UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Aleš Kavčič

# Razvoj mobilne aplikacije m-Študent na platformi Android

DIPLOMSKO DELO
VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: doc. dr. Rok Rupnik

Ljubljana, 2012

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00197/2012
Datum: 03.02.2012

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

Kandidat: **ALEŠ KAVČIČ**

Naslov: **RAZVOJ MOBILNE APLIKACIJE M-ŠTUDENT NA PLATFORMI ANDROID**

**THE DEVELOPMENT OF M-ŠTUDENT MOBILE APPLICATION FOR ANDROID PLATFORM**

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Na platformi Android razvijte mobilno aplikacijo m-Študent. Aplikacija naj bo zasnovana kot nadgradnja sistema e-Študent in naj študentom omogoča uporabo ključnih storitev. Pri načrtovanju aplikacije upoštevajte dejstvo, da nimate možnosti povezave na sistem e-Študent, zato vzpostavite nadomestno podatkovno bazo sistema e-Študent.

Mentor:

doc. dr. Rok Rupnik

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

# I Z J A V A  O  A V T O R S T V U

## diplomskega dela

Spodaj podpisani Aleš Kavčič,

z vpisno številko 63050140,

sem avtor diplomskega dela z naslovom:

Razvoj mobilne aplikacije m-Študent na platformi Android

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom

  doc. dr. Roka Rupnika,

- 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 29.03.2012 Podpis avtorja: _____

## Zahvala

Rad bi se zahvalil vsem, ki so me podpirali na moji poti do diplome. Na prvem mestu bi se zahvalil moji družini za podporo in spodbudo ob premagovanju ovir ter seveda zahvala tudi vsem kolegom in kolegicam na fakulteti. Še posebej bi se rad zahvalil mentorju ter sodelavcu Nejcu Ločniškarju in sošolcu Mateju Velikonji za velikodušno pomoč in nasvete.

# Kazalo vsebine

# Seznam uporabljenih kratic in simbolov

**API** – Application programming interface

**APK** – Application Package File

**EJB** - Enterprise Java Bean

**HTTP** - HyperText Transfer Protocol

**Java EE** – Java Enterprise Edition

**JPA** – Java Persistence API

**JPQL** - Java Persistence Query Language

**JSON** – JavaScript Object Notation

**JVM** – Java Virtual Machine

**OS** – Operating system

**POJO** – Plain Old Java Object

**RAM** - Random access memory

**SQL** - Structured Query Language

**SSL** - Secure Sockets Layer

**UML** – Unified Modeling Language

**URI** – Uniform Resource Identifier

**XML** – Extensible Markup Language

# Povzetek

Cilj diplomske naloge je izdelava mobilne aplikacije m-Študent, ki temelji na informacijskem sistemu e-Študent. Odjemalec m-Študent je aplikacija prilagojena za mobilne naprave z operacijskim sistemom Android. Aplikacija študentom omogoča prijavo in odjavo z izpita, pregled obvestil in ocen, ter naročanje potrdil. V uvodnih poglavjih je predstavljen že delujoči sistem e-Študent. V nadaljevanju diplomske naloge pa je opisan operacijski sistem Android ter razvojno okolje Eclipse. Opisana so tudi ostala programska orodja in tehnologije, ki so bila uporabljene pri razvoju aplikacije. V glavnem delu je predstavljena arhitekturna zasnova rešitve, princip delovanja odjemalca in strežnika, samo delovanje in uporaba aplikacije. V zaključku so navedeni še problemi, ki so nastali med razvojem in možnosti za izboljšavo aplikacije.

**Ključne besede:**
Android, Java, mobilna aplikacija, pametni telefoni, emulator, operacijski sistem, e-Študent, m-Študent

# Abstract

The goal of this thesis is to create mobile application m-Študent based on information system e-Študent. M-Študent client is an application adapted for mobile devices running on Android operating system. The application enables students to apply for exams, remove exam applications, read notices, check grades and order confirmations. The introductory chapter presents a system called e-Študent. The following chapters describe Android operating system, Eclipse development environment and other software tools and technologies that have been used to develop m-Študent application. The main part describes architectural design, client/server concept and presents application functions including its usage. Problems and ideas that appeared during development are explained in the conclusion.



**Key words:**

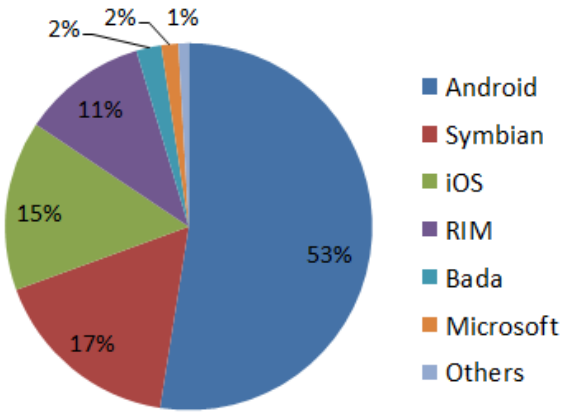
Android, Java, mobile application, smart phones, emulator, operating system, student's information system e-Študent, m-Študent

# 1. Uvod

Pametni mobilni telefoni iz dneva v dan postajajo vse bolj podobni običajnim računalnikom. Z njimi lahko poleg telefonskih pogovorov in pošiljanju kratkih spočil SMS namreč deskamo po svetovnem spletu, dostopamo do socialnih omrežij, neposredno objavljamo fotografije in filmske posnetke na spletnih dnevnikih in še bi lahko naštevali. Mobilne naprave so na poti, da postanejo novodobni osebni računalniki. Njihove zmogljivosti namreč dosegajo zmogljivosti osebnih računalnikov.

Mobilni telefon je množični medij prihodnosti, ki naj bi ponujal največ možnosti za razvoj in zaslužek. Pametni telefon je več kot le telefon, saj je hkrati tudi ura, koledar, fotoaparat, način dostopa do spleta in plačilni kanal. V treh letih so razvijalci ustvarili več kot 300 tisoč različnih mobilnih aplikacij, piše IDC **(**International Data Corporation). V letu 2010 so uporabniki po svetu prenesli 10,9 milijarde mobilnih aplikacij, pri čemer je bilo opaziti tudi razmah uporabe teh aplikacij na tabličnih napravah. [1]

Trenutno na trgu obstaja več mobilnih operacijskih sistemov. Gartnerjevi statistični podatki kažejo, da so pametni telefoni z **Androidom** v tretji četrtini leta 2011 zavzemali kar 52,5 % svetovni tržni delež. Zaradi neprestane rasti Androida pa so tržni delež izgubili ostali operacijski sistemi. Za Androidom sledijo naslednji operacijski sistemi: **Symbian** (16,9 % delež), **iOS** (15 % delež), **BlackBerry OS** (11 % delež), **Bada** (2,2 % delež) in **Windows Mobile** (1,5 % delež). Prikaz deležev prikazuje slika 1.1. [2]




Slika 1.1: Delež mobilnih operacijskih sistemov na svetovnem trgu v tretjem četrtletju 2011.

Po podatkih dveh največjih slovenskih mobilnih operaterjev, delež pametnih mobilnih telefonov v skupnem številu prodanih mobilnih telefonov ves čas narašča in pri Mobitelu

dosega že skoraj 50 %. Simobil podobno poroča, da je v zadnjem četrtletju leta 2010 uporaba pametnih telefonov narasla na 40 %. Najbolj popularen operacijski sistem pa je Android. [3]

Diplomsko delo se osredotoča na razvoj celovite rešitve m-Študent. Aplikacija je sprogramirana za mobilne naprave na katerih je operacijski sistem Android. Ima svoj strežnik, ki skrbi za posredovanje, hranjenje in obdelavo podatkov. V diplomski nalogi je najprej predstavljen tudi že obstoječi sistem e-Študent, ki je bila osnova za izgradnjo mobilne aplikacije m-Študent. V 3. poglavju diplomske naloge spoznamo še tehnologije in orodja, ki so bila uporabljena pri razvoju rešitve. Spoznamo kaj je Android in njegove sestavne dele ter programski jezik Java, ki je osnova za vse aplikacije, ki tečejo na Androidu. Poglavje 4 je sestavljeno iz dveh delov. V prvem delu je predstavljen razvoj odjemalca. Odjemalec je Android aplikacija, ki teče na mobilni napravi in komunicira s strežnikom. Najprej se seznanimo z delovanjem in arhitekturo odjemalca. V poglavju o arhitekturi se spoznamo z vsemi razredi in ostalo programsko kodo, ki je bila razvita v sklopu diplomskega dela. V drugem delu je opisan strežniški del rešitve. Tu spoznamo, kako se odjemalec povezuje s strežnikom, kako strežnik sprejema zahteve in se na njih odziva ter obdela podatke v podatkovni bazi.

# 2. Sistem e-Študent

## 2.1 Kaj je e-Študent

E-Študent je spletni 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 Univerze v Ljubljani 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. Sistem uporabljajo tudi mnoge druge fakultete. Sistem e-Študent je prikazan na sliki 2.1. [4]

Pred uvedbo sistema e-Študent se je uporabljalo sistem FNISID. Program FNISID - Fakultetni Nivo Informacijskega Sistema Izobraževalnih Dejavnosti je pripravil in ga tudi vzdrževal prof. dr. Viljan Mahnič s Fakultete za računalništvo in informatiko. Na obeh fakultetah (FRI in FE) sta bila na voljo po en študentom namenjen računalnik, s katerima so se prijavljali in odjavljali z izpitov. Program, ki je bil napisan leta 1991 in kasneje še dopolnjevan, je bil napisan v programskem jeziku Clipper in tekel pod operacijskim sistemom DOS v Novellovi mreži. Pred tem sistemom pa je vse potekalo preko papirnatih prijavnic. Najnovejši študentski informacijski sistem VIS, ki bo sčasoma nadomestil e-Študent, se že uporablja na nekaterih fakultetah. [5]

## 2.2 Uporaba e-Študenta

Ob vpisu na Fakulteto za računalništvo in informatiko, študent dobi tudi dostop do e-Študenta. Študent se lahko v sistem prijavi z uporabniškim imenom in geslom, vendar je tak način prijave omejen s številom prijav. Najbolj varno je, če se študent prijavi v sistem preko digitalnega potrdila. Za potrebe sistema e-Študent se uporabljajo digitalna potrdila izdajatelja SIGEN-CA, ki deluje v okviru Centra vlade za informatiko. Izdajatelj (overitelj) potrdil jamči za istovetnost lastnikov digitalnih potrdil. Z drugimi besedami, ko se uporabnik identificira z digitalnim potrdilom, ga sistem enolično in nedvoumno razpozna. Možna je tudi uporaba digitalnih potrdil certifikatne agencije @friCA, ki deluje interno na Fakulteti za računalništvo in informatiko. Za uporabo sistema potrebujemo le sodobni spletni brskalnik in delujočo internetno povezavo oziroma intranet na fakulteti.


Slika 2.1: Prijava v sistem e-Študent.

Z uvedbo e-Študenta se je razbremenilo študente na fakulteti, saj lahko preko sistema opravijo večino birokratskih zadev, katere so prej morali opravljati v študentski pisarni fakultete. Sistem uporabljajo tudi profesorji in asistenti Fakultete za računalništvo in informatiko ter študentski referat. [6]

Sistem e-Študent sestavljajo trije moduli:

- **e-profesor**, ki profesorjem in asistentom omogoča vpisovanje ocen, obveščanje študentov in analize rezultatov posameznih izpitnih rokov, kakor tudi dodajanje novih izpitnih rokov. Za vsakega pedagoškega delavca je v sistemu opredeljena svoja vloga, vendar bistvenih razlik med njimi 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.
- **e-študent**, ki študentu z uporabniškim imenom in geslom omogoča prijavo/odjavo na izpite in kolokvije, pregledovanje ocen (elektronski indeks), naročanje potrdila o vpisu oz. opravljenih izpitih, prebiranje obvestil, izpolnjevanje vpisnega lista preko interneta in ostala opravila v zvezi s študijskimi zadevami.
- **e-referat**, ki je namenjen zaposlenim v študentskem referatu in predstavlja jedro celotnega sistema, v katerem so omogočene vse funkcionalnosti, ki se nanašajo bodisi na študenta ali profesorja fakultete.

# 3. Uporabljena orodja in tehnologije pri razvoju aplikacije

V tem poglavju si bomo podrobno ogledali programska orodja in tehnologije, ki smo jih uporabili pri razvoju aplikacije m-Študent. Predvsem bomo predstavili operacijski sistem Android in njegove dele, vključno z razvojnim okoljem Eclipse ter s programskim jezikom Java v katerem se programirajo aplikacije za Androida. Potem si bomo ogledali še ogrodje aplikacijskega strežnika Apache Geronimo ter ostale tehnologije, uporabljene pri razvoju rešitve.

