UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Matej Velikonja

Razvoj mobilne aplikacije m-Študent za pedagoške delavce

DIPLOMSKO DELO
VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: doc. dr. Rok Rupnik

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: 00119/2011
Datum: 05.04.2011

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

Kandidat: **MATEJ VELIKONJA**

Naslov: **RAZVOJ MOBILNE APLIKACIJE M-ŠTUDENT ZA PEDAGOŠKE DELAVCE**

THE DEVELOPMENT OF MOBILE APPLICATION M-STUDENT FOR FACULTY STAFF

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Na podlagi poznavanja funkcionalnosti sistema e-Študent izdelajte analizo funkcionalnosti za mobilno aplikacijo m-Študent za pedagoške delavce. Mobilna aplikacija naj pedagoškim delavcem omogoča tiste storitve, ki bi jih po vaši oceni potrebovali v stanju mobilnosti. Na podlagi analize izdelajte načrt za mobilno aplikacijo m-Študent in jo razvijte na platformi Android.

Mentor:

doc. dr. Rok Rupnik

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Matej Velikonja,

z vpisno številko 63050125,

sem avtor diplomskega dela z naslovom:

Razvoj mobilne aplikacije m-Študent za pedagoške delavce

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom
  doc. dr. Rok Rupnik
- 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 26.10.2011 Podpis avtorja/-ice:

Zahvala

Na tem mestu bi se zahvalil vsem, ki so na kakršenkoli način pripomogli k nastanku te diplomske naloge.

Kazalo

Seznam uporabljenih simbolov in kratic

GPS - Global Positioning System; sistem globalnega določanja lege

SMS - Short Message Service; sistem za kratka sporočila

PC - Personal computer; osebni računalnik

CA - Certificate Agency; certificatna agencija

API - Application Programming Interface; vmesnik uporabniškega programa

SSL - Secure Sockets Layer; plast za varnost vtičnic

HTTPS - Hypertext Transfer Protocol Secure; protokol za varen prenos

JSON - JavaScript Object Notation; objektna notacija JavaScript

URI - Uniform Resource Identifier; enolični identifikator vira

SGL - Skia Graphics Engine; grafični pogon Skia

GL - Graphics Library; grafične knjižice

2D - Two Dimensional; dvo dimenzionalno

3D - Three Dimensional; tri dimenzionalno

SQL - Structured Query Language; sestavljeni jezik za poizvedbe

UML - Unified Modeling Language; poenoteni jezik modeliranje

BKS - Bouncy Castle; javanski paket za kriptografske algoritme

Povzetek

V diplomski nalogi je opisan razvoj celovite rešitve m-Študent, ki temelji na študentskem informacijskem sistemu e-Študent. Odjemalec m-Študent je aplikacija prilagojena za mobilne naprave z operacijskim sistemom Android. V delu se najprej spoznamo z obstoječim sistemom e-Študent. Diplomska naloga pa se nadaljuje z opisom Androida in ostalih uporabljenih orodij ter tehnologij. V jedru naloge se spoznamo s samim potekom razvoja rešitve. Najprej spoznamo kako je rešitev sestavljena, kateri so njeni ključni deli in kakšne so zahteve za poganjanje. V zaključku je predstavljen še manjši problem, do katerega smo prišli med razvojem in nekaj idej za nadaljnji razvoj.

Ključne besede:

Android, mobilna aplikacija, e-Študent, m-Študent

Abstract

The thesis describes the development of m-Študent solution, which is based on student's information system e-Študent. Client of this solution is application adapted to work with mobile devices with Android operating system. System e-Študent is described first. Afterwards Android and other technologies are presented. The development of the solution is described in the central part of the thesis. The architecture and key parts are presented and are followed by minimum requirements for running the application. Problem discovered during development and ideas for future are explained in the conclusion.

Key words:

Android, mobile application, student's information system e-Študent, m-Študent

Poglavje 1

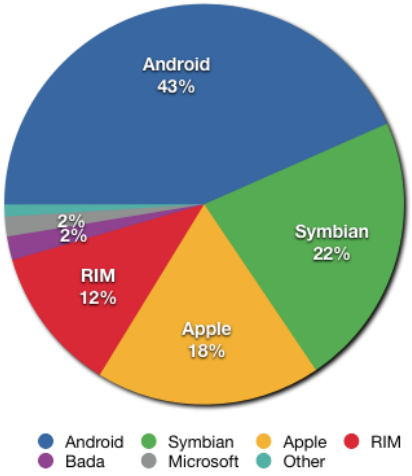
Uvod

Računalništvo postaja vedno bolj osebno in dostopno povsod in kadarkoli. Primer takih računalnikov so mobilne naprave, ki že kar nekaj časa niso namenjene samo telefonskih pogovorom in pošiljanju kratkih sporočil SMS. Sposobne so pošiljanja podatkov in videa, uporabo socialnih omrežij, zajem posnetkov in videoposnetkov, itd. Naprave so na poti, da postanejo novodobni PCji. Njihove zmogljivosti namreč dosegajo zmogljivostim osebnih računalnikov. Trenutno najbolj zmogljiva naprava na trgu ima kar 1,2 GHz dvojedrni procesor, 1 GB pomnilnika, zaslon občutljiv na dotik, več načinov povezovanje, GPS itd.

Na trgu mobilnih naprav se trenutno nahaja več operacijskih sistemov. Trenutno najbolj popularni operacijski sistemi so **Android** (43% delež), **Symbian** (22% delež) in **Applov iOS** (18% delež). Prikaz deležev lahko vidimo na sliki 1.1. [2]

Po podatkih dveh največjih slovenskih mobilnih operaterjev, se delež prodaje in uporabe pametnih mobilnih telefonov povečuje. Pri Mobitelu poročajo, da je uporabnikov, ki uporabljajo pametne mobilne telefone skoraj 50%. Pri Simobilu pa je uporaba malenkostno manjša in sicer 40%. Najbolj popularen operacijski sistem pa je Android. [3]

Diplomsko delo se osredotoča na razvoj celovite rešitve m-Študent. To je aplikacija spisana za uporabo v mobilnih napravah z operacijskim sistemom Android, skupaj z lastnim strežniškim delom, ki skrbi za posredovanje in hranjenje podatkov. V diplomskem delu je najprej predstavljen obstoječi sistem eŠtudent, ki je osnova za izgradnjo mobilne aplikacije. V poglavju 3, pa spoznamo orodja in tehnologije, ki so bile uporabljene pri razvoju celovite rešitve. Tako izvemo kaj je Android, kako je Android sestavljen, kaj je PHP. Poglavje 4 je sestavljeno iz dveh delov. V prvem delu je predstavljen razvoj odjemalca. Odjemalec je aplikacija, ki teče na mobilni napravi in komunicira s strežniškim



Slika 1.1: Delež operacijskih sistemov na mobilnem trgu v drugem četrtletju 2011.

delom. Najprej se seznanimo z zahtevami in delovanjem odjemalca, nato pa se podrobno posvetimo njegovi arhitekturi. V poglavju o arhitekturi se spoznamo z vsemi razredi in ostalimi datotekami, ki so bile razvite v sklopu diplomskega dela. V drugem delu pa je opisan strežniški del rešitve. Spoznamo kako se odjemalec lahko povezuje z njim in bolj podrobno o njegovih sestavnih delih.

Poglavje 2

Sistem e-Študent

2.1 Kaj je e-Študent

E-Študent je študijski informacijski sistem, ki uporabnikom omogoča oddaljen dostop do podatkov in storitev, ki so pomembne za izvajanje študijskega procesa. Sistem je Fakulteta za računalništvo in informatiko začela uporabljati v mesecu maju 2003. V sistem se lahko prijavijo samo registrirani uporabniki. E-študent uporabljajo študentje, profesorji, asistenti, študentska pisarna in ostalo vodstvo Fakultete za računalništvo in informatiko. Sistem e-Študent je prikazan na sliki 2.1.

Pred uvedbo sistema e-Študent se je uporabljajo sistem FNISID, ki je bil napisan v programskem jeziku Clipper. Slabost sistema je bila možna prijava le na dveh računalnikih, ki sta se nahajala na fakulteti. Pred tem sistemom pa je vse potekalo prek papirnatih prijavnic.

2.2 Uporaba e-Študenta

Ko se študent vpiše na Fakulteto za računalništvo in informatiko, dobi tudi dostop do e-Študenta. Prijavi se lahko z vpisno številko in geslom, lahko pa tudi z lastnim digitalnim potrdilom SIGEN-CA ali digitalnim potrdilom certifikatne agencije @friCA. Za uporabno sistema potrebujemo le sodobni spletni brskalnik z delujočo povezavo v internet oziroma intranet fakultete. Z uvedbo e-Študenta se je razbremenilo študente, saj lahko prek sistema opravijo večino birokratskih zadev, katere so prej morali opravljati v študentski pisarni fakultete. Sistem študentom omogoča prijave na izpit, pregled ocen in prejetih obvestil in naročanje potrdil o pravljenih izpitih ter vpisu. Prijavijo se lahko

Slika 2.1: Vstopna stran sistema e-Študent

tudi na elektronska sporočila, ki jih pošiljajo profesorji za obveščanje o izpitih in predmetih. Pred začetkom vsakega šolskega leta, pa si lahko s pomočjo e-Študenta tudi natisnejo vpisni list.

