDimen(R.dimen.spacer) Float spacer; // int
```

[34] Butterknife. http://jakewharton.github.io/butterknife/

3.3 Picasso

Picasso je biblioteka koja nam služi kako bi si pojednostavili samo učitavanje slike s mreže. Mnoge stvari koje mogu biti problematične Picasso rješava automatski poput:

- recikliranje ImageView objekta i otkazivanje učitavanja unutar adaptera
- kompleksne transformacije slike s minimalnim memorijskim zahtjevima
- automatska pred memorija[35]

Sliku možemo učitati u jednoj liniji:

```
Picasso.with(context).load(“http://i.imgur.com/DvpvklR.png").into(imag
eView);
```

Programski isječak 8 Učitavanje slike putem Picasso biblioteke

Ovako bi izgledalo učitavanje iste slike kad ne bi koristili Picasso biblioteku

```
public static Bitmap loadBitmap(String url) {
    Bitmap bitmap = null;
    InputStream inputStream = null;
    BufferedOutputStream outputStream = null;

    try {
        inputStream = new BufferedInputStream(new URL(url).openStream(),
IO_BUFFER_SIZE);
        final ByteArrayOutputStream byteArrayStream = new
ByteArrayOutputStream();
        outputStream = new BufferedOutputStream(byteArrayStream,
IO_BUFFER_SIZE);
         copy(inputStream, outputStream);
        outputStream.flush();

        final byte[] data = byteArrayStream.toByteArray();
        BitmapFactory.Options options = new BitmapFactory.Options();
        //options.inSampleSize = 1;

        bitmap = BitmapFactory.decodeByteArray(data, 0,
data.length,options);
    } catch (IOException e) {
         Log.e(TAG, "Error: " + url);
    } finally {
        closeStream(in);
        closeStream(out);
    }

    return bitmap;
```

[35] Picasso. http://square.github.io/picasso/

```
}
imageView.setImageBitmap(loadBitmap(“http://i.imgur.com/DvpvklR.png”);
```

Programski isječak 9 Ručno učitavanje slike s mrežnog iyvora

3.4 Otto

Biblioteku Otto smo koristili u projektu kao event bus. Biblioteka je dizajnirana da razdvaja dijelove aplikacije ali da im u isto vrijeme omogućava međusobnu komunikaciju. On je odvojen od Guava projekta ali je nadopunjen specijalizacijom za Android platformu.

Njegova je svrha u komunikaciji. Koncept je takav da omogućuje slanje podataka između odvojenih dijelova aplikacije. Svaki dio odnosno klasa koja je se moraju registrirati na glavnu Bus klasu jedne instance eng. *singleton* te pretplatiti na željeni event kojeg žele očekivati. Event se može poslati s bilo kojeg mjesta u aplikaciji zajedno s podatcima. U trenutku kada klasa više ne želi biti pretplaćena na bilo koji event ona može ukinuti registraciju.[36]

Bus klasa jedne instance:

```
public class Bus {
    private static com.squareup.otto.Bus sBus;

    private Bus() {
    }

    public static com.squareup.otto.Bus getInstance() {
        if (sBus == null) {
            sBus = new com.squareup.otto.Bus();
        }

        return sBus;
    }
}
```

Programski isječak 10 Bus singleton klasa

Inicijalizacija u samoj klasi Aplikacije:

```
public class PhotoArchiveAplication extends Application {

    public static Bus bus;
    public static Retrofit photoArvhiveClient;
    public static PhotoArchiveApi photoArchiveApi;
```

[36] Otto. http://square.github.io/otto/

```
    @Override public void onCreate() {
        super.onCreate();
        FacebookSdk.sdkInitialize(getApplicationContext());

        bus = new Bus(ThreadEnforcer.MAIN);
}
```

Programski isječak 11 Inicijaliyacija Bus klase

Registracija na glavni Bus objekt:

- `Bus.getInstance().register(this);`

Ukidanje registracije na glavni Bus objekt:

- `Bus.getInstance().unregister(this);`

Pretplaćivanje na event:

```
@Subscribe public void showSnackBar(final ShowSnackbarEvent
showSnackbarEvent) {
        snackbar = Snackbar
                .make(rootLayoutMain,
showSnackbarEvent.getMessage(), Snackbar.LENGTH_LONG);
        snackbar.show();
}
```

Programski isječak 12 Pretplata na Bus event

Kada ne bi koristili Otto biblioteku nešto slično se može napraviti sa LocalBrodecastReciver-om:

Deklariranje BrodecastReciver objekta za dohvat podataka s odgovarajućim ključem.

