UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Mitja Balažic

Izdelava aplikacije Študentska borza na mobilni platformi Android

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: doc. dr. Peter Peer

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00001/2010
Datum: 05.10.2010

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MITJA BALAŽIC**

Naslov: **IZDELAVA APLIKACIJE ŠTUDENTSKA BORZA NA MOBILNI PLATFORMI ANDROID**

STUDENT STOCK EXCHANGE APPLICATION DEVELOPMENT ON ANDROID MOBILE PLATFORM

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

Predstavite izobraževalni projekt Študentska borza ter njeno spletno aplikacijo za trgovanje. Nato predstavite orodja, ki so potrebna za razvoj na platformi Android ter opišite sam razvoj mobilne različice aplikacije za trgovanje. Opišite tako razvoj odjemalca kot tudi strežniškega dela.

Mentor:
doc. dr. Peter Peer

Dekan:
prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani/-a Mitja Balažic,

z vpisno številko 63050007,

sem avtor/-ica diplomskega dela z naslovom:

Izdelava aplikacije Študentska borza na mobilni platformi Android

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom
  doc. dr. Petra Peera
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 11.4.2011 Podpis avtorja/-ice:

Zahvala

Zahvaljujem se mentorju doc. dr. Petru Peeru za vse nasvete in strokovno pomoč pri izdelavi diplomskega dela.

Zahvaljujem se tudi staršem za omogočen študij in vsem ostalim za moralno podporo.

Kazalo

Seznam uporabljenih kratic in simbolov

2D – Two Dimensional; dvo dimenzionalno

3D – Three Dimensional; tri dimenzionalno

ADT – Android Development Tools; razvojna orodja za Android

ANR – Application Not Responding; aplikacija se ne odziva

API – Application Programming Interface; vmesnik uporabniškega programa

AVD – Android Virtual Device; navidezna naprava Android

CDMA – Code Division Multiple Access; hkraten dostop z deljenjem koda

CPU – Central Processing Unit; centralno procesna enota

DDMS – Dalvik Debug Monitor Server; strežnik za nadzor razhroščevanja Dalvik

EDGE – Enhanced Data rates for GSM Evolution; izboljšan podatkovni prenos za evolucijo GSM

EVDO – Evolution-Data Optimized; evolucijsko-podatkovno optimiziran

GPS – Global Positioning System; globalni sistem za pozicioniranje

GSM – Global System for Mobile Communications; globalni sistem za mobilne komunikacije

HTML – HyperText Markup Language; jezik za označevanje hiperteksta

HTTP – Hypertext Transfer Protocol; protokol za prenos hiperteksta

HTTPS – Hypertext Transfer Protocol Secure; protokol za varen prenos hiperteksta

IDE – Integrated Development Environment; vgrajeno razvojno okolje

IDEN – Integrated Digital Enhanced Network; vgrajeno izboljšano digitalno omrežje

ITU – International Telecommunication Union; mednarodna telekomunikacijska zveza

JSON – JavaScript Object Notation; objektna notacija JavaScript

JVM – Java Virtual Machine; navidezni stroj Java

LJSE – Ljubljana Stock Exchange; Ljubljanska borza

LTE – 3GPP Long Term Evolution; 3GPP dolgoročna evolucija

MMS – Multimedia Messaging Service; storitev za pošiljanje multimedijskih sporočil

PHP – PHP: Hypertext Preprocessor; splošno uporaben skriptni programski jezik

REST – Representational State Transfer; prenos z reprezentacijskimi stanji

SD – Secure Digital; tehnologija uporabljana v podatkovnih karticah

SDK – Software Development Kit; paket za razvoj aplikacij

SMS – Short Message Service; storitev za pošiljanje kratkih sporočil

SQL – Structured Query Language; sestavljeni jezik za poizvedbe

SSL – Secure Sockets Layer; varnost plasti vtičnic

TCP – Transmission Control Protocol; internetni protokol

TLS – Transport Layer Security; varnost transportne plasti

UMTS – Universal Mobile Telecommunications System; univerzalni mobilni telekomunikacijski sistem

URI – Uniform Resource Identifier; enolični identifikator vira

URL – Uniform Resource Locator; enolični krajevnik vira

WiMAX – Worldwide interoperability for Microwave Access; svetovna interoperatibilnost mikrovalovnega dostopa

XML – eXtensible Markup Language; razširljiv označevalni jezik

Povzetek

V diplomskem delu je opisan razvoj mobilne aplikacije za platformo Android, ki omogoča trgovanje v sistemu Študentska borza. V delu sta najprej opisana mobilni operacijski sistem Android ter razvojno okolje (SDK) v navezi z orodjem Eclipse. Za strežniški del aplikacije sta bila uporabljena spletni strežnik Apache na katerem teče PHP ter podatkovna baza MySQL. Pri komunikaciji med odjemalcem in strežnikom se uporabljajo tehnologije REST, JSON in SSL. V osrednjem delu je najprej opisana arhitektura celotne aplikacije. Nadalje je opisan proces izdelave aplikacije vse od začetne aplikacije "Pozdravljen svet" pa do testiranja in težav, ki so se porajale tekom izdelave. Razložena sta tudi uporabniški vmesnik in način uporabe aplikacije. Na kratko je opisan tudi strežniški del aplikacije, ki sicer ni poglavitna tema tega diplomskega dela. V zaključku so predstavljene ideje za nadaljnje delo, ki se nanašajo predvsem na širitev funkcionalnosti aplikacije, da bi bila uporabniška izkušnja čim bolj blizu tisti v spletnem vmesniku.

Ključne besede:

Android, mobilna aplikacija, Študentska borza, operacijski sistem

Abstract

The thesis describes the development of a mobile application that runs on Android platform and enables stock trading on Student stock exchange (Študentska borza). Mobile operating system Android and software development kit paired with Eclipse are first described. Web server Apache running PHP and MySQL database were used for server-side application. Technologies like REST, JSON and SSL are used for communication between client and server. Central part of the thesis describes application architecture. Furthermore, the whole process of application development that ranges from the first "Hello World" to testing, debugging and problem solving is described. User interface and application usage are also explained. Since server-side application is not the main part of this thesis the development is only briefly described. Ideas about adding additional functionality to make user experience closely resamble the one on the web are explained in the conclusion.

Key words:

Android, mobile application, Student stock exchange, operating system

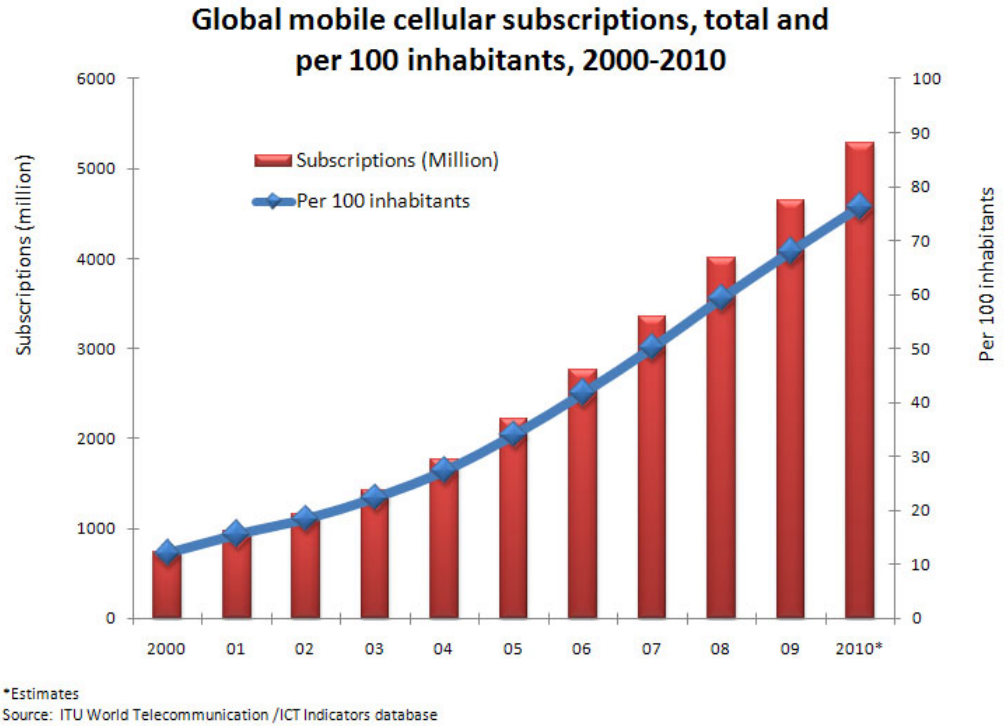
Poglavje 1

Uvod

Dandanes je uporaba mobilnih telefonov v velikem porastu. Navedimo zgolj en statistični podatek, ki zgovorno priča o tem. Po podatkih organizacije ITU (*angl. International Telecommunication Union*) se je število naročnikov po celem svetu v letu 2010 zelo približalo 5. miljardam (slika 1.1). Izmed 100 prebivalcev bi jih tako naj že kar 80 uporabljalo storitve mobilne telefonije.

Že dolgo pa mobilni telefoni niso namenjeni zgolj telefoniranju in pošiljanju kratkih sporočil, temveč se po funkcionalnosti naglo približujejo osebnim računalnikom. Omogočajo nam pregledovanje in pošiljanje elektronske pošte, brskanje po svetovnem spletu, uporabo socialnih omrežij kot sta Facebook in Twitter, pregled vremena, zajem visokoločljivostnih slik in videa, predvajanje multimedijskih vsebin itd. Tako je predvsem zaradi napredka strojne in programske opreme, ki poganjata mobilne telefone. Kot primer navedimo, da so trenutno najnaprednejši telefoni opremljeni z dvojedrnimi procesorji, ki tečejo pri frekvenci 1GHz, vsebujejo 1GB ali več pomnilnika, na multimedijskem področju pa so na voljo že kamere, ki zmorejo zajem visokoločljivostnih video posnetkov (1080p). Čeprav v zatonu, ima trenutno še vedno največji tržni delež operacijski sistem Symbian (36,6%), sledijo pa mu Android (25.5%), iOS (16,7%), BlackBerry OS (14,8%), Windows Mobile (2,8%), preostali delež pa predstavljajo ostali operacijski sistemi [1] (podatki so za 3. četrtletje 2010).

To diplomsko delo se osredotoča samo na operacijski sistem Android in razvoj aplikacij zanj. Najprej je na kratko predstavljen izobraževalni projekt Študentska borza, kateremu je aplikacija, izdelana v tem diplomskem delu, tudi namenjena. Poglavje 3, ki je bolj teoretično, vsebuje splošne lastnosti sistema Android, opis naveze paketa za razvoj aplikacij za Android ter razvojnega okolja Eclipse, na koncu pa so opisane tudi tehnologije, ki so uporabljene tako na strežniški kot odjemalčevi strani. V poglavju 4, ki predstavlja praktični



Slika 1.1: Število naročnikov mobilne telefonije na globalni ravni. Rdeči stolpci predstavljajo število naročnikov v miljonih in se razbirajo na levi skali, medtem ko modre točke predstavljajo število naročnikov na 100 prebivalcev in se razbirajo na desni skali.

del diplomskega dela, se najprej dotaknemo arhitekture celotne aplikacije. V nadaljevanju opišemo potek razvoja odjemalca vse od začetne aplikacije "Pozdravljen svet" pa do testiranja in težav, ki so se porajale med izdelavo. Razložena sta tudi uporabniški vmesnik in način upravljanja aplikacije. Na kratko je opisan tudi strežniški del aplikacije, ki sicer ni poglavitna tema tega diplomskega dela. V poglavju 5 so predstavljene ideje za nadaljnje delo, ki se nanašajo predvsem na širitev funkcionalnosti aplikacije, da bi bila uporabniška izkušnja čim bolj blizu tisti v spletnem vmesniku.

Poglavje 2

Študentska borza

2.1 Kaj je Študentska borza?

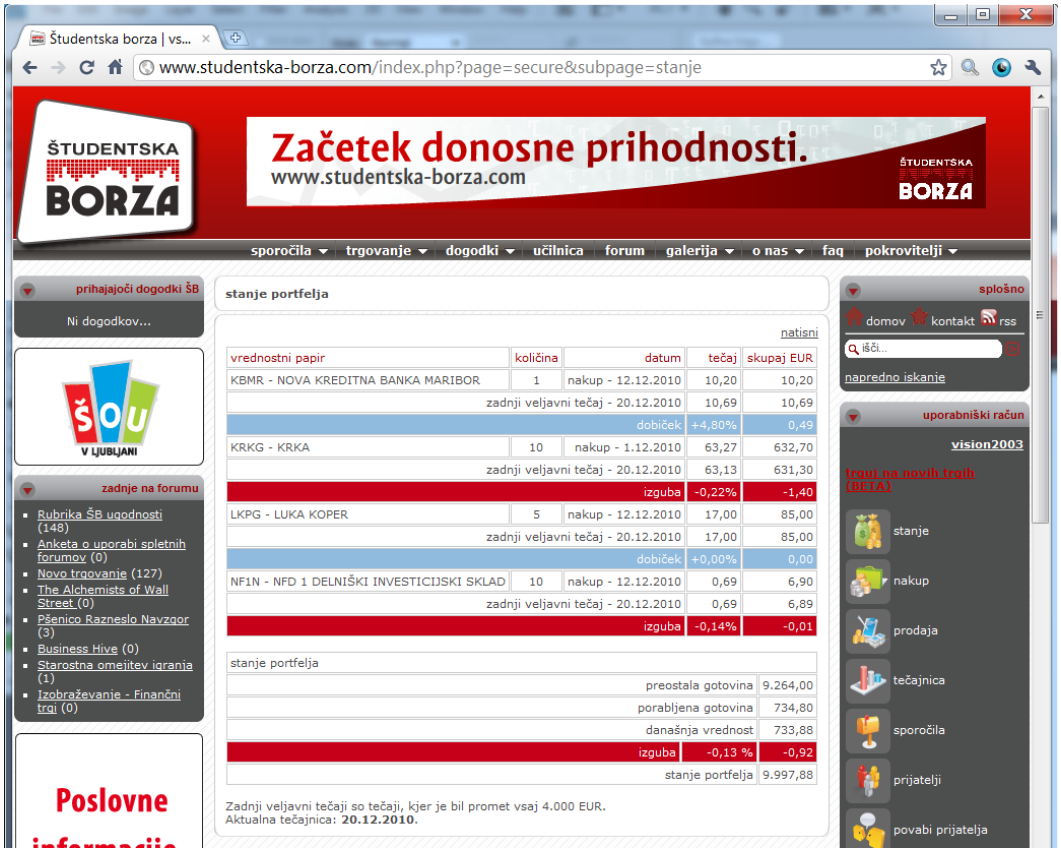
Študentska borza je projekt, ki ga izvaja Študentska organizacija Univerze v Ljubljani (ŠOU v Ljubljani) in je bil ustanovljen leta 1993. Je brezplačen izobraževalni sistem, ki na prijazen in vsem dostopen način poučuje o vseh relevantnih finančno-borznih in gospodarskih zadevah ter s tem omogoča uspešnejši vstop v poslovni svet [2].

