UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Peter Zlatnar

MOŽNOSTI PRENOVE SPLETNEGA SISTEMA FRI S TEHNOLOGIJO RUBY ON RAILS

Diplomska naloga
na visokošolskem strokovnem študiju

Mentor: izr. prof. dr. Blaž Zupan

Ljubljana, 2008

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

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

Zahvala

Kot prvemu se moram zahvaliti mentorju izr. prof. dr. Blažu Zupanu. Ker me je sprejel pod svoje okrilje, ter nato pomagal z nasveti in idejami pri izdelavi diplomske naloge. Zahvaljujem se mu tudi za potrpežljivost pri pregledovanju in popravljanju naloge, ter opozarjanju na marsikatero napako.

Posebna zahvala gre mojima staršema Dušanu in Ireni, ter punci Romani za izkazano podporo in potrpežljivost v času študija.

Zahvalil bi se tudi Mihu Keršiču, Mihu Bradaču, Mateju Grabnarju, Bojanu Petkovšku, Dejanu Stoparju in Vasji Femec s katerimi smo skupaj premagovali študijske obveznosti.

Hvala tudi vsem, ki so kakorkoli pripomogli k nastanku tega dela in jih poimensko nisem omenil.

Diplomsko nalogo posvečam staršema Dušanu in Ireni, ter punci Romani.

Kazalo

Seznam uporabljenih kratic in simbolov

Ajax

Asynchronous JavaScript and XML

MVC

Model View Controller

CoC

Convenction over Configuratoin

DRY

Don't repeat yourself

CMS

Content Managment System

FRI

Fakulteta za Računalništvo in Informatiko v Ljubljani

WYSIWYG

What You See Is What You Get

HTML

Hypertext Markup Language

XHTML

Extensible HyperText Markup Language

OO

Object - Oriented

CSS

Cascading Style Sheets

XML

Extensible Markup Language

JSP

JavaServer Pages

ASP

Active Server Pages

PHP

Hypertext Preprocessor

ERB

Embedded Ruby

DOM

Document Object Model

Povzetek

V diplomski nalogi smo preučili možnosti prenove spletnega sistema FRI s tehnologijo Ruby on Rails. Namen naloge je bil spoznati se s tehnologijo Ruby on Rails ter to tehnologijo uporabiti v namene študija možnosti izboljšanja spletne sistema FRI, predvsem v smislu poenostavitve upravljanja z vsebinami. Slednji problem smo v diplomi reševali z uporabo dodatkov za označevalni jezik ter metodami, ki slonijo na tehnologiji Ajax in omogočajo urejanje vsebin na mestu. V izdelani pilotni implementaciji smo dodatno uporabili tudi dodatek RedCloth ter knjižnici Prototype in Scriptaculous.

Ključne besede:

agilen razvoj spletnih strani, spletne tehnologije, upravljanje s spletnimi vsebinami, Ruby on Rails, Ruby, Ajax, Textile

Poglavje 1

Uvod

Na svetovnem spletu danes praktično ne poznamo več statičnih spletnih strani. Tudi če naše potrebe težijo k njim, za upravljanje s spletnimi vsebinami navadno izberemo odprtokodni sistem in ga prilagodimo, da ta ustreza zahtevam konkretne aplikacije. Pri tem imajo v zadnjem času seveda prednost sistemi za upravljanje z spletni vsebinami (*angl. content management systems*, CMS), ki poenostavijo urejanje vsebin in to uporabniku omogočijo neposredno na spletnih straneh brez uporabe posebnega, mnogokrat kompleksnega sistema za administriranje in urejanje vsebin. Ker sedanja implementacija spletnih strani na Fakulteti za računalništvo informatiko temelji na slednjem, torej kompleksnem sistemu urejanja vsebin, je bil namen pričujoče diplomske naloge izdelava implementacijske študije, v kateri bi urejanje vsebin bil bistveno poenostavljen.

Ker so po večini sistemi za upravljanje s spletnimi vsebinami kompleksne aplikacije je v namene njihove prilagoditve lažje, če si izberemo odprtokodno rešitev. Razvoj lastnega sistema ponavadi terja zelo veliko časa, naporov in je lahko povezano z velikimi stroški. Zavedati se moramo, da kljub temu tovrstne aplikacije niso narejene čez noč, in bi razvoj lastne aplikacije bil celo cenejši. Je pa uporaba odprtokodnih rešitev mnogokrat zato lahko bolj robustna, posodablja in testira jo večja skupina razvijalcev, ter lahko omogoča enostavnejšo uvedbo kasnejših posodobitev.

Glede na lastnosti in principe, ki jih za upravljanje spletnih strani in izdelavo spletnih aplikacija omogoča okolje Ruby on Rails (www.rubyonrails.org), smo za razvoj našega sistema izbrali to ogrodje. Predvsem nas je navdušilo dejstvo, da izbrano ogrodje omogoča hiter, enostaven in zelo fleksibilen razvoj spletnih aplikacij. Njegova podpora tehnologiji Ajax je bila ključnega pomena pri izvedbi diplomske naloge. Tako smo v delu združili uporabniški in admini-

stracijski del sistema v eno celoto a vseeno omejili dostop do administracijskih možnosti. Uporabnikom smo omogočili na mestu urejanje vsebin, podporo Textile označevalnemu jeziku in Ajax prijavo v sistem.