```
private static final String EXAMPLE_ACTION = "example";
private final BroadcastReceiver brodcastReciver = new
BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        if(EXAMPLE(intent.getAction())) {
            ...
        }
    }
};
```

Programski isječak 13 BrodecastReciver

Registracija na LocalBroadcastManager s potrebnim IntentFilterom

```
@Override
public void onResume() {
    super.onResume();
    LocalBroadcastManager manager =
LocalBroadcastManager.getInstance(getActivity());
    IntentFilter filter = new IntentFilter(EXAMPLE_ACTION);
    manager.registerReceiver(this.receiver, filter);
}
```

Programski isječak 14 Registracija na LocalBroadcastMenager

Ukidanje registracije

```
@Override
public void onPause() {
    super.onPause();
    LocalBroadcastManager manager =
LocalBroadcastManager.getInstance(getActivity());
    manager.unregisterReceiver(this.receiver);
}
```

Programski isječak 15 Ukidanje registracija na LocalBroadcastManager

Ovako bi se izvršavalo slanje

```
LocalBroadcastManager manager =
LocalBroadcastManager.getInstance(getActivity());
manager.sendBroadcast(intent);
```

Programski isječak 16 Slanje podataka putem LocalBroadcastManager klase

4. Aplikacija – Photo Archive

Prvobitna svrha ove aplikacije je da posluži kao primjer razvoja aplikacija za Android platformu. Ono čime se ona bavi je pohranjivanje i dijeljenje multimedijskog sadržaja koji je u ovom slučaju baziran samo na slikovne datoteke i fotografije.

Funkcije koje aplikacija omogućava:

- registraciju i login korisnika,
- uređivanje podataka korisnika,
- pregledavanje spremljenih slika na servisu,
- pregledavanje informacija o slici,
- pregledavanje informacija o korisniku,
- pregledavanje slika od korisnika,
- učitavanje pune veličine slike i njezino spremanje na disk,
- pregledavanja svih tag objekata kojima se obilježavaju slike te klikom na pojedini tag objekt dobivanje svih slika pod tim tag objektom,
- slanje slike na servis putem kamere ili izborom slike iz galerije već postojećih slika.

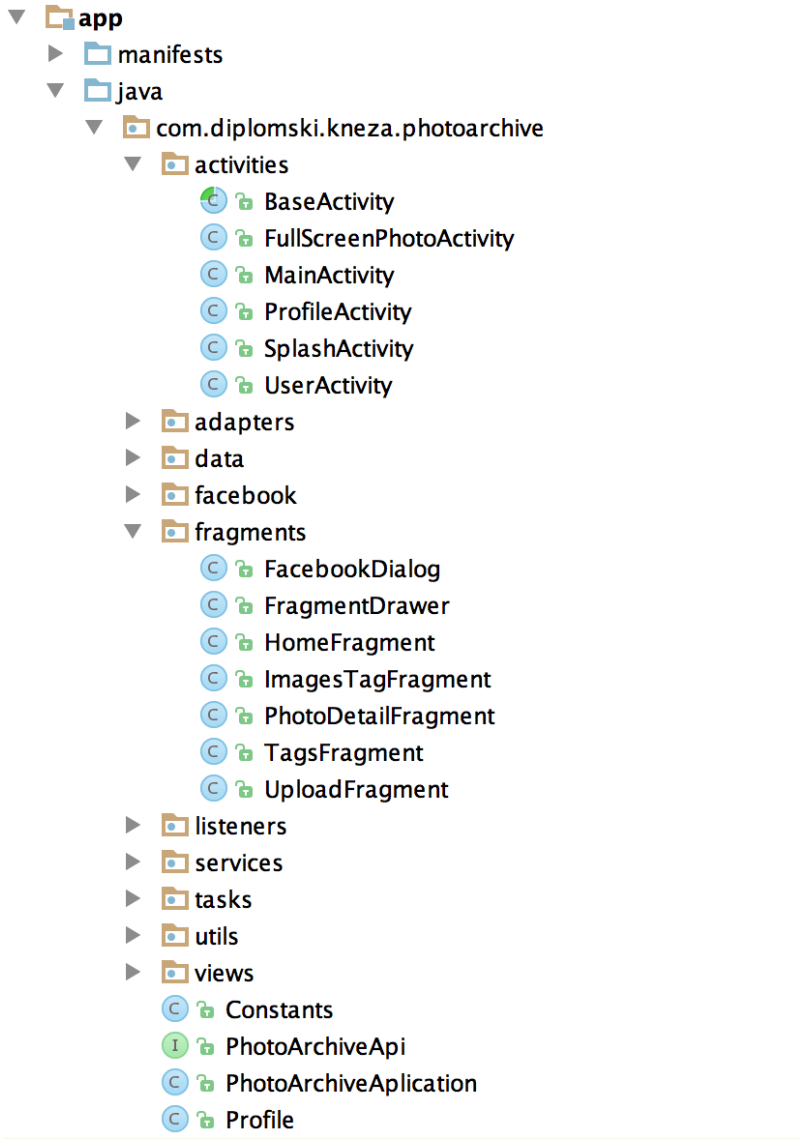
Moguće nadogradnje aplikacije:

- implementacija pretraživanja na serverskoj strani
- implementacija paginacije kojom bi se izbjeglo učitavanje svih slika odjednom
- dodavanje mnogih drugih opisa pojedine slike kako bi se bolje mogle katalogizirati u budućnosti

Uzevši ovakav projekt kao primjenu razvoja aplikacija na Android platformi razmišljanje je išlo u smjeru toga da je većina Android aplikacija danas spojena na mrežu te uspostavlja nekakvu komunikaciju sa servisom. Zapravo teško je naći neku popularnu aplikaciju koja nema spajanje na mrežu kao primarnu funkciju. Također današnje aplikacije često imaju primat na dijeljenju informacija te principu da korisnici sami stvaraju sadržaj. Na temelju toga odluka je donesena da Photo Archive bude prilog za ovaj diplomski rad. Za implementaciju aplikacije kao integrirani okoliš za razvoj koristio se Android Studio.

U narednom dijelu ćemo proći kroz neke dijelove njezine implementacije. Prvotno se usredotočiti na Activity klase i Fragment klase koje predstavljaju korisničko sučelje same aplikacije.

Aplikacije je strukturirana u pakete počevši s glavnim folderom java kojeg imaju svi Android projekti:



Slika 9 com.diplomski.kneza.photoarchive - primarni paket projekta s otvorenim paketima fragments i activities

4.1 Aktivnosti

BaseActivity

- ova aktivnost je apstraktna klasa koja primarno služi kao klasa roditelja koju nasljeđuju sve aktivnosti u projektu. Ona omogućuje dodavanje Toolbar komponente, komponente navigacijske ladice te dodavanje kontrole FloatingActionButton te dodavanja naziva naslova pojedine aktivnosti.

FullScreenPhotoActivity

- aktivnost koja služi za učitavanje pune slike s servisa,
- ona prima URL te uz pomoću Picasso biblioteke s servisa učitava sliku,
- implementira se biblioteka PhotoView za kontrole približavanja slike
- https://github.com/chrisbanes/PhotoView.

MainActivity

- aktivnost nasljeđuje sve funkcije klase BaseActivity,
- sadrži glavni tlocrt a`ctivity_main.xml` u kojem se nalazi navigacijska ladica s pripadajućim fragmentom,
- implementira mogućnost SnackBar kontrole,
- ona je roditeljska aktivnost svim fragmentima.

ProfileActivity

- ova aktivnost se brine o informacijama korisnika koji je trenutno ulogiran u aplikaciju
- omogućuje uređivanje korisničkih podataka koji se spremaju na servisu ali i lokalno
- određivanje lokacije korisnika obavlja se ručno pomoću klase Geocoder koja kako korisnik upisuje rijeci u listi ispod nudi moguće rezultate te automatski putem GPS-a uređaja koristeći se tehnologijom GooglePlayService
- klasa BackgroundLocationService se brine za dohvaćanje lokacije koja je zapravo pozadinski servis koji koriste GooglePlayServise posebice GoogleApiClient za primanje lokacije te koja nakon što dobije lokaciju javlja ProfileActivity klasi putem biblioteke Otto eventom da je lokacija dohvaćena te se ista sprema lokalno u SharedPrefrences

SplashActivity

- početka aktivnost koja se prva pali pokretanjem aplikacije
- u manifest datoteci označena kao Intent Filter:

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

Programski isječak 17 Primjer intent-filter funkcija

- omogućuje funkcije login i registriranje koje su ujedno dvije servisne metode kojima se korisnik služi upisujući svoje podatke
- sadrži i mogućnost logina putem Facebook API-a (nije implementirano na servisu)

UserActivity

- aktivnost koja se brine o prikazu podataka o korisnicima te njegovim objavljenim slikama
- podatci: puno ime, email, telefon, lokacije te korisnička slika

4.2. Fragmenti

FragmentDrawer

- fragment koji se predstavlja korisničko sučelje navigacijske ladice koja se brine o navigaciji kroz samu aplikaciju
- Sadrži:
    - sliku profila na čiji pritisak se otvara ProfileActivity
    - Home koji otvara HomeFragment
    - Tags koji otvara TagsFragment
    - Upload koji otvara UploadFragment
    - Log of koji izvršava odjavljivanje na servisu te vraća korisnika na SplashActivity ekran

HomeFragment

- obnaša funkciju prvog fragmenta tj. ekrana koji se otvara ulaskom u aplikaciju nakon prijavljivanja u sustav u SplashActivity klasi
- sadrži dvije liste pogonjene kontrolom RecyclerView
    - lista novih slika
    - lista popularnih slika
- obje liste su predstavljene u horizontalnom formatu te svaka jedinica lista je predstavljena kao umanjena slika s naslovom ispod iste
- klikom na pojedinu sliku se otvara PhotoDetailFragment

PhotoDetailFragment

- fragment koji predstavlja informacije o navedenoj slici