2.2 Trgovanje

Uporabniki vsak mesec dobijo navidezen znesek, ki ga porabijo pri trgovanju na Študentski borzi. Trgujejo po zelo podobnem sistemu in pravilih, ki veljajo na pravi borzi. Tako se lahko brez tveganj in popolnoma brezplačno naučijo osnov borznega trgovanja. Vsak mesec zmagovalec prejme denarno nagrado v višini 300 EUR neto. Nagrajeni so tudi uporabniki od 2. do 5. mesta, saj prejmejo naročnino na revijo Kapital. Zmagovalci četrtletij so nagrajeni s kotizacijo na Finančno-borzni konferenci LJSE (*angl. Ljubljana Stock Exchange*) ter celoletno naročnino na revijo Navtika. Uporabniki, ki v četrtletju zasedejo 2., 3. in 4. mesto, prejmejo 6-mesečno brezplačno uporabo analitičnega trgovalnega orodja TeleTrader Professional SEE. Najboljši v celem letu prav tako prejme 6-mesečno brezplačno uporabo analitičnega trgovalnega orodja, celoletno naročnino na revijo Navtika in diplomo iz rok uglednega strokovnjaka na enem od seminarjev Študentske borze.

2.3 Spletna stran

Spletna stran www.studentska-borza.com (slika 2.1) je središče znanj, ki jih ponuja Študentska borza. Na njej uporabniki najdejo vse o finančno-borznem in gospodarskem področju. Naredijo lahko prve korake pri trgovanju in si prislužijo nagrado ali sodelujejo pri zanimivih debatah na forumu. Spletna stran je tudi zabavna, saj nudi povezovanje z drugimi uporabniki, sodelovanje pri izobraževalnih kvizih, ogled fotografij in še mnogo drugih stvari.



Slika 2.1: Spletna stran Študentska borza.

2.4 Spletna skupnost

Uporabniki skupaj sestavljajo spletno skupnost. Lahko si ustvarijo svoj profil, sodelujejo na forumu, povabijo prijatelje, izmenjujejo mnenja, povedo o

čem razmišljajo, trgujejo in v popolnosti izkoristijo vse ugodnosti, ki jih nudi Študentska borza.

2.5 Seminarji in okrogle mize

Jasno je, da so na začetku vsake poti zelo pomembne tudi izkušnje "prekaljenih starih mačkov". S tem namenom Študentska borza za vse zainteresirane vsako leto organizira seminarje (okrogle mize) in delavnice, kjer gostijo najbolj ugledne in izkušene goste iz sveta slovenskega borzništva in gospodarstva.

Poglavje 3

Uporabljena orodja in tehnologije pri razvoju aplikacije

3.1 Platforma Android

3.1.1 Splošno o Androidu

Android je odprtokodni mobilni operacijski sistem prvotno razvit s strani podjetja Android Inc., ki temelji na operacijskem sistemu Linux. Leta 2005 ga je kupil Google in septembra 2008 je izšla prva uradna različica [3].

Android ima ogromno skupnost razvijalcev, ki razvijajo aplikacije in s tem razširjajo nabor funkcionalnosti naprav na katerih teče. Trenutno je na voljo več kot 100.000 aplikacij, ki jih lahko prenesemo iz Android Marketa[1] ali pa jih pridobimo kako drugače (na primer neposredno iz spletnih strani).

Kot zanimivost na tem mestu povejmo, da trenutna različica operacijskega sistema sestoji iz 12 milijonov vrstic kode. Od tega je 3 milijone vrstic kode napisane v XML-ju, 2,8 milijona vrstic v C-ju, 2,1 milijona vrstic v Javi in 1,75 milijona vrstic v C++, preostalo pa v drugih programskih jezikih.

[1] Android Market je Googlova spletna trgovina, od koder lahko uporabniki naložijo aplikacije neposredno na njihovo napravo.

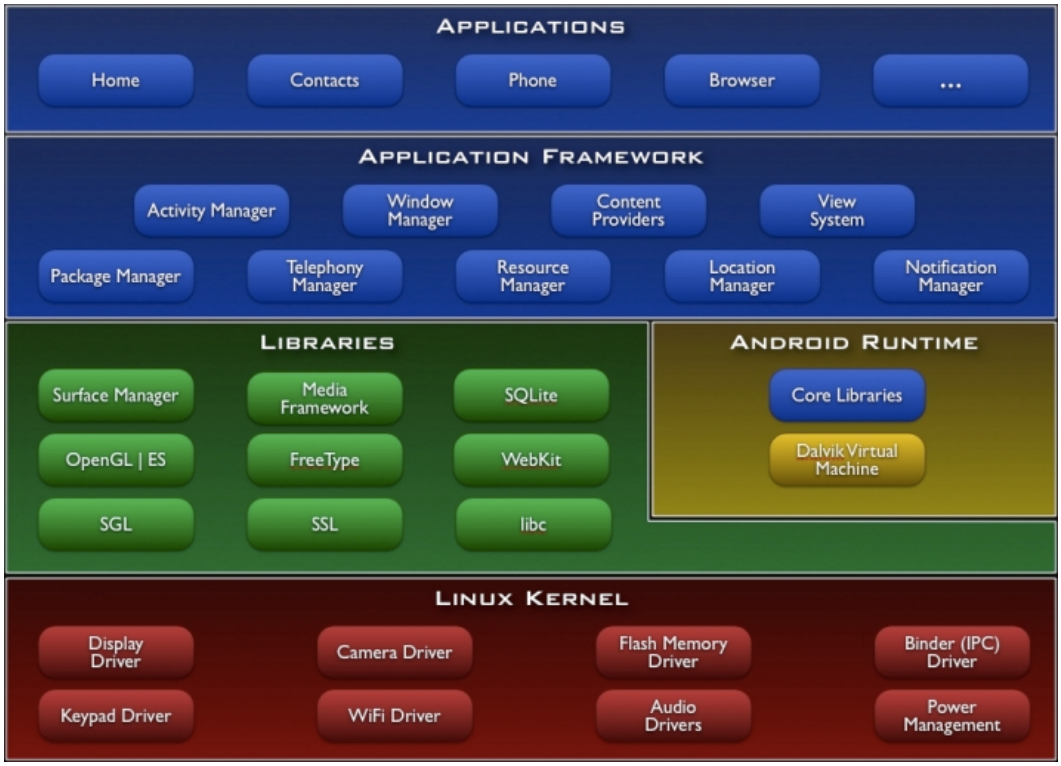
Sistem Android se ponaša s številnimi funkcijami, trenutno pa vsebuje naslednje:

- **Aplikativno ogrodje** – omogoča ponovno uporabo in zamenjavo komponent
- **Povezljivost** – podpira tehnologije: GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE in WiMAX
- **Sporočanje** – SMS in MMS
- **Podpora Javi** – kljub temu, da so Android aplikacije napisane v Javi, se te (prevedene v Java Byte Code) ne izvajajo na virtualnem stroju Java, temveč v virtualnem stroju Dalvik, ki je optimiziran za mobilne naprave z omejenim pomnilnikom in CPE
- **Vgrajen brskalnik** – temelji na odprtokodnem pogonu WebKit in JavaScript pogonu brskalnika Chrome V8
- **Optimizirana grafika** – lastna 2D grafična knjižnica; 3D grafika temelji na specifikaciji OpenGL ES 1.0, ki opcijsko omogoča tudi strojno pospeševanje
- **Shranjevanje podatkov** – podatki se shranjuje v relacijski podatkovni bazi SQLite
- **Podpora multimediji** – podpira formate: WebM, H.263, H.264, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, BMP ter pretočne formate kot so: RTP/RTSP, HTML (HTML5 <video>)
- **Bluetooth**
- **Strojna podpora napravam** – podpira različne naprave kot so: kamera, GPS, kompas, merilnik pospeška, zaslon občutljiv na dotik, žiroskop, merilnik magnetnega polja, senzor za zaznavanje bližine in tlaka, termometer itd.
- **Razvijalsko okolje** – vsebuje emulator, razhroščevalnik, orodja za pomnilniško in zmogljivostno profiliranje, dokumentacijo ter vrsto primerov
- **Podpora večdotičnosti**

- **Večopravilnost** – je večopravilni sistem, kar pomeni, da se lahko več aplikacij izvaja hkrati
- **Glasovne funkcije** – omogoča glasovno iskanje in narek (trenutno samo v angleščini)
- **Privezovanje** (*angl. Tethering*) – mobilno napravo se lahko uporabi kot brezžično / žično dostopno točko, ki omogoča, da delimo internetno povezavo mobilne naprave z ostalimi napravami.

3.1.2 Arhitektura

Slika 3.1 opisuje glavne komponente operacijskega sistema Android, ki so predstavljene kot sklad plasti [4]. V nadaljevanju so posamezne od teh še posebej opisane.



Slika 3.1: Arhitektura operacijskega sistema Android.

Aplikacije

Android že vsebuje poglavitne aplikacije vključno z odjemalcem za elektronsko pošto, SMS aplikacijo, koledarjem, Google Maps, brskalnikom, imenikom itd. Vse aplikacije so napisane v Javi.

Aplikativno ogrodje

Ker je Android odprto aplikativno ogrodje, omogoča razvijalcem razvijanje izjemno bogatih in inovativnih aplikacij. Razvijalci lahko tako izkoristijo strojno opremo naprave, dostopajo do lokacijskih podatkov, poganjajo servise v ozadju, nastavljajo budilke itd.

Razvijalci imajo dostop do enakega vmesnika API (*angl. Application Programming Interface*) kot aplikacije, ki so že vgrajene v Android.

Komponente aplikacije

Arhitektura sistema Android omogoča ponovno uporabo komponent. To pomeni, da lahko aplikacije nudijo svojo funkcionalnost ostalim aplikacijam, prav tako pa lahko uporabljajo funkcionalnosti ostalih aplikacij (seveda, če je to s strani teh aplikacij omogočeno).

Da je to mogoče, mora biti sistem sposoben zagnati proces aplikacije, kadar se to zahteva in ustvariti vse javanske objekte. Zato, z razliko od ostalih sistemov, aplikacije v Androidu nimajo enotne vstopne točke (na primer metoda `main()`), temveč sestojijo iz komponent, ki se lahko po potrebi zaganjajo.

Te komponente so: aktivnosti, storitve, sprejemniki in ponudniki vsebine.

Aktivnosti (*angl. activities*)

Aktivnost je edini del, ki ga uporabnik dejansko vidi in predstavlja uporabniški vmesnik. Lahko prikazuje seznam, besedilo, galerijo, ali karkoli drugega, kar želimo, da je uporabniku vidno. Tako recimo aplikacija za sporočila sestoji iz ene aktivnosti, ki prikazuje seznam stikov, druge aktivnosti, ki omogoča vnos besedila sporočila in ostalih aktivnosti, ki omogočajo pregled sporočil in spreminjanje nastavitev.

Tako lahko vsaka aplikacija sestoji iz ene aktivnosti ali pa mnogih, kot pravkar omenjena aplikacija za sporočila. Kaj točno v aplikaciji aktivnosti predstavljajo in koliko jih je, je odvisno od same zasnove aplikacije. Navadno je ena aktivnost vstopna točka (torej tista, ki se uporabniku najprej prikaže –

na primer obrazec za vnos uporabniškega imena in gesla), ta pa nato sproži naslednjo aktivnost.

Vsaka aktivnost ima svoje okno za izris vsebine. Navadno to okno zaseda celoten zaslon, lahko pa je tudi manjše in se pojavi na vrhu ostalih oken. Aktivnost lahko proži tudi dodatna okna – na primer pojavno okno, kamor uporabnik vnese uporabniško ime in geslo.

Za vizualno predstavitev vsebine skrbijo pogledi, ki so izpeljani iz razreda `View`. Vsak pogled nadzira določen pravokoten del zaslona. Pogledi so hierarhično organizirani, zato vsak starš določa razporeditev svojih otrok. Pogledi čisto na dnu hierarhije (listi) skrbijo za dejanski izris vsebine v delih zaslona, katerim so namenjeni. Android ima že vgrajenih mnogo pogledov kot so: gumbi, vnosna polja, drsni elementi, seznami itd.

Storitve (*angl. services*)

Storitev, tako kot je v navadi tudi pri ostalih sistemih, ne vsebuje dejanskega uporabniškega vmesnika, temveč teče v ozadju. Lep primer storitve je nalaganje datotek preko brskalnika. Uporabnik zažene brskalnik, izbere datoteko, ki jo želi naložiti in zažene sam proces nalaganja. Ker je ta datoteka lahko velika, nima smisla, da uporabnik ves čas spremlja prenos. Zato je brskalnik zasnovan tako, da zažene storitev prenosa datoteke, ki teče v ozadju. Uporabnik lahko tako brskalnik v vmesnem času zapre in z napravo počne kaj drugega.