Poglavje 2

Metode in orodja

V tem poglavju bomo na kratko predstavili metode in orodja, ki smo jih uporabili pri razvoju sistema razvitega v diplomski nalogi.

2.1 Metode in pristopi k razvoju spletnih strani

2.1.1 Agilen razvoj spletnih aplikacij

Agilen razvoj spletnih aplikacij [5] je model, ki ga lahko uporabljamo pri razvoju spletnih aplikacij. Njegov koncept je zelo podoben agilnemu razvoju navadnih, npr. namiznih aplikacij s to razliko, da je omejen na spletne aplikacije. V primerjavi z ostalimi modeli razvoja spletnih aplikacij je učinkovitejši in močnejši pri projektih s kratkimi časovnimi roki, vzpodbuja osebno komunikacijo, ter v ekipo razvoja vključuje tako tehnično osebje, kot stranke aplikacije.

Za model agilnega razvoja spletnih aplikacij je značilno vzporedno opravljanje planiranja, analiziranja zahtev, načrtovanja, kodiranja, testiranja in dokumentiranja aplikacije. Zelo močan vpliv na proces razvoja aplikacije imajo stranke, ki so neposredno vpete v ta proces. Ravno sodelovanje strank pri razvoju je razlog za hitrejši in učinkovitejši razvoj spletnih aplikacij. Na eni strani to pomeni veliko več ustne komunikacije, na drugi strani pa veliko manj tehnične dokumentacije. S tem ko so stranke vpete v proces razvoja aplikacije imajo razvijalci vedno na voljo povratne informacije. To pomeni krajši časovni rok razvoja, manj napak v kodi, aplikacija pa je bolj usmerjena k uporabniškim zahtevam in tako tudi prijaznejša do uporabnikov. Naloge kot so planiranje, načrtovanje, kodiranje, testiranje, itd. se odvijajo istočasno glede na uporabniške zahteve. To pa v končni fazi pomeni krajši razvojni čas aplikacije in

uporabniško prijaznejšo dokumentacijo.

Za model agilnega razvoja spletnih aplikacij so značilni:

- kratki časovni roki,
- majhne razvojne skupine,
- natančno planiranje,
- kratka in neformalna dokumentacija.

2.1.2 Ne ponavljaj se za sabo

Princip DRY (angl. *Don't Repeat Yourself*) zagovarja pisanje čim manj kode, kar dosežemo s tem, da se v kodi čim manj ponavljamo. Vsako definicijo zapišemo le enkrat in jo nato večkrat uporabimo.

2.1.3 Določila nad nastavitvami

Ta princip zagovarja minimalno nastavljanje aplikacije. To dosežemo tako, da se držimo določenih pravil, ki nam jih predpisuje tehnologija. Nastaviti moramo samo tiste dele aplikacije za katere orodje ni ustvarilo določil oz. za katero ni bilo pred nastavitev.

2.1.4 Model, pogled, krmilnik

Princip model - pogled - krmilnik (angl. *Model View Controller*) predstavlja arhitekturo aplikacije, ki loči kodo za dostop do podatkov od kode za prikaz podatkov uporabnikom. V prvem primeru govorimo o modelu v drugem pa o pogledu. Oba dela aplikacije pa povezuje krmilnik.

2.2 Orodja

Praktični del diplomske naloge je bil v celoti narejen v programskem jeziku Ruby 1.8.6 (www.ruby-lang.org/en/downloads), spletnem ogrodju Rails 2.0.2 (www.rubyonrails.org/down) in podatkovni bazi SQLite (www.sqlite.org/download.html). Za urejanje kode in pisanje LaTex besedila je bil uporabljen urejevalnik besedil TextMate (macromates.com). Vsa orodja so tekla na operacijskem sistemu Mac OS X 10.5 Leopard (www.apple.com/macosx).

2.2.1 HTML

HTML je kratica za *HyperText Markup Language*, ki je jezik za objavljanje na svetovnem spletu. Je podmnožica mednarodnega standarda za izmenjavo elektronskih dokumentov imenovan SGML (angl. *Standard Generalized Markup Language*). Računalniki potrebujejo programsko opremo, da lahko ta jezik razumejo. Ta programska oprema vključuje grafične brskalnike, tekstovne in govorne naprave.

HTML je označevalni jezik (angl. *Markup Language*). S posebnimi elementi se označi besedilo, ki pove brskalniku, kako naj strukturo dokumenta prikaže. Vsak element je sestavljen iz treh delov: začetne značke, vsebine in končne oznake. Značka (angl. *tag*) je besedilo, ki se nahaja med znakoma `<` in `>`. Zaključna značka vsebuje znak `/` takoj za znakom `<`. Npr. element `span` element vsebuje začetno značko, `<span>`, in zaključno značko, `</span>`.