## 3.1 Android

### 3.1.1 Splošno o Androidu

Android je operacijski sistem oziroma platforma, namenjena mobilnim napravam. Vključuje operacijski sistem, aplikacije in programsko opremo, ki ta dva nivoja povezuje. Android temelji na Linuxovem jedru in ga razvija podjetje Google v sodelovanju s 47-imi podjetji, ki so združena v konzorcij Open Handset Alliance (OHA) in je bilo predstavljeno v drugi polovici leta 2007. Združenje OHA teži k pospešenemu razvoju inovacij na področju mobilnih storitev in razvoju odprtih standardov na področju prenosnih naprav. V tem združenju najdemo podjetja, kot so Google, HTC, Intel, Broadcom, Nvidia, T-Mobile, China Mobile, Samsung, LG Electronics, Texas Instruments in Motorola... Na sliki 3.1 vidimo Android logotip [7]


Slika 3.1: Android logotip.

Ime Android pomeni robota, ki s svojim videzom ter delovanjem spominja na človeka. Z razvojem Androida je začelo podjetje Android Inc., ki 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. Android je širši javnosti bolje poznan kot rezultat Googlovega projekta Gphone. Koda (odprta) tega projekta je bila dana v javnost pod imenom Android Open Source Project (AOSP).

Google je dostop do izvorne kode platforme Android omogočil 21. oktobra 2008 pod odprto kodno licenco Apache License. Google je objavil celotno izvorno kodo, vključno z mrežno in telefonsko kopico. Android je odprtokoden in povsem brezplačen kar pomeni, da so potrošnikom na voljo cenejše in bolj funkcionalne mobilne naprave ter večje število inovativnih storitev. Proizvajalcem mobilnih telefonov Android omogoča znižanje stroškov razvoja programske opreme ter hitrejšo realizacijo svojih konceptov, prav tako pa jim ob fleksibilnosti sistema še vedno prinaša možnost krojenja lastne identitete z razvojem samosvojih rešitev. Podjetja, ki se ukvarjajo s programsko opremo, bodo odslej enostavneje in cenejše izdelovala programske komponente, ki bodo na voljo velikemu številu uporabnikov, samostojni razvijalci programske opreme pa bodo z lažjo in cenejšo distribucijo ter komercializacijo lahko svoje ideje pokazali celemu svetu.

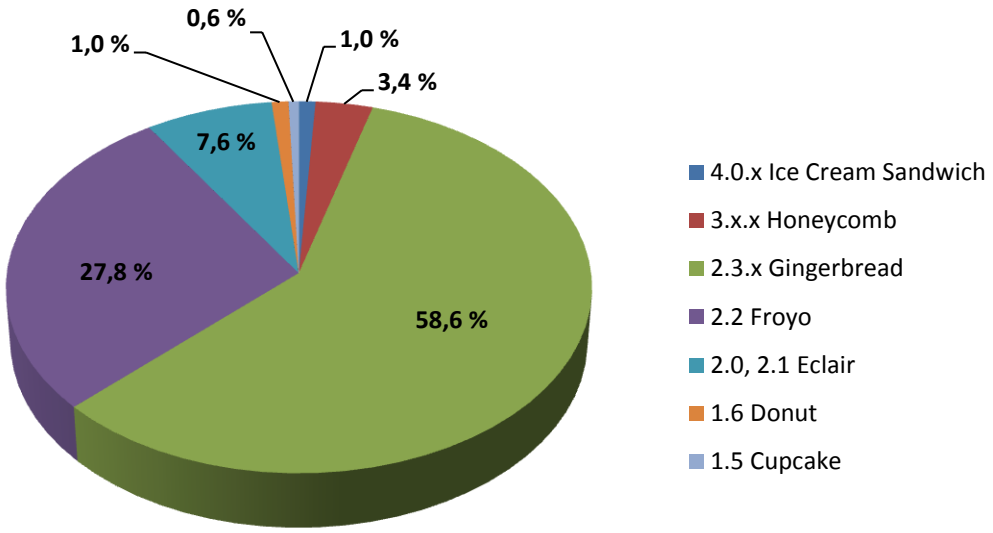
### 3.1.2 Različice operacijskega sistema Android

Od svojega prihoda na tržišče, to je od septembra 2008 do danes, je platforma Android doživela že veliko nadgradenj in popravkov. Vsaka posodobitev ogrodja API je narejena tako, da je nova verzija kompatibilna s svojo predhodno verzijo. Različice se predstavljajo s kodnimi imeni, ki so hkrati tudi imena sladic. Kodna imena se v industriji uporabljajo, kadar se želijo prikriti informacije o izdelavi novih projektov. V tabeli 1 [8], so naštete do sedaj izdane ali znane verzije OS Android, njihovo kodno ime, datum izdaje in nekaj glavnih lastnosti posamezne verzije.

| Verzija | Kodno ime | Datum izdaje | Lastnosti |
|---|---|---|---|
| 1.0 | / | 9.2.2007 | Prva različica operacijskega sistema, ki je prišla na trg s prvimi pametnimi telefoni. Sistem je vseboval funkcije, kot so budilka, testni prikaz (demo) uporabniškega vmesnika, pregledovalnik za internet, kamero, itd. |
| 1.5 | Cupcake | 30.4.2009 | Ta različica je sprožila pravo poplavo pametnih telefonov na tržišču in med uporabniki. Poleg različice 1.0 so tej različici dodatno dodali še možnosti dodajanja medijskih datotek neposredno na internet, možnost bluetooth povezave, animacije na ekranu, itd. |
| 1.6 | Donut | 15.9.2009 | Dodan mu je bil nov, preglednejši, uporabnejši Android Market za prenos programov, skupaj z Open Handset Alliance so pripravili telefon HTC Hero in na njem uspešno zagnali to različico operacijskega sistema. |
| 2.0/2.1 | Eclair | 26.10.2009 | Ta različica je bila nekaj novega, saj so jo pričeli izdelovati od samega začetka in ni popravek. Zaradi ponovne izdelave, so se vsem uporabnikom prejšnjih različic operacijskega sistema telefoni nadgradili v celoti. Posledice so bile občutna pohitritev odzivnega časa operacijskega sistema, novi uporabniški vmesniki, bluetooth 2.1 in podobno. |
| 2.2 | Froyo | 20.5.2010 | Popravek se je sprva pojavil na telefonu HTC Nexus One, sčasoma pa tudi na ostalih pametnih telefonih. Ključne značilnosti popravka so bile nalaganje aplikacij na spominsko kartico telefona, vizualno popravljeni in spremenjeni uporabniški vmesniki, ter spremenjen Android market, ki je sedaj omogočal samodejne posodobitve aplikacij. |
| 2.3 | Gingerbread | 6.12.2010 | Ker so se pri tem popravku večinoma osredotočili na strojno opremo, vsebuje popravek dva nova senzorja (giroskop, barometer). Dodana so tudi orodja za kopiranje in lepljenje datotek. |
| 3.0/3.1 | Honeycomb | 24.2.2011/ 10.5.2011 | Zasnovan za tablične računalnike. Zaslon upošteva večje ekrane. Več-jedrni procesor in novi multimedijski standardi. |
| 4.0 | Ice cream sandwich | 19.10.2011 | Na trg je prispel skupaj s telefonom Galaxy Nexus. V obilici popravkov in dodatnih funkcionalnosti sistema izstopajo strojno pospešen grafični vmesnik, prenova grafičnega vmesnika, odklep s prepoznavo obraza, nov spletni brskalnik, izboljšana aplikacija za kamero, itd. |

Tabela 1: Različice operacijskega sistema Android

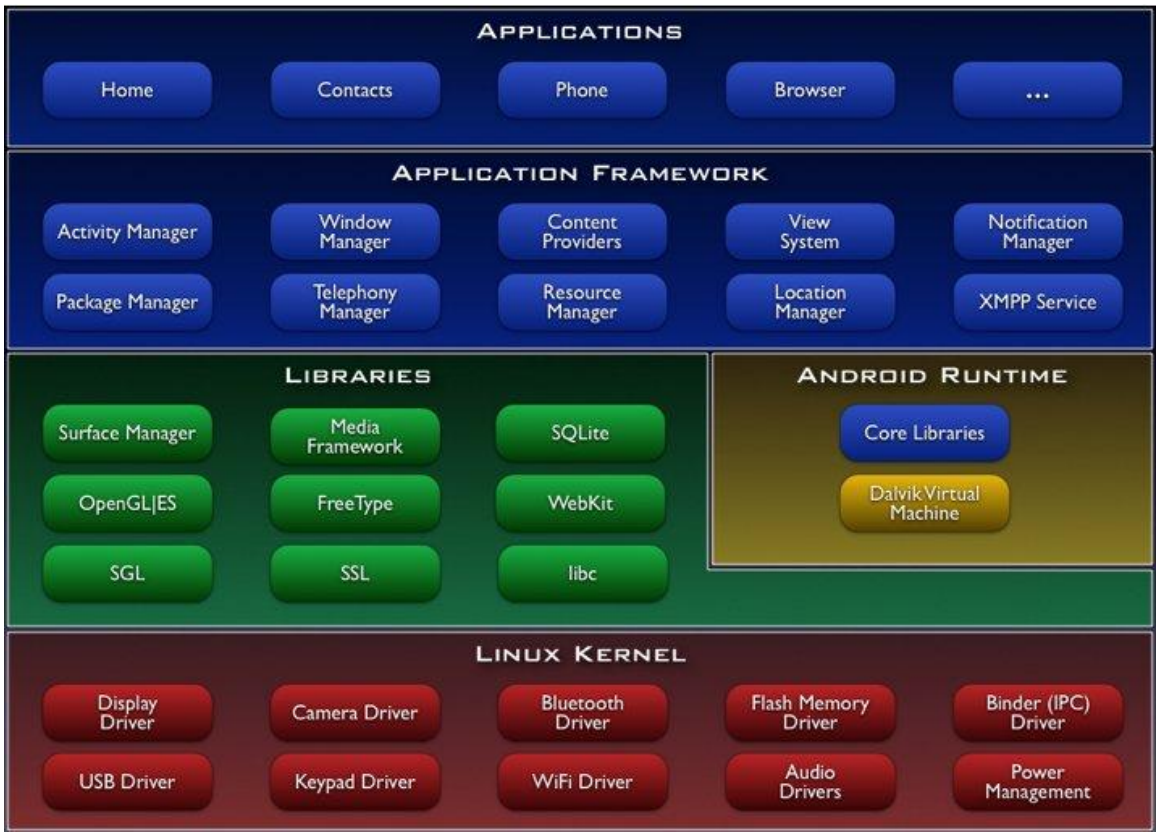
Trenutno najbolj uporabljena verzija je 2.3.x Gingerbread in sicer z 58.6 % deležem na trgu. Na sliki 3.2 lahko vidimo še ostale deleže verzij.




Slika 3.2: Delež uporabe različnih verzij operacijskega sistema Android do 1. februarja 2012.

### 3.1.3 Arhitektura

Slika 3.2 opisuje glavne komponente operacijskega sistema Android, ki so predstavljene kot sklad plasti. Arhitektura Androida je sestavljena iz petih elementov: aplikacije, aplikacijsko ogrodje, knjižnice, jedro Linuxa in prevajalnik. V nadaljevanju so posamezni deli še posebej opisani. [9]




Slika 3.3: Arhitektura operacijskega sistema Android.

**Aplikacije *(angl. applications)***

Vse te aplikacije so napisane v programskem jeziku Java. Poleg nje se za izdelavo programov uporablja tudi XML. Prevedena javanska koda vključno z vsemi potrebnimi viri se zapakira v datoteko s končnico .apk. To datoteko se smatra za aplikacijo in predstavlja osnovo za distribucijo in nameščanje aplikacije na mobilni telefon. Ob nakupu mobilnega telefona je nekaj osnovnih aplikacij, na primer Android Market, koledar, zemljevidi, brskalnik, kontakte, odjemalec e-pošte, program za SMS sporočila, že naloženih. Vsaka aplikacija se požene v svojem Linux procesu. Vse vrste aplikacij se nahajajo na aplikacijskem nivoju.