Sistem uporabljajo tudi pedagoški delavci Fakultete za računalništvo in informatiko. Pedagoški delavec je lahko profesor ali asistent. Za vsakega je v sistemu opredeljena svoja vloga, vendar bistvenih razlik med njima ni. V sistem se lahko prijavijo izključno samo z lastnim digitalnim potrdilom SIGEN-CA ali digitalnim potrdilom certifikatne agencije @friCA. Prijava z uporabniškim imenom in geslom ni omogočena. Profesorji lahko v e-Študentu vnašajo obvestila študentom, pregledujejo statistične podatke za ocene pri svojih predmetih, pregledujejo rezultate kvizov na Moodle sistemu (https://ucilnica.fri.uni-lj.si/) itd.

Poglavje 3

Uporabljena orodja in tehnologije pri razvoju aplikacije

3.1 Android

3.1.1 Splošno o Androidu

Android je kopica programov za mobilne naprave, ki vključuje operacijski sistem, aplikacije in programsko opremo, ki ta dva nivoja povezuje. Zgrajen je na jedru Linuxa. Android ima veliko skupnost razvijalcev, ki pišejo aplikacije za razširitev funkcionalnosti naprav. Trenutno je na voljo preko 200.000 aplikacij za mobilne naprave z Android platformo.

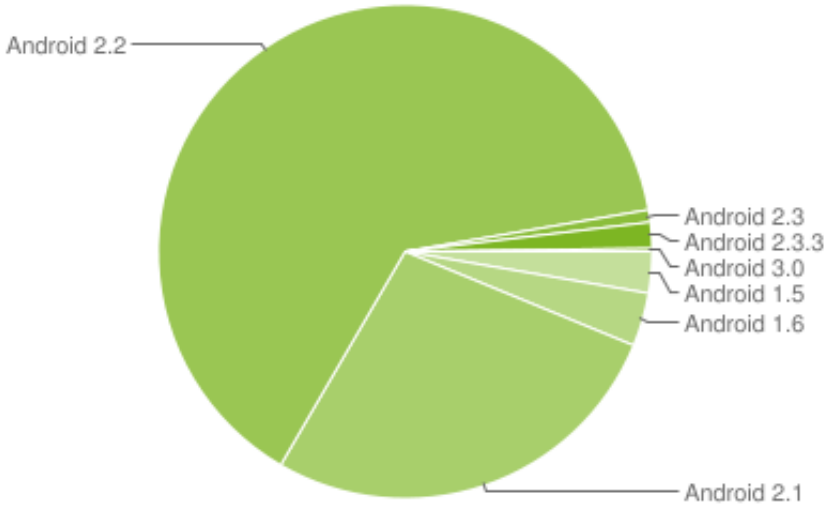
Podjetje Android Inc. je bilo ustanovljeno oktobra 2003 z namenom, da bi razvili mobilne naprave, ki se bolje zavedajo uporabnikove lokacije in njegovih osebnih želja. Google je Android Inc. kupil avgusta 2005. 5. novembra 2007 pa je bil ustanovljen konzorcij Open Handset Alliance. Namen konzorcija je razvoj odprtih standardov za mobilne naprave. Isti dan so predstavili tudi svoj prvi izdelek, Android.

Android je na voljo pod odprto kodno licenco Apache License od 21. oktobra 2008. Google je objavil celotno izvorno kodo, vključno z mrežno in telefonsko kopico. Tudi seznam problemov je na voljo vsem v pogled in možnost komentiranja. Čeprav je Android odprto koden, pa ga proizvajalci mobilnih naprav ne smejo uporabljati, če naprava ni v skladu z Googlovim dokumentom Compatibility Definition Document.

Android je od prvotne izdaje imel že kar nekaj posodobitev. Te posodobitve so namenjene tako odpravljanju hroščev, kot dodajanju novih funkcionalnosti. Do sedaj so bile izdane oziroma najavljene naslednje verzije Androida:

- Beta - izdana 5. novembra 2007;
- 1.0 - izdana 23. septembra 2008;
- 1.1 - izdana 9. februarja 2009;
- 1.5 Cupcake - izdana 30. aprila 2009;
- 1.6 Donut - izdana 15. septembra 2009;
- 2.0/2.1 Eclair - izdana 26. oktobra 2009;
- 2.2 Froyo - izdana 20. maja 2010;
- 2.3 Gingerbread - izdana 6. decembra 2010;
- 3.0 Honeycomb (samo za tablične računalnike) - izdana 24. februarja 2011;
- 3.1 Honeycomb (samo za tablične računalnike) - izdana 10. maja 2011;
- 3.2/4.0 Ice Cream Sandwich - najavljena na konferenci Google I/O 2011.

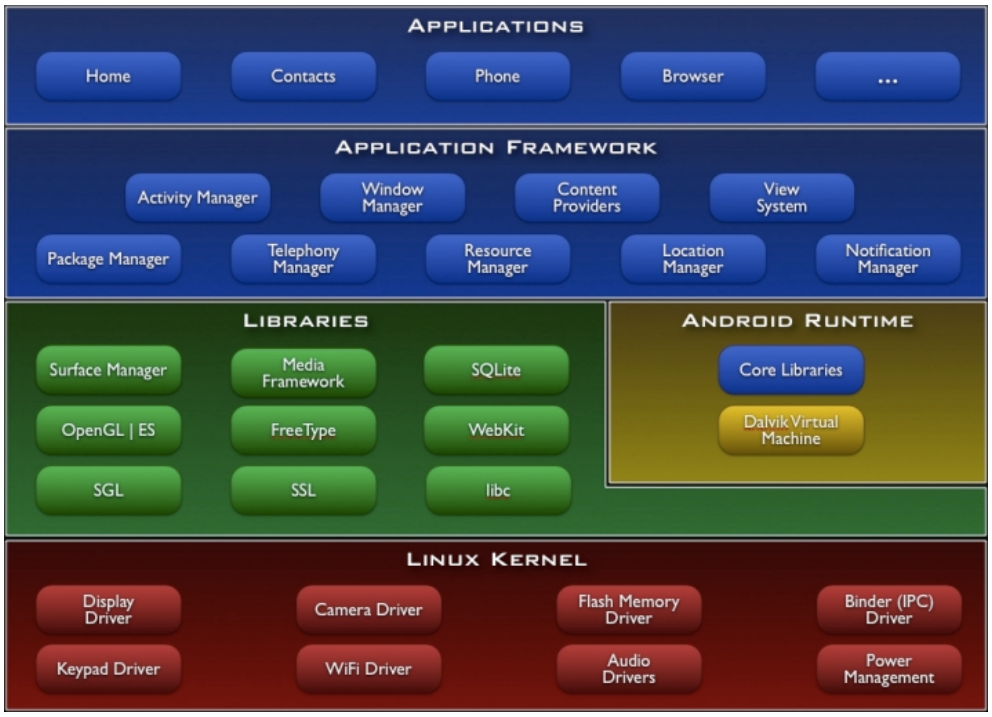
Trenutno najbolj uporabljena verzija je 2.2.x Froyo in sicer s 65.9% deležem na trgu. Na sliki 3.1 lahko vidimo še ostale deleže verzij.



Slika 3.1: Podatki so bili zbrani v roku dveh tednov s koncem 2. maja 2011.

3.1.2 Arhitektura

Diagram na sliki 3.2 prikazuje glavne komponente operacijskega sistema Android. Nekaj najpomembnejših komponent je opisanih v nadaljevanju.



Slika 3.2: Arhitekutura Androida

Aplikacije (*angl. applications)*

Android že vsebuje nekaj najpomembnejših aplikacij kot so odjemalec e-pošte, program za SMS, koledar, zemljevidi, brskalnik, kontakte in drugo. Ostale aplikacije pa si lahko uporabnik namesti sam. Vse vrste aplikacij se nahajajo na aplikacijskem nivoju in uporabljajo iste knjižnice API.

Aplikacijsko ogrodje (*angl. application framework)*

Z aplikacijskim ogrodjem razvijalci dobijo dostop do razredov s katerimi lahko razvijejo aplikacijo za Android. Omogoča tudi abstraktni dostop do strojne opreme in razvoja grafičnega vmesnika.

Knjižice (*angl. libraries)*

Knjižice so spisane v programskem jeziku C/C++. Skrbijo za povezavo s komponentami sistema Android. Nekaj takih knjižic je naštetih spodaj:

- medijska knjižica, ki skrbi za predvajanje zvoka in video vsebin;
- grafične knjižice, ki vsebujejo SGL in OpenGL za 2D in 3D grafiko;
- SQLite za podporo podatkovnim bazam;
- SSL in WebKit za varno internetno izkušnjo.

Jedro Linuxa (*angl. Linux kernel)*

Samo jedro Linuxa vsebuje servise kot so gonilniki komponent, procesi, upravljanje s spominom, varnost, mreženje in upravljanje z energijo. Zgrajeni so na verziji Linuxa 2.6.

Arhitektura aplikacij Android (*angl. Android Application Architecture)*

Arhitektura Androida spodbuja koncept souporabe komponent, kar omogoča da lahko delimo aktivnosti, servise in podatke z drugimi aplikacijami z lastnimi varnostnimi ukrepi. Mehanizem, ki omogoča zamenjavo tovarniških aplikacij, dovoli razvijalcem izpostavitev delov lastne aplikacije drugim razvijalcem. S tem lahko razvijalci razvijejo lasten uporabniški vmesnik ali dodatke za že obstoječo aplikacijo.

Spodaj so na kratko opisani arhitekturni gradniki vseh aplikacij Android, ki sestavljajo ogrodje za razvoj lastne programske opreme za mobilne naprave z operacijskim sistemom Android:

- **Upravljavec aktivnosti** (*angl. Activity Manager)* ureja življenjski cikel aktivnosti;
- **Pogledi** (*angl. Views)* se uporabljajo za gradnjo uporabniškega vmesnika aktivnosti aplikacij;
- **Upravljavec obvestil** (*angl. Notification Manager)* omogoča dosledno in nevsiljivo obveščanje uporabnikov;
- **Ponudniki vsebin** (*angl. Content Providers)* delijo podatke z ostalimi aplikacijami;

- **Upravljavec z lokacijo** (*angl. Location Manager)* omogoča dostop do lokacijskih storitev;

- **Upravljavec virov** (*angl. Resource Manager)* podpira dostop do zunanjih virov, kot so slike ali nizi.