- prikazuje samu umanjenu sliku s gumbom na dnu za otvara FullScreenPhotoActivity klase koja učitava punu veličinu iste slike te omogućuje kontrole uvećavanja
- prikazuje sve tag objekte slike
- informacije o slici te korisnika koji je zaslužan za pohranjivanje slike na serveru te klikom na korisnikovu sliku ili ime se otvara UserActivity
- FloatingActionButton kontrola koji ima implementaciju spremanja slike na disk u zasebni PhotoArchive folder te koji u slučaju da je slika već skinuta javlja da je ista skinuta i omogućuje korisnikov pregled folder nakon skidanja ako korisnik ima instaliran neki od menadžera za foldera na uređaju

TagsFragment

- fragment koji predstavlja prvih 13 tag objekata koje servis izbaci u listi
- koristi se kontrola RecyclerView za prikaz liste
- klikom na pojedini tag objekt dobivamo listu svih slika koje su označene tim tag objektom

UploadFragment

- fragment koji se brine za slanje slike s podatcima na servis
- omogućuje učitavanje slike iz galerije uređaja te učitavanje slike s kamere
- FloatingActionButton implementira slanje pozivanjem dvije metode s servisa
    - `@POST("api/image/createimage")`
    - `@POST("api/images/upload")`

4.3. Ostale klase

Aplikacija uz ove navedene dijelove još sadrži mnoge druge pomoćne klase koje se brinu o raznim stvarima poput FacebookLogin klase koja se primarno brine za login i komunikaciju s Facebook API-em te dohvaćanje podataka s iste kroz zatražena dopuštenja. Razne klase u data paketu poput: Image, Tag, User, LoginInformation, NavDrawerItem itd. su klase koje služe sa skladištenje podataka i stvaranje objekata od istih te klase za obrađivanje JSON podataka putem GSON biblioteke.

Paket adapters sadrži klase koje nasljeđuju RecyclerView.Adapter te služe za komunikaciju između korisničkog sučelja i raznih lista podataka.

Paket listeners sadrži klase sučelja:

- OnActivityResultNotifier - javlja rezultat operacije nad Activity klasom
- OnActivityResultListener - sluša na isti rezultat
- OnClickListener - sluša na onClick evente
- OnFacebookLoginListener - javlja da je Facebook login završen

Paket tasks sadrži klase koje nasljeđuju AsyncTask klasu

- DownloadImageTask - brine se o skidanju slike s servera na uređaj
- GetSuggestionsTask - implementira Geocoder funkcionalnost te u pozadinskom zadatku izračunava moguće rezultate kako korisnik upisuje lokaciju te ih prezentira na korisničkom sučelju
- SetLocationTask - pretvara lokacijsku longitudu i latitudu u naziv grada u kojem se nalazi korisnik

Paket Utils sadrži klasu DetailsTransition koja se brine o tranziciji dok klasa Utils sadrži razne statičke pomoćne metode za sam projekt. Klasa CircleTransformBorder se brine o transformaciji samog formata slike tokom skidanja te se implementira putem Picasso biblioteke.

Klasa Profile

- klasa je implementirana kao objekt jedne instance kako bi se izbjegla stalna inicijalizacija klase
- ona obnaša funkciju spremanja podataka u SharedPrefrences sustav koji omogućuje spremanje jednostavnih objekata poput String, Integer i sl. unutar memorije same aplikacije te predstavlja jednostavan API za spremanje i dohvaćanje istih.

Primjer implementacije spremanja i dohvaćanja String objektnih podataka o korisničkom e-mail-u putem klase Profil.