**Aplikacijsko ogrodje *(angl. application framework)***

V aplikacijskem ogrodju se nahajajo vse sistemske aplikacije ki se uporabljajo za razvoj aplikacij. Razvijalci lahko tako izkoristijo strojno opremo naprave, dostopajo do lokacijskih podatkov, poganjajo servise v ozadju itd. 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:

- Pogledi *(angl. Views)* se uporabljajo za gradnjo uporabniškega vmesnika (seznami, vnosna polja, gumbi …) aktivnosti aplikacij;
- Ponudniki vsebin *(angl. Content providers)* omogočijo aplikacijam, da si delijo podatke z drugimi aplikacijami, kot je npr. Kontakti;
- Upravljalnik lokacij *(angl. Location Manager)* omogoča dostop do lokacijskih storitev;
- Upravljalnik virov *(angl. Resource Manager)* podpira dostop do zunanjih virov, kot so nizi, grafika in slike;
- Upravljalnik obvestil *(angl. Notification Manager)* omogoča dosledno obveščanje uporabnikov;
- Upravljalnik aktivnosti *(angl. Activity Manager)* ureja življenjski cikel aktivnosti aplikacij.

**Knjižnice *(angl. libraries)***

Nad jedrom se nahaja plast, ki vsebuje knjižnice. Te niso aplikacije same po sebi, temveč skrbijo za povezavo s komponentami sistema Android.. Spisane so v jeziku C ali C++. Nekaj primerov knjižnic je navedenih spodaj:

- grafični knjižnici Skia (2D grafika) in OpenGL ES (3D grafika) skrbita za podporo grafiki;
- knjižnica za podporo medijem (angl. Media codecs) zagotavlja kodeke, ki omogočajo predvajanje zvoka in video vsebin, npr. AAC, MP3, MPEG-4;
- knjižnica SQLite za podporo podatkovnim bazam;
- knjižnica Webkit, ki zagotavlja delovanje spletnega brskalnika

### Izvajalno okolje (angl. Android runtime)

V plasti nad jedrom se poleg knjižnic nahaja tudi Android runtime oziroma izvajalno okolje. Izvajalno okolje vsebuje dva pomembna dela, in sicer: Java knjižnice in virtualni stroj Dalvik. Nizka učinkovitost CPU, majhen RAM, počasen notranji pomnilnik in omejena moč baterije so glavne značilnosti mobilnih naprav, zaradi katerih so pri Googlu čutili potrebo po spremembi Java knjižnic in Java virtualnega stroja (JVM). Glavne Java knjižnice, ki se nahajajo znotraj izvajalnega okolja, so tako alternativna in omejena implementacija standardnih Java knjižnic, virtualni stroj Dalvik pa so razvili eksplicitno za Android. [10]

Tudi na Android platformi se izvorna javanska koda prevede v *.class* datoteko, vendar se nato s pomočjo orodja »dx«, *.class* datoteka v nadaljevanju pretvori v datoteko *.dex* oziroma Dalvik Executable File. Slednja se izvrši v Dalvik VM. Dalvik VM je spisan tako, da naprava lahko učinkovito zažene več virtualnih strojev naenkrat. To je pomembna lastnost, saj vsaka aplikacija na Androidu teče znotraj svojega procesa. Vsak proces zahteva svoj primerek virtualnega stroja.

### Jedro Linuxa (angl. Linux kernel)

Na dnu arhitekture Androida se nahaja Linux jedro, ki je zgrajeno na verziji Linuxa 2.6. Jedro upravlja z gonilniki naprave, pomnilnikom, omrežjem in procesi, skrbi za varnost in zagotavlja ostale storitve operacijskega sistema. Jedro deluje kot vez med strojno in programsko opremo …

### Življenjski cikel aplikacij

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 življenjski cikel aktivnosti.
Vsaka aktivnost ima štiri bistvena stanja:

- Aktivnost je aktivna *(running)* in je na zaslonu v ospredju (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 *(paused)*, je še vedno živa (to pomeni, da so vsa stanja in spremenljivke aktivnosti shranjene), vendar jo lahko sistem v primeru pomanjkanja pomnilnika zaustavi.

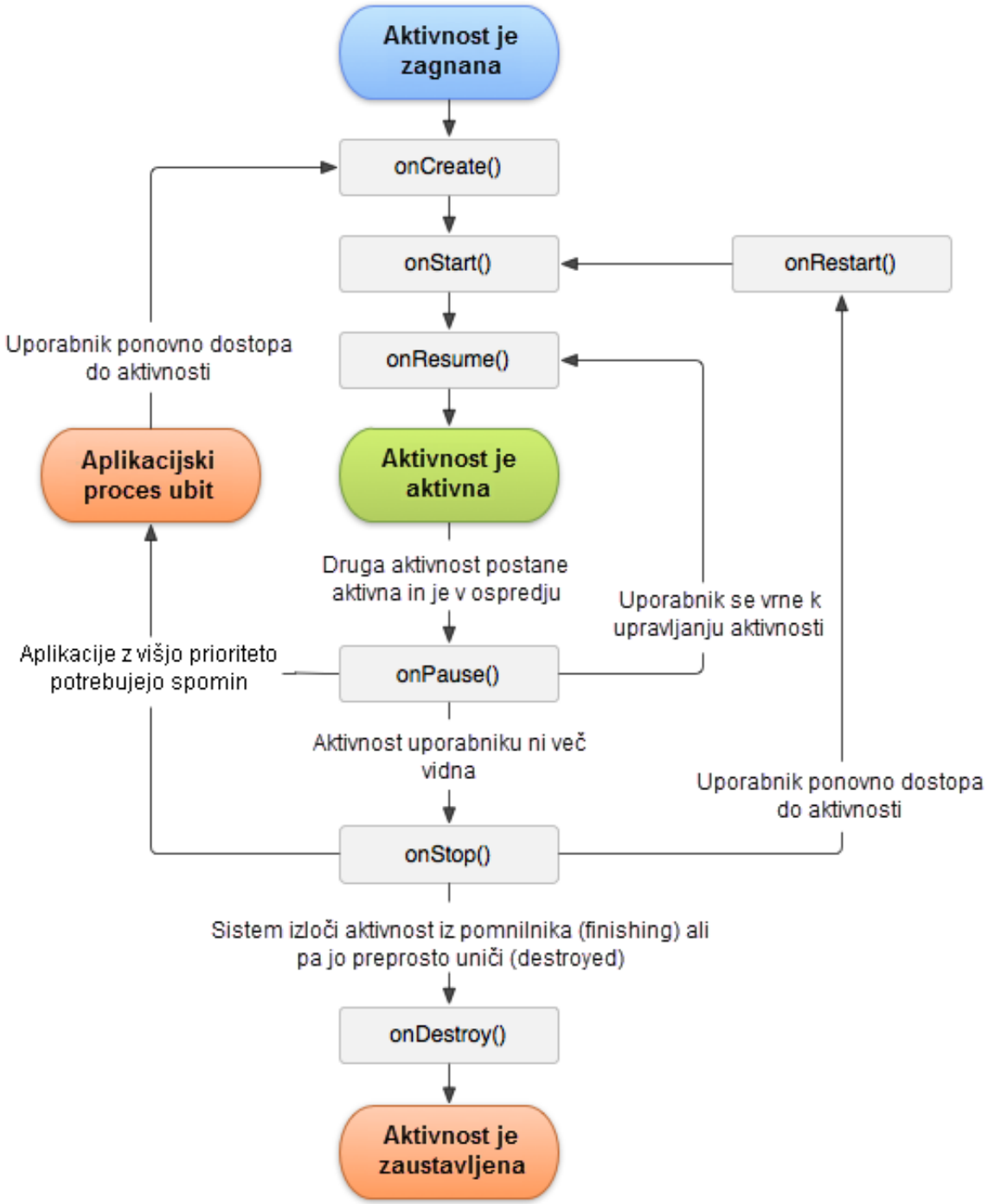
- Aktivnost je zaustavljena *(stopped)*, ker je neka druga aktivnost postala aktivna. V tem stanju aktivnost uporabniku ni več vidna. Sistem v tem stanju še vedno hrani vsa stanja in spremenljivke aktivnosti, vendar jo bo sistem uničil, če bo potreboval pomnilnik za kaj drugega.
- Če je aktivnost prekinjena *(paused)* ali zaustavljena *(stopped)*, 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.

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 *(angl. entire lifetime)* 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 *(angl. visible lifetime)* aktivnosti se zgodi med metodama *onCreate()* in *onStop()*. V tem času je aktivnost uporabniku vidna (čeprav ni nujno, da je v ospredju). Med tema dvema metodama običajno upravljamo z viri, ki so nujno za prikaz aktivnosti.
- Cikel, ko je aktivnost v ospredju *(angl. foreground lifetime)*, 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 sistem sam, 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.



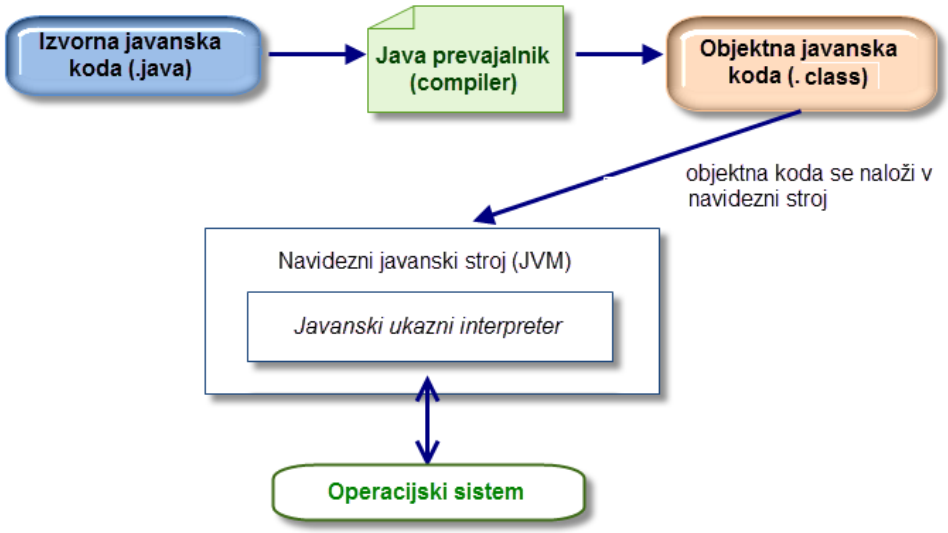

Slika 3.4: Življenjski cikel aktivnosti aplikacij.

## 3.2 Programska orodja

V tem poglavju si bomo podrobno ogledali konkretna programska orodja, ki smo jih uporabili pri razvoju mobilne aplikacije. Pri razvoju aplikacije smo uporabili veliko orodij, ki so odprtokodna in prosto dostopna. V nadaljevanju si bomo pobliže ogledali vsakega izmed njih.

### 3.2.1 Java EE

**Java** je objektno usmerjeni, prenosljivi programski jezik, ki ga je razvil James Gosling s sodelavci v podjetju Sun Microsystems. Njena glavna prednost je neodvisnost od operacijskega sistema, kar pomeni, da javanska aplikacija, zgrajena v Windows okolju brez kakršnihkoli prilagajanj, deluje tudi v Unix okolju. To omogoča JVM (angl. Java Virtual Machine) oziroma navidezni javanski stroj, čigar naloga je prevajanje vmesne kode (datoteke s končnico .class) v platformi prilagojene ukaze (diagram delovanja na sliki 3.5) in upravljanje s pomnilnikom med izvajanjem, kar zmanjša obseg dela programerja.




Slika 3.5: Diagram delovanja JVM.