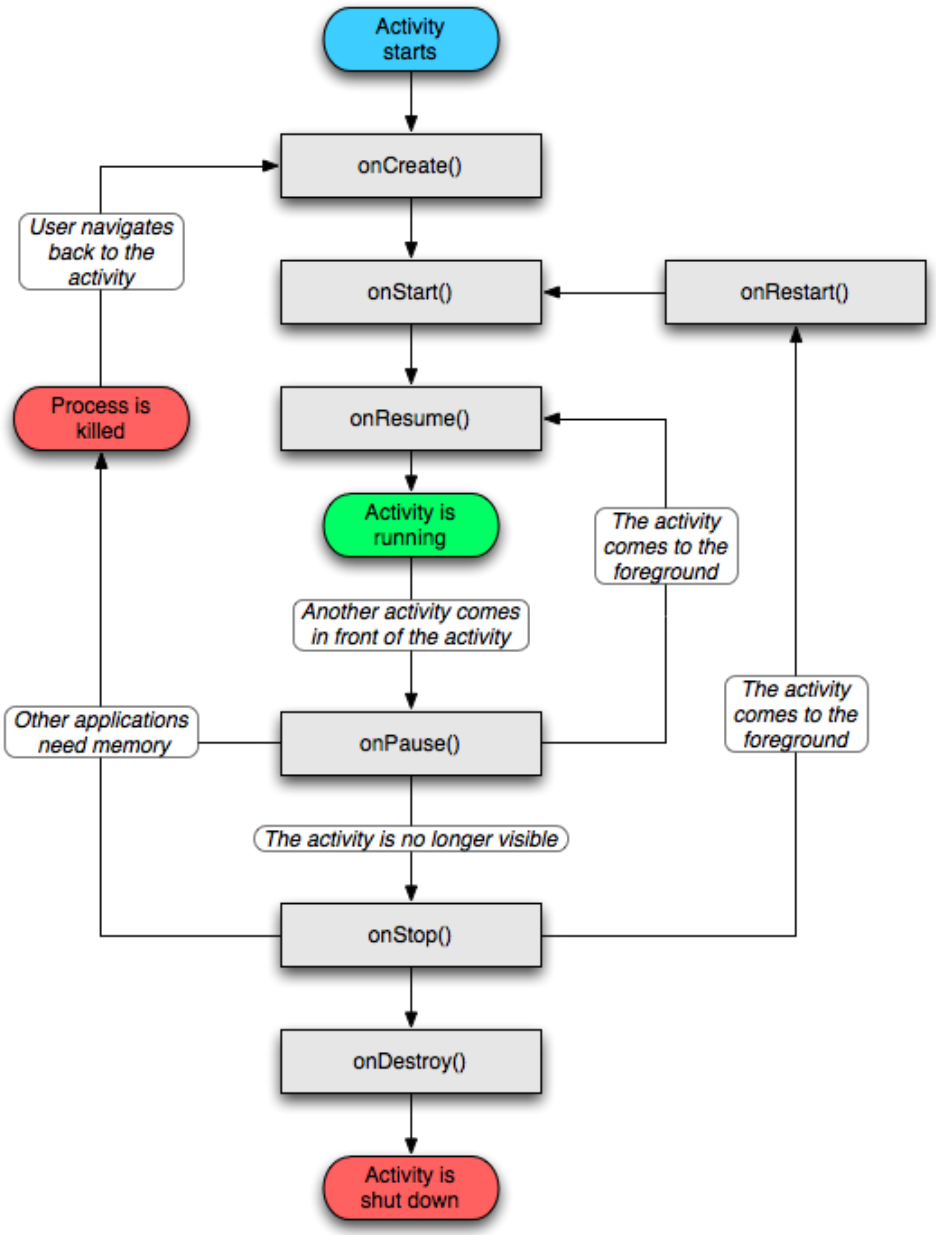
Življenjski cikel aplikacij

Življenjski cikel aplikacij upravlja izključno sistem in sicer glede na uporabnikove potrebe, proste vire in tako naprej. Na primer, uporabnik si lahko zaželi zagnati internetni brskalnik, vendar se na koncu sistem odloči ali bo zagnal aplikacijo ali ne. Sistem upravlja z aplikacijami glede na logične in opredeljene smernice. Če uporabnik upravlja z neko aplikacijo, ji sistem dodeli najvišjo prednost. Po drugi strani, če aplikacija ni vidna, se sistem lahko odloči, da jo zapre in tako sprosti vire. Cikel je prikazan na sliki 3.3.

Ko sistem zažene aktivnost, pokliče metodo `onCreate()`, nato mu sledi še `onStart()`. V primeru, da je bila aplikacija že ustavljena z `onStop()`, pa se `onCreate()` ne kliče. V trenutku, ko je poklicana metoda `onStart()`, aktivnost še ni vidna uporabniku. Takoj za to metodo, pa se požene `onResume()` in v tem trenutku lahko uporabnik upravlja z aktivnostjo. Ko se uporabnik odloči, da bo prešel v neko drugo aktivnost, se pokliče `onPause()`, ki mu sledi še `onStop()`. Od tukaj naprej se nato izvede ali `onRestart()` ali pa `onDestroy()`. Metoda `onRestart()`, se požene v primeru, ko uporabnik ponovno dostopa do aktivnosti. Če pa želi sistem sprostiti vire oziroma se v kodi kliče `finish()`, pa je poklicana metoda `onDestroy()`. Sistem pa lahko aktivnost uniči tudi brez klica teh dveh metod.

3.2 PHP

PHP je odprto kodni skriptni jezik, ki je bil razvit za gradnjo dinamičnih spletnih strani [6]. PHP je nastal leta 1995 s prvotnim imenom *Personal Home Page Tools (PHP Tools).* Rasmus Lerdorf ga je razvil, ker je potreboval statistiko dostopov do svoje spletne strani. Kasneje je funkcionalnosti razširil do te meje, da se je lahko jezik uporabljajo za gradnjo dinamičnih spletnih strani. Leta 1998 je bil izdan PHP 3 z novim imenom *PHP: Hypertext Preprocessor.* Z verzijo 5.0, ki je bila izdana leta 2005, pa je PHP postal objektno orientiran jezik.



Slika 3.3: Življenski cikel aktivnosti aplikacij

3.3 Orodja

3.3.1 Eclipse

Eclipse je več jezikovno orodje za razvoj programske opreme. Spisan je v programskem jeziku Java in izdan pod odprto kodno licenco. Ker ima dobro razvit sistem za vtičnike, se jih je sčasoma nabralo kar veliko število. Ti podpirajo razvoj opreme v različnih programskih jezikih, kot so Java, C, C++, Perl, PHP. Z vtičniki pa lahko dostopamo do oddaljenih podatkovnih baz, uporabljamo orodje za vodenje različic SVN, pišemo dokumente v LaTeXu, ...

3.3.2 Apache Subversion (SVN)

SVN je sistem za nadzor verzij. Razvojniki ga uporabljajo za vzdrževanje trenutne in zgodovinskih verzij datotek, kot so izvorne kode, spletne strani in razni dokumenti. Izdan je pod odprto kodno licenco in je brezplačen za uporabo.

3.3.3 Zend Framework

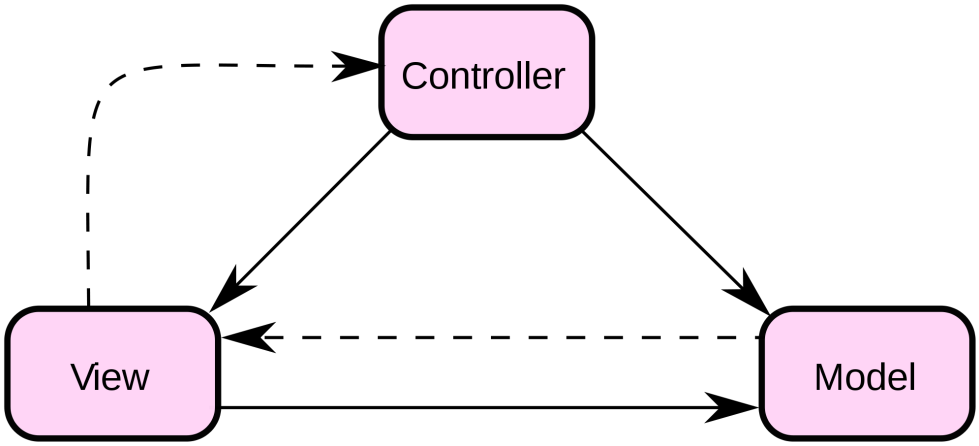
Zend Framework je odprto kodno, objektno orientirano ogrodje za gradnjo spletnih aplikacij. Implementirano je v PHP 5 in izdano pod licenco *New BSD Licence*.

Zend Framework ni tipično programsko ogrodje, saj uporaba vseh delov, ni nujna. Bolj je podoben zbirki knjižic, ki se jih da povezati v izredno močno ogrodje za izdelavo aplikacij. Vse knjižice so testirane z uporabo PHPUnit testov, kar pomeni izredno visoko kvaliteto kode.