V Androidu se je možno povezovati tudi na storitve, ki so že v teku. Ko smo enkrat s tako storitvijo povezani, lahko z njo komuniciramo preko vmesnika, ki ga ta storitev definira. Za zgornji primer bi ta storitev lahko, na primer, prikazovala potek prenosa.

Omenimo še, da tako aktivnosti kot storitve tečejo v glavni niti programa. Da se izognemo blokiranju ostalih komponent uporabniškega vmesnika (in slavnemu pogovornemu oknu ANR (*angl. Application Not Responding*)), je pametno zahtevnejše in časovno potratnejše storitve poganjati v ločenih nitih [5] (pogl. 8).

Sprejemniki (*angl. broadcast receivers*)

Sprejemnik je komponenta, ki sprejema razpršena sporočila (*angl. broadcast announcements*), ki navadno izvirajo iz sistema samega. Primer takih sporočil je, na primer, menjava časovnega pasu, sporočilo, da je baterija skoraj prazna, sporočilo, da je zajemanje slike končano itd. Aplikacije lahko tudi same pošiljajo sporočila in tako oznanijo, da je neko opravilo končano (na primer

prenos datoteke je končan).

Vsaka aplikacija lahko ima enega ali več sprejemnikov, ki sprejemajo samo tista sporočila, ki so za aplikacijo pomembna. Vsak sprejemnik v osnovi razširja razred `BroadcastReceiver`.

Načeloma sprejemniki nimajo uporabniškega vmesnika, lahko pa ob sprejetju sporočila poženejo aktivnost ali pa prikažejo obvestilo preko razreda `NotificationManager`.

Ponudniki vsebine (*angl. content providers*)

Kot že ime nakazuje, ponudnik vsebine nudi določene dele podatkov aplikacije ostalim aplikacijam. Podatki so lahko shranjeni na datotečnem sistemu, v podatkovni bazi SQLite ali kakršnikoli drugi smiselni obliki. Vsak ponudnik vsebine razširja razred `ContentProvider` in implementira standarden nabor metod, ki omogočajo ostalim aplikacijam dostop (branje in shranjevanje) do teh podatkov. Aplikacije sicer teh metod ne kličejo neposredno, temveč to storijo preko razreda `ContentResolver`.

Zaganjanje komponent – prožilci

Ponudnik vsebine se zažene ob klicu objekta `ContentResolver`. Ostale tri komponente (aktivnosti, storitve in sprejemniki) pa se zaganjajo preko asinhronih sporočil, ki jim pravimo prožilci (*angl. intents*). Prožilec je v splošnem objekt tipa `Intent` in vsebuje vsebino sporočila. Za aktivnosti in storitve sta to med drugim ime akcije, ki se jo zahteva ter naslov URI (*angl. Uniform Resource Identifier*). Tako lahko, na primer, prožilec poda neki aktivnosti zahtevo, da ta prikaže sliko ali pa dovoli uporabniku urediti tekst. Za sprejemnike prožilec vsebuje ime akcije, ki se je izvedla. To je lahko, na primer, sporočilo, da je bil pritisnjen gumb za zajem slike s kamero.

Za zaganjanje različnih komponent, obstajajo različne metode. Te so:

- Aktivnost se zažene tako, da metodi `Context.startActivity()` ali pa metodi `Activity.startActivityForResult()` podamo prožilec. Zagnana aktivnost lahko kasneje do tega prožilca (in morebitnih podatkov, ki jih ta vsebuje) dostopa preko metode `getIntent()`.

  Aktivnost pogosto zažene novo aktivnost. Če od te aktivnosti pričakuje določen rezultat, namesto metode `startActivity()` pokliče sorodno metodo `startActivityForResult()`. Na primer, če aktivnost zažene aktivnost v kateri lahko uporabnik izbere fotografijo, lahko pričakuje kot

rezultat izbrano fotografijo. Rezultat se vrne kot objekt tipa `Intent` s klicem metode `onActivityResult()` prvotne aktivnosti.

- Storitev se zažene tako, da metodi `Context.startService()` podamo prožilec. Android potem pokliče metodo `onStart()` storitve in ji poda ta prožilec.

  Podobno lahko prožilec podamo preko metode `Context.bindService()` in s tem vzpostavimo trajno povezavo s storitvijo. Storitev dobi v tem primeru prožilec preko metode `onBind()`. Za primer navedimo aktivnost, ki preko klica metode `bindService()` vzpostavi povezavo s storitvijo za prenos datotek (iz primera za prenos datotek opisanega pri storitvah) in tako uporabniku sporoči napredek prenosa.

- Pošiljanje razpršenih sporočil izvedemo tako, da podamo prožilec metodam `Context.sendBroadcast()`, `Context.sendOrderedBroadcast()`, `Context.sendStickyBroadcast()` oz. katerikoli variaciji le-teh. Android nato ta sporočila dostavi vsem poslušojočim sprejemnikom preko klica metode `onReceive()`.

Zaustavljanje komponent

Ker je ponudnik vsebine aktiven, samo dokler se odziva zahtevkom objekta `ContentResolver` in poslušalec, samo dokler se odziva razpršenim sporočilom, ni potrebe, da bi ti dve komponenti zaustavljali.

Drugače pa je z aktivnostmi, saj te definirajo uporabniški vmesnik. To pomeni, da so aktivne, dokler poteka interakcija med aktivnostjo in uporabnikom. Podobno je s storitvami, ki lahko tečejo dalj časa. Zato Android nudi metode, s katerimi lahko tako aktivnosti kot storitve enostavno zaustavimo:

- Aktivnost zaustavimo s klicem metode `finish()`. Aktivnost lahko zaustavi tudi drugo aktivnost (to je tisto, ki jo je zagnala s klicem metode `startActivityForResult()`) s klicem metode `finishActivity()`.

- Storitev zaustavimo s klicem metode `stopSelf()` ali pa s klicem metode `Context.stopService()`.

Na tem mestu omenimo še zelo pomembno lastnost sistema Android. Ta ima namreč privilegij, da lahko komponente zaustavi tudi sam. To se zgodi, kadar te dolgo časa niso aktivne ali pa sistem potrebuje pomnilnik zaradi takšnega ali drugačnega razloga. O tem je več zapisanega kasneje (glej stran 17).

Manifest

Da lahko Android zažene neko komponento, mora vedeti, da ta obstaja. Zato aplikacije deklarirajo vse svoje komponente v manifestu, ki je del paketa Android (datoteka s končnico .apk).

Manifest ni pravzaprav nič drugega kot strukturirana XML datoteka, ki se imenuje `AndroidManifest.xml` [5] (pogl. 3). Poleg deklaracije komponent aplikacije se v tej datoteki definirajo knjižnice, ki jih aplikacija uporablja (poleg privzete Android knjižnice), določijo pa se tudi pravice aplikacije (na primer pravica za uporabo podatkovne povezave ali pa kompasa).

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
    <application . . . >
        <activity android:name="com.example.project.Activity"
                  android:icon="@drawable/small_pic.png"
                  android:label="@string/freneticLabel"
                  . . .  >
        </activity>
        . . .
    </application>
</manifest>
```

Slika 3.2: Primer manifesta.

Na sliki 3.2 je predstavljen osnovni primer manifesta. Element `<activity>` ima 3 atribute. Atribut `name` predstavlja ime razreda aktivnosti (oz. podrazreda razreda `Activity`). Atributa `icon` in `label` kažeta na datoteki z ikono in oznako aktivnosti.

Ostale komponente se deklarirajo podobno – element `<service>` predstavlja storitev, `<receiver>` sprejemnika in `<provider>` ponudnika vsebine. Aktivnosti, storitve in ponudnike vsebine, ki niso deklarirani v manifestu, sistem ne vidi in se jih zato ne da zagnati. To pa ne velja za sprejemnike, saj so ti lahko deklarirani v manifestu, lahko pa jih tudi dinamično ustvarimo v sami kodi in jih registriramo v sistemu s klicem metode `Context.registerReceiver()`.

Filtri prožilcev (*angl. intent filters*)

Prožilec lahko izrecno poimenuje ciljno komponento. V tem primeru Android najde ustrezno komponento (glede na deklaracijo v manifestu) in jo

zažene. Če pa ciljna komponenta ni izrecno poimenovana, mora Android sam najti najbolj ustrezno. To stori tako, da primerja prožilec s filtri prožilcev potencialnih komponent. Filtri prožilcev neke komponente Androidu povedo kakšne prožilce lahko komponenta sprejme. Kot vse ostale pomembne stvari, so tudi filtri prožilcev deklarirani v manifestu.

Razširimo prejšnji primer z dvema filtroma prožilcev:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
  <application . . . >
      <activity android:name="com.example.project.Activity"
              android:icon="@drawable/small_pic.png"
              android:label="@string/freneticLabel"
              . . .  >

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

        <intent-filter . . . >
            <action
              android:name="com.example.project.BOUNCE" />
            <data android:mimeType="image/jpeg" />
            <category
              android:name="android.intent.category.DEFAULT" />
        </intent-filter>

      </activity>
      . . .
  </application>
</manifest>
```

Slika 3.3: Manifest z dvema filtroma prožilcev.

Prvi filter na sliki 3.3 je kombinacija akcije `android.intent.action.MAIN` ter kategorije `android.intent.category.LAUNCHER` in se uporablja pogosto. S tem filtrom je določeno, da bo zaganjalnik aplikacije najprej pognal to ak-

tivnost. Z drugimi besedami to pomeni, da je ta aktivnost pravzaprav vstopna točka aplikacije. Drugi filter deklarira akcijo, ki jo aktivnost lahko izvede na določenem podatkovnem tipu (v tem primeru je to slika tipa `JPEG`).

Vsaka komponenta lahko ima poljubno število filtrov, vsak od teh pa deklarira različne zmožnosti. V primeru, da aktivnost nima nobenega filtra, jo lahko aktivirajo samo prožilci, tako da jo navedejo izrecno (poimensko).

Življenjski cikel aktivnosti

Vsaka komponenta aplikacije ima svoj življenjski cikel – od začetka, ko Android komponente zažene in do konca, ko jih zaustavi. Med tema dvema skrajnima točkama so lahko komponente aktivne ali neaktivne (v primeru aktivnosti to pomeni, da so te uporabniku vidne oziroma nevidne). Sedaj pa si podrobneje poglejmo za to diplomsko delo zanimiv življenjski cikel - življenjski cikel aktivnosti.

Vsaka aktivnost ima tri stanja:

- Aktivnost je aktivna in je na zaslonu v ospredju (tehnično gledano to pomeni, da je na vrhu sklada aktivnosti). Taka aktivnost ima trenutni fokus za uporabnika.
- Aktivnost je bila prekinjena in je izgubila fokus, vendar je še vedno vidna na zaslonu. S tem je mišljeno, da nad njo leži okno neke druge aktivnosti. Ko je aktivnost prekinjena, je še vedno živa (to pomeni, da so vsa stanja in spremenljivke aktivnosti shranjene), vendar jo lahko sistem v primeru pomanjkanja pomnilnika zaustavi.
- Aktivnost je zaustavljena, ker je neka druga aktivnost postala aktivna. V tem stanju aktivnost uporabniku ni več vidna. Sistem v tem stanju še vedno hrani vsa stanja in spremenljivke aktivnosti, vendar jo bo sistem uničil, če bo potreboval pomnilnik za kaj drugega.

Če je aktivnost prekinjena ali zaustavljena, jo lahko sistem izloči iz pomnilnika s klicem metode `finish()` ali pa proces enostavno ubije. Ob ponovni zahtevi za prikaz te aktivnosti, jo mora sistem ponovno zagnati in obnoviti njeno stanje.

Pri prehajanju aktivnosti med posameznimi stanji se prožijo naslednje metode:

```
void onCreate(Bundle savedInstanceState)
void onStart()
```

```
void onRestart()
void onResume()
void onPause()
void onStop()
void onDestroy()
```

Vse te metode lahko prepišemo in s tem poskrbimo, da se ob prehodu med stanji izvedejo primerne rutine (npr. za shranjevanje stanja ob zaustavitvi). Vsaka aktivnost mora implementirati metodo `onCreate()` za začetno inicializacijo objekta, mnoge pa implementirajo tudi metodo `onPause()`.

Skupaj teh sedem metod tvori celoten življenjski cikel aktivnosti. Glede na vsa stanja so možni trije cikli (slika 3.4):

- Celoten življenjski cikel aktivnosti se začne s klicem metode `onCreate()` in konča z edinim klicem metode `onDestroy()`. Aktivnost v tem primeru opravi vso začetno inicializacijo v metodi `onCreate()` in sprosti vse vire v metodi `onDestroy()`.