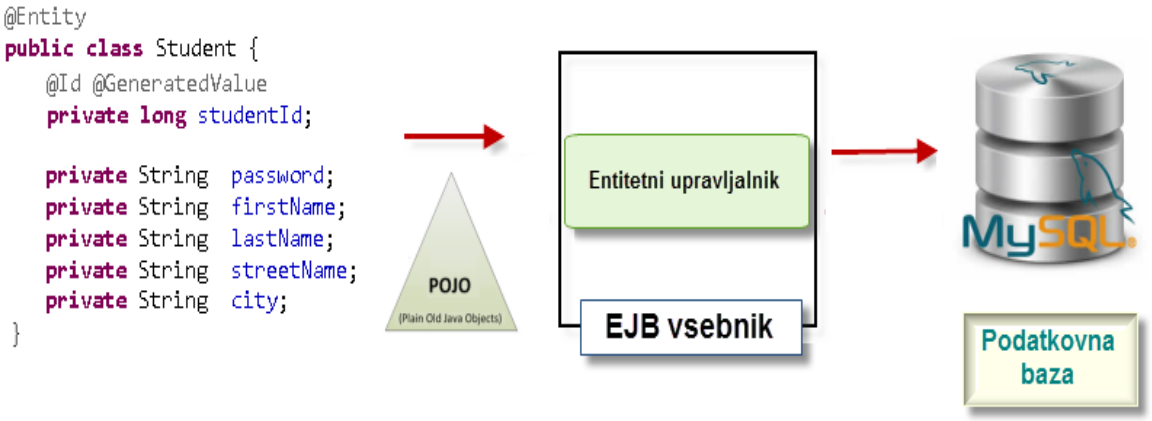
Java EE (angl. Java Enterprise Edition) je javansko razvojno okolje, ki se od običajne oblike Jave (angl. Java Standard Edition) razlikuje v tem, da vsebuje knjižnice, ki omogočajo razvoj modularnih, porazdeljenih aplikacij, ki tečejo na aplikacijskem strežniku. Aplikacijski strežnik upravlja s transakcijami, skrbi za varnost, nadgradnjo in upravlja z vsemi komponentami, ki so na njem nameščene. Tako je razvijalcem omogočeno, da se osredotočijo na poslovno logiko sistema in ne na njegovo infrastrukturo. Java EE programski model

definira štiri tipe aplikacijskih komponent, ki jih implementacija specifikacije podpira: aplikacijski odjemalci, appleti, servleti (strežniški programčki), JSP in EJB. [11]

V nadaljevanju si bomo pogledali tehnologije Java EE okolja, ki smo jih uporabili za razvoj strežniškega dela naše mobilne aplikacije.

#### 3.2.1.1 EJB in JPA

EJB (angl. Enterprise Java Bean) je tehnologija za realizacijo poslovno-storitvenega nivoja. Za našo aplikacijo je bistvena njena zmožnost dela z entitetnimi zrni. Entitetno zrno je tip javanskega zrna, ki predstavlja podatek, zapisan v podatkovni bazi. Entitetna zrna (angl. Entity beans) so povsem navadni javanski razredi POJO, opremljeni z ustreznimi anotacijami (angl. annotations), ki določajo način preslikave v podatkovno bazo. Anotacije intepretira aplikacijski strežnik in jih upošteva pri svojem delovanju. Oznako za anotacijo predstavlja znak @, ki ji sledi niz znakov, s katerimi želimo dodatno označiti našo kodo. Vsa skrb glede ustrezne tvorbe in preslikave je prepuščena vsebniku EJB (angl. EJB container), ki skrbi tudi za veliko ostalih opravil (slika delovanja na sliki 3.6). Vsebnik tako skrbi za podrobnosti podatkovne baze, zmožen je kreiranja podatkovne baze iz entitetnih zrn, vzdržuje podatkovno shemo, skrbi za transakcije, omogoča sočasno dostopanje do istih virov, upravlja s povezavami na bazo, itd. [12]

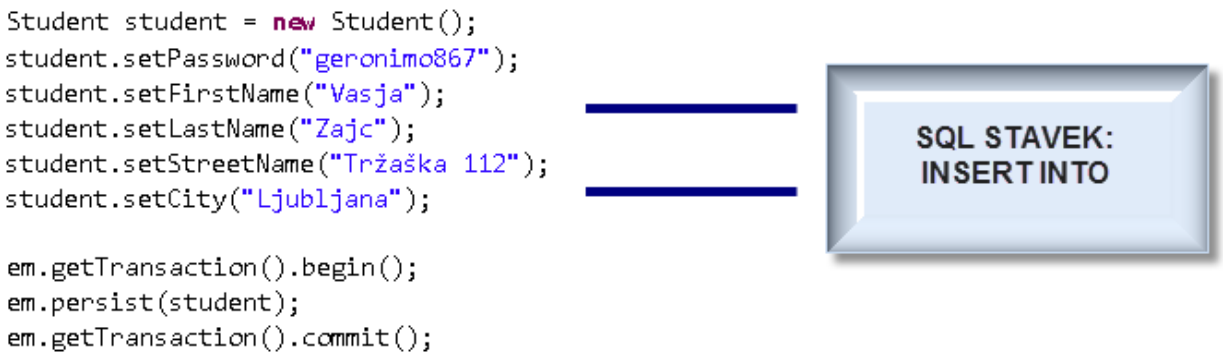



Slika 3.6: Shema delovanja arhitekture EJB.

Aplikacije, zgrajene z modelom EJB, so transakcijske, nadgradljive in varne, prav tako pa niso odvisne od aplikacijskega strežnika, za tega je važno le to, da podpira specifikacijo EJB.

Vmesnik JPA (Java Persistence API) je del specifikacije EJB in skrbi za objektno-relacijsko preslikavanje v podatkovno bazo. Javanski razred, označen z JPA anotacijami predstavlja tabelo v podatkovni bazi, objekt istega razreda pa en vnos v tabeli. (primer preslikave iz javanskega objekta v zapis v bazi na sliki 3.7). Za poizvedbe podatkov iz entitet vmesnik JPA uporablja objektno orientiran poizvedbeni jezik Java Persistence Query Language (JPQL). Jezik je zelo podoben jeziku SQL, le da namesto s tabelami operira z entitetnimi objekti JPA. S pomočjo JPA lahko razvijalec tudi brez uporabe jezika SQL vpisuje, briše in spreminja podatke v bazi. Jedro JPA predstavlja objekt Entitetni upravljalnik (angl. Entity Manager), ki sinhronizira stanje v podatkovni bazi, skrbi za istočasno izvajanje transakcij in za odpiranje in zapiranje povezav na bazo.

Obstaja več implementacij vmesnika JPA: TopLink, Spring, Hibernate in OpenJPA. Za potrebe naše mobilne aplikacije smo uporabili OpenJPA, ker je enostaven in že priložen aplikacijskemu strežniku Apache Geronimo, ki je opisan v nadaljevanju.




Slika 3.7: Primer vnosa v podatkovno bazo z JPA.

#### 3.2.1.2 Servlet

Servlet (strežniški programček) je javanski razred, ki sprejme zahtevo *http*, v kateri se prenašajo parametri. Zahtevo procesira in po potrebi vrne odgovor oz. se na zahtevo odziva. HttpServlet je najbolj pogosto uporabljen za generiranje dinamičnih spletnih strani kot odgovor na zahtevo. Kot dobra praksa se smatra uporabo servletov za izvajanje poslovne logike. V našem primeru servlet sprejme zahtevo *http*, ki jo pošlje Android.

#### 3.2.1.3 Aplikacijski strežnik

Aplikacijski strežnik je odgovoren za strežniško stran v aplikacijah odjemalec/strežnik kot tudi za podatke, ki so dosegljivi za odjemalce. Na njem tečejo Java EE komponente kar pomeni, da na njem v našem primeru teče storitev EJB in servleti. Strežnik je zadolžen za dodajanje, brisanje in spreminjanje komponent, ki se na njem izvajajo. Skrbi pa tudi za učinkovito izvajanje procedur.

V Java EE okolju aplikacijski strežnik za komponente, ki na njem tečejo, predstavlja podaljšan JVM. Ta transparentno skrbi za povezave do baze na eni strani povezave do spletnega klienta na drugi. Moderni javanski aplikacijski strežniki imajo v večini primerov že priložene tudi spletne strežnike, ki skrbijo za povezave http, SSL seje in zahtevke http. Strežnik za našo aplikacijo predstavlja rešitev za izvajanje Java EE aplikacij, brez nameščanja dodatnih komponent.

Poleg drugih aplikacijskih strežnikov kot so Glassfish, Jetty, Resin in Tomcat, smo se odločili za aplikacijski strežnik Apache Geronimo, ki v polnosti podpira Java EE verzije 6. Vanj je vključenih veliko javanskih komponent, za nas pa so pomembne predvsem naslednje:

- Apache Tomcat (strežnik http, vsebnik za izvajanje servletov)
- Apache OpenEJB (vsebnik za EJB)
- Apache OpenJPA (ogrodje JPA)

### 3.2.2 Eclipse

Eclipse je poleg Netbeans najbolj razširjeno prosto dostopno razvojno okolje za razvoj običajnih in Java EE aplikacij. Omogoča enostavno nameščanja razširitev in vtičnikov. Eclipse ima vdelan močan razhroščevalnik (angl. debugger), v njem pa je možno uporabiti tudi aplikacijski strežnik, s katerim lahko lokalno testiramo delovanje aplikacije brez nameščanja. Za Eclipse smo se odločili zato, ker je brezplačen, podpira širok nabor razširitev in vtičnikov, ki olajšajo programiranje in je vanj enostavno vključiti katerikoli aplikacijski strežnik.

Za razvoj odjemalca smo uporabili Eclipse IDE z vtičnikom ADT. Z namestitvijo tega vtičnika smo lahko uporabili emulator, s pomočjo katerega zaženemo, razhroščujemo in testiramo aplikacije. V času razvoja mobilne aplikacije 90 % testiranja opravimo ravno na emulatorju in dejanskih mobilnih naprav ne uporabljamo. Zgled emulatorja je prikazan na sliki 3.8. V desnem delu okna emulatorja se nahaja celoten nabor možnih gumbov za izvajanje akcij tako z uporabo miške kot tudi tipkovnice. Levi del okna emulatorja vsebuje ekran, kjer se prikazuje izvajanje aplikacij. Vsakemu emulatorju lahko v upravljalniku AVD (angl. Android Virtual Device) določimo njegove strojne in programske lastnosti.


Slika 3.8: Okno emulatorja z OS Android 2.2

### 3.2.3 JSON

JSON (angl. JavaScript Object Notation) je preprost format za podatkovno izmenjavo. Izhaja iz programskega jezika JavaScript in je namenjen predstavitvi enostavnih podatkovnih struktur. 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. JSON temelji na dveh strukturah:

- Zbirka parov ime/vrednost. V različnih jezikih je realizirana kot *objekt*, zapis, struktura.
- Urejen seznam vrednosti. V večini jezikov je realiziran kot polje, vektor, *seznam* ali zaporedje. [13]

Primer: uporaba JSON v naši aplikaciji za pridobivanje prijav na izpitni rok:

```
{"procedure":"getExamApplications","parameters":{,"username":"63050150"}}
```

### 3.2.4 MySQL

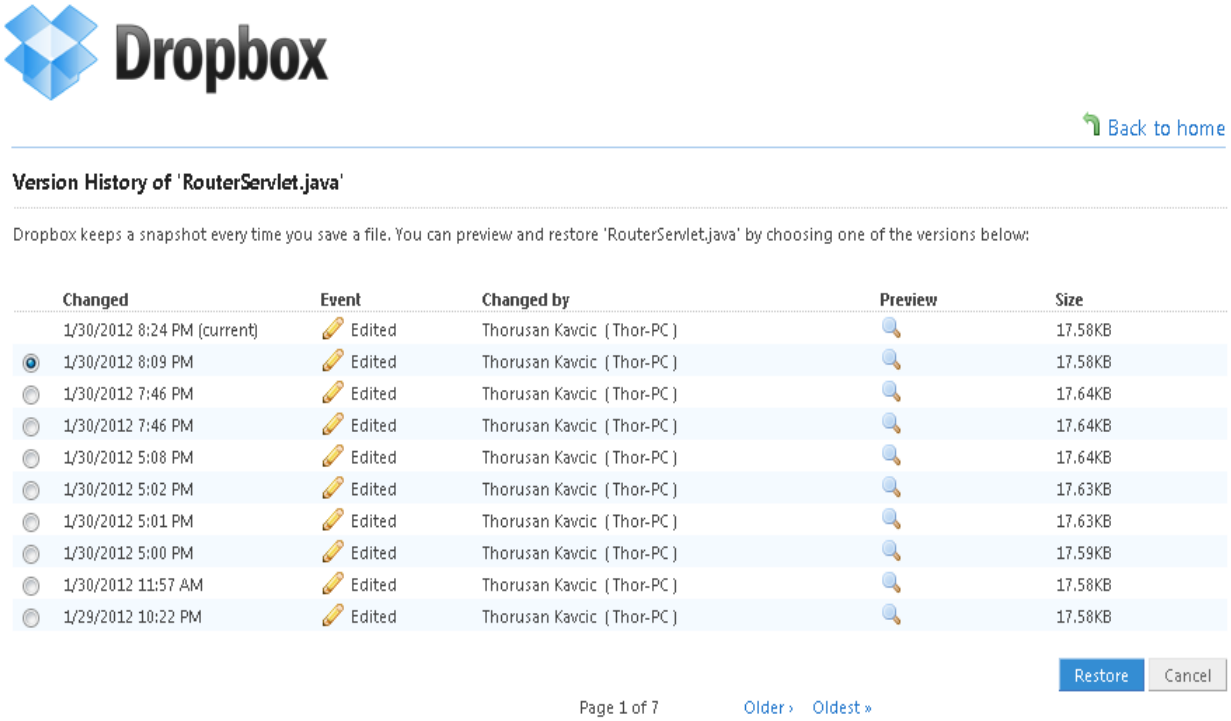
MySQL je brezplačen sistem za upravljanje s podatkovnimi bazami. MySQL je odprtokodna implementacija relacijske podatkovne baze, ki za delo s podatki uporablja jezik SQL. SQL je najbolj priljubljen jezik za dodajanje, dostop in urejanje vsebine podatkovne baze. Njegove odlike so: hitrost, zanesljivost, enostavnost uporabe in fleksibilnost. MySQL sistem deluje na principu odjemalec - strežnik, pri čemer lahko strežnik namestimo kot sistem, porazdeljen na več strežnikov. Sistem je sestavljen iz strežnika SQL, ki podpira različne odjemalske programe in knjižnice, administrativna orodja in velik razpon aplikacijskih programskih vmesnikov.