Obstaja način, kako označiti naslove, odstavke, sezname tabele in še druge elemente. HTML je hipertekstovni jezik (angl. *Hypertext Language*). Hipertekst uporablja način na sklicevanje drugih spletnih dokumentov, ki so med seboj povezani. Ko se v spletnem dokumentu klikne povezavo, se v resnici kliče hiperpovezavo.

2.2.2 XHTML

XHTML se od HTMLja razlikuje predvsem po tem, da ga pišemo po strožji pravilih, ter da težimo k ločitvi vsebine (angl. *content*) od izgleda (angl. *style*). Večina modernih spletnih strani zato namesto klasičnega HTMLja uporablja XHTML.

XHTML prav tako kot HTML pišemo z značkami (angl. *tag*). Z njimi npr. definiramo odseke strani, del besedila ki ga želimo odebeliti, vstavljamo slike in flash animacije, oblikujemo tabele ter ustvarjamo obrazce.

Poglejmo si enostaven primer XHTML strani:

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
2   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
3   <html>
4    <head>
5      <title>Naslov strani</title>
6      <meta http-equiv="Content-Type" content="text/html;
7      charset=windows-1250" />
8    </head>
9    <body>
10     <p>Primer odstavka.</p>
11   </body>
```

```
12 </html>
```

Za razliko od klasičnega HTML jezika je XHTML jezik bolj strikten in čist jezik. Naj naštejemo nekaj bistvenih razlik med omenjenima jezikoma:

- vse označbe in parametre pišemo strogo z malimi črkami,
- vsi XHTML elementi morajo biti zaključeni,
- vsi XHTML elementi morajo biti pravilno gnezdeni,
- XHTML dokument mora vsebovati deklaracijo tipa dokumenta `<!DOCTYPE ...>` ,ki je lahko `strict` ali `transitional`.

2.2.3 Rails

Ruby on Rails je odprtokodno spletno ogrodje, katere namen je hitrejši, enostavnejši in produktivnejši razvoj spletnih aplikacij (rails.si). Ogrodje je sestavljena iz dveh ključnih komponent, programskega jezika Ruby in ogrodja Rails. Ruby je objektno usmerjen interpreterski skriptni jezik. Razvil ga je Yukihiro Matsumoto (www.ruby-lang.org/en/about) leta 1995. Rails pa je ogrodje za razvoj spletnih aplikacij, napisan v Rubyu. Razvil ga je David Heinemeier Hanssonn, kot posledico dela na projektu Basecamp (www.basecamphq.com) za podjetje 37signals. Prva verzija je bila izdana leta 2004. Leta 2006 pa je podjetje Apple najavilo, da bo distribuiralo ogrodje Rails z naslednjo različico njihovega operacijskega sistema Mac OS X 10.5 (weblog.rubyonrails.org/2006/8/7/ruby-on-rails-will-ship-with-os-x-10-5-leopard). Trenutno izdana verzija je 2.1.

Če na kratko povzamemo je Ruby on Rails ogrodje za učinkovit razvoj spletnih aplikacij. Podrobnejše ga bomo spoznali v nadaljevanju naloge.

2.2.4 Ruby

Ruby je dinamičen, odprtokodni programski jezik s poudarkom na enostavnosti in izboljšanju produktivnosti. Ima zelo elegantno sintakso, ki je zelo podobna naravnemu jeziku. Je lahko razumljiv za branje in enostaven za pisanje. Razvil ga je Yukihiro Matsumoto iz Japonske. Za razvoj novega programskega jezika se je odločil, ker ni bil z nobenim obstoječim popolnoma zadovoljen. Med razvojem se je osredotočil predvsem na to, da bo jezik omogočal hiter in enostaven razvoj pri tem pa razvijalcem povzročal čim manj glavobolov.

Prva različica jezika se pojavi leta 1995, trenutno verzija pa je 1.8.7. Avtor je navdih za razvoj jezika črpal iz jezikov kot so Lisp, Smaltalk in Perl. Ruby

je objektno usmerjen programski jezik vendar primeren tudi za proceduralno in funkcijsko programiranje [1].

Programerji ki so naredili prehod na Ruby trdijo, da so veliko bolj srečni pri razvoju aplikacij, kot so bili v ostalih programskih jezikih [2]. Programi v programskem jeziku Ruby so praviloma krajši kot v ostalih standardnih jezikih. Za primer si oglejmo kodo, ki izpiše *"Hello, World!"* v Javi:

```
1 public class HelloWorld {
2   public static void main(String []args) {
3     System.out.println("Hello World");
4   }
5 }
```

Kar bi v Rubyu napisali nekako takole:

```
1 puts 'Hello World'
```

Ruby je objektno usmerjen

Avtor programskega jezika Ruby je želel skriptni jezik, ki bo zmogljivejši od Perla in bolj objektno usmerjen kot je Python. Kadar pišemo objektno usmerjeno kodo navadno modeliramo objekte iz realnega življenja. Tipično bomo skozi proces modeliranja odkrili različne kategorije stvari, ki jih bomo želeli zajeti v kodi. V spletnem sistemu je koncept uporabnika ena izmed takšnih kategorij. V Rubyu bomo tako definirali razred, ki bo predstavljal vsako od teh entitet. Razred je kombinacija stanja (recimo ime uporabnika) in metod ki uporabljajo ta stanja (metoda, ki nam izpiše ime uporabnika).

Ko imamo definirane razrede bomo želeli ustvariti poljubno število ponovitev teh razredov. V primeru spletnega sistema bomo za vsakega uporabnika želeli ustvariti nov primerek razreda. Primerkom razredov pravimo `objekti` (angl. *objects*).

V Rubyu objekte ustvarimo s klicem konstruktorja `new`.

```
1 uporabnik1 = User.new("Peter")
2 uporabnik2 = User.new("Marko")
```

V zgornjem primeru smo naredili dva objekta razreda `User` in sicer `user1` in `user2`. Vsak objekt ima edinstveno identifikacijsko številko ali `id` objekta. V vsakem objektu lahko določimo spremenljivke objekta (angl. *instance variable*), to so spremenljivke v katerih so edinstvene vrednosti posameznega objekta. Spremenljivke objekta hranijo stanje objekta. Recimo vsak naš uporabnik bo imel spremenljivko objekta, ki bo hranila ime uporabnika.

Znotraj vsakega razreda lahko definiramo metode objektov (angl. *instance methods*). Te metode lahko kličemo znotraj razreda ali izven razreda, odvisno od določil. Preko teh metod so nam dostopne spremenljivke objekta. Pokličemo jih tako da pošljemo sporočilo objektu, ki vsebuje ime metode in zahtevane atribute. Poglejmo si enostaven primer:

```
5.times { print "We *love* Ruby -- it's outrageous!" } # Pet krat izpise tekst v navednicah.
```

V mnogih programskih jezik števila in ostali primitivni tipi niso objekti, v Rubyu pa so vse vrednosti objekti. Tako je tudi število `5` objekt nad katerim kličemo metodo `.times`. Vsakemu bitu informacije in kode lahko dodelimo svoje atribute in metode. Celo vrednosti kot so `true`, `false` in `nil` (Ruby verzija za `null` vrednost) so objekti.

Osnovne značilnosti jezika Ruby

Za začetek si poglejmo enostaven primer kode v Rubyu. In sicer bomo napisali metodo, ki vrne niz kateremu bomo dodali ime osebe.

```
# Metoda za izpis pozdrava
def sayHello(name)
  result = "Pozdravljen, " + name
  return result
end

# Klic metode
puts sayHello("Peter") # nam izpise Pozdravljen Peter
```

Kot lahko opazimo je sintaksa Rubya zelo čista in elegantna. Vrstic ni potrebno zaključevati s podpičjem, dokler vsak ukaz pišemo v novo vrstico. Komentarji se v Rubyu začnejo z znakom za števila `#` in so enovrstični.

Metode definiramo z rezervirano besedo `def`, ki ji sledi ime metode (v našem primeru `sayHello`) in morebiti argumenti metode v oklepajih. Ruby ne uporablja zavitih oklepajev za telo metode, kot recimo Java in PHP. Metodo enostavno zaključimo z `end`. Vse med ključnima besedama `def` in `end` je telo metode. V našem primeru lokalni spremenljivki `result` dodelimo niz `"Pozdravljen"`, ter ime ki ga podamo preko parametra metode. Zadnja vrstica vrne rezultat, kot vrednost `result` spremenljivke.

V našem primeru lahko vidimo kako kreiramo nov niz, ki je kot smo že dejali objekt. Najlažje to storimo tako, da med enojne ali dvojne navednice zapiše niz znakov. V primeru uporabe enojnih navednic Ruby vse med navednicama

smatra kot niz. Če uporabimo dvojne navednice, veljajo določena pravila. Prvo je upoštevanje ubežnih sekvenc; nizov znakov, ki se začnejo s poševnico nazaj in določenim znakom, kot je recimo za novo vrstico, tabulator, itd. Drugo pa je da lahko v nizu pišemo izvršljive ukaze. Izraz v konstruktu `# izraz` se nadomesti z njegovo vrednostjo. Zgornji primer bi lahko napisali tudi takole:

```
1 # Metoda za izpis pozdrava
2 def sayHello(name)
3   result = "Pozdravljen, #{name}"
4   return result
5 end
```

Končno lahko našo metodo še nekoliko poenostavimo. Vrednost, ki jo vrnejo metode v Rubyu je vrednost zadnje izvršenega izraza. Po tem takem se lahko brez skrbi znebimo `return` stavka.