```
private SharedPreferences prefs;
private static Profile instance = null;
public static final String EMAIL = "com.hajjnet.salam.data.EMAIL";

private Profile(Context context) {
```

```
        prefs =
PreferenceManager.getDefaultSharedPreferences(context.getApplication
Context());
    }

    public static Profile getInstance(Context context) {
        if (instance == null)
            instance = new Profile(context);

        return instance;
    }

public void setEmail(String email) { prefs.edit().putString(EMAIL,
email).apply(); }
public String getEmail() { return prefs.getString(EMAIL, ""); }
```

Programski isječak 18 Primjer implementacije klase Profile

6. Zaključak

Android kao platforma za razvoj mobilnih aplikacije za dijeljenje i arhiviranje multimedijskog sadržaja se pokazala kao uspješna. Android-ov aplikacijski okvir i životni vijek predstavljaju odličan temelj na kojemu se mogu raditi i modificirati aplikacije koristeći se događajima životnog vijeka poput onCreate i onStop. Klase naziva Activity i Fragment pružaju odlične mogućnosti za upravljanjem korisničkim sučeljem a manipulacija .xml koda daje standardizirani i već poznati način stvaranja View kontrola.

Unatoč problemu pokrivenosti mnogih različitih mobilnih uređaja s različitim veličinama ekrana i različitim verzijama Androida, platforma je fluidna i svjesno obavlja svoje zadaće. Android se sam brine o većini stvari vezanih za kompatibilnost za koje se razvojni programer može ako želi dodatno pobrinuti. Problem pokrivenosti raznih mobilnih uređaja će uvijek biti postojan ali kako se Android kao platforma razvija te mobilni uređaji postaju sve hardverski brži i jači takvih problema će biti sve manje.

Android kao najzastupljeniji operacijski sustav u svijetu donosi sa sobom mnoge biblioteke koje uvelike olakšavaju proces razvoja aplikacija te omogućavaju programeru da se usredotoči na samu bit aplikacije i pomogne mu u vezi raznih tehničkih stvari poput mrežnih upita, spajanja na servise, slanje podataka među aplikativnih komponenti itd.