Uporabili smo tudi odprtokodno orodje MySQL Workbench za vizualno načrtovanje podatkovnih zbirk. Orodje v enotnem okolju za razvijalce in administratorje združuje modeliranje podatkov, ustvarjanje podatkovne zbirke, upravljanje sprememb in dokumentacijo.

### 3.2.5 Dropbox

Priljubljena spletna storitev v oblaku Dropbox omogoča enostavno deljenje datotek preko spleta s pomočjo sinhronizacije. Omogoča pa tudi hranjenje in pregledovanje več različic iste datoteke, kar je osnovna zahteva za kontrolo revizij. Pregledovanje različic v Dropboxu je prikazano na sliki 3.9. Večina modernih razvojnih projektov uporablja orodja za kontrolo verzij datotek izvorne kode (angl. revision control system), ker pa je bil naš projekt manjšega obsega in ker je na njem delal sam en razvijalec, smo si želeli enostavnega načina nadzora nad datotekami.

Za Dropbox smo se odločili zato, ker je brezplačen in enostaven za uporabo. Vse, kar je bilo potrebno narediti za kontrolo verzij datotek izvorne kode, je bila postavitev delovnega okolja v mapo, ki je pod Dropbox nadzorom. Ker se lahko na isti Dropbox račun prijavljamo iz kateregakoli računalnika in celo mobilnega telefona, je na ta način mogoče delo z istim delovnim okoljem s poljubne lokacije.

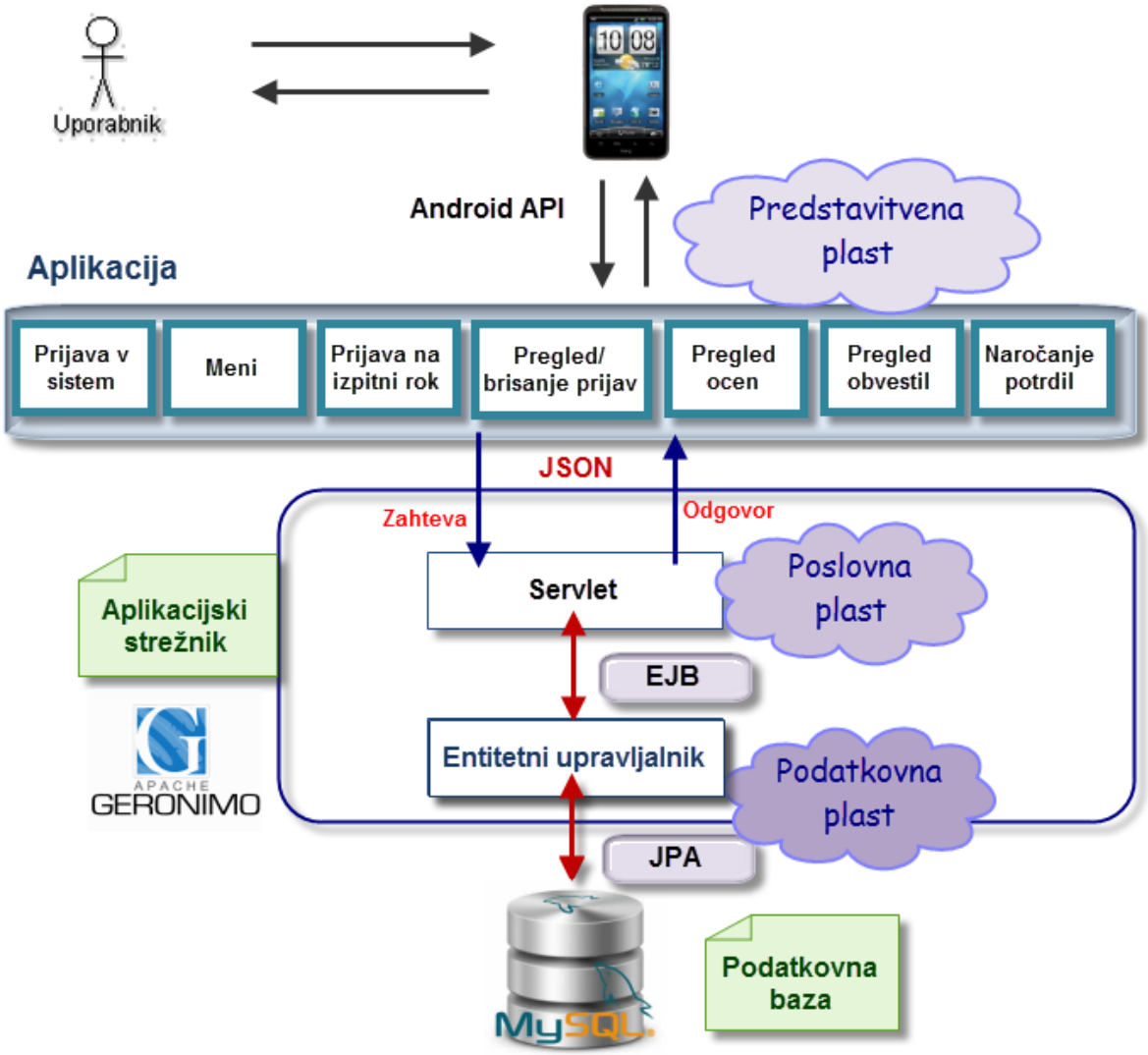



Slika 3.9: Primer sledenja verzijam datoteke.

# 4. Razvoj aplikacije

## 4.1 Arhitektura

Zgradbo same arhitekture rešitve najlažje predstavimo na sliki 4.1. Uporabnik neposredno komunicira samo z napravo, kjer teče sistem Android. Mobilna naprava je torej vez med uporabnikom in aplikacijo. Rešitev m-Študent je sestavljena iz odjemalca in strežnika. Odjemalec je aplikacija za mobilne naprave z operacijskim sistemom Android, strežnik pa je sestavljen iz aplikacijskega strežnika Apache Geronimo in podatkovne baze MySQL. Odjemalec in strežnik med seboj komunicirata preko povezave http, podatki med njima se prenašajo v formatu JSON.




Slika 4.1: Arhitekturna zasnova rešitve m-Študent.

Na sliki imamo pet ključnih komponent: uporabnika, mobilno napravo s sistemom Android, uporabniški del aplikacije, strežniški del aplikacije in podatkovno bazo. S puščicami je označena komunikacija med komponentami, ponekod pa so zraven tudi protokoli in druge lastnosti komunikacije.

Odjemalec, ki teče na mobilni napravi vsako zahtevo posreduje do strežnika, ki se na zahtevo primerno odzove. Strežnik po potrebi s pomočjo EJB naredi poizvedbo na podatkovni bazi, podatke obdela in vrne odgovor. Odgovor iz strežnika je lahko uspešen ali neuspešen. Odjemalec ga v vsakem primeru prebere in uporabniku odgovori na primeren način.

Mobilna aplikacija uporablja tristopenjsko arhitekturo in sicer:

- **predstavitvena plast:** predstavitveno plast (odjemalec) predstavljajo Androidove aktivnosti, ki skrbijo za prikaz podatkov, ki jim jih posreduje servlet.
- **poslovna plast:** logiko poslovne plasti predstavlja storitev EJB v navezi s servletom z imenom *RouterServlet*. EJB s pomočjo entitetnega upravljalnika (angl. entity manager) servletu dostavlja entitetna zrna, servlet pa izvaja obdelavo in izpis v formatu JSON posreduje aktivnostim.
- **podatkovna plast:** nalogo podatkovne plasti opravlja knjižnica JPA, ki skrbi za preslikavo entitetnih zrn v podatkovno bazo.

## 4.2 Odjemalec

### 4.2.1 Delovanje preproste aktivnosti

Pred začetkom pisanja diplomskega dela o razvoju na platformi nismo vedeli praktično ničesar, zato smo si zadali kot prvo nalogo spoznavanje z njo. Vendar pa lahko najdemo veliko dokumentacije na Androidovi spletni strani, ki je precej nazorna. Za začetek smo napisali vsem programerjem dobro poznano aplikacijo "Pozdravljen svet" (angl. Hello World), ki je prikazana na sliki. 4.2.

```
package com.example.helloandroid;

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

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

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

Slika 4.2: Programska koda aplikacije "Pozdravljen svet".

Kot smo že povedali v teoretičnem delu, lahko uporabniški vmesnik ustvarimo in prikažemo samo v komponenti *Activity*. Ta aplikacija vsebuje samo eno aktivnost, ki se imenuje HelloAndroid. V splošnem mora vsaka aktivnost razširjati (angl. extends) oz. dedovati razred *Activity*.

V aplikacijo najprej uvozimo (ang. import) vse potrebne knjižnice. V razredu HelloAndroid je ključnega pomena metoda **onCreate(Bundle savedInstanceState)**, 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 aktivnosti 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* to besedilo prikažemo v uporabniškem vmesniku te aktivnosti.


Slika 4.3: Zagnana aplikacija “Pozdravljen svet”.

Na slikah 4.2 in 4.3 smo postavitev uporabniškega vmesnika določili programsko. To pomeni, da smo vmesnik zgradili kar v programski kodi samega programa. Takšen pristop ni dober, ker majhne spremembe v postavitvi lahko povzročijo korenite spremembe same aplikacije. Zaradi tega Android omogoča načrtovanje uporabniškega v vmesnika v tako imenovanih postavitvenih datotekah XML (angl. XML layout file).

Splošna struktura postavitvene datoteke je enostavna: je drevo XML elementov, kjer je vsako vozlišče ime razreda View in njegovih podrazredov. Takšna struktura hitro in enostavno gradnjo uporabniških vmesnikov. Model izhaja iz spletnega razvoja, kjer sta predstavitev (uporabniški vmesnik) ter logika aplikacije ločeni.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">

        <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Hello, World!"
        android:textAppearance="?android:attr/textAppearanceMedium"/>

</LinearLayout>
```

Slika 4.4: Primer postavitvene datoteke.

V primeru na sliki 4.4 imamo v postavitveni datoteki samo en element – TextView, ki bo prikazoval naše besedilo. Element ima štiri atribute, ki jih ne bomo podrobneje razlagali. Pomemben atribut za nas je **android:text**, ki skrbi za prikaz našega besedila. Potrebno je spremeniti še metodo *onCreate*, kjer se ne bo več dinamično kreiral objekt TextView in se mu ne nastavi besedila. Vse, kar je potrebno v tem primeru narediti, je določitev postavitvene datoteke za prikaz v uporabniškem vmesniku. Popravljena metoda *onCreate* je prikazana na sliki 4.5.

```
public void onCreate(Bundle savedInstanceState)
{
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
}
```

Slika 4.5: Aplikacija “Pozdravljen svet” z uporabo postavitvene datoteke.

### 4.2.2 Delovanje odjemalca

Odjemalec za delovanje potrebuje mobilno napravo z operacijskim sistemom Android, katerega verzija mora biti vsaj 2.2. Aplikacija deluje na različnih resolucijah zaslona. Ker se povezuje na strežnik, nujno potrebuje delujočo internetno povezavo.

Obstaja več različic operacijskega sistema Android, zato smo se pri razvoju odjemalca odločili za verzijo 2.2, ki je trenutno ena najbolj uporabljenih verzij. Na sliki 3.2 vidimo, da ima verzija 2.2 kar 27,8 % delež na trgu. Trenutno je večina mobilnih naprav opremljena z vsaj to verzijo operacijskega sistema. Zaradi dobre arhitekture operacijskega sistema Android pa aplikacija brez večjih problemov deluje tudi na napravah, ki imajo novejšo različico Androida.