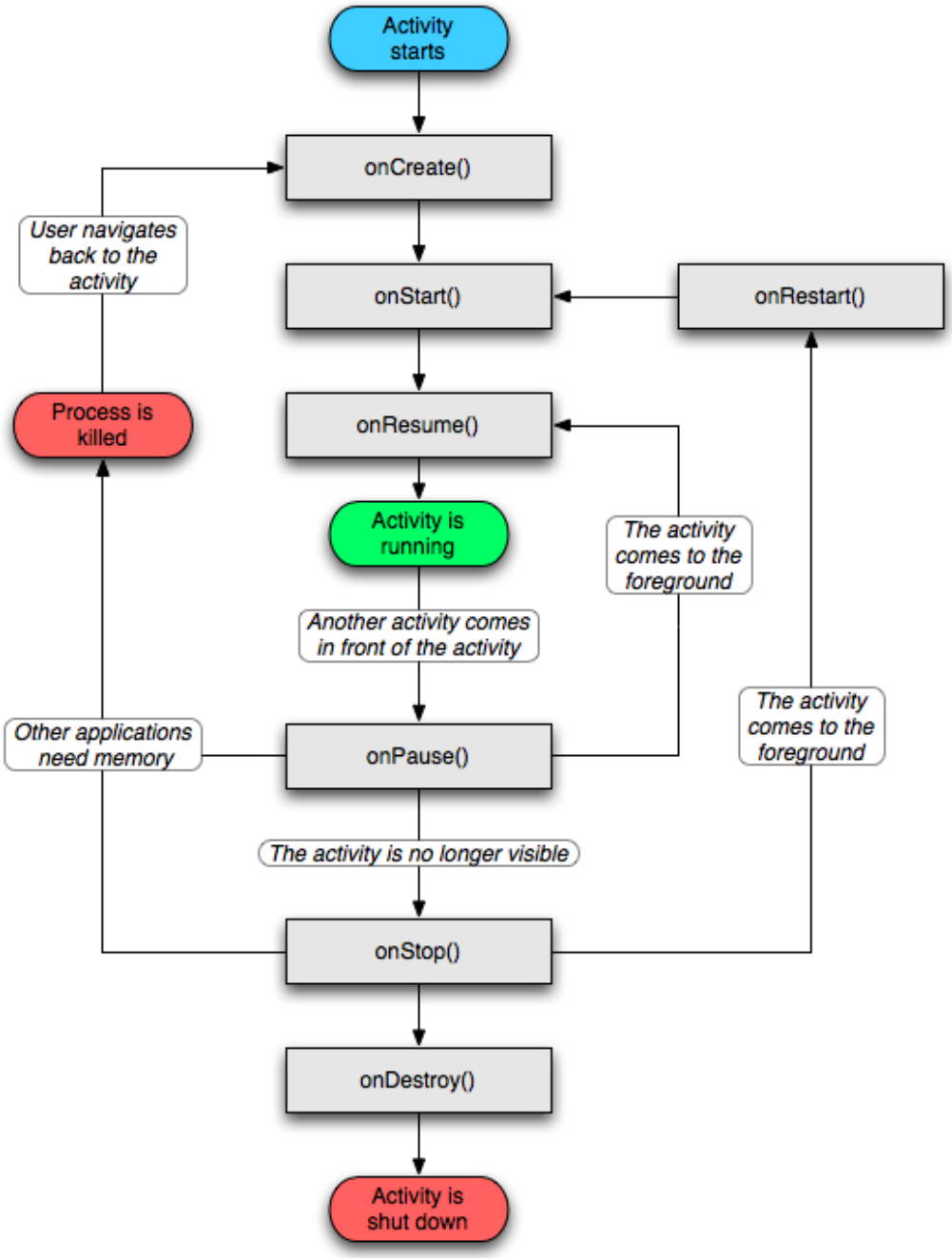
- Vidni življenjski cikel aktivnosti se zgodi med metodama `onCreate()` in `onStop()`. V tem času je aktivnosti uporabniku vidna (čeprav ni nujno, da je v ospredju). Med tema dvema metodama običajno upravljamo z viri, ki so nujni za prikaz aktivnosti.

- Cikel, ko je aktivnost v ospredju, se zgodi med metodama `onResume()` in `onPause()`. V tem času je aktivnost v ospredju (pred vsemi ostalimi aktivnostmi) in poteka interakcija z uporabnikom. Aktivnosti pogosto prehajajo med tema dvema stanjema (metodama), zato morajo biti rutine, ki jih izvajamo v teh dveh metodah, kratke in nepotratne.

Shranjevanje stanja

V primeru, da namesto uporabnika zaustavi aktivnost sam sistem, se pričakuje, da bo ob vrnitvi v aktivnost, ta v istem stanju kot ob zapustitvi.

Za takšne primere shranjevanja stanja aktivnosti se uporablja metoda `onSaveInstanceState()`. Android to metodo pokliče preden bi lahko bila aktivnost uničena – to je pred klicem metode `onPause()`. Metoda kot argument dobi objekt tipa `Bundle`, kamor lahko shranimo podatke v obliki `key-value` parov. Ko se aktivnost ponovno zažene, dobita metodi `onCreate()` in `onRestoreInstanceState()` isti objekt. To nam omogoča enostavno obnovitev stanja aktivnosti.



Slika 3.4: Življenjski cikel aktivnosti.

Velja opozoriti, da z razliko od metode `onPause()`, metodi `onSaveInstanceState()` in `onRestoreInstanceState()` nista del metod življenjskega cikla, ker se jih ne pokliče vedno. Kot primer navedimo, da se metoda `onSaveInstanceState()` ne pokliče, ko uporabnik, na primer, pritisne tipko `nazaj`. V tem primeru uporabnik ne pričakuje, da se bo vrnil v aktivnost in zato tudi ni potrebno shranjevanje stanja.

Knjižnice

Android vsebuje množico C/C++ knjižnic, ki jih uporabljajo različne komponente sistema. Nekatere izmed teh knjižnic so:

- **Sistemska knjižnica C** – implementacija standardne sistemske knjižnice C (libc) prilagojene za vgrajene naprave
- **Multimedijske knjižnice** – osnovane na knjižnici OpenCORE; podpirajo predvajanje in snemanje avdija/videa, delo s slikami, itd.
- **LibWebCore** – pogon brskalnika, ki poganja v Android vgrajeni brskalnik
- **SGL** – 2D grafični pogon
- **3D knjižnice** – implementacija temelji na OpenGL ES 1.0; knjižnice uporabljajo ali strojno 3D pospeševanje ali pa optimizirano programsko 3D izrisovanje
- **FreeType** – knjižnica za delo s fonti
- **SQLite** – zmogljiva relacijska podatkovna baza, ki je na voljo vsem aplikacijam.

Izvajalno okolje

Kot že prej omenjeno, se Android aplikacije ne izvajajo na virtualnem stroju Java, temveč za to skrbi virtualni stroj Dalvik. Prevedene aplikacije se zato transformirajo v format Dalvik Executable (.dex), ki jih razume virtualni stroj Dalvik. Vsaka aplikacija teče v ločenem procesu, Dalvik pa je narejen tako, da lahko mobilna naprava brez težav poganja več instanc le-tega. To zagotavlja, da aplikacije ne vplivajo druga na drugo v primeru nepravilnosti (in v najhujšem primeru prisilne zaustavitve). Funkcije kot so nitkanje in nizkonivojsko upravljanje s pomnilnikom virtualnemu stroju Dalvik seveda zagotavlja Linux jedro.

Linux jedro

Android je osnovan na Linux jedru 2.6, ki skrbi za osrednje sistemske funkcije kot so varnost in zaščita, upravljanje s pomnilnikom, upravljanje s procesi, razporejanje opravil, dostop do datotečnega sistema, omrežni sklad, navidezni datotečni sistem itd. [5] (pogl. 1)

3.2 Android SDK

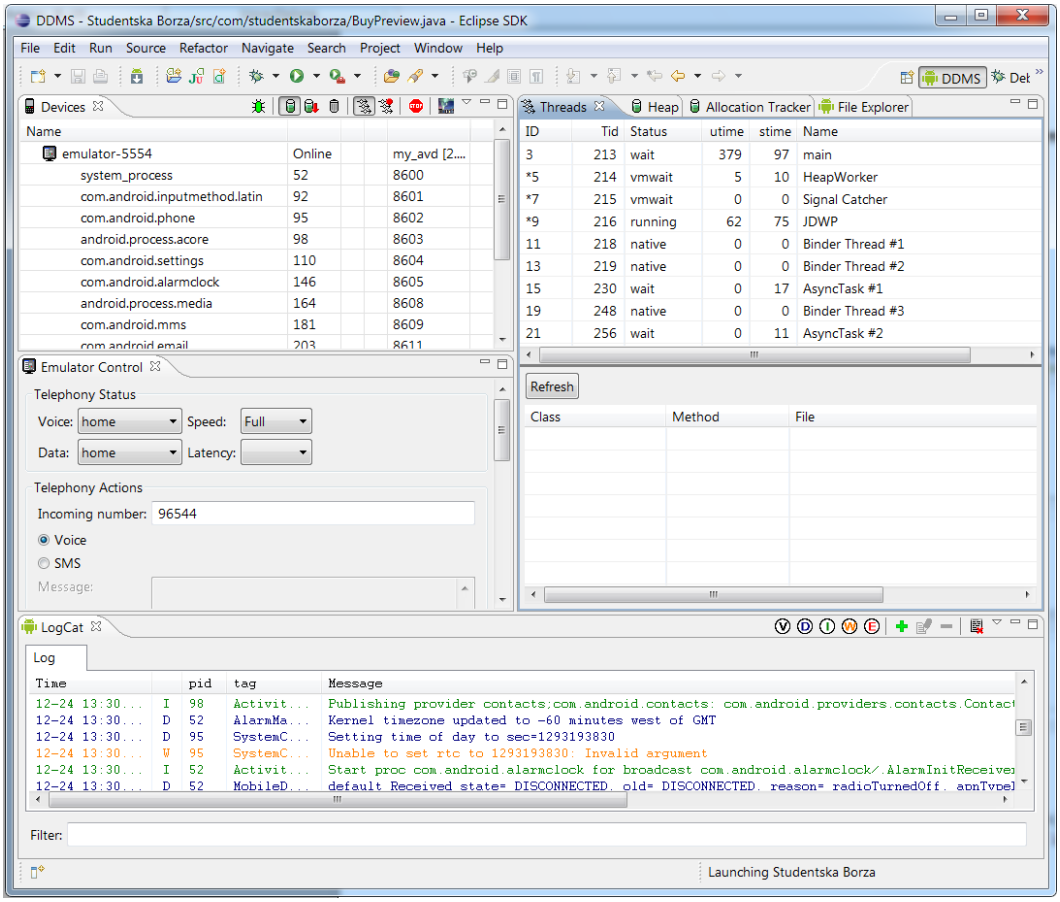
Paket za razvoj aplikacij (*angl. Software Development Kit*) Android vsebuje množico koristnih razvijalskih orodij [6]. Sem sodijo razhroščevalnik, knjižnice, emulator (ki temelji na emulatorju QEMU), dokumentacija, primeri kode in vodiči. Trenutno podprti operacijski sistemi so Linux, Mac OS X (10.4.9 ali kasnejši) ter Windows XP ali kasnejši. Aplikacije je možno napisati v vsakem urejevalniku besedila in jih potem prevesti in poganjati iz ukazne lupine. Sicer pa je uradno podprto in s strani Googla priporočeno razvojno okolje (*angl. Integrated Development Environment*) Eclipse v navezi z vtičnikom ADT (*angl. Android Development Tools*).

3.2.1 Razhroščevalnik

Omenimo, da Android SDK vsebuje razhroščevalnik DDMS (*angl. Dalvik Debug Monitor Server*). Ta deluje kot vmesnik med razvojnim orodjem (na primer Eclipse) in samo aplikacijo. Komunikacija med njima poteka tako, da se DDMS ob zagonu aplikacije poveže na navidezni stroj.

Glavne funkcionalnosti DDMS (slika 3.5) so:

- Nadzor naprav – tu se nahaja seznam vseh naprav in pripadajočih navideznih strojev posamezne naprave. Za vsak navidezni stroj lahko (na desni strani) spremljamo njegove niti, porabo kopice itd.
- Simulacija klicev in sporočil – na emulatorju lahko simuliramo dohodni klic in sporočilo.
- Simulacija položaja – navidezno lahko nastavimo trenutni položaj naprave
- Podatkovne in glasovne nastavitve – nastavljamo lahko stanje povezave (na primer doma, gostovanje, iskanje omrežja) in hitrost povezave (na primer GSM, UMTS, EDGE)



Slika 3.5: Razhroščevalnik DDMS v uporabljenem IDE okolju Eclipse.

- Dnevnik Logcat – sistemski dnevnik, kamor lahko pišemo tudi iz aplikacije preko knjižnice Log (to je bil v tem diplomskem delu najbolj uporabljan način razhroščevanja).

3.2.2 Emulator

Za lažji razvoj je v Android SDK vključen tudi emulator. To je nekakšna navidezna naprava, ki teče na računalniku in posnema pravo mobilno napravo.

Vsakemu emulatorju lahko v upravljalniku AVD (*angl. Android Virtual Device*) določimo njegove strojne in programske lastnosti. Programska lastnost je samo ena in sicer različica operacijskega sistema Android (podprte različice so: 1.5, 1.6, 2.0, 2.1, 2.2 in trenutno najnovejša 2.3). Nekatere izmed zanimivejših strojnih lastnosti, ki jih lahko nastavljamo so:

Slika 3.6: Android emulator.

- Velikost SD kartice
- Ločljivost zaslona
- Predvajanje in zajem zvoka
- Kamera (ali jo emulator ima ali ne)
- Delovanje na baterijo
- GPS
- Fizična tipkovnica
- GSM modem.

Na tem mestu velja omeniti, da je emulator še vedno navidezna naprava, ki ima svoje omejitve. Tako na njem, na primer, ne moremo uporabljati povezave Bluetooth, zaznavati, če so na napravo recimo priključene slušalke itd. [5] (pogl. 2).

3.3 Orodja in tehnologije na strežniški strani

Spodaj opisane tehnologije so bile uporabljene v tem diplomskem delu na strežniški strani aplikacije.

3.3.1 Spletni strežnik Apache

Apache je eden izmed bolj znanih spletnih strežnikov, ki je igral izjemno pomembno vlogo pri razvoju svetovnega spleta [7].

Lahko se pohvali s številnimi funkcionalnostmi, ki so pogosto implementirane kot moduli, ki razširjajo osnovno funkcionalnost strežnika. Te funkcionalnosti segajo od podpore jezikom, ki tečejo na spletnem strežniku (na primer Perl, Python, Tcl, PHP) pa vse do avtentikacijskih shem (`mod_access`, `mod_auth`, `mod_digest`). Omogoča tudi kriptirani povezavi SSL in TLS, filtriranje, prilagojene dnevniške zapise itd.

Za potrebe tega diplomskega dela je zanimiva predvsem naveza s skriptnim jezikom PHP in pa podpora kriptirani povezavi SSL.

3.3.2 PHP

PHP (*angl. PHP Hypertext Preprocessor*) je večnamenski odprtokodni skriptni jezik, ki je bil prvotno ustvarjen za spletni razvoj, konkretneje dinamično generiranje spletnih strani [8].