Photo Archive aplikacija za dijeljenje i arhiviranje multimedijskog sadržaja je poslužila kao odličan primjer aplikacije koja se temelji na dijeljenju sadržaja što u današnjem društvu jako popularno. Photo Archive aplikacija je prototip te ona pokazuje samo djelić mogućnosti Androida kao platforme ali u isto vrijeme omogućuje dodatnu nadogradnju. Upotrebom biblioteka navedenih u diplomskom radu došlo se do stabilne aplikacije jer nad svakoj od tih biblioteka stoji tim stručnjaka i testera koji osiguravaju stabilan rad biblioteke.

7. Literatura

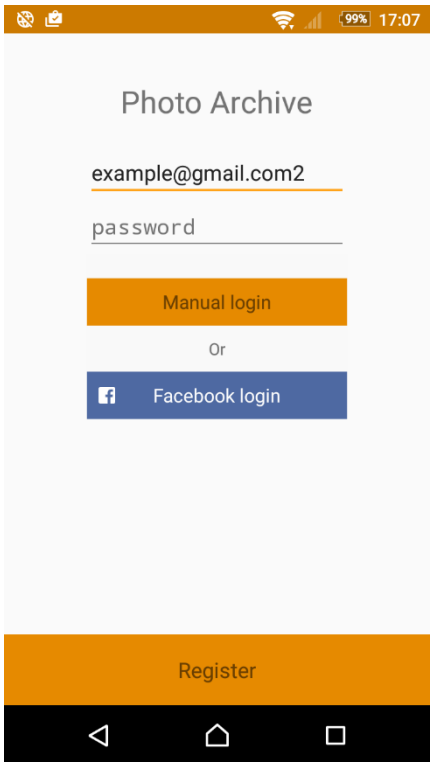
1. Activity. Android Developers. https://developer.android.com/reference/android/app/Activity.html
2. An Overview of the Android Architecture. Techotopia. http://www.techotopia.com/index.php/An_Overview_of_the_Android_Architecture
3. Android 1.5 Platform. Android Developers. https://developer.android.com/about/versions/android-1.5.html
4. Android 1.6 Platform. Android Developers. https://developer.android.com/about/versions/android-1.6-highlights.html
5. Android 2.0 Platform Highlights. Android Developers. https://developer.android.com/about/versions/android-2.0-highlights.html
6. Android 2.2 Platform Highlights. Android Developers. https://developer.android.com/about/versions/android-2.2-highlights.html
7. Android 6.0 Marshmallow. Android Developers. https://developer.android.com/about/versions/marshmallow/index.html
8. Android KitKat. Android Developers. https://developer.android.com/about/versions/kitkat.html
9. Android Lolipop. Android Developers. https://developer.android.com/about/versions/lollipop.html
10. Android Pre-History. Android Central. http://www.androidcentral.com/android-pre-history
11. Android's Early Days. Android Central. http://www.androidcentral.com/androids-early-days
12. Art and Dalvik. Android. https://source.android.com/devices/tech/dalvik/index.html
13. Art and Dalvik. Android. https://source.android.com/devices/tech/dalvik/index.html
14. Butterknife. http://jakewharton.github.io/butterknife/
15. Configuring ART. Android. https://source.android.com/devices/tech/dalvik/configure.html#how_art_works