Z Zend Framewokom se na izredno enostaven način lahko implementira MVC (Model-View-Controller). MVC je arhitekturni vzorec, ki se uporablja v razvoju programskih sistemov[7]. Vzorec ločuje uporabniški vmesnik od logike aplikacije in modelov ter s tem omogoča neodvisen razvoj in testiranje vsakega dela posebej. Omenjeni vzorec je prikazan na sliki 3.4.

3.3.4 Nginx

Nginx je hiter in lahek spletni strežnik, ki zna upravljati tudi z elektronsko pošto. Izdan je pod odprto kodno licenco. Nginx je znan po hitrem serviranju statičnih vsebin. Pri tem je njegova obremenitev na sistem minimalna. Za serviranje zahtev uporablja asinhron pristop. S tem se pod veliko obremenitvijo obnaša veliko bolj predvidljivo kot strežniki, ki uporabljajo nitni ali



Slika 3.4: Diagram vzorca MVC

procesni pristop. Ima tudi SSL podporo, za varno komunikacijo prek interneta. Vsem odgovorom pa lahko zmanjša velikost z uporabo *gzip* kompresije.

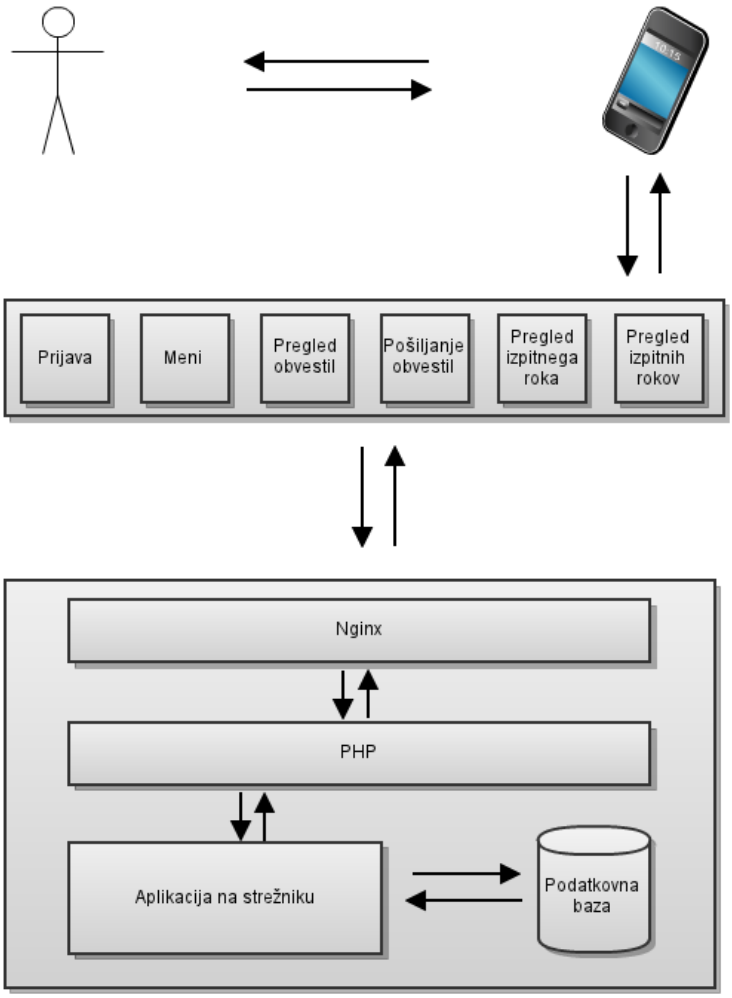
Poglavje 4

Razvoj aplikacije

4.1 Arhitektura

Rešitev m-Študent je sestavljena iz odjemalca in strežnika. Odjemalec je aplikacija za mobilne naprave z operacijskim sistemom Android, strežnik pa je skupek spletnega strežnika, spletne aplikacije in podatkovne baze. Odjemalec in strežnik komunicirate preko varne HTTPS povezave v formatu JSON.

Uporabnik komunicira z mobilno napravo, ta pa zahteve posreduje odjemalcu. Vsako zahtevo uporabnika razreši odjemalec. Če odjemalec nima dovolj podatkov za razrešitev zahteve, sproži poizvedbo na strežnik. Strežnik poizvedbo prebere in se primerno odzove. Odgovor iz strežnika je lahko uspešen ali neuspešen. V primeru uspešnega odgovora, ga odjemalec prebere in odgovori na uporabniku primeren način. Arhitektura in potek komunikacije sta predstavljena na diagramu 4.1.



Slika 4.1: Arhitektura rešitve m-Študent

4.2 Odjemalec

4.2.1 Delovanje

Odjemalec za delovanje potrebuje mobilno napravo z operacijskim sistemom Android. Verzija Androida mora biti vsaj 2.2. Deluje na različnih velikostih

ekranov. Zaradi povezovanja na strežnik, nujno potrebuje delujočo internetno povezavo.

Ker je verzij operacijskega sistema Androida več, smo se morali pri izdelavi odjemalca odločiti od katere verzije naprej bo aplikacija delovala. Za verzijo 2.2 smo se odločili, ker je trenutno najbolj uporabljena. Na sliki 3.1 vidimo, da ima verzija 2.2 65.9% delež na trgu. Večina mobilnih naprav, ki jih je na trgu trenutno mogoče kupiti so opremljeni z vsaj to verzijo operacijskega sistema. Zaradi same zgradbe Androida pa aplikacija brez težav deluje tudi na napravah z novejšo verzijo.

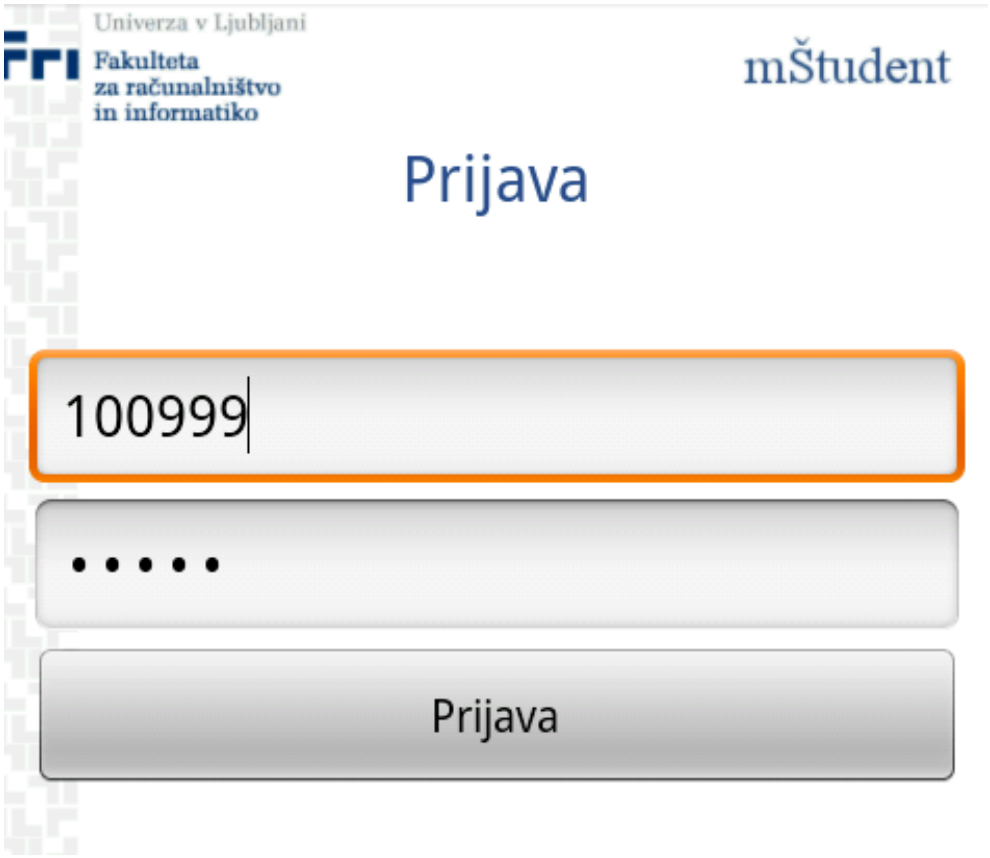
4.2.2 Uporaba

UML diagram uporabe odjemalca lahko vidimo na sliki 4.3. Uporabnik se na vstopni aktivnosti, ki je vidna na sliki 4.2, najprej prijavi z uporabniškim imenom in geslom. Aplikacija nato preveri veljavnost vpisanih podatkov. V primeru kakršnekoli napake, uporabnika obvesti o tem. Če je uporabnik vpisal pravilne podatke, ga premakne v glavni meni aplikacije.

V glavnem meniju aplikacija najprej pozdravi uporabnika z imenom in priimkom. Uporabnik se lahko nato odloči ali bo upravljal z obvestili ali z izpitnimi roki. Svojo odločitev izbere s klikom na enega izmed gumbov. Aplikacija ga nato premakne v primerno aktivnost.

V primeru, ko se uporabnik odloči, da bo upravljal z obvestili, mu aplikacija v aktivnosti prikaže seznam do sedaj poslanih obvestil. Seznam obvestil vsebuje čas in datum, ko je bilo obvestilo poslano in naslov obvestila. S klikom na eno izmed obvestil se uporabniku prikaže celotno obvestilo. Uporabnik novo obvestilo doda s klikom na gumb *Dodaj novo obvestilo*. Aplikacija nato uporabniku ponudi vrsto obvestil, ki jih lahko pošlje. Obvestilo lahko pošlje vsem udeležencem predmeta, katerega nosilec je ali pa vsem udeležencem nekega pisnega izpita. Glede na izbiro, se v naslednji aktivnosti prikaže seznam vseh predmetov oziroma vseh izpitnih rokov. S klikom na enega izmed gumbov aplikacija uporabnika premakne v naslednjo aktivnost. Tu uporabnik napiše naslov in besedilo obvestila. S klikom na gumb *Pošlji*, uporabnik sporoči aplikaciji naj začne z razpošiljanjem obvestil.