To je doseženo tako, da je koda PHP vstavljena v dokument HTML, to pa potem obdela spletni strežnik s PHP modulom (v našem primeru Apache z modulom `mod_php`) in tako ustvari spletno stran. Kot splošnonamenski programski jezik lahko PHP deluje tudi kar kot program v ukazni vrstici. Interpreter PHP izvede samo tisti del kode v dokumentu, ki je znotraj mej `<?php ?>`, ostali del kode pa pusti nedotaknjen. Zato končni uporabnik spletne strani nikoli ne vidi kode PHP, temveč samo interpretirano datoteko HTML.

Primer na sliki 3.7 prikazuje strežniško datoteko PHP. Apache (oz. PHP modul) v njej obdela samo tisti del kode, ki je znotraj mej `<?php ?>`. V tem primeru je vse kar naredi izpis niza "Ta tekst je izpisal interpreter PHP.".

3.3.3 SSL

SSL (*angl. Secure Sockets Layer*) je kriptografski protokol, ki zagotavlja varno komunikacijo v omrežju. SSL deluje na programski plasti, ki je med plastema

```
<html>
   <head>
       <title>Primer PHP</title>
   </head>
   <body>
      <?php
         echo "Ta tekst je izpisal interpreter PHP.";
      ?>
   </body>
</html>
```

Slika 3.7: Primer kode v jeziku PHP.

protokola HTTP in TCP. Uporablja enkripcijski sistem s certifikatom (to je z zasebnim in javnim ključem), ki je poznan iz sistema RSA.

Precej poenostavljeno poteka kriptirana komunikacija med odjemalcem in strežnikom preko SSL tako:

1. Odjemalec preveri certifikat in se tako prepriča, da je strežnik s katerim bo komuniciral verodostojen.
2. Odjemalec in strežnik se dogovorita o tipu enkripcije. Izbereta takšen tip, ki ga oba razumeta.
3. Odjemalec in strežnik si izmenjata unikatna gesla, s katerimi bosta kriptirala sporočila pred pošiljanjem.
4. Odjemalec in strežnik si začneta pošiljati kriptirana sporočila.

V tem diplomskem delu je protokol SSL uporabljen v navezi s protokolom HTTP. Oba protokola skupaj tvorita protokol HTTPS (*angl. Hypertext Transfer Protocol Secure*), ki je uporabljen za pošiljanje občutljivih podatkov.

3.3.4 MySQL

MySQL je brezplačen sistem za upravljanje s podatkovno bazo. Natančneje je to odprtokodna implementacija relacijske podatkovne baze, ki uporablja jezik SQL (*angl. Structured Query Language*).

SQL je najbolj priljubljen jezik za dodajanje, dostop in urejanje vsebine podatkovne baze. Njegove odlike so: hitrost, zanesljivost, enostavnost uporabe in fleksibilnost.

Na Študentski borzi uporabljamo, ravno zaradi zgoraj naštetih odlik, MySQL že od samega začetka, zato je tudi del tega diplomskega dela.

3.3.5 REST

REST (*angl. Representational State Transfer*) arhitektura sestoji iz odjemalcev in strežnikov. Odjemalci pošiljajo zahteve strežnikom; strežniki obdelajo zahteve in vrnejo primerne odgovore. Naštejmo nekaj zahtev REST arhitekture, ki so pomembne za razumevanje tega diplomskega dela [9]:

- **Odjemalec-strežnik** – odjemalec in strežnik sta ločena preko enotnega vmesnika. To pomeni, da, na primer, shranjevanje podatkov ni domena odjemalca ampak je prepuščeno zgolj strežniku. Prav tako se, na primer, strežnika ne tiče uporabniški vmesnik in stanje uporabnika, ki je prepuščeno odjemalcu.
- **Brez stanj** (*angl. Stateless*) – strežnik ne hrani nikakršnih stanj oz. konteksta odjemalca med posameznimi zahtevami. Vsaka zahteva mora vsebovati vse podatke (na primer uporabniško ime in geslo), ki so potrebni za strežbo le-te.
- **Enotni vmesnik** – vmesnik med odjemalcem in strežnikom poenostavi in razdvoji arhitekturo, kar omogoča, da se vsak od njiju razvija neodvisno od drugega. Kar je tukaj pomembno je, da so posamezni viri definirani kot zahtevki (na primer prek različnih URI-jev). Tako na primer, strežnik na nek zahtevek ne odgovori s pošiljanjem celotne baze, temveč pošlje samo nekaj podatkov predstavljenih kot objekt JSON.

V tem diplomskem delu se REST arhitektura uporablja v navezi s HTTP (*angl. Hypertext Transfer Protocol*) za pošiljanje zahtevkov in posredovanje odgovorov.

3.3.6 JSON

JSON (*angl. JavaScript Object Notation*) je odprt standard za podatkovno izmenjavo. Izhaja iz programskega jezika JavaScript in je namenjen predstavitvi enostavnih podatkovnih struktur in asociativnih tabel [10]. Kljub očitni navezi z JavaScriptom, je neodvisen od jezika in ga lahko uporabljamo praktično v vsakem programskem jeziku. Pogosto se uporablja za serializacijo in prenos strukturiranih podatkov po omrežju. Primarno je namenjen izmenjavi podatkov med strežnikom in spletno aplikacijo kot alternativa jeziku XML.

Osnovni podatkovni tipi v JSON so:

- Število (integer ali float)
- Niz (*angl. string*)
- Boolean (true ali false)
- Tabela (*angl. array*); niz vrednosti ločenih z vejico, ki so vklešče med oglata oklepaja
- Objekt (*angl. object*); zbirka parov key:value ločenih z vejico, ki so vklešče med zavita oklepaja.

Primer na sliki 3.8 prikazuje objekt JSON za osebo. Ime in priimek objekta sta niza, starost je število, naslov je objekt, telefonske številke pa so tabela objektov.

JSON je v tej aplikaciji uporabljen za pošiljanje podatkov nazaj s strežnika na mobilno napravo.

```
{
     "ime": "Janez",
     "priimek": "Novak",
     "starost": 39,
     "naslov":
     {
         "hisniNaslov": "Prešernova 7",
         "mesto": "Ljubljana"
     },
     "telefonskaStevilka":
     [
         {
           "tip": "doma",
           "stevilka": "212 555-1234"
         },
         {
           "tip": "mobitel",
           "stevilka": "646 555-4567"
         }
     ]
 }
```

Slika 3.8: Primer objekta JSON za osebo.

Poglavje 4

Razvoj aplikacije

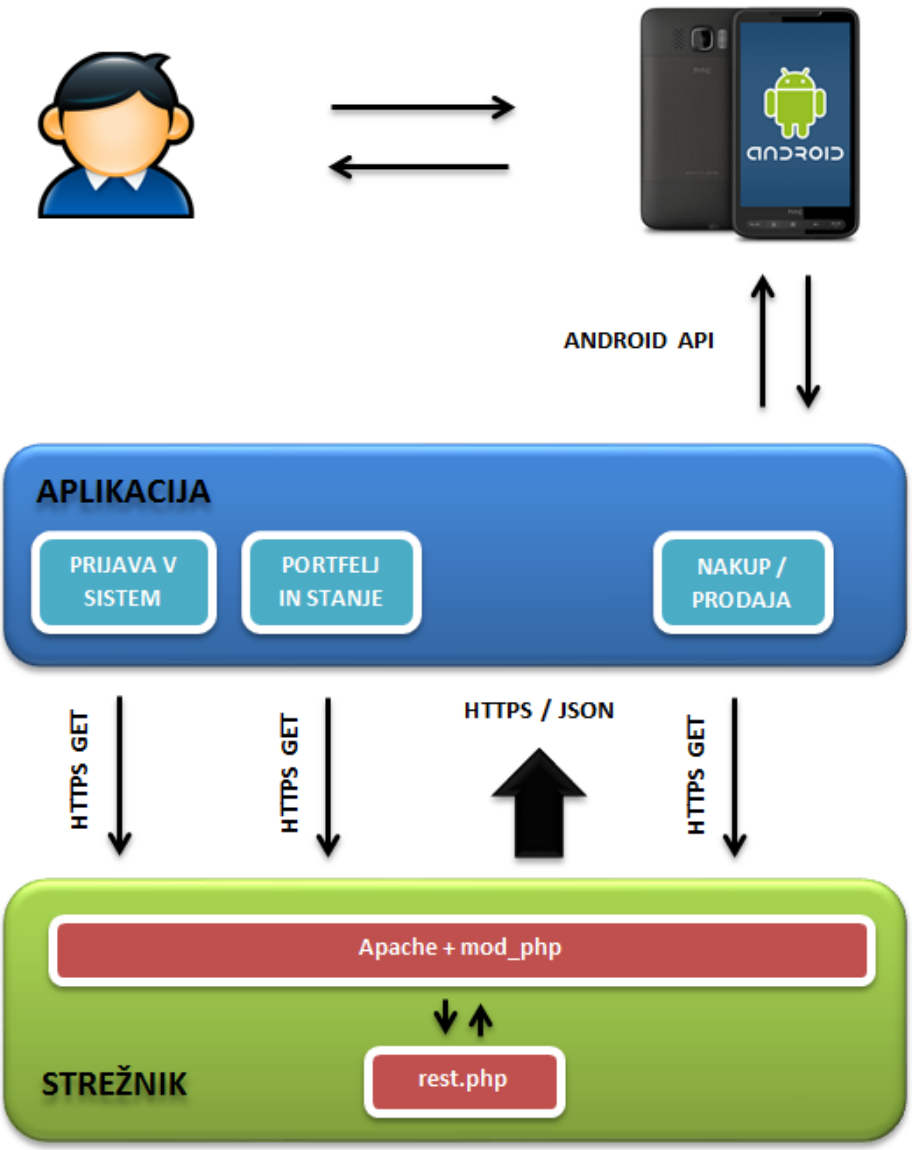
4.1 Arhitektura

Zgradbo same aplikacije najlažje predstavimo z diagramom na sliki 4.1.

Uporabnik neposredno komunicira samo z napravo, kjer teče sistem Android. Mobilna naprava je torej vez med uporabnikom in aplikacijo. Na tem mestu poudarimo, da za delovanje aplikacije ni nujno, da ima mobilna naprava zaslon občutljiv na dotik. Res je, da ta izboljša uporabniško izkušnjo, vendar lahko aplikacijo uporabljajo tudi uporabniki mobilnih naprav, ki imajo vgrajene, na primer, smerne tipke, ali pa kakšno drugo napravo, ki omogoča navigacijo po aplikaciji.

Aplikacija in mobilna naprava (natančneje sistem Android) komunicirata preko vmesnika API. Android aplikaciji nudi sistemske rutine, dostop do strojne opreme naprave, lokacijske informacije, ponovno uporabo komponent ostalih aplikacij itd. Aplikacija nudi uporabniku preko uporabniškega vmesnika rutine za prijavo v sistem, pregled stanja in portfelja in nakup ter prodajo vrednostnih papirjev.

Na najnižjem nivoju se nahaja strežnik, ki je lokacijsko neodvisen. Na njem teče spletni strežnik Apache (ali kakšen drug ekvivalent) v navezi s PHP-jem. Znotraj je nameščena skripta `rest.php`, ki se vedno izvede na samem strežniku in ne na odjemalčevi strani. Ta obdela odjemalčeve zahtevke in vrne primerne rezultate. Komunikacija med strežnikom in aplikacijo na mobilni napravi poteka preko kriptirane povezave HTTPS. Odjemalec pošilja različne zahtevke tipa GET, strežnik pa nanje odgovarja z vračanjem paketov tipa JSON, kjer so zapakirane povratne informacije, ki jih aplikacija potem obdela in po potrebi tudi prikaže.



Slika 4.1: Zgradba razvite aplikacije. Na diagramu imamo štiri ključne komponente: uporabnika, mobilno napravo s sistemom Android, uporabniški del aplikacije ter strežniški del aplikacije. S puščicami je označena komunikacija med komponentami, ponekod pa so zraven tudi protokoli in druge lastnosti komunikacije.

4.2 Razvoj odjemalca

4.2.1 Hello World

Ker pred začetkom pisanja tega diplomskega dela o razvoju na platformi Android nisem vedel praktično ničesar, je bila prva naloga spoznavanje z njo.

Z veseljem lahko povem, da je dokumentacija s strani Googla na spletu precej nazorna. Kot je pri programiranju v navadi, sem se najprej seznanil z osnovami razvoja, potem pa sem za začetek spisal že tradicionalno aplikacijo "Pozdravljen svet" (*angl. Hello World*), ki je prikazana na sliki 4.2.