16. Dalvik bytecode. Android Developers. https://source.android.com/devices/tech/dalvik/dalvik-bytecode.html
17. David Ehringer, March, 2010. The Dalvik Virtual Machine Arhitecture. http://davidehringer.com/software/android/The_Dalvik_Virtual_Machine.pdf

18. Debugging ART Garbage Collection. Source Android. https://source.android.com/devices/tech/dalvik/gc-debug.html
19. Gingerbread. Android Developers. https://developer.android.com/about/versions/android-2.3-highlights.html
20. Handling Runtime Changes. Android Developers. https://developer.android.com/guide/topics/resources/runtime-changes.html
21. Honeycom. Android Developers.https://developer.android.com/about/versions/android-3.0-highlights.
22. Ice Cream Sandwich. Android Developers. https://developer.android.com/about/versions/android-4.0-highlights.html
23. Jelly Bean. Android Developers. https://developer.android.com/about/versions/jelly-bean.html
24. Keeping The Device Awake. Android Developers. https://developer.android.com/training/scheduling/wakelock.html
25. Meier Reto. Professional Android 4 application development. 2012
26. Otto. http://square.github.io/otto/
27. Picasso. http://square.github.io/picasso/
28. Platform Architecture. Android Developers. https://developer.android.com/guide/platform/index.html
29. Retrofit. https://square.github.io/retrofit/
30. Security Tips. Android Developers. https://developer.android.com/training/articles/security-tips.html
31. Stefan Brahler. Analysis of the Android Architecture. 2010. https://os.itec.kit.edu/downloads/sa_2010_braehler-stefan_android-architecture.pdf
32. What is Linux. Linux. https://www.linux.com/what-is-linux

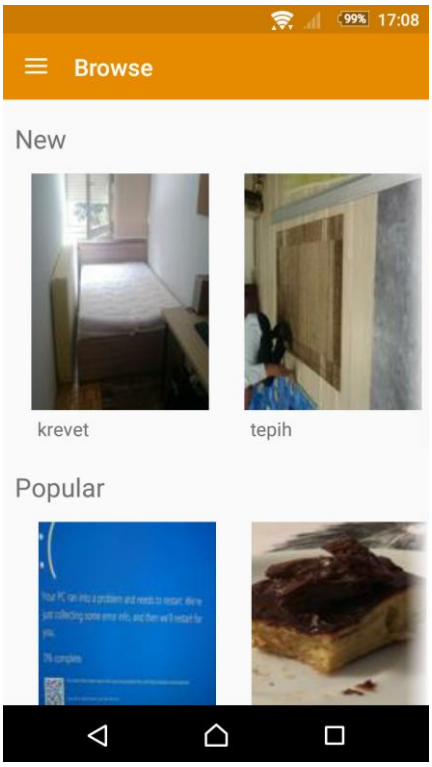
8. Slike

9. Programski isječci

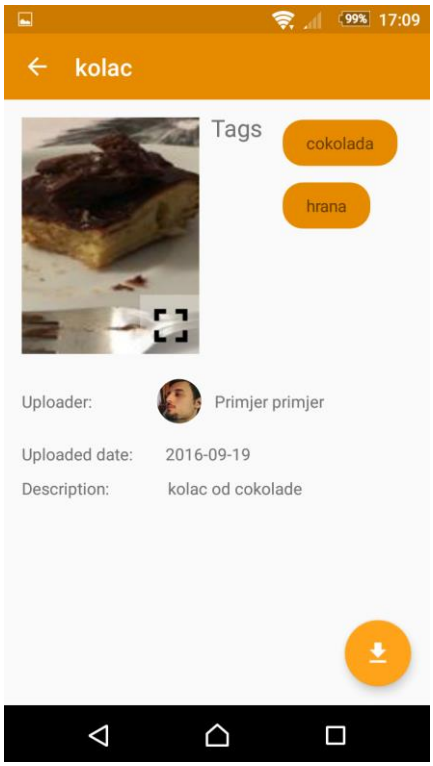
Prilog 1 - Slikovni isječci Photo Archive



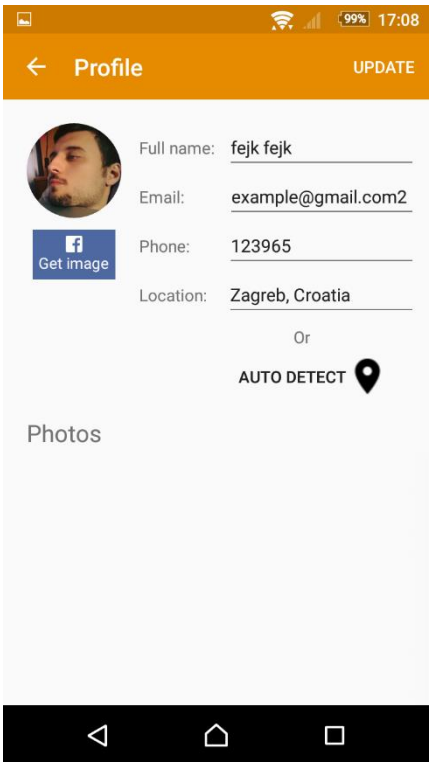
Slika 10 LoginActivity



Slika 11 HomeFragment

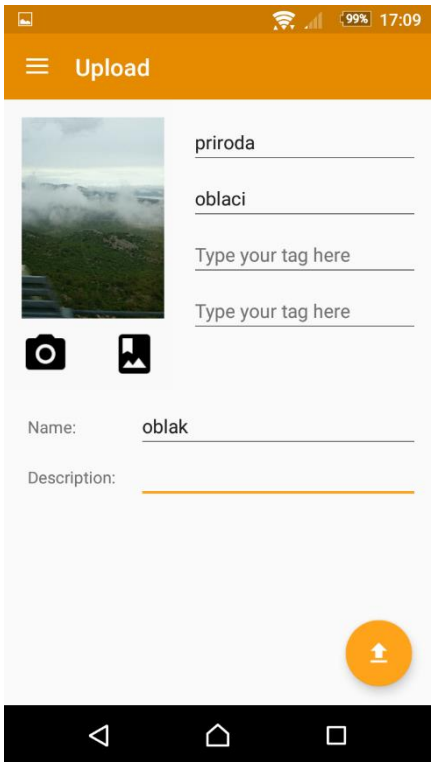


Slika 12 PhotoDetailFragment



Slika 13 ProfileActivity

Slika 14 TagsFragment



Slika 15 UploadFragment

Prilog 2 – PhotoArchiveApi.class