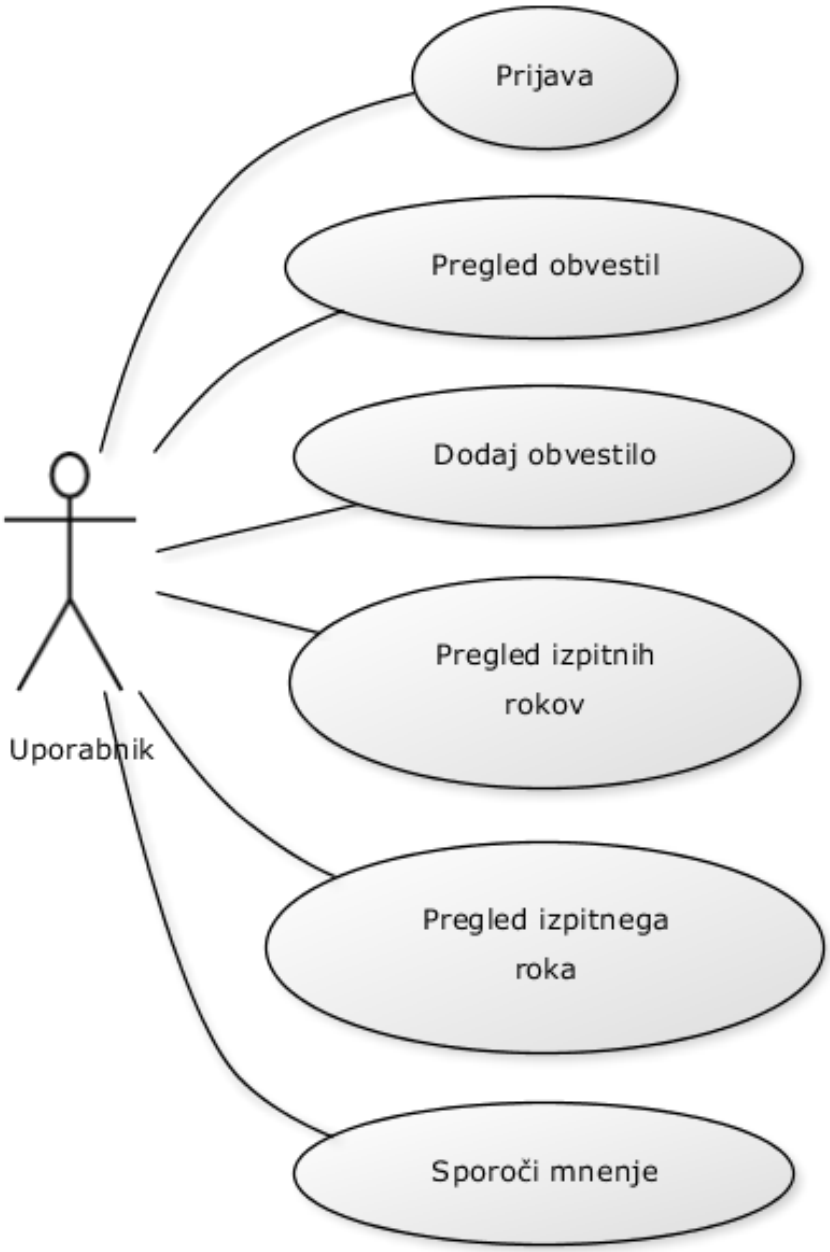
Če se uporabnik v glavnem meniju odloči, da bo upravljal z izpitnimi roki, mu aplikacija prikaže seznam vseh izpitnih rokov. Seznam vsebuje ime predmeta in datum izpitnega roka. Z izbiro enega izmed elementov v seznamu, aplikacija premakne uporabnika v aktivnost, ki vsebuje podrobnosti izpitnega roka. Uporabnik lahko vidi datum, čas in lokacijo ter seznam študentov, ki so se prijavili. V primeru, da je izpitni rok že zaključen in so vnesene vse



Slika 4.2: Vstopna aktivnost aplikacije m-Študent

ocene, pa se zraven vsakega študenta prikaže še njegova ocena, prav tako pa tudi povprečna ocena tega izpitnega roka.

Ker želimo od uporabnikov dobiti čim več predlogov za izboljšavo, mnenj in poročil o morebitnih težavah, smo naredili še aktivnost, ki se ukvarja s tem. Na vsakem koraku lahko uporabnik pritisne na fizični gumb za nastavitve. Klik prikliče dodatni meni. V tem meniju lahko uporabnik klikne na *Sporoči nam svoje mnenje*. Aplikacija ga nato premakne na aktivnost, kjer lahko v besedilno polje vpiše svoje predloge ali kritike. Z gumbom *Pošlji* pa nam svoje sporočilo tudi posreduje.



Slika 4.3: UML diagram uporabe odjemalca

4.2.3 Arhitektura

Odjemalec je zgrajen iz naslednjih datotek, ki vsebujejo istoimenske razrede:

- mStudentActivity.java;
- ActivityExam.java;
- ActivityExamList.java;
- ActivityLogin.java;
- ActivityMenu.java;
- ActivityMessageForm.java;
- ActivityMessageFormCourses.java;
- ActivityMessageFormExams.java;
- ActivityMessages.java;
- ActivityMessageListRecipients.java;
- Attendee.java;
- Course.java;
- Exam.java;
- HttpParams.java;
- mObject.java;
- Msg.java;
- mStudent.java;
- mStudentHttpsClient.java;
- Service.java.

Za izgled odjemalca pa skrbijo:

- exam.xml;

- examlist.xml;
- login.xml;
- menu.xml;
- message_form.xml;
- message_list.xml;
- messages.xml;
- tellus.xml.

Najpomembnejše datoteke in razredi so opisani v nadaljevanju.

mStudentActivity

Datoteka *mStudentActivity.java* vsebuje abstraktni razred **mStudentActivity**. Vsi razredi, ki opisujejo aktivnosti v aplikaciji, dedujejo ta razred. Vse aktivnosti v aplikaciji imajo namreč nekaj skupnih spremenljivk in metod. Najpomembnejše metoda je `onCreate(Bundle savedInstanceState)`, ki poskrbi za pravilno inicializacijo spremenljivk ter nastavi nekaj lastnosti aktivnostim. V razredu sta tudi metodi `onCreateOptionsMenu(Menu menu)` in `onOptionsItemSelected(Men` `item)`, ki skrbita za prikazovanje dodatnega menija omenjenega v poglavju 4.2.2.

ActivityLogin

Razred *ActivityLogin* skrbi za prvo aktivnost s katero se sreča uporabnik. Sestavljen je iz več metod. Najpomembnejše pa so `onCreate(Bundle s)`, `onClick(View v)` in `run()`.

V prvi metodi se, kot v vseh ostalih aktivnostih aplikacije, najprej nastavi uporabniški vmesnik s klicem funkcije `setContentView(R.layout.login)`. V naslednjih korakih metoda preveri, če je uporabniško ime in geslo že shranjeno v pomnilniku naprave. V primeru da je, se ti dve vrednosti avtomatsko nastavita v polje za vnos, da ju uporabniku ni potrebno vsakič vnašati. Metoda tudi preveri, če je naprava povezana v internetno omrežje, sicer uporabnika obvesti, da brez povezave ne more nadaljevati.

Metoda `onClick(View v)` lovi klik na gumb *Prijava* (viden na sliki 4.2). Najprej preveri, če sta polji za vnos uporabniškega imena in gesla izpolnjeni.

Nato zažene dialog, ki uporabnika obvesti, da se bo nekaj dalj časa dogajalo. Nato pa zažene nit s klicem `thread.start()`.

Ker v aktivnosti uporabljamo večnitnost, moramo imeti implementirano metodo `run()`. Metoda ne počne nič drugega kot da pokliče servis za avtentikacijo in rezultat posreduje rokovalcu `handler`.

Rokovalec skrbi za komunikacijo med aktivnostjo in nitjo. Ko nit pokliče rokovalca, ta najprej uniči dialog, nato pa se glede na prejet rezultat odloča o nadaljnjem postopku. V primeru, da je bila avtentikacija uspešna, shrani vnosna polja v pomnilnik naprave in premakne uporabnika v aktivnost *ActivityMenu*. Če avtentikacija ni uspela, uporabnika obvesti o napaki. Izvorna koda rokovalca je vidna na sliki 4.4.

```
private Handler handler = new Handler() {
  public void handleMessage(Message msg) {
    dialog.dismiss();

    if (msg.what == 0) {
      Toast t = Toast.makeText(
        getApplicationContext(),
        "Napacni podatki. Avtentikacija ni uspela.",
        Toast.LENGTH_SHORT
      );
      t.show();
    } else {
      SharedPreferences.Editor editor =
        preferences.edit();
      editor.putString(
        "username",
        username.getText().toString()
      );
      editor.putString(
        "password",
        password.getText().toString()
      );
      editor.commit();

      Intent menu = new Intent(
        getApplicationContext(),
        ActivityMenu.class
      );

      startActivityForResult(menu, 0);
    }
  }
};
```

Slika 4.4: Izvorna koda rokovalca v aktivnosti *ActivityLogin*.