```
1 # Metoda za izpis pozdrava
2 def sayHello(name)
3   result = "Pozdravljen, #{name}"
4 end
```

Za konec pregleda osnovnih značilnosti jezika poglejmo še pravila za določevanje imen. Ruby uporablja določila, ki nam glede na prvi znak imena povejo kaj določeno ime predstavlja. In sicer lokalne spremenljivke, parametre metod in imena metod pišemo z malo črko ali podčrtajem. Globalnim spremenljivkam dodamo znak dolar `$`, med tem ko spremenljivkam primerkov objektov dodamo znak viseča opica `@`. Imena razredov, modulov in konstant pišemo z veliko začetnico. Na primer:

```
1 user     # Lokalna spremenljivka
2 $user    # Globalna spremenljivka
3 @user    # Spremenljivka objekta
4 @@user   # Razredna spremenljivka
5 User     # Razred ali konstanta
```

Tabele

Tabele so Rubyu indeksirane zbirke podatkov, ki hranijo zbirke objektov dosegljivih preko ključev. Delimo jih na navadne in na zgoščene tabele (angl. *hash tables*). Pri navadnih tabelah je ključ indeks objekta, med tem ko nam zgoščene tabele omogočajo kot ključ katerikoli objekt. V obeh tabelah lahko dodajamo elemente po potrebi. Navadne tabele nam omogočajo učinkovitejši dostop,

zgoščene pa večjo prilagodljivost. V obeh tabelah lahko hranimo objekte katerega koli tipa. Tako imamo lahko objekte, ki predstavljajo cela števila (angl. *integer*), decimalna števila (angl. *floating point*) ali nize znakov (angl. *string*).

Tabelo najenostavnejše naredimo tako, da v oglatih oklepajih navedemo elemente tabele.

```
a = [ 1, 'drevo', 3.14] # Tabela z 3 elementi
```

Če želimo narediti prazno tabelo v oglatih oklepajih ne navedemo nobenega elementa ali kličemo konstruktor `new`.

```
praznaTabela1 = []
praznaTabela2 = Array.new
```

Za ustvarjanje tabele besed nam Ruby omogoča elegantno rešitev z uporabo ukaza %w.

```
besede = %w {drevo polje reka mesto}
# besede = ['drevo','polje','reka','mesto']
```

Zgoščene tabele so zelo podobne navadnim. Namesto oglatih oklepajev, raje uporabljamo zavite oklepaje. Pri kreiranju nove tabele navedemo en objekt za vrednost in drug za ključ elementa. Poglejmo si primer enostavne zgoščene tabele.

```
uporabnik = {
  'ime' => 'Peter',
  'priimek' => 'Zlatnar',
  'email' => 'peter.zlatnar@email.si'
}
```

Do elementov tabele dostopamo tako, da namesto indeksa navedemo ključ elementa.

```
uporabnik['ime']     # Vrne Peter
uporanik['priimek'] # Vrne Zlatnar
```

Krmilni stavki

Ruby pozna vse običajne krmilne stavke, kot jih imajo drugi programski jeziki. Od `if` pogojnega stavka do `while` zanke. Glavna razlika med krmilnimi stavki Ruby jezika in tistimi iz Jave in C jezika je, da Ruby ne uporablja oklepajev. Tako ne za telo struktur, kot tudi ne za pogoje struktur.

```
if stevec > 10
  puts "Poizkusi ponovno!"
elsif poizkusi == 3
  puts "Izgubili ste."
else
  puts "Vnesite stevilo:"
end
```

Poleg izpuščanja oklepajev nam Ruby omogoča elegantnejši način pisanja kontrolnih struktur, ki imajo samo en stavek v telesu.

```
if hitrost > 130
  puts "Pozor! Prevelika hitrost."
end

# Lahko krajse zapisemo

puts "Pozor! Prevelika hitrost." if hitrost > 130
```

Bloki in iteratorji

Na kratko bomo opisali še bloke in iteratorje, ki dajejo Rubyu posebno moč. Blok je v Rubyu kos kode, ki ga lahko povežemo s klicem metode, skoraj tako kot bi šlo za njene parametre. To pa je zelo močna lastnost. Blok je enostavno košček kode med zavitima oklepajema ali med ključnima besedama `do...end`.

```
{ puts "Lepo Pozdravljeni."} # To je blok kode
```

Blok lahko tudi povežemo s klicem metode. V sami metodi nam je blok dosegljiv s klicem `yield` stavka. V spodnjem primeru smo definirali metodo, ki nato dvakrat kliče `yield`. Klicu metode dodamo blok, ki ga potem v telesu metode dvakrat kličemo.

```
def callBlock
  yield
  yield
end

# Klicemo metodo in ji dodamo blok
callBlock { puts "To je blok." }

# Kar nam izpise
To je blok.
```

```
11 To je blok.
```

Ruby uporablja bloke za implementacijo iteratorjev, metod ki se sprehodijo skozi zbirko elementov in vrnejo trenutni element. Poglejmo si enostaven primer.