### 4.2.3 Uporaba

Diagram primerov uporabe oziroma UML diagram (angl. Unified Modeling Language) je prikazan na sliki 4.7. Uporabnik se najprej v sistem prijavi z uporabniškim imenom in geslom. Aplikacija nato preveri veljavnost vpisanih podatkov. V primeru kakršnekoli napake, uporabnika o tem tudi opozori. Če je uporabnik vpisal prave podatke, mu aplikacija dovoli dostop do glavnega oziroma opcijskega menija aplikacije, ki ga lahko vidimo na sliki 4.6.

V opcijskem meniju ima uporabnik oziroma v našem primeru študent več možnosti. Uporabnikova prva možnost je, da se prijavi na izpit. S pritiskom na gumb »Prijava na izpit« pride v ospredje aktivnost, ki uporabniku omogoča prijavljanje na izpitne roke. V novem oknu lahko študent izbere predmet in izpitni rok na katerega se bo prijavil. Predmet je opisan z imenom predmeta in profesorjem oziroma nosilcem, ki predmet izvaja. Ko študent izbere predmet, ima v spustnem meniju (angl. drop-down menu) za izbiro izpitnega roka na voljo več izpitnih rokov. Vsak izpitni rok je določen z datumom, časom in lokacijo oziroma predavalnico v kateri se piše izpit. Uporabnik se s pritiskom na gumb »Prijavi« prijavi na izpitni rok.

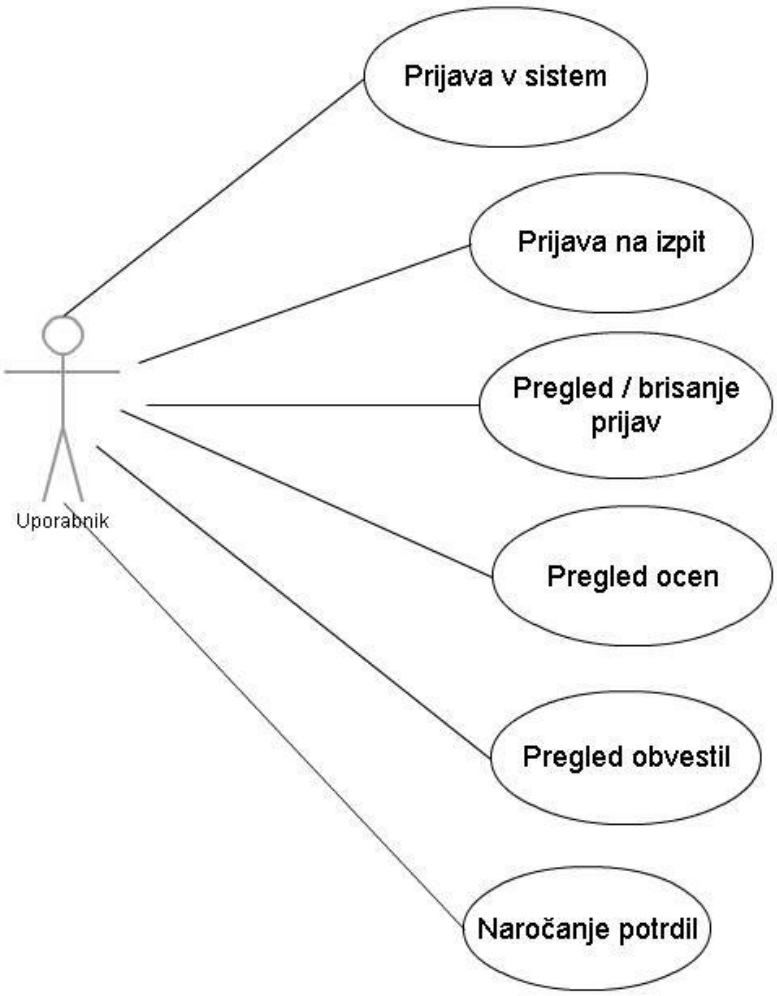
V primeru da se uporabnik odloči, da se bo odjavil od izpitnega roka, mu ob pritisku na gumb »Pregled/brisanje prijav« aplikacija prikaže seznam izpitnih rokov, na katere se je že prijavil. Vsak izpitni rok predstavlja gumb, ki ima na sebi napisano ime predmeta, datum, čas in lokacijo izpita. S pritiskom na gumb se uporabniku v novem oknu izpišejo podrobnosti izpitnega roka. Čisto na dnu stoji gumb »Odjava« s katerim se študent odjavi od izpita. Z dotikom na gumb, se pojavi dialog, ki uporabnika vpraša, če se ta želi res odjaviti od izpita. Na voljo sta 2 možnosti. Uporabnik pritisne na gumb »OK« in se tako odjavi od izpita, aplikacija ga premakne nazaj v opcijski meni. Lahko pa se uporabnik odloči za drugo možnost in s pritiskom na gumb »Cancel« prepreči odjavo od izpitnega roka in se vrne nazaj na

seznam prijavljenih izpitnih rokov. V primeru da študent ni prijavljen na noben izpitni rok, se mu izpiše opozorilo: »Trenutno niste prijavljeni na noben izpit!«.


Slika 4.6: Opcijska aktivnost aplikacije m-Študent.

Uporabnik lahko s pritiskom na gumb »Pregled ocen« preveri, kakšno ima oceno za vsak predmet in kakšna je skupna povprečna ocena. Podobno lahko uporabnik z dotikom na gumb »Obvestila« pogleda, če mu je profesor, asistent ali študentski referat poslal obvestilo. Sistem poišče vsa uporabnikova sporočila in mu odpre seznam različnih obvestil. Vsako obvestilo vsebuje ime pošiljatelja, datum in uro oziroma čas poslanega obvestila.

Zadnja uporabnikova možnost je naročanje potrdil. Naroči lahko potrdila o opravljenih izpitih ali potrdila o vpisu. S pomočjo spustnega menija lahko uporabnik naroči od 1 do 5 potrdil. Količina naročanja vsake vrste potrdil je omejena na pet potrdil na mesec.




Slika 4.7: Diagram primerov uporabe odjemalca.

### 4.2.4 Arhitektura

Odjemalec je zgrajen iz datotek, ki vsebujejo istoimenske razrede in jih lahko vidimo v 1.stolpcu tabele 2. Kot smo že povedali, vsaki aktivnosti pripada tudi postavitvena datoteka (angl layout file), ki skrbi za izgled odjemalca in ima končnico .xml.

| Datoteka z izvorno kodo (razred oz. aktivnost) | Postavitvena datoteka |
|---|---|
| ApplyExamListActivity.java | apply_exam_list.xml |
| ExamApplicationsInfoActivity.java | exam_applications_info.xml |
| ExamApplicationsReviewActivity.java | exam_applications_review.xml |
| GradesAverageActivity.java | grades_average.xml |
| GradesMobileTabActivity.java | grades_mobile_tab.xml |
| GradesSubjectsActivity.java | grades_subjects.xml |
| MainScreenLoginActivity.java | main_screen_login.xml |
| NoticesActivity.java | notices.xml |
| OrderConfirmationsActivity.java | order_confirmations.xml |
| StudentOptionsActivity.java | student_options.xml |

Tabela 2: Datoteke, ki sestavljajo odjemalca.

Pri odjemalcu sta pomembna še 2 razreda, in sicer:

- **SessionManager.java** – skrbi za sejo
- **RequestHandler.java** – razred, ki skrbi za zahteve (angl. requests) in jih pošilja do aplikacijskega strežnika

Najpomembnejše datoteke in razredi so opisani v nadaljevanju.

- **RequestHandler**

Razred *RequestHandler* skrbi za pošiljanje zahtev aplikacijskemu strežniku. Na splošno je narejeno, da aktivnosti mobilnega odjemalca kličejo metode razreda rokovalca RequestHandler, ki pošlje zahtevo http aplikacijskemu strežniku. Aplikacijski strežnik naredi potrebne poizvedbe in obdelavo podatkov ter odgovor vrne nazaj rokovalcu *RequestHandler*.

V primeru prijave uporabnika v sistem je narejeno tako, da aktivnost *MainScreenLoginActivity* kliče metodo *loginSend*. V metodi *loginSend* se najprej naredi JSON objekt, v katerem se določi ime zahtevane procedure in potrebni parametri. Pri drugih metodah, ki jih kličejo druge aktivnosti je princip isti, drugačni so samo parametri in ime procedure. V metodi *LoginSend* imamo ime procedure prijava (angl. login), parametra pa sta

uporabniško ime (angl. username) in geslo (angl. password), ki ju je uporabnik vpisal v vnosni polji »Vpisna številka« in »Geslo«. Pred pošiljanjem zahteve na strežnik se JSON objekt šifrira s standardnim **Base64** kodiranjem.

- **JSON struktura <u>zahteve</u> (login request):**

```
{ "parameters" :
        { "password" : "geronimo867",
          "username" : "63050150"
        },
  "procedure" : "login"
}
```

- **JSON struktura <u>odgovora</u> (login response):**

```
{ "returnCode" : 0,
  "returnMessage" : "Login uspešen!"
}
```

- **MainScreenLoginActivity**

Aktivnost *MainScreenLoginActivity* skrbi za prvo aktivnost s katero se sreča uporabnik. Sestavljen je iz več metod. Najpomembnejši pa sta *onCreate* in *onClick*. V metodi *onCreate* se, kot v vseh ostalih aktivnostih aplikacije, najprej nastavi postavitvena datoteka oz. uporabniški vmesnik s klicem funkcije *setContentView*. Če uporabnik dvakrat pritisne v vnosni polji za vnos vpisne številke in gesla, se mu vsebina vnosnega polja izbriše. Ob enem pritisku na vnosno polje pa polje dobi fokus in uporabnik lahko vnese podatke.

Metoda *onClick* se sproži ob pritisku na gumb Prijava. Gumb ima namreč definiranega poslušalca z imenom *setOnClickListener*, ki preverja kdaj uporabnik pritisne na ta gumb. V metodi *onClick* se najprej preveri, če je mobilna naprava povezana v internetno omrežje, sicer uporabnika z opozorilnim dialogom (angl. alert dialog) obvesti, da ni povezan z internetom.

Potem metoda razreda *RequestHandler* z imenom *LoginSend* preveri, če sta polji za vnos uporabniškega imena in gesla pravilno izpolnjeni. Metoda posreduje zahtevo do aplikacijskega strežnika. Strežnik naredi potrebne poizvedbe in vrne odgovor metodi. Metoda *LoginSend* vrne **boolean** vrednost resnično (angl. true) ali neresnično (angl. false). V primeru, da uporabnik ni vnesel pravih podatkov (vrnjena vrednost: neresnično), mu aplikacija napiše, da je vnesel napačno ime ali geslo. Če pa uporabnik vnese pravilne podatke (vrnjena vrednost: resnično) se mu prikaže dialog z napisom, da se je uspešno prijavil v sistem in ga aplikacija premakne v aktivnost **StudentOptionsActivity**.

V primeru uspešne prijave se vzpostavi seja in se z metodo razreda ***SessionManager*** z imenom *setUsername* nastavi atribut uporabniško ime (angl. username), ki je v bistvu enolično določena vpisna številka študenta. S tem dosežemo, da lahko potem kasneje kjerkoli v aplikaciji identificiramo trenutnega uporabnika z metodo *getUsername*.


Slika 4.8: Prijavno okno mobilne aplikacije.