ActivityExamList

Uporabnik začne uporabljati razred *ActivityExamList*, ko iz glavnega menija preide na seznam vseh izpitnih rokov. Sestavljen je iz dveh metod in dveh podrazredov. V metodi `onCreate(Bundle s)` se nastavi uporabniški vmesnik, takoj za tem pa prikaže dialog napredka in zažene nit za posodobitev podatkov.

Delovanje niti je napisano v metodi `run()`. Tukaj se kliče servis za pridobitev seznama objektov izpitnih rokov. Rezultat shrani in pokliče rokovalca.

Rokovalec gre čez seznam izpitnih rokov in za vsakega ustvari gumb z napisom imena izpita in datumom izpitnega roka. Vsakemu gumbu doda tudi `OnClickListener listener`, ki posluša klike na gumb. Gumb doda v uporabniški vmesnik s klicem `ll.addView(b)`, kjer je `ll LinearLayout` in `b` ustvarjen gumb. Po koncu dodajanja gumbov, skrije še dialog napredka, ustvarjenega v metodi `onCreate(Bundle s)`. Razred `OnClickListener listener`, posluša klike na gumb. Ob kliku na gumb, se v razred *mStudent* shrani izbran izpitni rok in uporabnika premakne v aktivnost *ActivityExam*, ki je opisana v nadaljevanju.

ActivityExam

Aktivnost *ActivityExam*, skrbi za prikaz podrobnosti nekega izpitnega roka. Iz razreda *mStudent* prebere shranjen izpitni rok in ga shrani v lokalno spremenljivko `Exam exam`. Iz objekta `exam` lahko nato prebere vse potrebne informacije, ki jih aktivnost prikaže uporabniku. Informacije, ki jih prikaže uporabniku so napisane v poglavju 4.2.2. Izgled aktivnosti je opisan v datoteki *exam.xml* in prikazan na sliki 4.5.

ActivityMessages

Aktivnost, ki upravlja prikaz zadnjih nekaj poslanih obvestil, se imenuje *ActivityMessages*. Aktivnost v metodi `onCreate(Bundle s)` nastavi uporabniški vmesnik, kazalec na seznam obvestil `LinearLayout linearLayout` in gumbu za dodajanje obvestil dodeli poslušalca klikov.

Ker lahko uporabnik do aktivnosti dostopa tako iz glavnega menija, kot z gumbom nazaj iz aktivnosti, ki sledijo tej aktivnosti, smo morali implementirati tudi metodo `onResume()`, ki se kliče vsakič ko aktivnost postane vidna uporabniku. V tej metodi se nastavi in zažene dialog napredka in nit za posodobitev podatkov. Nit pokliče servis za vračanje obvestil in jih posreduje rokovalcu.

Rokovalec `handler` zna ukrepati v dveh različnih situacijah. V primeru, da ni nobenega aktualnega obvestila, uporabnika o tem obvesti s kratkim sporočilom. Če servis vrne nova obvestila pa počisti trenutni seznam in ga posodobi z novimi obvestili. Vsako obvestilo v seznamu prikazuje naslov in datum s časom, ko je bilo sporočilo poslano. Ko uporabnik klikne na eno izmed obvestil, to prestreže poslušalec in prikaže celotno besedilo obvestila.

Aktivnost prikaže tudi gumb za dodajanje novih obvestil. Poslušalec klikov prestreže klik in izvede se koda napisana v `onClick(View v)`. V metodi se zgradi dialog za izbiro elementov. V našem primeru so ti elementi vrsta obvestil, ki jih uporabnik lahko pošlje. Dialog se zgradi s pomočjo graditelja `Builder builder`, ki je sestavni del operacijskega sistema Android. Metoda graditelju nastavi naslov, elemente in poslušalca klikov na element. Določi mu tudi posebno lastnost `cancelable`, ki omogoča preklic dialoga s klikom na fizični gumb nazaj.

ActivityMessageListRecipients

Do aktivnosti *ActivityMessageListRecipients* uporabnik pride, ko želi poslati novo obvestilo. V metodi `onCreate(Bundle s)` se glede na vrsto obvestila, ki ga je uporabnik izbral izpiše sporočilo o tem. Podatek o vrsti izpita preberemo s pomočjo metode `this.getIntent().getExtras().getInt("msgType")`, ki nam vrne številčno vrednost. Vrednosti so naslednje:

- 0 - obvestila, ki se pošiljajo vsem udeležencem nekega predmeta;
- 1 - obvestila, ki se jih pošilja prijavljenim na izpit.

Vrednost zapišemo v lastnost razreda `private int msgType`. Nato se pokliče nit za posodobitev podatkov in prikaže dialog napredka.

V niti se glede na številčno vrednost vrste obvestila, zapisane v `private int msgType`, kličejo primerne storitve v servisu. Vrsta *predmet nosilca* kliče servisno storitev `service.getCourses()`, medtem ko vrsta *pisni izpit* kliče `service.getExams()`. Več o servisnih storitvah je napisano v poglavju 4.2.3. Po koncu klicev servisnih storitev se podatke posreduje rokovalcu, ki je podoben rokovalcem iz drugih aktivnosti.

ActivityMessageForm

ActivityMessageForm je abstraktni razred, ki ga uporabljata razreda *ActivityMessageFormCourses* in *ActivityMessageFormExams*. Ker sta oba omenjena

razreda izredno podobna, je bilo smiselno vpeljati abstrakcijo. Razreda se razlikujeta le v metodi `run()` in `getWelcomeText()`.

Aktivnost *ActivityMessageForm* vpeljuje abstraktno metodo `getWelcomeText()`. Metoda se kliče med ustvarjanjem aktivnosti. Uporablja se za prikaz vrste obvestila, ki ga želi uporabnik poslati.

Metoda `send()` preveri polja, ki jih je uporabnik izpolnil. V primeru, da je eno izmed polj prazno, uporabnika obvesti o tem. Če je s polji vse v redu, prikaže dialog napredka in požene nit.

Service

Service je razred, ki implementira vzorec samskosti (*angl. singelton*). Vzorec je pomemben, ker omogoča dostop do iste instance razreda iz vseh delov aplikacije. Da smo to omogočili smo morali v razred implementirati lastnost `Service self`, ki drži kazalec do instance razreda. Zraven tega pa smo morali implementirati še tri nove metode:

- `private Service()` prepiše konstruktor razreda in s tem onemogoči zunanji dostop;
- `public static synchronized Service getInstance()` vrača aktivno instanco razreda;
- `public Object clone() throws CloneNotSupportedException` onemogoči kloniranje razreda.

Razred skrbi za pretvarjanje strežniških podatkov v podatke primerne za posredovanje aktivnostim aplikacije. Metoda `connect()` sprejme tri argumente:

- `String func` je ime funkcije, ki jo kličemo na strežniku;
- `Vector<HttpParams> params` je seznam parametrov, ki jih posredujemo strežniku;
- `boolean usePasskey` sporočimo ali je za funkcijo potreben dostopni ključ.

Njena naloga je, da s pomočjo razreda *mStudentHttpsClient*, opisanega v poglavju 4.2.3, komunicira s strežnikom in rezultat v formatu JSON pretvori v primerne objekte. Metoda najprej iz seznama parametrov sestavi URI na katerega se poveže s pomočjo prej omenjenega razreda. Po komunikaciji s strežnikom, preveri vrnjeno glavo (*angl. header*) odziva. Če je v glavi koda odziva 200, pomeni

da je bil zahtevek pravilen. Tako lahko nadaljuje s procesiranjem rezultata. Rezultat, ki je v formatu JSON, pretvori v *JSONObject* in ga shrani v lastnost `JSONObject lastResults`. Če je celoten postopek uspešen, vrne rezultat `true`.

Servisna storitev *auth*, prejme dva argumenta in sicer `String username` in `String password`, kjer je prvi uporabniško ime uporabnika, drugi pa njegovo geslo. Iz argumentov sestavi seznam parametrov in skupaj z imenom funkcije kliče prej opisano metodo *connect*. Iz lastnosti *lastResults* nato prebere vrnjen dostopni ključ in naziv uporabnika. V primeru da je bil celoten postopek uspešen vrne rezultat `true`.

Razred omogoča klice še na naslednje servisne storitve:

- `Vector<Exam> getExams()` - vrača seznam izpitnih rokov;
- `Vector<Course> getCourses()` - vrača seznam predmetov;
- `boolean sendMsg(String t, String b, int id, String type)` - pošlje novo obvestilo z naslovom *t*, besedilom *b*, identifikacijsko številko predmeta oziroma izpitnega roka *id* in vrsto obvestila *type*;
- `Vector<Msg> getMessages()` - vrača seznam vseh obvestil;
- `boolean sendComment(String text, String referrer)` - pošlje komentar z besedilom *text* in imenom aktivnosti *referrer*.

mStudentHttpsClient

Razred *mStudentHttpsClient* je namenjen povezovanju prek protokolov *http* in varnega protokola *https*. Protokol *http* uporablja vrata 80, medtem ko *https* uporablja vrata 443. Deduje elemente objekta *DefaultHttpClient*. Za varno povezovanje uporablja samo podpisani certifikat *BKS (BouncyCastle)*, ki je priložen aplikaciji. Do certifikata dostopa prek lastnosti `Context context` s pomočjo metode `getResources().openRawResource(R.raw.keystoreprod)`, kjer je *keystoreprod* ime certifikata shranjenega v mapi *raw*.

Slika 4.5: Slika prikazuje aktivnost ActivityExam. Aktivnost prikazuje vse podrobnosti izpitnega roka, ki zanimajo končnega uporabnika.