```
public interface PhotoArchiveApi {

	@POST("api/user/register")
	Call<Void> register(@Body RegisterInformation
	loginInformation);

	@POST("api/user/login")
	Call<Void> login(@Body LoginInformation loginInformation);

	@POST("api/user/logoff")
	Call<Void> logoff();

	@PUT("api/image/click/{click}")
	Call<Void> photoClick(@Body Image image, @Path("click") int
	click);

	@GET("api/user/{id}")
	Call<UserResponse> getUser(@Path("id") String id);

	@POST("api/user/edit")
	Call<UserResponse> editUser(@Body UserResponse userResponse);

	@GET("api/image/all")
	Call<ArrayList<ImageWithTags>> getAllImages();

	@GET("api/image/allimages")
	Call<ArrayList<Image>> getAllImagesWithoutTags();

	@GET("api/image/popular")
	Call<ArrayList<ImageWithTags>> getPopularImages();

	@GET("api/image/alltags")
	Call<ArrayList<ServiceTag>> getAllTags();

	@GET("api/image/userimages")
	Call<ArrayList<ImageWithTags>> getUserImages();

	@GET("api/image/{id}")
	Call<ArrayList<ImageWithTags>> getImageById(@Path("id") int
	id);

	@Multipart
	@POST("api/images/upload")
	Call<ResponseUpload> upload(@Part("description") RequestBody
	description,
	@Part MultipartBody.Part file);
```

```
    @POST("api/image/createimage")
    Call<Void> uploadInfo(@Body UploadInfo uploadInfo,
    @Header("Authorization") String token);

    @FormUrlEncoded
    @POST("/api/token")
    Call<TokenResponse> getToken(@Field("username") String
    username, @Field("password") String pass);

    @GET("api/image/tag/{tagId}")
    Call<ArrayList<ImageWithTags>> getImagesByTag(@Path("tagId")
    int id);

    @GET("api/user/userid/{id}")
    Call<UserResponse> getUserById(@Path("id") String createdbyId);
}
```