```
1 tabela = %w{ peter marko matej luka}  # Kreiramo tabelo
2 tabela.each { |oseba| puts oseba }  # Gremo skozi tabelo in
    izpisemo osebe
3
4 # Izpis
5 peter
6 marko
7 matej
8 luka
```

Mnogi konstrukti zank iz jezikov C in Java so v jeziku Ruby enostavni klici metod.

```
1   3.times { print "Ruby! " }   # Izpise "Ruby! Ruby! Ruby! "
2   1.upto(9) {|x| print x }     # Izpise "123456789"
```

2.2.5 SQLite

SQLite je programska knjižnica napisana v jeziku C velikosti 500KB, ki se sama vzdržuje, deluje kot ločen strežniški proces, je ni potrebno posebno nameščati in implementira transakcijski podatkovni strežnik. To so glavne značilnosti te res zelo majhne knjižnice. Na domači spletni strani knjižnice (www.sqlite.org) navajajo, da je SQLite najpogosteje uporabljena podatkovna zbirka na svetu. Uporablja se v neštetih namiznih aplikacijah, kot tudi v mnogih dlančnikih, pametnih telefonih in glasbenih predvajalnikih. Med drugimi podatkovno zbirko SQLite najdemo v Mozilli Firefox, Mac operacijskih sistemih, Skype programu za izmenjavo sporočil in telefonskih klicev, Symbian operacijskemu sistemu za mobilne telefone in mnogih spletnih straneh narejenih v PHP skriptnem jeziku.

2.2.6 CSS

CSS (angl. *Cascading Style Sheets*) omogoča avtorjem spletnih strani, da predpišejo obliko posameznih elementov. Večina elementov v HTML je namreč namenjena logičnemu oblikovanju, kjer samo določimo kakšne vrste je posamezen element (slika, tabela, vrstica v tabeli, celica v vrstici, seznam,

točka seznama, odstavek, indeks, eksponent, naslov, aktivna povezava ...), brskalnik pa te elemente oblikuje po svoje.

Z uporabo stilov CSS lahko elementom določimo celo vrsto oblikovnih lastnosti, med katere spadajo ozadje, robovi, razmiki, odmiki, pisava, poravnava, barva itd... CSS nam omogoča, da oblikovne lastnosti določimo ločeno od vsebine, kar poveča preglednost napisane kode.

2.2.7 AJAX

Ajax je kratica za *"Asynchronous JavaScript and XML"* in označuje način ustvarjanja interaktivnih spletnih strani. Sam po sebi Ajax ni tehnologija, temveč način združevanja različnih neodvisnih tehnologij. Kadar se v delu sklicujemo na Ajax tehnologijo, tako mislimo na skupek naslednjih tehnologij:

- XHTML (ali HTML) in CSS za predstavitev besedil, informacije o besedilo ali način prikaza besedila,
- DOM (angl. *Document Object Model*) v povezavi z jezikom JavaScriptom za dinamičen prikaz informacij in interakcijo z njimi,
- objekt XMLHttpRequest za asinhrono izmenjavo podatkov s strežnikom. XMLHttp je vmesnik za programiranje aplikacij (*angl. application program interface*, API), ki ga lahko uporablja JavaScript za prenos in manipulacijo XML (*angl.* Extensible Markup Language) s strežnika ali na strežnik preko HTTP protokola.

Pri spletnih straneh brez tehnologije Ajax akcija uporabnika (npr. pritisk na gumb) sproži pošiljanje zahteve na strežnik preko HTTP protokola. Strežnik akcijo obdela in uporabniku pošlje nazaj HTML stran. Taki (t.i. zaporedni) izmenjavi podatkov med strežnikom in klientom pravimo sinhrona izmenjava. Slaba stran takega pristopa je, da mora uporabnik med procesiranjem zahteve vsakič čakati na odziv strežnika. Ajax odstrani čakanje z uvedbo vmesnika med strežnikom in klientom (t.i. Ajax engine), ki ga internetni brskalnik naloži namesto internetne strani. Vmesnik je napisan v JavaScriptu in največkrat implementiran v skritem okvirju ter tako neviden za uporabnika. Njegova naloga je tako ustvarjanje tega, kar uporabnik vidi, kot tudi komunikacija s strežnikom, ki je pogojena s prestrezanjem zahtev uporabnika. Vsaka zahteva sproži JavaScript klic do Ajax vmesnika. Če je zahteva dovolj preprosta, vmesnik opravi vse potrebno, kadar pa potrebuje podatke s strežnika, le - temu pošlje XML zahtevo asinhrono, brez vpliva na uporabnikovo komunikacijo z aplikacijo.