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class HelloAndroid extends Activity
{
   public void onCreate(Bundle savedInstanceState)
   {
       super.onCreate(savedInstanceState);

       TextView tv = new TextView(this);
       tv.setText("Hello, Android");
       setContentView(tv);
   }
}
```

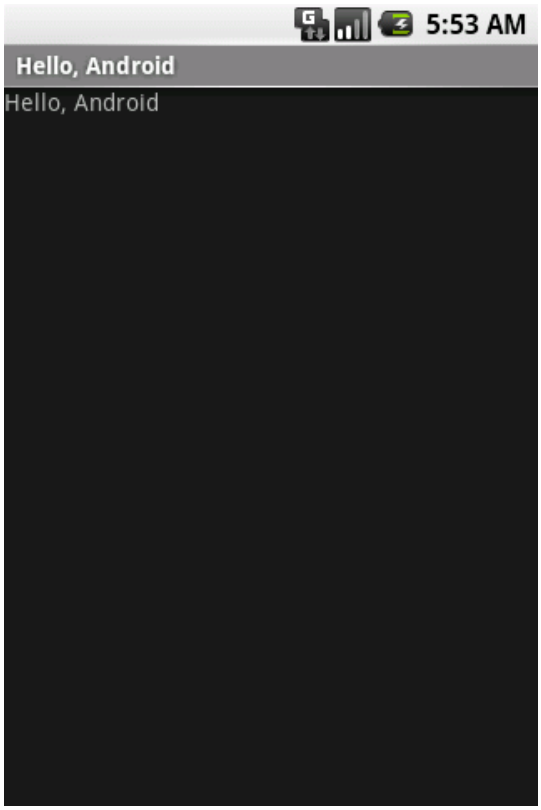
Slika 4.2: Aplikacija Pozdravljen svet.

Kot je bilo povedano že v teoretičnem delu, lahko uporabniški vmesnik ustvarimo in prikažemo samo v komponenti `Activity`. Ta aplikacija vsebuje samo eno aktivnost – `HelloAndroid`. V splošnem mora vsaka aktivnost razširjati razred `Activity` (oz. kateregakoli od njegovih derivatov).

V aplikaciji najprej uvozimo vse potrebne knjižnice. V samem razredu `HelloAndroid` je ključnega pomena metoda `onCreate(BundlesavedInstanceState)`, ki se izvede ob zagonu te aktivnosti. Tukaj navadno zgradimo uporabniški vmesnik in opravimo vso inicializacijo. Kot parameter dobi ta metoda objekt `savedInstanceState` tipa `Bundle`, ki vsebuje zbirko parov tipa key:value. Na tem mestu je dovolj, da vemo, da takšen objekt pride prav pri obnavljanju aktivnost v njenem življenjskem ciklu.

Znotraj metode `onCreate()` najprej ustvarimo objekt tipa TextView, ki bo prikazal naše besedilo. V naslednji vrstici mu določimo katero besedilo naj prikaže, nato pa s klicem metode `setContentView(tv)` to besedilo prikažemo v uporabniškem vmesniku te aktivnosti.

Naslednji korak je seveda zagon aplikacije (slika 4.3).



Slika 4.3: Zagnana aplikacija Pozdravljen svet.

Uporaba postavitvene datoteke

V primeru na slikah 4.2 in 4.3 smo postavitev uporabniškega vmesnika (*angl. User interface layout*) določili programsko. To pomeni, da smo uporabniški vmesnik zgradili kar v kodi samega programa. Takšen pristop je slab iz preprostega razloga; majhne spremembe v postavitvi lahko terjajo korenite spremembe same aplikacije.

Zaradi tega Android omogoča načrtovanje uporabniškega vmesnika v tako imenovanih postavitvenih datotekah XML (*angl. XML layout file*). Razlaga tega koncepta je najenostavnejša kar na primeru (slika 4.4).

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/...."
     android:id="@+id/textview"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:text="Hello, Android" />
```

Slika 4.4: Primer postavitvene datoteke.

Splošna struktura postavitvene datoteke je enostavna: je drevo XML elementov, kjer je vsako vozlišče ime razreda View (in njegovih derivatov) [11] (pogl. 5). Takšna struktura omogoča enostavno in hitro gradnjo uporabniških vmesnikov. Ta model izhaja iz spletnega razvoja, kjer sta predstavitev (uporabniški vmesnik) ter logika aplikacije ločeni.

V primeru na sliki 4.4 imamo v postavitveni datoteki samo en element – TextView, ki bo prikazoval naš tekst. Element ima 5 atributov, ki jih ne bomo podrobneje razlagali. Omenimo samo, da je atribut `android:text` dejansko besedilo, ki ga želimo prikazati. Sedaj spremenimo še metodo `onCreate()`, kjer ne bomo več dinamično kreirali objekta TextView in mu nastavili besedila. Vse kar moramo v tem primeru narediti, je določiti postavitveno datoteko za prikaz v uporabniškem vmesniku. Popravljena metoda `onCreate()` je prikazana na sliki 4.5.

4.2.2 Študentska borza

Opis razredov

Na sliki 4.6 so prikazani razredi, ki tvorijo celotno aplikacijo.

Razredi v zgornjem delu slike, ki so povezani s puščicami, predstavljajo aktivnosti. Vsaka od teh aktivnosti je definirana v manifestu in ima svojo

```
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);
}
```

Slika 4.5: Aplikacija Pozdravljen svet z uporabo postavitvene datoteke.

postavitveno datoteko, ki določa uporabniški vmesnik. Puščice med aktivnostmi nakazujejo, kako se lahko aktivnosti med seboj kličejo. S tem definirajo tudi vse možne poti v aplikaciji. Iz slike je razvidno, da po zagonu aplikacije začnemo v aktivnosti `Login` (seveda to velja samo za prvi zagon; če je aplikacija bila predhodno zagnana, pridemo nazaj v zadnjo aktivnost). Od tu naprej pa preko aktivnosti `Main` (ta je pravzaprav samo ogrodje, ki nam v aplikaciji omogoča zavihke) lahko izbiramo med aktivnostmi (zavihki) `Balance`, `PortfolioGroup`, `BuyGroup`. Iz teh aktivnosti lahko nadalje prehajamo na podaktivnosti, ki se prikazujejo v istem zavihku kot njihovi starši.

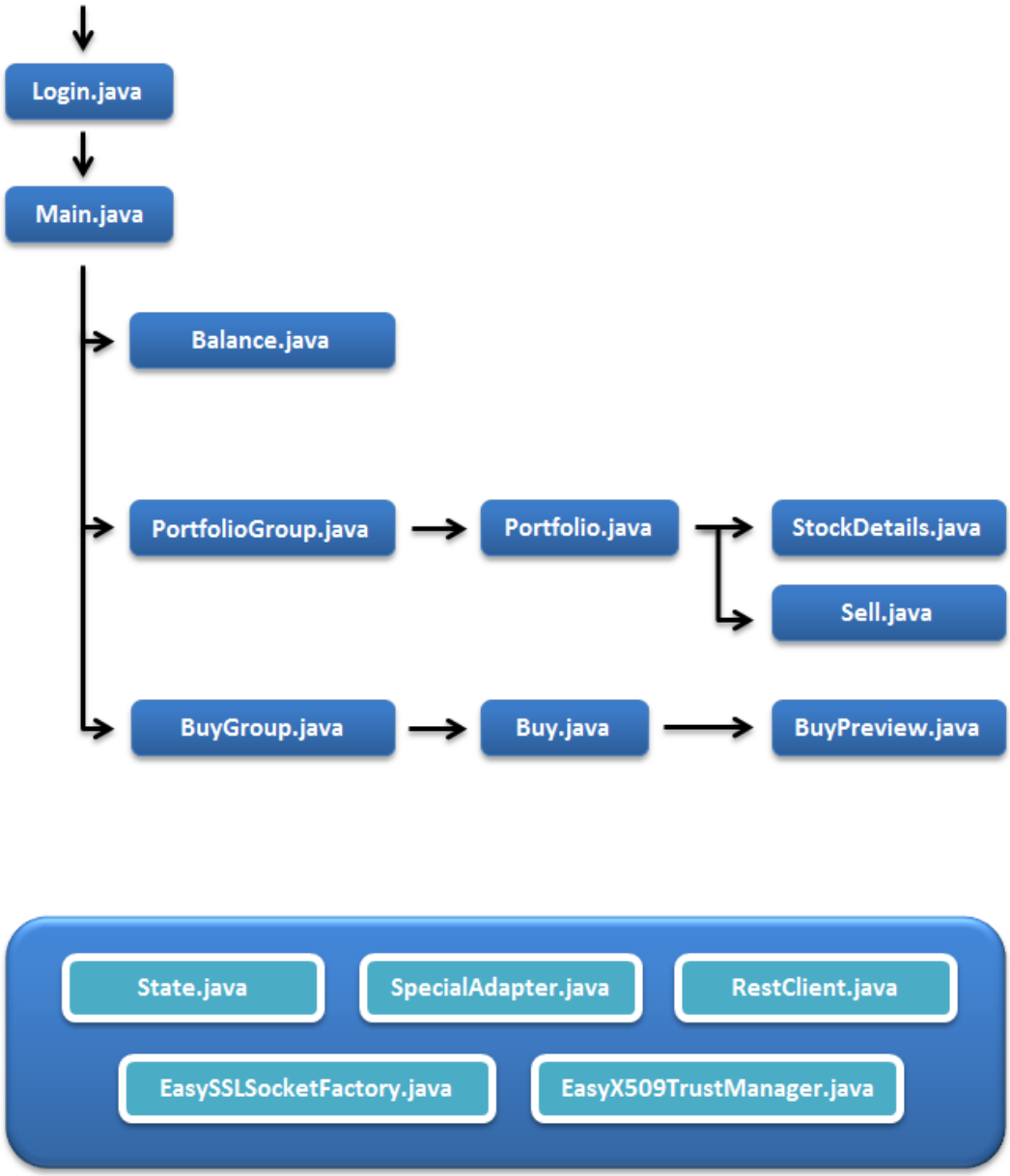
V spodnjem delu slike (v oblačku) so pomožni razredi, ki jih aktivnosti za svoje delovanje potrebujejo. To so, na primer, razred za delo s SSL certifikati, implementacija odjemalca REST itd. Ti razredi niso aktivnosti, zato tudi nimajo definiranega uporabniškega vmesnika in jih ni potrebno navesti v manifestu.

Sedaj pa si podrobneje poglejmo, čemu so posamezni razredi namenjeni.

Login.java

`Login` je začetna aktivnost. Kot že ime nakazuje, omogoča prijavo uporabnika v sistem Študentska borza. Za prikaz uporablja postavitveno datoteko `/res/layout/login.xml`. Na tem mestu omenimo še dobro prakso (slika 4.7), ki je uporabljena praktično v celotni aplikaciji. Ko z gumbom sprožimo prijavo, se komunikacija s strežnikom izvrši v asinhronem opravilu (ločena nit). S tem omogočimo, da glavna nit programa, v kateri teče uporabniški vmesnik, ostane odzivna in nemotena.

Vsak razred, ki razširja razred `AsyncTask`, mora prepisati metode `doInBackground()`, `onPreExecute()` in `onPostExecute()`. Najprej se pokliče metoda `onPreExecute()`, kjer navadno opravimo inicializacijo (v večini aktivnosti te aplikacije v tej metodi prikažemo animacijo, ki prikazuje nalaganje nečesa in sporoča uporabniku, naj počaka). Za tem se pokliče metoda `doInBackground()`, ki opravi glavni del dela (v tej aplikaciji se v tem delu izvede dostop



Slika 4.6: Razredi, ki sestavljajo aplikacijo Študentska borza.

do strežnika). Na koncu se pokliče še metoda `onPostExecute()`, kjer obravnavamo, s strani strežnika, vrnjene rezultate. Kot je razvidno iz slike je metoda `doInBackground()` tipa Boolean. Vrednost, ki jo ta metoda vrača se kot pa-

rameter `result` posreduje metodi `onPostExecute()` in ji s tem sporoča, če se je opravilo izvedlo uspešno.

Main.java

`Main` je aktivnost, v katero pridemo po uspešni prijavi v sistem Študentska borza. Razširja razred `TabActivity` in je samo nekakšno ogrodje, ki nam omogoča zavihke `Stanje`, `Portfelj` in `Nakup`. Zavihke lahko realiziramo na dva načina; tako da je vse skupaj ena aktivnost in s pritiski na različne zavihke samo menjavamo postavitve, ali pa vsak zavihek predstavlja popolnoma ločeno aktivnost. V tej aplikaciji je uporabljena slednja možnost, saj se mi je na prvi pogled zdela preprostejša in elegantnejša. Vendar videz vara, saj ta način prinaša kup pasti in težav (opisano v poglavju 4.2.5) in če bi izbiral še enkrat, bi izbral prvo možnost. V tem razredu tudi lovimo zahtevo za kreiranje menija in tega ustvarimo glede na trenutno izbran zavihek.

Balance.java

`Balance` je aktivnost, ki jo po uspešni prijavi najprej zagledamo. Nudi nam informacije o trenutnem stanju računa. Podobno kot pri aktivnosti `Login.java` poteka tudi tukaj komunikacija s strežnikom v asinhronem opravilu. Prenos podatkov iz strežnika in osveževanje pogleda sta popolnoma ločena, prenos podatkov za stanje računa in prenos podatkov za portfelj pa sta, iz performančnih razlogov, združena. Tako recimo pri osvežitvi portfelja samo pokličemo metodo za osvežitev pogleda stanja računa in se izognemo ponovnemu prenašanju podatkov. Enako ob prenosu podatkov pri stanju računa samo osvežimo pogled portfelja.

PortfolioGroup.java

`PortfolioGroup` razširja razred `ActivityGroup` in obstaja samo zato, da lahko znotraj posameznega zavihka prikazujemo več različnih aktivnosti [12] – v tem primeru so to `Portfolio`, `StockDetails` in `Sell`.

Portfolio.java

`Portfolio` je glavna aktivnost za upravljanje s portfeljem. Prikazuje nam vrednostne papirje, ki jih trenutno posedujemo. Za vsak papir lahko od tukaj poženemo aktivnosti `StockDetails` za podrobnejši prikaz informacij o vrednostnem papirju ter `Sell` za prodajo vrednostnega papirja. `Portfolio` je tudi

```
public void onCreate(Bundle savedInstanceState)
{
     loginButton.setOnClickListener(new View.OnClickListener()
     {
        public void onClick(View view)
        {
                /* when clicked on login button, fire AsyncTask which
                 * will handle login in seperate thread
                 */
                new LoginTask().execute();
        }
     });
}

private class LoginTask extends AsyncTask<String, Void, Boolean>
{
     protected Boolean doInBackground(String... params)
     {
          /* ... */
     }

     protected void onPostExecute(Boolean result)
     {
          /* ... */
     }

     protected void onPreExecute()
     {
          /* ... */
     }
}
```

Slika 4.7: Proženje asinhronega opravila v ločeni niti.

edina aktivnost, ki vsebuje tako imenovani kontekstualni meni, ki se prikaže, če na posameznem papirju pridržimo prst (pritisnemo za dalj časa).