4.3 Strežnik

Ker je povezovanje z bazo, izvajanje poizvedb in preurejanje podatkov izredno procesorsko zahtevno, smo morali razviti strežnik, čigar naloga je čim večja razbremenitev odjemalca. Odjemalec se prek varnega *HTTPS* protokola povezuje s strežnikom in si z njim izmenjuje podatke. Strežnik komunicira s podatkovno bazo in odgovarja na zahteve odjemalca. Podatke posreduje v formatu *JSON*, ki je zaradi enostavnosti najbolj primeren za mobilne naprave.

4.3.1 Delovanje

Strežnik je aplikacija napisana v programskem jeziku PHP. Za delovanje potrebuje fizični ali virtualni strežnik z navedeno konfiguracijo:

- operacijski sistem **Ubuntu**, verzije 11.04;
- spletni strežnik **nginx**, verzije 1.05;
- jezik **PHP**, verzije 5.3.5;
- ogrodje **Zend Framework**;
- protokol **FastCGI/PHP-FPM** za povezavo spletnega strežnika in PHP;
- podatkovno bazo **MySQL**, verzije 5.5.54;
- sistem za nadzor verzij **Subversion**, verzije 1.6.12.

4.3.2 Uporaba

Odjemalec uporablja strežnik za pridobivanje podatkov iz podatkovne baze in pripravo v primeren format. Odjemalec za dostop do različnih podatkov, uporablja ustrezne metode. Metode so združene v treh modulih. Modul *Student*, uporabljajo odjemalci za študente, modul *Default* skrbi samo za procesiranje napak, do katerih mogoče pride med izvajanjem aplikacije, modul *Teacher* pa uporabljajo odjemalci namenjeni pedagoškim delavcem. Vsak modul pa je razdeljen še v primerne krmilnike. V nadaljevanju so opisane metode v modulu *Teacher*, katere smo razvili za uporabo v diplomskem delu.

Za dostop do metode moramo klicati URI, ki je naslednje oblike: **https://naslov-strežnika/modul/krmilnik/metoda/morebitni-dodatni-argumenti**. Morebitne dodatne argumente se podaja kot **ime/vrednost/**. Če metoda ni podana, se uporabi metoda *Index*. V primeru, da manjka modul ali krmilnik, pa se proži

napaka. Vse metode, razen posebej omenjenih, morajo imeti kot dodaten argument veljaven ključ z imenom argumenta *passkey*.

Krmilnik Auth

Metoda Index Metoda sprejme dva argumenta in sicer *username* in *password*, kjer je *username* uporabniško ime uporabnika in *password* osebno geslo uporabnika. Če kateri argument manjka, aplikacija vrže napako. Ta metoda za delovanje ne potrebuje ključa.

Metoda preveri, če je kombinacija uporabniškega imena in gesla pravilna. V primeru pravilnosti vrne ključ *passkey*, ki je potreben za uporabo ostalih metod. V primeru nepravilnosti pa proži napako.

Krmilnik Comment

Metoda Send Metoda sprejme dva argumenta in sicer *text* in *referrer*, kjer je *text* besedilo komentarja in *referrer* ime aktivnosti iz katere je komentar poslan.

Metoda skrbi za shranjevanje komentarjev uporabnikov.

Krmilnik Courses

Metoda List Metoda vrne seznam vseh predmetov, katerih nosilec je prijavljen uporabnik. Za vsak predmet, vrne naslednje lastnosti:

- *title* - ime predmeta;
- *id* - unikatna številka predmeta.

Krmilnik Exams

Metoda List Metoda vrne seznam vseh izpitnih rokov, katerih nosilec je prijavljen uporabnik.

Za vsak izpitni rok vrne naslednje lastnosti:

- *title* - ime predmeta za katerega je izpitni rok;
- *id* - unikatna številka izpitnega roka;
- *avgGrade* - povprečna ocena izpitnega roka,v primeru da izpitni rok še nima povprečne ocene, vrne vrednost 0.0;

- *location* - lokacija izpitega roka;
- *datetime* - datum in čas izpitnega roka;
- *attendees* - seznam prijavljenih študentov na izpitni rok.

Za vsakega prijavljenega študenta pa vrne še naslednje lastnosti:

- *name* - ime in priimek študenta;
- *id* - vpisna številka študenta;
- *grade* - ocena pisnega dela, če ni na voljo je vrednost 0;
- *gradeOral* - ocena ustnega dela, če ni na voljo je vrednost 0.

Krmilnik Messages

Krmilnik skrbi za upravljanje s sporočili pedagoškega delavca študentom.

Metoda List Metoda vrača seznam vseh obvestil, ki jih je uporabnik že poslal. Za vsako sporočilo vrne tri lastnosti:

- *title* - naslov sporočila;
- *body* - besedilo sporočila;
- *datetime* - datum in čas pošiljanja.

Metoda Send Metoda sprejme argumente, preveri njihovo pravilnost ter pošlje obvestilo.

Sprejme naslednje argumente:

- *title* - naslov sporočila;
- *body* - besedilo sporočila;
- *type* - vrsta sporočila, lahko je *courses* (za prijavljene k predmetu) ali pa *exams* (za prijavljene na izpitni rok);
- *id* - unikatna številka predmeta oziroma izpitnega roka.

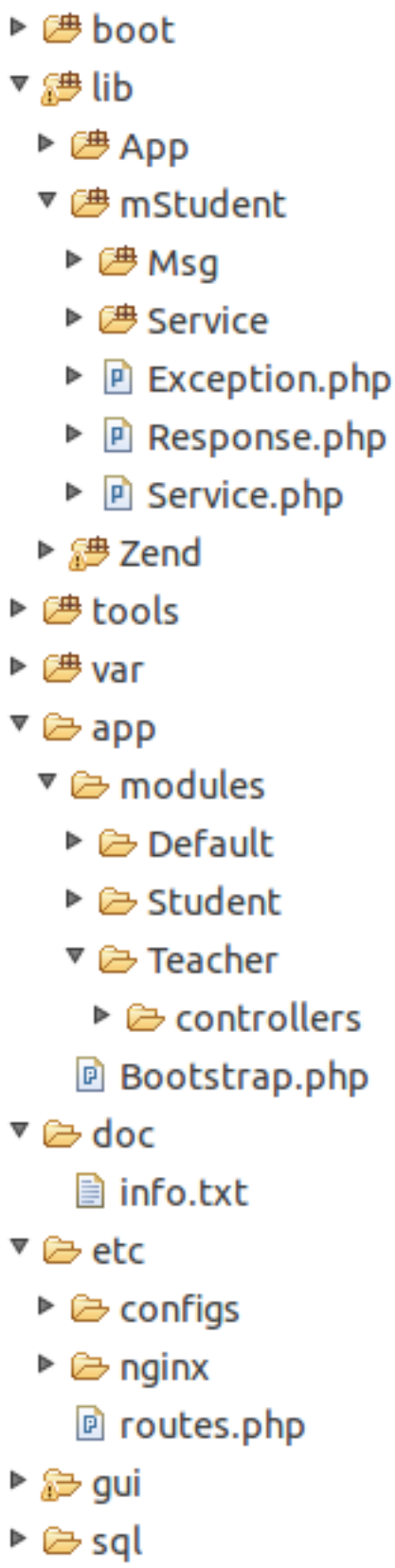
4.3.3 Arhitektura

Datotečna struktura

Strežniška aplikacija implementira vzorec MVC z uporabo ogrodja Zend Framwork. Ker ima omenjeno ogrodje svoj način strukturiranja, smo morali strukturo aplikacije temu prilagoditi. Aplikacija je razdeljena v več map, kjer vsaka mapa vsebuje nove pod mape in datoteke. Datotečna struktura je prikazana na sliki 4.6.

Glavne mape aplikacije so naslednje:

- *app* - vsebuje jedro aplikacije, kot so moduli, krmilniki in metode;
- *boot* - vsebuje datoteko *index.php*, kjer je napisana procedura za zagon aplikacije;
- *doc* - vsebuje razne dokumente za pomoč pri razumevanju in razvijanju aplikacije;
- *etc* - vsebuje razne nastavitvene datoteke za povezovanje s podatkovno bazo, uporabo spletne strežnika *nginx*, certifikate ipd;
- *gui* - vsebuje datoteke, ki so javno dostopne zunanjosti;
- *lib* - vsebuje tako zunanje knjižice, kot knjižice, ki so bile spisane med razvojem aplikacije;
- *sql* - vsebuje poizvedbe sql s katerimi lahko postavimo strukturo baze;
- *tools* - vsebuje razna orodja za pomoč med razvojem;
- *var* - mapa, kjer se shranjujejo datoteke proizvedene med življenjskim ciklom aplikacije.



Slika 4.6: Slika prikazuje strukturo strežnika po mapah.