Statične spletne strani danes nadomeščajo dinamične in interaktivne. Pri slednjih je največji poudarek na interakciji uporabnika s stranjo. Ajax, kot pristop, omogoča izdelavo popolnoma interaktivnih strani. Odličen primer Ajax aplikacije je vsem dobro znan Googleov Gmail, ki na noben drug način ne bi mogel biti učinkoviteje zgrajen.

Poglavje 3

Razvoj spletnih aplikacij z Ruby on Rails

Kot smo že omenili je Ruby on Rails odprtokodno spletno ogrodje za učinkovit razvoj spletnih aplikacij. Predno pa se posvetimo razvoju spletnih aplikacij z ogrodjem Rails moramo najprej razložiti nekaj osnovnih principov, ki jih ogrodje Rails zagovarja.

3.1 Temeljna principa ogrodja Ruby on Rails

Temeljna principa ogrodja Rails sta "Določila nad nastavitvami" in "Ne ponavljaj se za sabo".

3.1.1 Določila nad nastavitvami

Ta princip pravi, da je potrebno nastaviti le tisto kar je ne nastavljivega. To v praksi pomeni, da ogrodje Rails ustvari privzete vrednosti za ključne povezovalne vidike aplikacije in nam hkrati s pravili določa poimenovanje razredov, metod in atributov. Eno izmed pravil za poimenovanje v ogrodju pravi da se razred z imenom `Page` preslika v tabelo podatkovne baze z imenom `pages`. Če bi želeli omenjeni razred preslikati v kakšno drugo tabelo, bi morali to eksplicitno navesti. Vendar z upoštevanjem teh določil lahko napišemo prelepo kodo, z manj tratenja časa ob nastavitvah.

3.1.2 Ne ponavljaj se za sabo

Ta princip stoji za tem, naj bo vsak košček znanja v sistemu izražen na samo enem mestu. Za zagotavljanje tega principa ogrodje Rails izkorišča moč Ruby programskega jezika, ki omogoča visoko povezljivost komponent. To pomeni da nam ni potrebno ročno določevati povezav. Ogrodje Rails nam tako z upoštevanjem principa DRY in CoC omogoča grajenje spletnih aplikacij z manj kode in minimalnimi nastavitvami aplikacije.

Agilen razvoj spletnih aplikacij

Malo prej v delu smo na kratko omenili agilen razvoj spletnih aplikacij in principe, ki jih zagovarja. Ogrodje Rails je zelo agilno in idealno orodje za uveljavljanje agilnega razvoja spletnih aplikacij. Prilagodljivost je močno zasidrana v osnovnih gradnikih ogrodja. Samo ogrodje trdno stoji za principi agilnega razvoja, ki jih avtorji navajajo v Agilnem manifestu [4]:

- posamezniki in interakcije nad procesi in orodji,
- delujoča programska oprema nad obširno dokumentacijo,
- sodelovanje z strankami nad pogajanji o pogodbenih obveznostih,
- odzivati se na spremembe nad sledenje planu.

Ogrodje Rails je narejeno za posameznike in interakcije. Ne zahteva zahtevnih orodij, kompleksnih nastavitev in obsežne dokumentacije. Odlično se počuti v majhni skupini razvijalcev, enostavnem urejevalniku besedil in koščki Ruby kode. To pa vodi do preglednosti; vsaka sprememba aplikacije je takoj vidna strankam.

Rails se ne odpoveduje dokumentaciji. Ravno nasprotno, ogrodje omogoča zelo enostaven način izdelave HTML dokumentacije za celotno aplikacijo. Vendar pa razvoj ni voden na podlagi dokumentacije, tako v Rails projektih ne boste našli obsežne dokumentacije. Namesto tega boste našli majhno skupino razvijalcev in uporabnikov, ki bodo raziskovali njihove potrebe in iskali odgovore na te potrebe. Našli boste rešitve, ki se bodo spreminjale skozi razvojni cikel s tem ko bodo zahteve postajale vse bolj jasne. Našli boste ogrodje, ki vam bo že v začetnih fazah omogočalo delujočo aplikacijo. Sicer bo še zelo groba po robovih a bo že omogočala seznanjenost uporabnikov s prvimi občutki, kaj lahko pričakujejo v kasnejših fazah razvoja.

Na ta način ogrodje Rails vzpodbuja sodelovanje med razvijalci in uporabniki. Ko uporabniki oziroma stranke vidijo, kako hitro se ogrodje Rails lahko

odziva na spremembe, začnejo zaupati v ekipo in verjeti, da jim lahko dostavi aplikacijo glede na trenutne zahteve in ne glede na to kar je bilo zahtevano predno se je začelo razvijati aplikacijo.