Ker se lahko v tem zavihku pojavijo tri aktivnosti vključno z aktivnostjo `Portfolio`, rabimo za primer, ko aplikacijo zapustimo, ali pa jo zaradi potreb po pomnilniku zapre sistem, dodatno logiko za obnovitev tega stanja. Če je uporabnik v aktivnosti `StockDetails` oz. `Sell`, bi sistem brez te logike po ponovnem zagonu aplikacije v zavihku pognal začetno aktivnost (t.j. aktivnosti `Portfolio`), česar pa seveda uporabnik ne pričakuje in ne želi. Temu se izognemo tako, da v metodi `onSaveInstanceState(Bundle outState)`, ki se pokliče tik pred zapustitvijo aplikacije, v pomnilnik naprave shranimo potrebne podatke za obnovitev stanja (to pomeni, da jih zapakiramo v `Bundle`) in jih potem v metodi `onCreate(Bundle savedInstanceState)` obnovimo. Primer tega je prikazan na sliki 4.8.

StockDetails.java

`StockDetails` je zelo enostavna aktivnost, ki v navezi s postavitveno datoteko `/res/layout/stock_details.xml` prikaže podrobnosti o vrednostnem papirju, ki smo ga izbrali v portfelju. Do tega pridemo tako, da v portfelju pritisnemo (izberemo) vrednostni papir.

Sell.java

Aktivnost `Sell` nam omogoča prodajo vrednostnih papirjev. Do nje pridemo preko kontekstualnega menija v portfelju. Spremembe izračunov se računajo sproti ob spremembi količine vrednostnih papirjev, ki jih želimo prodati. To je doseženo z metodo `addTextChangedListener()`, ki nam omogoča spremljanje objekta `EditText`, ob spremembi pa pokliče ustrezno rutino za izračun novih vrednosti. V tej aktivnosti in aktivnosti `Buy` je bilo nekaj težav s fizično tipkovnico in registriranjem pritiska na tipko `nazaj`. To je podrobneje opisano v poglavju 4.2.5.

Ob uspešni prodaji se prav tako osvežita zavihka za stanje in portfelj (iz portfelja se odstrani ustrezno količino tega vrednostnega papirja, v stanju pa se vrednost prodaje prišteje k stanju na računu).

BuyGroup.java

Aktivnost `BuyGroup` podobno kot `PortfolioGroup` razširja razred `ActivityGroup` in obstaja samo zato, da lahko znotraj posameznega zavihka prikazujemo več različnih aktivnosti [12] – v tem primeru sta to `Buy` ter `BuyPreview`.

```
public void onCreate(Bundle savedInstanceState)
{
   if (savedInstanceState != null)
   {
      try
      {
         stockDetailsID = savedInstanceState.getInt(
                             "stockDetailsID");

         sellID = savedInstanceState.getInt("sellID");
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
   }
}

protected void onSaveInstanceState(Bundle outState)
{
   outState.putInt("stockDetailsID", stockDetailsID);
   outState.putInt("sellID", sellID);

   super.onSaveInstanceState(outState);
}
```

Slika 4.8: Primer shranjevanja stanja v metodi onSaveInstanceState() in obnavljanja stanja ob zagonu aktivnosti v metodi onCreate().

Buy.java

Aktivnost `Buy` nam omogoča nakup vrednostnih papirjev. Podobno kot aktivnost `Portfolio` tudi `Buy`, ob zaprtju aplikacije, shranjuje in obnavlja stanje, saj v tem zavihku lahko poleg `Buy` teče še aktivnost `BuyPreview`. V meniju je omogočeno tudi osveževanje pogleda, kar terja nov prenos podatkov iz strežnika.

BuyPreview.java

Aktivnost `BuyPreview` je na las podobna aktivnosti `Sell`. Razlika je le v logiki, ki se za to aktivnost izvede na strežniku ter v dostopu (do te aktivnosti pridemo v zavihku za nakup, ko izberemo želeni vrednostni papir).

State.java

State je zelo pomemben razred, ki razširja razred `Application`. Predstavlja nekakšno globalno odložišče. V ta objekt lahko namreč piše ali bere vsaka aktivnost kadarkoli. Vsebuje spremenljivke za uporabniško ime, geslo, podatke o stanju in portfelju, rutine za prenos stanja in portfelja itd.

Podoben princip netrajnega shranjevanja podatkov je uporaba razreda `SharedPreferences`. Slednji način v tem diplomskem delu ni uporabljen, ker omogoča samo shranjevanje primitivnih tipov v obliki parov `key:value`.

SpecialAdapter.java

Razred `SpecialAdapter` je razširitev razreda `SimpleAdapter`, ki je v aplikaciji uporabljen za objekte `ListView`. V splošnem objekt tipa `SimpleAdapter`, ki je napolnjen s podatki za seznam, vežemo na `ListView`, ta pa sam poskrbi za prikaz. V primeru, da hočemo prikaz kakorkoli spreminjati, moramo `SimpleAdapter` razširiti z ustreznimi popravki. V aplikaciji je uporabljen, na primer, za spremembo barve teksta pri pregledu spremembe tečaja (pozitivna sprememba tečaja se obarva zeleno, negativna pa rdeče), ali pa za barvanje elementov seznama v portfelju.

EasySSLSocketFactory.java ter EasyX509TrustManager.java

Ta dva pomožna razreda omogočata uporabo certifikatov SSL (pri uporabi kriptirane povezave), ki so lastnoročno podpisani [13]. Privzeto to namreč ni mogoče, saj mora biti vsak certifikat SSL podpisan s strani ustrezne organizacije.

RestClient.java

Razred `RestClient` je uporabljen v vsaki komunikaciji s spletom in je tisti razred, ki dejansko prenese podatke iz podanega naslova URL. V razredu so definirane nastavitve povezave TCP (npr. različica protokola HTTP, kodiranje ciljne spletne strani, čas za iztek povezave itd.) in funkcije, ki izvedejo prenos.

4.2.3 Uporabniški vmesnik in uporaba aplikacije

Uporabniški vmesnik je bil zasnovan z uporabnikom v mislih. To pomeni, da je bil cilj čim bolj enostaven in intuitiven uporabniški vmesnik. Barvna shema je bila nekoliko prilagojena celostni grafični podobi Študentske borze, ne pa v celoti. Ves vmesnik je definiran v ustreznih postavitvenih datotekah, kar nam omogoča hitro in učinkovito spreminjanje samega videza aplikacije. Poudarimo tudi, da je bil uporabniški vmesnik zasnovan z mislijo, da naj na vseh napravah izgleda približno enako. Android nam to omogoča z uporabo abstraktnih enot, ki so bile uporabljene tudi v tej aplikaciji.

Ob zagonu aplikacije se nam najprej prikaže okno za prijavo v sistem Študentska borza (slika 4.9). Za prijavo v sistem v ustrezni polji vnesemo uporabniško ime in geslo ter pritisnemo gumb `Prijava`.

Po uspešni prijavi se nam odpre glavna aktivnost s tremi zavihki. Prvi zavihek je namenjen pregledu stanja (slika 4.10). Pri tem posamezne postavke pomenijo naslednje:

- **Preostala gotovina** – sredstva, ki jih še imamo na voljo za nakup vrednostnih papirjev.
- **Porabljena gotovina** – sredstva, ki smo jih že porabili za predhodne nakupe vrednostnih papirjev.
- **Današnja vrednost** – današnja vrednost sredstev iz zgornje postavke. Če je poslovanje pozitivno, je ta postavka višja od zgornje, v nasprotnem primeru pa nižja.
- **Sprememba** – razlika med današnjo vrednostjo in porabljeno gotovino.
- **Stanje portfelja** – vrednost celotnega portfelja z upoštevanjem še nekaterih dodatnih faktorjev.

V meniju aplikacije (do njega pridemo s pritiskom fizične tipke za meni na mobilni napravi) nam je na voljo tudi gumb za osvežitev stanja (ta sproži ponoven prenos podatkov s strežnika in osvežitev pogleda).

Slika 4.9: Zaslonska maska prijave v sistem Študentska borza.

Naslednji zavihek prikazuje trenutno stanje portfelja (slika 4.11). Za vsak kupljen vrednostni papir se prikaže: simbol vrednostnega papirja, količina kupljenih papirjev, sprememba tečaja ter polno ime vrednostnega papirja. Čisto ob strani posameznega vrednostnega papirja je z barvo predstavljen tudi status le-tega. Na dnu zaslona je tudi legenda, ki ob pritisku na posamezno barvo, prikaže, kaj ta pomeni. Pomen barv je naslednji:

- **Zelena** – vrednostni papir je mogoče prodati.
- **Rdeča** – vrednostnega papirja ni mogoče prodati, ker je bil kupljen po trenutno veljavni tečajnici.
- **Vijolična** – vrednostnega papirja ni mogoče prodati zaradi premalega prometa.
- **Modra** – vrednostnega papirja ni mogoče prodati, ker na pravi borzi poteka trgovanje.

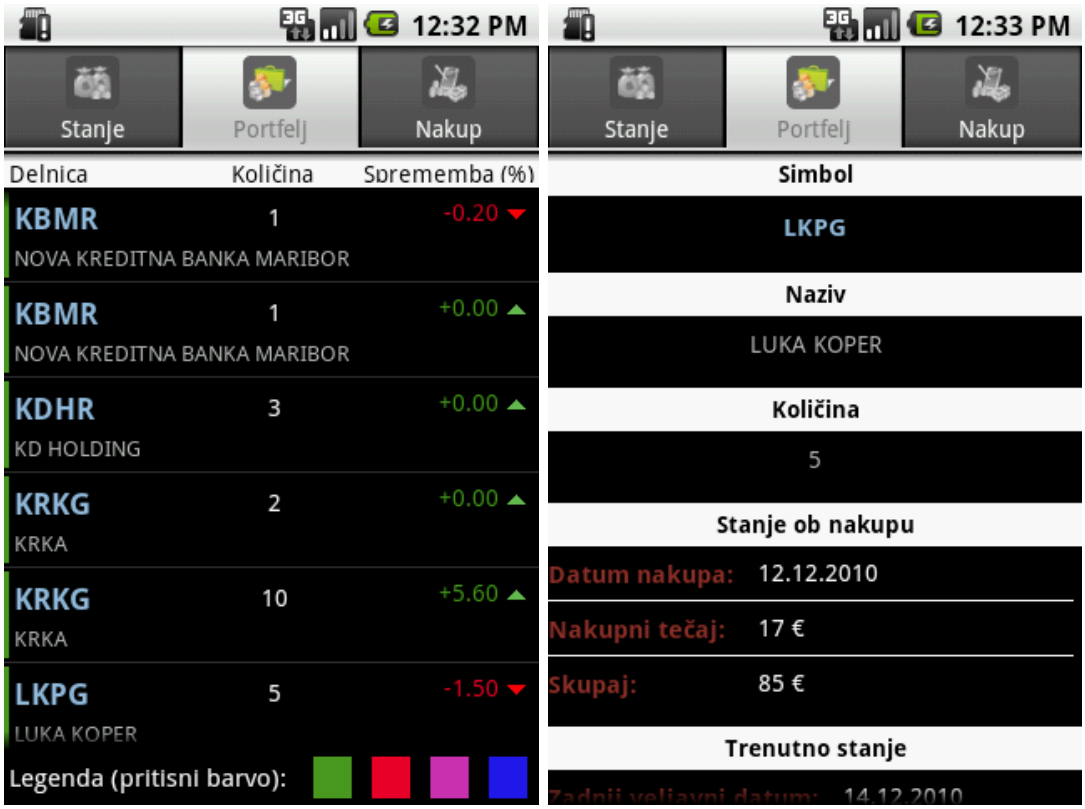
Slika 4.10: Zaslonska maska aktivnosti za prikaz trenutnega stanja. Viden je tudi meni, ki nam omogoča osvežitev stanja.

Samo v primeru, da je vrednostni papir zelene barve, lahko na posameznem papirju pridržimo prst (pritisnemo za dalj časa) in v kontekstualnem meniju izberemo opcijo za prodajo. Tako kot pri stanju, nam je tudi tukaj v meniju na voljo gumb za osvežitev portfelja.

V primeru, da na posamezen papir samo pritisnemo (torej ne pridržimo), se nam odpre podrobnejši pogled vrednostnega papirja. Ta prikazuje:

- Simbol vrednostnega papirja
- Naziv vrednostnega papirja
- Količino kupljenih papirjev
- Stanje ob nakupu
  - Datum nakupa

  - Nakupni tečaj
  - Skupno vrednost nakupa
- Trenutno stanje
  - Datum zadnjega veljavnega tečaja
  - Zadnji veljavni tečaj
  - Skupno vrednost po zadnjem veljavnem tečaju
- Spremembo (v EUR in %).



Slika 4.11: Zaslonski maski prikaza portfelja ter podrobnejšega pogleda za posamezni vrednostni papir.

Prodajo vrednostnega papirja (slika 4.12) izvedemo tako, da v polje za količino vnesemo, koliko vrednostnih papirjev želimo prodati. Predogled prodaje se ob spremembi količine osveži avtomatično. Vsebuje naslednje podatke:

- Količino vrednostnih papirjev
- Prodajni tečaj
- Vrednost prodaje
- Staro stanje (pred prodajo)
- Novo stanje (po prodaji).