- **StudentOptionsActivity**

To aktivnost prikazuje slika 4.6. Aktivnost vsebuje 5 gumbov, od katerih ima vsak registriranega poslušalca za pritisk na gumb. Ko uporabnik pritisne na gumb, se izvrši metoda *onClick*, ki uporabnika premakne v primerno aktivnost. Možne aktivnosti so:

- ApplyExamListActivity (prijava na izpitni rok)
- ExamApplicationsReviewActivity (pregled in brisanje prijav)
- GradesMobileTabActivity (pregled ocen)
- OrderConfirmationsActivity (pregled obvestil)
- NoticesActivity (naročanje potrdil)

- **ApplyExamListActivity**

Uporabnik začne uporabljati razred ApplyExamListActivity, ko iz opcijskega menija preide na seznam predmetov in izpitnih rokov. V metodi onCreate se nastavi uporabniški vmesnik in pokliče metoda razreda RequestHandler z imenom *requestSubjects*, ki poskrbi zato, da se seznam predmetov naloži v Androidovo komponento **spinner** oziroma spustni meni z imenom spinnerPredmet. V tem primeru kot odgovorimo dobimo JSON tabelo oziroma polje (angl. JSON array), ki vsebuje seznam predmetov. JSON struktura za zahtevo in odgovor:

- **JSON struktura <u>zahteve</u> za pridobivanje predmetov:**

```
{ "parameters" : { "username" : "63050150" },
  "procedure" : "getSubjects"
}
```

- **JSON struktura odgovora za pridobivanje predmetov:**

```
{ "result" : [
    { "creditPoints" : 8,
      "lecturer" : "doc. dr. Mojca Ciglarič",
      "name" : "Računalniške komunikacije",
      "subjectId" : 2401
    },
    { "creditPoints" : 4,
      "lecturer" : "dr. Gasper Fijavž",
      "name" : "Diskretne Strukture",
      "subjectId" : 2601
    },
  ],
 "returnCode" : 0,
 "returnMessage" : "Good good"
}
```

Zelo podobno je pri pridobivanju izpitnih rokov. Spustni meni spinnerPredmet ima nastavljenega poslušalca *onItemSelected*, ki zazna spremembo, ko uporabnik nekaj izbere v tem spustnem meniju. Ob spremembi predmeta v tem spustnem meniju se pokliče metoda *fillExams*, ki za parametre dobi enolično določeno identifikacijsko številko predmeta in trenutnega uporabnika. Tokrat se kliče metoda razreda RequestHandler, ki se imenuje *requestExams*. Metoda poskrbi, da se v spustni meni spinnerIzpitniRok naložijo vsi možni izpitni roki za trenutno izbrani predmet. JSON struktura za zahtevo in odgovor:

- **JSON struktura zahteve za pridobivanje izpitnih rokov:**

```
{ "parameters" :
   { "chosenSubject" : "2401",
     "username" : "63050150"
   },
 "procedure" : "getExams"
}
```

- **JSON struktura odgovora za pridobivanje izpitnih rokov:**

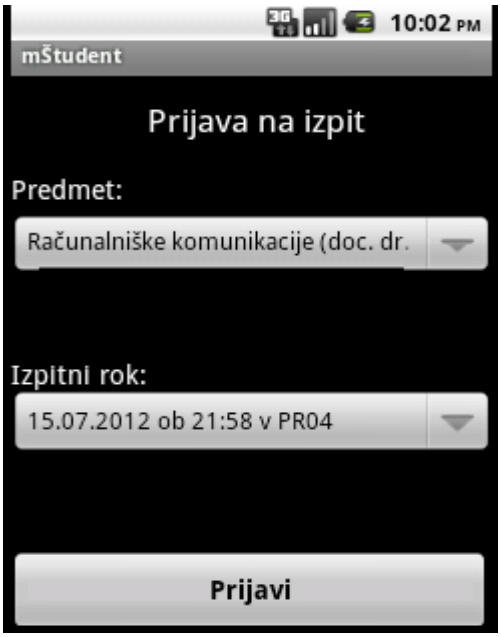
```
{ "result" : [
    { "date" : 1341051938000,
      "examId" : 8201,
      "location" : "PR03"
    },
    { "date" : 1342382328000,
```

```
      "examId" : 2901,
      "location" : "PR04"
    }
  ],
 "returnCode" : 0,
 "returnMessage" : "Good good"
}
```


Na koncu je pomemben gumb »Prijavi« in njegova metoda onClick, ki uporabnika prijavi na izpitni rok. Metoda najprej pridobi identifikacijsko številko predmeta in izpitnega roka na katerega se uporabnik želi prijaviti. Nato pokliče metodo razreda RequestHandler z imenom *applyForExam*, ki za parametre dobi vpisno številko uporabnika in malo prej omenjeni identifikacijski številki. V primeru, da je uporabnik na ta predmet že prijavljen, se mu prikaže opozorilo in se mu izpiše, da je na ta predmet že prijavljen. Uporabnik je namreč lahko za določen predmet prijavljen samo na en izpitni rok naenkrat. V nasprotnem primeru se uporabniku izpiše, da je bila prijava uspešna. Slika 4.9 prikazuje to aktivnost.

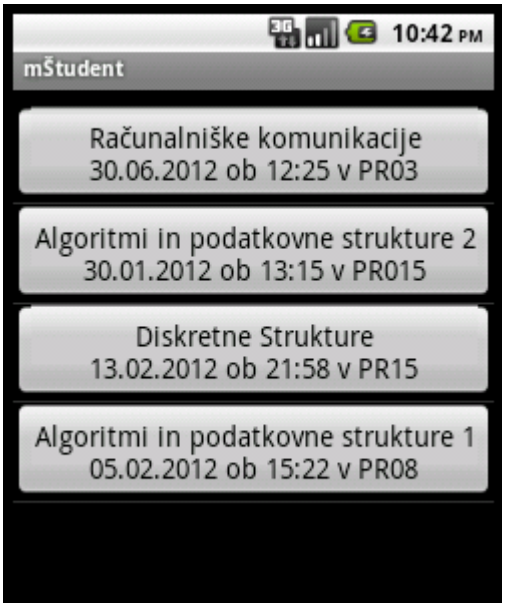
- **ExamApplicationsReviewActivity**

Aktivnost ExamApplicationsReviewActivity je zadolžena za prikaz že vnesenih prijav. Najprej se v metodi onCreate pokliče metoda *requestExamApplications*, ki dobi seznam prijav. Nato se za vsako prijavo ustvari gumb z imenom izpita, datumom, uro in lokacijo izpitnega roka. Za kreiranje seznama smo naredili seznam (angl. listView) po meri, ki vsebuje gumbe. Naredili smo še poseben razred ***ViewHolder***, ki vsak gumb ustvari in mu določi trenutno pozicijo. To omogoča, da lahko kasneje ob prehodu v novo aktivnost dobimo podatke od vsakega izpitnega roka posebej.




Slika 4.9: Prijava na izpitni rok.

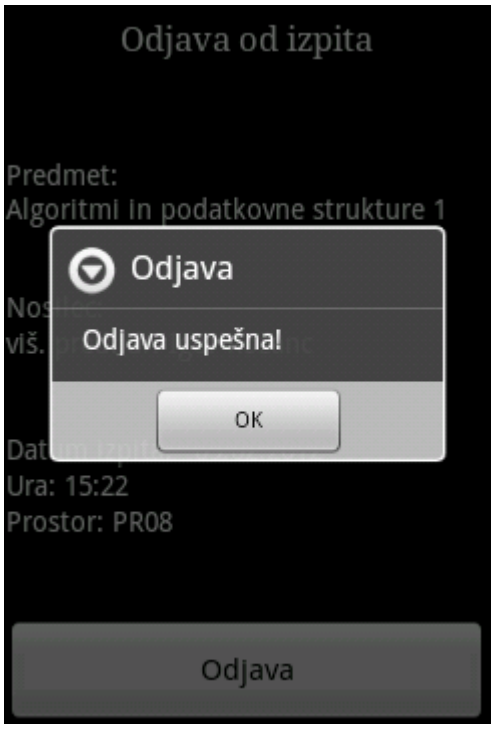
Vsakemu gumbu smo dodali poslušalec *onClickListener*, ki posluša pritiske na gumb. Ob pritisku na gumb uporabnika premakne v aktivnost *ExamApplicationsInfoActivity*, ki je opisana v nadaljevanju. V novo aktivnost se prenesejo tudi podatki izbranega izpitnega roka. Seznam prijav lahko vidimo na sliki 4.10.




Slika 4.10: Seznam prijav na izpite.

- **ExamApplicationsInfoActivity**

Ko pride v ospredje ta aktivnost, uporabnik najprej vidi podatke izpitnega roka, od katerega se želi odjaviti. Gumb »Odjava« ima zopet registriranega poslušalca, torej ob pritisku na gumb najprej aplikacija s pomočjo opozorilnega uporabnika vpraša, če se res želi odjaviti od izbranega izpita. Če uporabnik klikne na gumb »Cancel« ga aplikacija vrne nazaj v aktivnost s seznamom prijav. Pri drugi možnosti, ko uporabnik klikne na gumb »OK« pa se kliče metodo razreda RequestHandler z imenom *removeExamApplication*, ki uporabnika odjavi od izbranega izpitnega roka. Uporabniku se prikaže opozorilni dialog z napisom, da je bila odjava uspešna. Aplikacija potem premakne uporabnika nazaj v opcijski meni. Primer uspešne odjave lahko vidimo na sliki 4.11.




Slika 4.11: Prikaz aktivnosti, ki skrbi za odjavo od izpitnega roka.

- **GradesMobileTabActivity:**
  - **GradesAverageActivity**
  - **GradesSubjectsActivity**

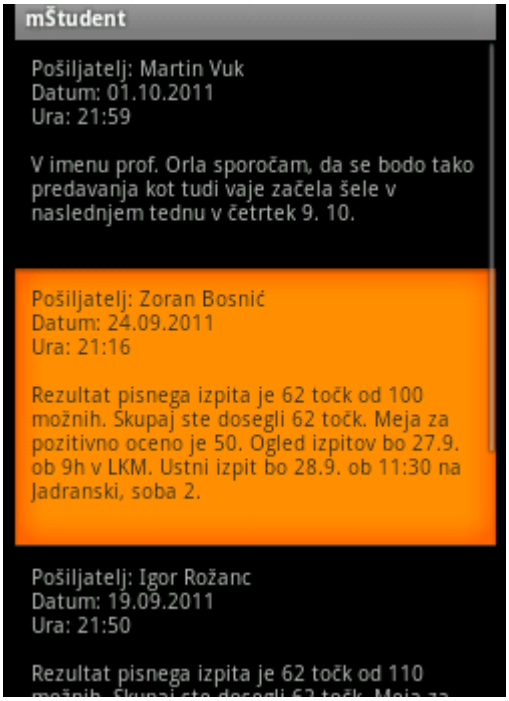
Vse 3 aktivnosti skrbijo za prikaz ocen. Razred *GradesMobileTabActivity* v tem primeru razširja razred *TabActivity*, ker je to aktivnost z zavihki. V metodi onCreate se najprej kot vedno določi postavitvena datoteka, nato pa se določi še zavihke. Prvi zavihek predstavlja aktivnost *GradesAverageActivity* in prikazuje ocene po predmetih, drugi zavihek pa razred *GradesSubjectsActivity*, ki prikazuje povprečno oceno vseh predmetov. Ustvarimo objekt razreda *TabHost* in vanj dodamo obe aktivnosti. Razred *TabHost* je v bistvu vsebnik (angl. container) za aktivnosti z zavihki.

Obe aktivnosti kličeta metodo razreda RequestHandler z imenom *requestGrades*, ki za parametre dobi uporabniško ime trenutnega uporabnika. Aktivnost *GradesSubjectsActivity* dobi JSON tabelo in v seznam doda ime predmeta ter oceno. Vse to se potem izpiše v seznamu, ki ima tudi drsnik, in sicer v primeru, da je seznam ocen predolg. *GradesAverageActivity* izračuna povprečno oceno vseh predmetov in jo prikaže. Na sliki 4.12 lahko vidimo kako aktivnost *GradesSubjectsActivity* prikaže ocene.


Slika 4.12: Prikaz ocen s pomočjo aktivnosti GradesMobileTabActivity**.**

- **NoticesActivity**

Aktivnost najprej v metodi onCreate pokliče metodo *requestNotices*, ki tudi tokrat za argument dobi uporabniško ime trenutnega uporabnika. Metoda vrne JSON tabelo obvestil. Vsako obvestilo ima vsebino, pošiljatelja, uro in datum poslanega obvestila. Vsa obvestila potem v metodi onCreate izpišemo v komponenti seznam (angl. ListView), ki vključuje tudi drsnik (angl. scrollview) za premikanje po obvestilih. Primer obvestil lahko vidimo na sliki 4.13.




Slika 4.13: Prikaz obvestil.

- **OrderConfirmationsActivity**

Podobno kot pri prijavi na izpitni rok, sta tudi v tej aktivnosti dva spustna menija (angl. spinner). Aktivnost skrbi za naročanje potrdil in sicer v prvem spustnem meniju uporabnik lahko izbere potrdilo o opravljenih izpitih ali potrdilo o vpisu. V drugem spustnem meniju pa uporabnik lahko izbere količino naročenih potrdil. Ob pritisku na gumb »Naroči« se kliče metoda onClick, ki pokliče metodo razreda RequestHandler z imenom *setConfirmations*, ki poskrbi za prenos podatkov do aplikacijskega strežnika, ki podatke obdela in vnese v podatkovno bazo.

## 4.3 Strežnik

Odjemalec se preko protokola http povezuje z aplikacijskim strežnikom, mu pošilja zahteve in si z njim izmenjuje podatke v JSON formatu. Strežnik preko EJB komunicira s podatkovno bazo in odgovarja na zahteve odjemalca.