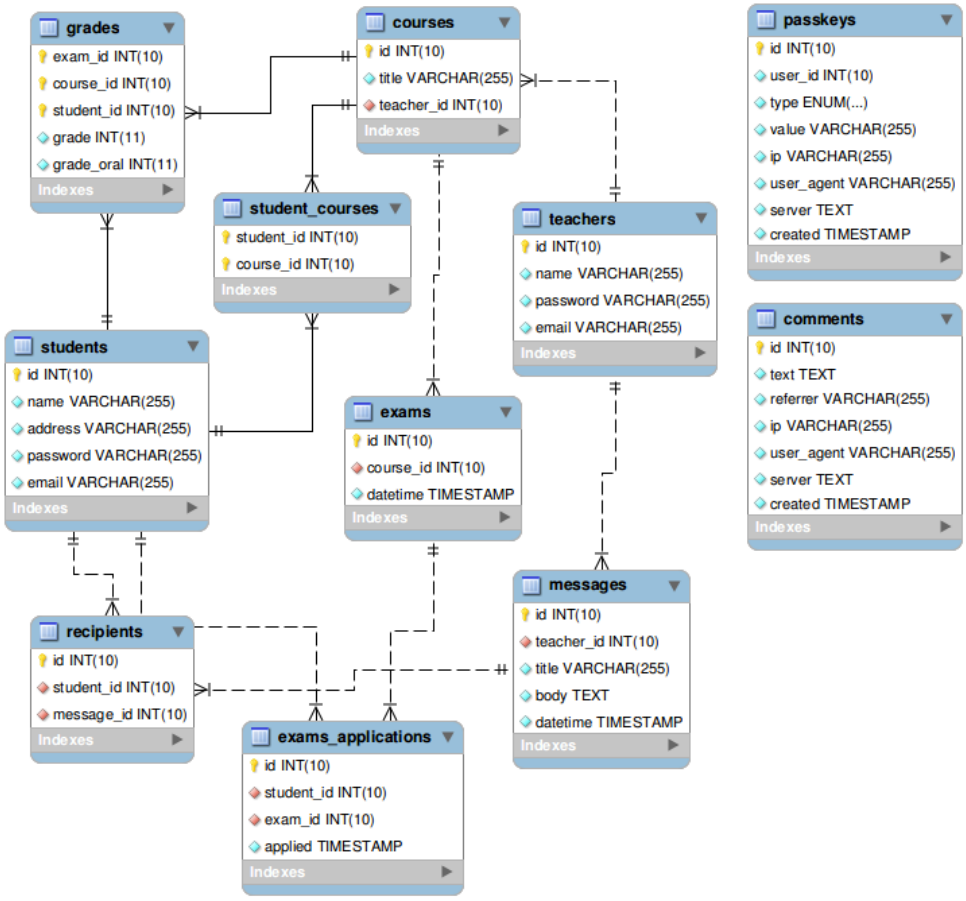
Podatkovna baza

Kot lahko vidimo na sliki 4.7, je podatkovna baza sestavljena iz 12 tabel, ki so med sabo povezane z različnimi relacijami. Posebnost sta samo tabeli *passkeys* in *comments*, ki sta edini brez relacij. Tabela *passkeys* vsebuje uporabniške ključe za dostop do metod strežniške aplikacije in shranjuje različne vrednosti, ki jih aplikacija potrebuje. Tabela *comments* pa vsebuje komentarje uporabnikov. Ker ne potrebuje podatkov iz drugih tabel, je brez relacij.

Tabela *students* vsebuje podatke o študentih, in sicer njegovo ime, naslov, geslo in elektronsko pošto. Tabela, ki vsebuje podatke o pedagoških delavcih pa se imenuje *teachers* in vsebuje njihova imena, gesla in elektronske naslove.

Vsak pedagoški delavec je lahko nosilec enega ali več predmetov (tabela *courses*). Vsak predmet pa ima lahko več ocen (tabela *grades*), za vsakega študenta po eno ustno oceno (polje *grade_oral*) in eno oceno pisnega dela (polje *grade*). Vsak predmet pa ima lahko tudi več izpitnih rokov (tabela *exams*), vsakega z različnim datumom in časom. Študent se na izpitni rok lahko prijavi s prijavnico (tabela *exams_applications*), ki vsebuje podatke o študentu, izpitnem roku in času prijave. Vsak študent je lahko prijavljen tudi na več predmetov prek tabele *student_courses*.

Pedagoški delavec lahko ima več sporočil (tabela *messages*), ki vsebujejo ime sporočila (polje *title*), besedilo sporočila (polje *body*) in čas, ko je bilo sporočilo ustvarjeno. Vsako sporočilo ima več prejemnikov (tabela *recipients*). Prejemnik je študent z natanko enim sporočilom.



Slika 4.7: Slika prikazuje strukturo podatkovne baze.

Poglavje 5

Sklepne ugotovitve

V diplomskem delu je opisan razvoj celovite rešitve m-Študent. To je aplikacija spisana za uporabo v mobilnih napravah z operacijskim sistemom Android, skupaj z lastnim strežniškim delom, ki skrbi za posredovanje in hranjenje podatkov.

Uspešno nam je uspelo razviti vse funkcionalnosti, ki smo si jih zadali na začetku. Z aplikacijo se lahko pedagoški delavec prijavi v sistem, pregleduje prijave na izpit in pregleduje ter pošilja obvestila študentom. Aplikacija je bila tudi stestirana na različnih mobilnih napravah z različnimi velikosti zaslonov. Vse teste je uspešno prestala.

Ker nam tekom izdelave ni uspelo pridobiti dostopa do podatkovne baze e-Študenta, smo morali bazo razviti sami. Trenutna rešitev tako deluje na lastnem strežniku, z lastno podatkovno bazo. Da bi aplikacija imela uporabno vrednost, bi jo morali povezati s podatkovno bazo e-Študenta. Tako bi odjemalec prikazoval realne informacije in obvestila bi resnično prišla do študentov. Rešitev smo razvili tako, da bo morebiten prehod na bazo e-Študenta čim bolj enostaven. Odjemalca ob prehodu ne bo potrebno spreminjati, ampak se spremeni samo strežniški del, ki skrbi za SQL poizvedbe na podatkovno bazo.

Zaenkrat je v aplikaciji omogočena prijava samo z uporabniškim imenom in geslom. Zaradi boljše varnosti bi bilo bolje, da bi se v morebitnem nadaljnjem razvoju spremenilo aplikacijo tako, da bi podpirala tudi prijavo s certifikatom, kot je to omogočeno v sistemu e-Študent. Zaenkrat poteka komunikacija med odjemalcem in strežnikom z lastnoročno podpisanim certifikatom, ki skrbi samo za zaščito povezave in ne prijavo v sistem.

Dodatek A

Namestitev strežniške aplikacije

Za namestitev aplikacije na virtualni oziroma fizični strežnik potrebujemo programsko opremo opisano v poglavju 4.3.1. Nameščanje delov strežniške aplikacije pa je opisano v nadaljevanju.

A.1 Namestitev datotečne strukture

Datotečna struktura se nahaja v skladišču *Subversiona*. Najprej na strežniku ustvarimo mapo v kateri se bo datotečna struktura nahajala. V to mapo, z ukazom `svn co https://velikonja.svn.beanstalkapp.com/mstudent .` iz skladišča shranimo vse mape in datoteke. Za dostop do skladišča potrebujemo ustrezno uporabniško ime in geslo.

Ker aplikacija za delovanje potrebuje ogrodje Zend Framework, moramo to ogrodje namestiti v mapo `trunk/lib/Zend`. Namestitveni postopki ogrodja so opisani v dokumentu Zend Framework Downloads [8].

A.2 Nastavitev spletnega strežnika

Za delovanje aplikacije moramo tudi ustrezno nastaviti spletni strežnik *Nginx*. Nastavitvena datoteka aplikacije se nahaja v `trunk/etc/nginx`, glede na to, ali nameščamo aplikacijo na produkcijsko ali razvojno okolje, uporabimo ustrezno datoteko. Datoteko skopiramo v mapo iz katere *Nginx* bere nastavitvene datoteke. Datoteko nato ustrezno spremenimo, da deluje v postavljenem okolju. V večini primerov moramo samo spremeniti naslov na katerem se aplikacija oglaša (`server_name`) in poti do datotečne strukture ter certifikatov. Nato ponovno zaženemo *Nginx*.

A.3 Namestitev podatkovne strukture

Kot omenjeno v poglavju 4.3.3, se datoteke s poizvedbami SQL nahajajo v mapi `trunk/sql`. Najprej ustvarimo novo zbirko podatkov z imenom *mstudent*. Nato poženemo ukaz
`mysql -u uporabnik -p geslo mstudent < pot-do-sql-poizvedb/mysql.sql`, kjer je `uporabnik`, uporabniško ime osebe, ki ima dostopne pravice do zbirke podatkov *mstudent*. S tem se celotna podatkovna struktura uvozi v ustvarjeno zbirko podatkov.

Aplikaciji pa moramo sporočiti še dostopne informacije do zbirke podatkov. Najprej prekopiramo datoteko `etc/configs/api.php` v `etc/configs/api.local.php`, v novi datoteki poiščemo nastavitve za *mysql* in jih ustrezno popravimo. Namestitev podatkovne struture je tako končana.

A.4 Testiranje namestitve

Namestitev aplikacije testiramo tako, da v poljubni brskalnik vpišemo naslednji URI: **https://naslov-strežnika/test**, kjer je naslov strežnika spremenljivka `server_name` omenjena v poglavju A.2. Če se v brskalniku izpiše niz {`"db":true`}, potem aplikacija deluje pravilno.

Slike

Literatura

[1] (2011) Android (operating system); dostopno na:
http://en.wikipedia.org/wiki/Android_(operating_system).

[2] (2011) Smartphone; dostopno na:
http://en.wikipedia.org/wiki/Smartphone.

[3] (2011) Tudi v Sloveniji pohod pametnih mobilnih telefonov in Androida;
dostopno na:
http://www.dnevnik.si/novice/znanost/1042430377.

[4] Sayed Hashimi, Satya Kometineni, Dave MacLean "Pro Android 2," 2010.

[5] (2011) Prijava na izpite; dostopno na:
http://www.fri.uni-lj.si/si/izobrazevanje/prijava_na_izpite/.

[6] (2011) PHP Manual; dostopno na:
http://si2.php.net/manual/en/index.php.

[7] (2011) Trygve Reenskaug, The original MVC reports; dostopno na:
http://heim.ifi.uio.no/ trygver/2007/MVC_Originals.pdf.

[8] (2011) Zend Framework Downloads; dostopno na:
http://framework.zend.com/download/overview.