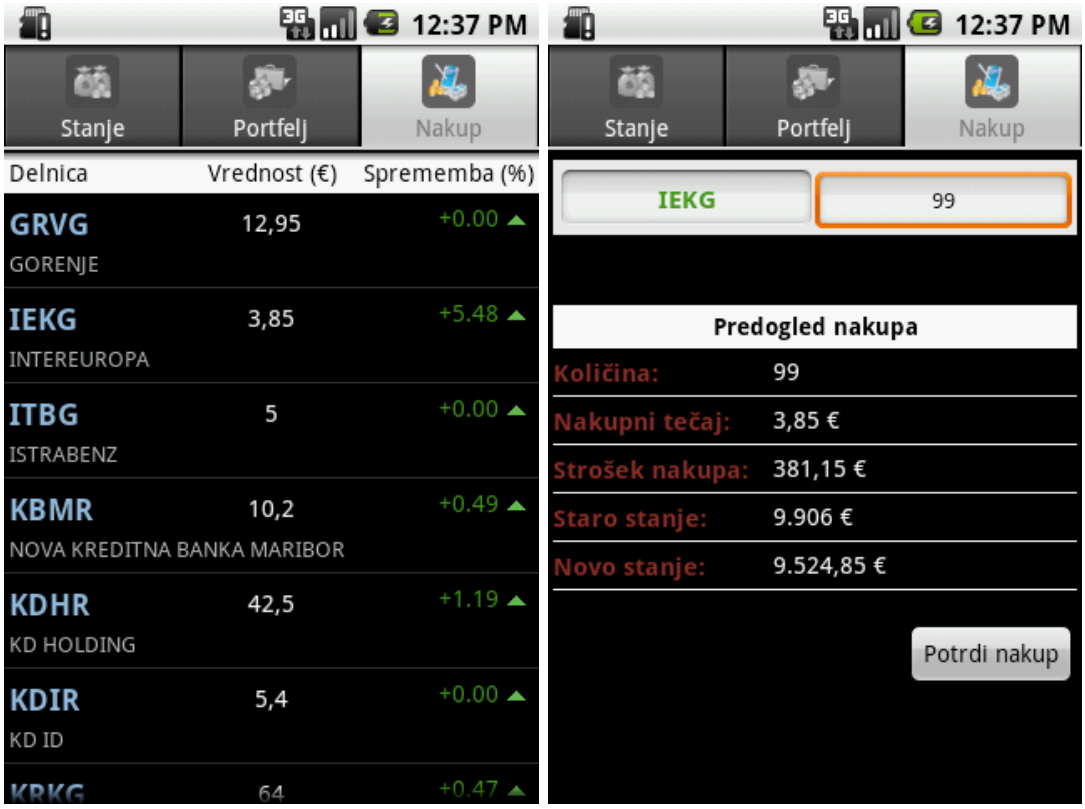
Če izberemo neprimerno količino vrednostnih papirjev za prodajo (bodisi premajhno bodisi preveliko), nas sistem na to opozori in onemogoči prodajo (s tem, da skrije gumb za potrditev prodaje).

Ko smo s predogledom zadovoljni, prodajo zaključimo s pritiskom na gumb `Potrdi prodajo`.

Slika 4.12: Zaslonska maska prodaje vrednostnega papirja.

Zadnji zavihek je namenjen nakupu vrednostnih papirjev (slika 4.13). Je na las podoben zavihku za portfelj – za vsak vrednostni papir, ki je na voljo za

nakup, se prikaže: simbol vrednostnega papirja, vrednost v EUR, sprememba tečaja ter polno ime vrednostnega papirja. S pritiskom na posamezen papir se nam odpre aktivnost za nakup. Predogled nakupa vsebuje enake podatke kot prodaja in tudi sam nakup poteka enako kot prodaja, zato tukaj tega ne bomo še enkrat opisovali.



Slika 4.13: Zaslonski maski nakupa in predogleda nakupa vrednostnega papirja.

4.2.4 Testiranje aplikacije

Aplikacija je bila testirana tako na emulatorju kot na dveh fizičnih mobilnih napravah. Sama narava aplikacije je takšna, da ne zahteva uporabo naprav kot so kompas ali pa GPS, katerih testiranje na emulatorju je lahko precej težavno. Zato je bila večina testiranja opravljena kar na emulatorju, ki je, po zaslugi DDMS, veliko priročnejši predvsem pri razhroščevanju. Z različnimi AVD-ji (*angl. Android Virtual Device*) so bile posnemane naslednje naprave:

- Samsung Galaxy Spica i5700 (zaslon 3.2", ločljivost 320×480 pik, Android 2.1)
- Samsung Galaxy S (zaslon 4", ločljivost 480×800 pik, Android 2.2)
- HTC Wildfire (zaslon 3.2", ločljivost 240×320 pik, Android 2.1).

Ko je bila aplikacija končana, je bila stestirana tudi na fizičnih napravah Samsung Galaxy Spica i5700 ter Samsung Galaxy S. Zaradi narave aplikacije in testiranja na emulatorju, hroščev, razen dveh izjem, praktično ni bilo.

Kot zanimivost omenimo, da aplikacija na sami napravi deluje veliko hitreje kot v emulatorju, kljub temu da je naprava precej manj zmogljiva, kot pa računalnik na katerem je tekel emulator. To je po svoje seveda razumljivo, saj mora emulator znotraj operacijskega sistema, na kateremu teče, poganjati tako ARM Linux jedro kot tudi ARM JVM (*angl. Java Virtual Machine*). To ga naredi počasnega, vendar zagotavlja, da se ta obnaša kot dejanska naprava.

4.2.5 Težave pri razvoju

Težava pri gumbu nazaj v zavihkih

Ena izmed težav, ki jih prinaša način z zavihki, kjer je vsak zavihek ločena aktivnost, se pojavi pri pritisku gumba `nazaj` na mobilni napravi. Težava se pojavi pri podpostavitvah (na primer, ko v portfelju pritisnemo na določen vrednostni papir in se nam v zavihku odpre podrobnejši pogled tega papirja), saj se Android, namesto da bi se vrnil na prejšnjo aktivnost v zavihku, vrne na aktivnost pred glavno aktivnostjo `TabActivity` (v našem primeru se, namesto v aktivnost `Portfolio`, vrne na predhodnika aktivnosti `Main`, torej v aktivnost za prijavo v sistem `Login`), česar pa seveda nočemo. Težava je rešena z uporabo `ActivityGroup` ter nekaj dodatne logike [12].

Težave s kriptirano povezavo

Na Študentski borzi zaenkrat nimamo podpisanega certifikata SSL, zato se uporabljata dva pomožna razreda, ki omogočata uporabo certifikatov SSL (pri uporabi kriptirane povezave), ki so lastnoročno podpisani [13]. Privzeto to namreč ni mogoče, saj mora biti vsak certifikat SSL podpisan s strani ustrezne organizacije.

Kriptirana komunikacija poteka po ločenem naslovu, ki ne vsebuje domene Študentske borze. V ta namen se že pri prijavi v sistem, najprej nekriptirano, pošlje zahtevek na strežnik za kriptiran naslov. Ko strežnik ta naslov posreduje odjemalcu, poteka vsa nadaljnja komunikacija po kriptirani povezavi.

Združen prenos podatkov za stanje in portfelj

Vsaka vzpostavitev kriptirane povezave in prenos podatkov terjata dodatna delovna sredstva (*angl. overhead*), kar se še posebej pozna na mobilnih napravah, kjer so viri omejeni [11] (pogl. 1). Zato je prenos podatkov o stanju in portfelju združen, kljub začetnim načrtom, po katerih naj bi bil ločen. S tem se osveževanje stanja in portfelja občutno pohitri, poleg tega pa prenesemo manj podatkov.

4.3 Razvoj strežniškega dela

Razvoj strežniškega dela pravzaprav ni tema tega diplomskega dela, saj so rutine že bile spisane za spletni vmesnik Študentske borze. Tako je bilo potrebno določene funkcije samo prilagoditi.

Strežniški del obsega samo eno datoteko – `rest.php`. Zahtevki se vršijo z dostopom do naslovov URL (*angl. Uniform Resource Locator*) podobnim naslovu na sliki 4.14.

```
https://www.studentska-borza.com/android/
rest.php?action=login&username=janez&password=klobasa
```

Slika 4.14: Primer naslova URL preko katerega aplikacija pošlje zahtevek strežniku.

V naslovu so vsebovani parametri tipa GET, katerih vrednosti se, zaradi prenosa po varni povezavi HTTPS, pred pošiljanjem samodejno kriptirajo. V primeru iz slike 4.14, so to parametri `action`, `username` in `password`. Ker je uporabljena arhitektura REST, moramo v vsakem zahtevku za avtentikacijo

poslati tudi uporabniško ime in geslo. Strežnik iz naslova izlušči kaj od njega želimo, potem pa pokliče ustrezno funkcijo (slika 4.15), ki zahtevek obdela in vrne objekt JSON.

```
$action = (String)$_GET['action'];
$username = mysql_real_escape_string($_GET['username']);
$md5pass = md5(mysql_real_escape_string($_GET['password']));

switch ($action)
{
     case "getHttpsUrl":  getHttpsUrl($httpsUrl);
                          break;

     case "login":        performLogin($username, $md5pass);
                          break;

     /* ... */
}
```

Slika 4.15: Del strežniške kode, ki iz naslova izlušči parametre in ustrezno ukrepa.

Objekt JSON lahko generiramo tako, da ustvarimo asociativno tabelo, potem pa pokličemo funkcijo `json_encode()`, ki pretvori tabelo v objekt JSON (slika 4.16). Tega nato vrnemo odjemalcu.

```
$array = array('loginResult'=>1)
$returnJSON = json_encode($array);

echo $returnJSON;
```

Slika 4.16: Primer kreiranja odgovora na zahtevek za prijavo v sistem Študentska borza v obliki objekta JSON.

Poglavje 5

Sklepne ugotovitve in ideje za nadaljnje delo

V diplomskem delu je opisan razvoj aplikacije za trgovanje na Študentski borzi, ki teče na mobilni platformi Android.

Uspelo mi je izpolniti vse naloge, ki sem si jih na začetku zadal. Z aplikacijo je možno kjerkoli (seveda samo v primeru, da je na voljo internetna povezava) in kadarkoli pregledati stanje na računu, pregledati portfelj in navsezadnje vrednostne papirje tudi kupiti in prodati. Aplikacija je tehnološko dovršena in stestirana v dovoljšnji meri za uporabo v produktnem okolju.

Jasno je, da aplikacija nikoli ne bo ponujala vseh funkcionalnosti spletnega vmesnika Študentske borze. Vseeno pa bi bilo dobro vgraditi še naslednje funkcionalnosti, ki so navedene v padajočem vrstnem redu po pomembnosti:

1. Registracija – aplikacija je trenutno namenjena že registriranim uporabnikom. Novi uporabniki se v sami aplikaciji ne morejo registrirati, lahko pa to seveda storijo preko spletnega vmesnika. Dobro bi bilo, če bi bila možna registracija novih uporabnikov kar neposredno v aplikaciji.

2. Pregled rezultatov – v aplikaciji bi bilo zelo priročno, če bi lahko pregledovali rezultate trgovanja, kot to ponuja spletni vmesnik. Tako bi lahko na napravi pregledovali trenutne, mesečne, četrtletne ter letne rezultate, na voljo pa bi bil tudi arhiv vseh rezultatov.

3. Forum – Študentska borza vsebuje forum, ki je bogata zakladnica znanja, ki jo prispevajo uporabniki sami. Aplikacija bi lahko vsebovala vmesnik za prebiranje tem ter sodelovanje v njih.

4. Sporočila – aplikacija bi lahko vsebovala vmesnik za pregled in pošiljanje sporočil med uporabniki, podoben tistemu na spletu.

5. Pregled novic – na Študentski borzi se dnevno objavljajo sveža borzna poročila, ostale novice iz sveta borzništva, ugodnosti za uporabnike ter sporočila v zvezi s samo Študentsko borzo. Vse to bi lahko bilo dostopno tudi v aplikaciji.

6. Pregled in prijava na dogodke – aplikacija bi lahko vsebovala sistem za pregled in prijavo na dogodke (delavnice, seminarji, okrogle mize itd.), ki jih organizira Študentska borza.

7. Galerija – aplikacija bi lahko vsebovala tudi galerijo dogodkov Študentske borze. Slike bi se seveda nalagale sproti in ne bi bile vsebovane v sami aplikaciji. Implementacija bi bila pravzaprav precej enostavna, saj Android že ponuja pogled `Gallery`, ki omogoča točno to, kar želimo.

Na Študentski borzi smo ravno na prehodu na novo trgovanje, ki bo zajemalo še ostale najbolj priljubljene svetovne borze. Aplikacija je bila razvita s tem v mislih, zato bodo po prehodu potrebni le manjši popravki. Ko bo vse to izvedeno, bomo seveda aplikacijo ponudili za brezplačen prenos na Android Marketu in s tem omogočili še priročnejše izobraževanje mladih na finančno-borznem in gospodarskem področju.

Slike

Literatura

[1] Mobile operating system.
Dostopno na: *http://en.wikipedia.org/wiki/Mobile_operating_system*
(zadnji obisk: 17. 3. 2011)

[2] Na kratko o Študentski borzi.
Dostopno na: *http://www.studentska-borza.com/datoteke/Na kratko o Studentski borzi.pdf*
(zadnji obisk: 17. 3. 2011)

[3] Android (operating system).
Dostopno na: *http://en.wikipedia.org/wiki/Android_(operating_system)*
(zadnji obisk: 17. 3. 2011)

[4] Android dokumentacija.
Dostopno na: *http://developer.android.com/index.html*
(zadnji obisk: 17. 3. 2011)

[5] R. Meier, Professional Android™Application Development, Indianapolis: Wiley Publishing, 2009

[6] Android SDK.
Dostopno na: *http://developer.android.com/sdk/index.html*
(zadnji obisk: 17. 3. 2011)

[7] Apache HTTP Server.
Dostopno na: *http://en.wikipedia.org/wiki/Apache_HTTP_Server*
(zadnji obisk: 17. 3. 2011)

[8] PHP.
Dostopno na: *http://en.wikipedia.org/wiki/Php*
(zadnji obisk: 17. 3. 2011)

[9] Representational State Transfer.
Dostopno na: *http://en.wikipedia.org/wiki/Representational_State_Transfer*
(zadnji obisk: 17. 3. 2011)

[10] JSON.
Dostopno na: *http://en.wikipedia.org/wiki/Json*
(zadnji obisk: 17. 3. 2011)

[11] M. L. Murphy, The Busy Coder's Guide To Android Development, CommonsWare, 2008

[12] Android: TabActivity Nested Activities.
Dostopno na: *http://blog.henriklarsentoft.com/2010/07/android-tabactivity-nested-activities/*
(zadnji obisk: 17. 3. 2011)

[13] Self Signed SSL acceptance Android.
Dostopno na: *http://stackoverflow.com/questions/1217141/self-signed-ssl-acceptance-android*
(zadnji obisk: 17. 3. 2011)