### 4.3.1 Delovanje

Strežnik je aplikacija, napisana v programskem jeziku Java. Glavni del strežnika predstavlja servlet oziroma tako imenovani strežniški programček.
Minimalne zahteve za delovanje:

- operacijski sistem Windows 7 ali Linux,
- aplikacijski strežnik Apache Geronimo, verzije 2.2 ali višje,
- programski jezik Java, verzije 1.6 ali višje,
- podatkovno bazo MySQL, verzije 5.1.59 ali višje.

### 4.3.2 Uporaba

Odjemalec uporablja strežnik za pridobivanje podatkov iz podatkovne baze in pripravo v primeren format. Odjemalec za dostop do različnih podatkov metode razreda RequestHandler, ki so bile omenjene že prej pri aktivnostih. Za dostop do strežnika moramo klicati URI:
http://mstudent.servehttp.com:8080/mStudentWAR/RouterServlet?requestString*="jsonObjekt"*
Spremenljivka **jsonObjekt** se spreminja glede na to kakšne podatke želimo od strežnika.

### 4.3.3 Arhitektura

Glavni del strežnika predstavlja servlet oz. tako imenovani strežniški programček z imenom ***RouterServlet***. Na strežniški (servlet) strani metoda service, najprej dekodira parameter zahteve, ki je bil šifriran s standardnim **Base64** kodiranjem. Z dekodiranjem dobimo JSON objekt iz katerega izluščimo ime zahtevane procedure in njene parametre. Ko je to znano, se pokliče metoda entitetnega rokovalca (angl. entity handler), ki pokliče metodo razreda *StudentEJBBean*, ki dejansko naredi potrebne poizvedbe na podatkovni bazi. Razred StudentEJBBean je v bistvu implementacija EJB rokovalca (angl. EJB handler).

V primeru da želimo dobiti seznam predmetov, servlet preusmeri zahtevo na metodo entitetnega rokovalca *SubjectsHandler* z imenom *getSubjectsData*, ki vrne JSON tabelo

predmetov. V tej metodi se kliče metoda EJB rokovalca, ki za argument dobi vpisno številko trenutnega uporabnika. Metoda *getSubjects* najprej dobi povezavo z entitetnim upravljalnikom (angl. entity manager) s klicem metode *createEntityManager*.. Z njegovo metodo *createQuery* lahko tako naredimo poizvedbo s pomočjo že omenjenega jezika JPQL. Kot rezultat dobimo seznam predmetov, na katere se lahko uporabnik prijavi. Na koncu še zapremo povezavo z entitetnim upravljalnikom. Izvorno kodo metode getSubjects se lahko vidi na sliki 4.14.

```
public JSONArray getSubjects(long studentId)    // get subjects for student specified with student id
{
      EntityManager em = emfStudent.createEntityManager();
      JSONArray subjectsArray = new JSONArray();

      try {
            Student student = em.find(Student.class, studentId);  // find student with specified id

            Query q = em.createQuery("SELECT x FROM Subject x WHERE x.yearCourse.yearCourseId = :idParam");
            q.setParameter("idParam", student.getYearCourse().getId());

            List<Subject> results = q.getResultList ();

            for(Subject subject : results)
            {
                  subjectsArray.put(subject.toJSONObject());      // put notice to JSON array
            }
            return subjectsArray;
      } catch(Exception e)    {
                  System.out.println(e.getMessage());
                  return new JSONArray();
      } finally {
                  em.close();
}
```

Slika 4.14: Izvorna koda metode getSubjects.

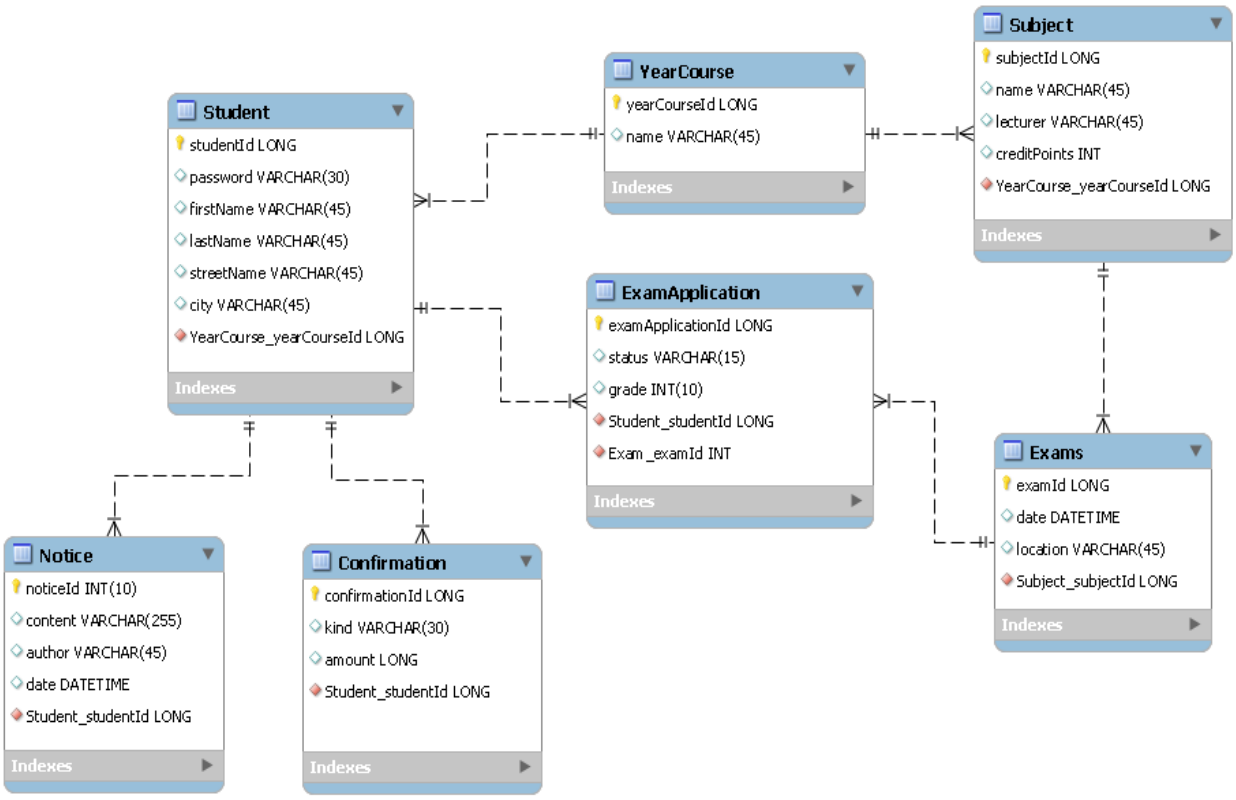
Pomembni so še ostali entitetni rokovalci:

- **ApplyExamHandler** – skrbi za prijavo uporabnika na izpitni rok
- **ConfirmationsHandler** – skrbi za pridobivanje obvestil
- **ExamApplicationsHandler** – skrbi za pridobivanje seznama prijav
- **ExamsHandler** – skrbi za pridobivanje izpitnih rokov
- **GradesGandler** – skrbi za pridobivanje ocen
- **LoginHandler** – preveri identiteto uporabnika pri prijavi v sistem
- **NoticesHandler** – skrbi za pridobivanje seznama obvestil
- **RemoveExamApplicationHandler** – skrbi za brisanje prijave iz podatkovne baze

## 4.4 Podatkovna baza

Podatkovna baza, ki je predstavljena na sliki 4.15, je sestavljena iz 7-ih tabel. Tabele so med seboj povezane z različnimi relacijami. Tabela *Student* vsebuje podatke o študentih in sicer njegovo geslo, ima, priimek, naslov ulice in ime mesta, v katerem živi. Tabela Notice shranjuje obvestila za vsakega študenta posebej. Vsako obvestilo ima vsebino, avtorja, datum in čas poslanega obvestila. Študent lahko naroči tudi dve različni vrsti potrdil (tabela *Confirmation*) in največ 5 potrdil vsake vrste na mesec.

Tabela *YearCourse* opisuje v katerem letniku je študent. V vsakem letniku je lahko več študentov. Za vsak letnik obstaja več predmetov, ki so definirani v tabeli *Subject*. Predmet je opisan z imenom, predavateljem in številom kreditnih točk. Vsak predmet pa ima lahko tudi več izpitnih rokov (tabela *Exams*), vsakega z različnim datumom, časom in lokacijo. Študent se na izpitni rok lahko prijavi s prijavnico (tabela ExamApplications), ki vsebuje podatke o študentu, izpitnem roku, statusu prijave in končni oceni za to prijavo.




Slika 4.15: Struktura podatkovne baze.

# 5. Sklepne ugotovitve

V diplomski nalogi smo izdelali mobilno aplikacijo m-Študent. To je aplikacija spisana za uporabo v mobilnih napravah z operacijskim sistemom Android, skupaj z lastnim strežniškim delom, ki skrbi za hranjenje in posredovanje podatkov.

Glede na zadane cilje nam je uspelo uspešno razviti skoraj celotno funkcionalnost. Z mobilno aplikacijo se študent lahko prijavi v sistem, prijavi ali odjavi z izpitnega roka, pregleduje ocene in obvestila ter naroča potrdila. Aplikacija preverjeno deluje tudi na drugih mobilnih napravah z različnimi velikosti zaslonov.

Podatkovno bazo smo morali razviti sami, ker nam tekom izdelave diplomske naloge ni uspelo pridobiti dostopa do podatkovne baze e-Študenta. Trenutna rešitev deluje na aplikacijskem strežniku Apache Geronimo in podatkovni bazi MySQL. Če bi želeli, da bi bila aplikacija uporabna, bi jo morali povezati s podatkovno bazo e-Študenta. Tako bi odjemalec prikazoval realne podatke in bi se lahko študenti resnično prijavili na izpit ter izkoriščali še druge možnosti, ki jih rešitev ponuja. Da bo morebiten prehod na bazo e-Študenta čim bolj enostaven, smo rešitev razvili tako, da odjemalca ni potrebno spreminjati, ampak se spremeni samo strežniški del, ki skrbi za SQL poizvedbe na podatkovno bazo.

Aplikacija trenutno podpira prijavo samo z uporabniškim imenom in geslom. Priložnost za izboljšavo vidimo v tem, da bi se v morebitnem nadaljnjem razvoju rešitve spremenilo aplikacijo tako, da bi podpirala tudi prijavo s certifikatom, kot je to že narejeno v sistemu e-Študent. Tako bi bila uporaba aplikacije veliko bolj varna. Možnost izboljšave bi bila tudi, da bi aplikacija študenta za vsak izpitni rok pravočasno obvestila, do kdaj se še lahko prijavi na izpitni rok. Lahko bi naredili tudi sinhronizacijo izpitnih rokov s koledarjem na mobilni napravi. Tako bi uporabnik točno vedel, kdaj ima določene izpitne roke.

# Seznam slik

# Literatura:

[1] (2010) Leta 2010 prenesenih 10,9 milijarde mobilnih aplikacij; dostopno na: http://www.ris.org/db/26/11898/Novice.

[2] (2011) Android drži v rokah kar 52,2-odstotni tržni delež; dostopno na: http://www.racunalniske-novice.com/novice/mobilna-telefonija/dogodki-in-obvestila/android-drzi-v-rokah-kar-polovico-trznega-deleza.html.

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

[4] (2012) Prijava na izpite; dostopno na: http://www.fri.uni-lj.si/si/izobrazevanje/prijava_na_izpite.

[5] (2011) O uvajanju računalniško podprte priprave urnikov in prijav na izpite; dostopno na: http://www.ff.uni-lj.si/hp/ff/urniki.html.

[6] (2012) Informacije o digitalnih potrdilih; dostopno na: http://estudent.fri.uni-lj.si/Navodila_prijava.html.

[7] (2012) Kaj je Android?; dostopno na: http://slo-android.si/prispevki/kaj-je-android.html.

[8] (2012) Android (operating system); dostopno na: http://en.wikipedia.org/wiki/Android_operating_system.

[9] (2012) What is Android?; dostopno na: http://developer.android.com/guide/basics/what-is-android.html.

[10] Burnette E., Hello, Android 3rd edition. Raleigh, Dallas: Pragmatic Programmers LLC, 2010.

[11] (2012) Java EE; dostopno na: http://docs.oracle.com/javaee/.

[12] Enterprise JavaBeans; dostopno na: http://en.wikipedia.org/wiki/Enterprise_JavaBeans.

[13] JSON; dostopno na:
http://json.org/json-sl.html.