Vse to je združeno v ključnem sporočilu ogrodja, biti sposoben se odzivati na spremembe. Z močno, na trenutke že obsedeno implementacijo DRY principa nam Rails zagotavlja, da spremembe vplivajo na čim manj kode, kar ne bi mogli trditi za ostale tehnologije.

3.2 Arhitektura aplikacij Rails

3.2.1 Uvod

Ena izmed glavnih značilnosti ogrodja Rails od vas zahteva, da se držite določenih pravil strukturiranosti aplikacije. To pa je zelo učinkovito saj močno poenostavi razvoj aplikacij, kodo pa naredi veliko bolj pregledno. Vsak košček kode ima točno določeno mesto v aplikaciji. Govorimo o arhitekturi Model - Pogled - Krmilnik v nadaljevanju navedeno z MVC kratico.

3.2.2 Model, pogled, krmilnik arhitektura na splošno

Leta 1979 je Trygve Reenskaug [4] predstavil novo arhitekturo za razvoj interaktivnih aplikacij. V njegovi arhitekturi so aplikacije razdeljene na tri glavne komponenete: modele, poglede in krmilnike.

Model

Model uporabljamo pri vzdrževanju stanja aplikacije. Včasih je to stanje prehodno in traja le nekaj transakcij med uporabnikom in aplikacijo. Včasih pa je stanje trajno in shranjeno izven aplikacije, pogosto v relacijski podatkovni bazi.

Model predstavlja več kol le shranjene podatke v podatkovni bazi. Vsebuje namreč tudi vsa poslovna pravila, ki veljajo za določene podatke. Z implementacijo poslovnih pravil v modelu zagotovimo, da nam ničesar drugega v aplikaciji ne more pokvariti podatkov. Model tako služi za preverjanje podatkov (t.i. validaciji) ob vhodu in njihovo shranjevanje v podatkovni bazi.

Pogled

Pogled uporabljamo pri oblikovanju uporabniškega vmesnika, navadno na podlagi podatkov v modelu. Na primer: spletni sistem organizacije ima v podat-

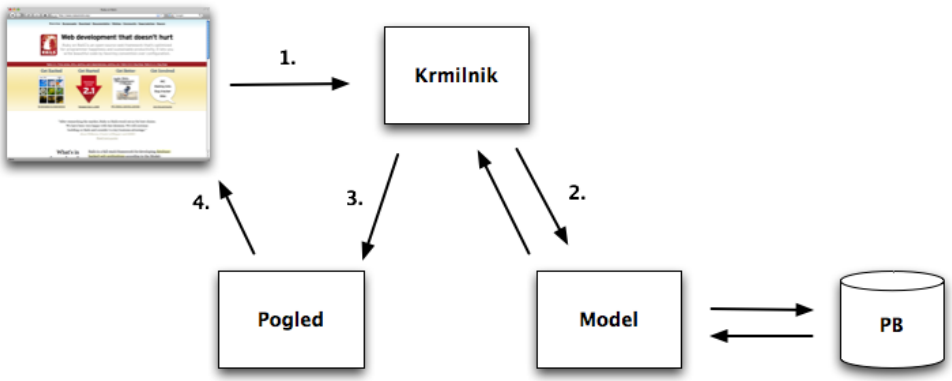
kovni bazi shranjeno listo vseh zaposlenih, ki jo želimo prikazati obiskovalcem. Ta lista nam je dostopna preko modela, vendar pa je pogled tisti, ki bo dostopal do liste in jo pripravil za prikaz uporabnikom. Čeprav se preko pogledov uporabniki srečajo z mnogimi načini vnosa podatkov, sam pogled nikoli ne služi obdelavi vnesenih podatkov. Naloga pogleda je samo prikaz podatkov.

Kot smo že omenili je pogled tesno povezan z modelom. Do istega modela načeloma dostopa več pogledov, ki pa služijo različnim namenom. In sicer od prikaza podatkov uporabnikom do urejanja in dodajanja podatkov administratorjev.

Krmilnik

Krmilnik je kot nekakšen logični center aplikacije, ki skrbi za vodenje aplikacije, ter komunikacijo med modelom in pogledom. Tipično prejme zahtevo z strani brskalnika, glede na njo komunicira z modelom in nato preda zahtevane podatke pogledu, ki jih prikaže uporabniku.

Arhitekturo sestavljeno iz treh komponent, to je iz modelov, pogledov in krmilnikov, prikazuje shema na sliki 3.1 in se bomo nanj v prihodnjem besedilo sklicevali z angleško kratico MVC (angl. *Model View Controller*).



Slika 3.1: MVC arhitektura spletnih aplikacij.

Akcije (povezave med komponentami arhitekture) na sliki 3.1 so naslednje:

1. Brskalnik pošlje zahtevo krmilniku.

2. Krmilnik se poveže z modelom, ki mu preda zahtevane podatke.

3. Krmilnik preda podatke pogledu.

4. Pogled podatke uredi in jih pošlje brskalniku, ki jih prikaže uporabniku.

MVC arhitektura za vsak košček kode predvideva svoje mesto v aplikaciji. Za razliko od klasičnega razvoja spletnih aplikacij, kjer imamo v enem paketu vse od predstavitve podatkov, dostopa do podatkovne baze, poslovnih pravil do upravljanja z dogodki. To pa je odlično izhodišče za nepregledno in težko berljivo kodo. Na drugi strani pa nam MVC arhitektura z izhodiščnim ogrodjem, ter upoštevanjem določil ponuja odlična izhodišča za lepo, pregledno kodo in tako produktivnejši razvoj.

Vendar pa so v preteklosti razvijalci bili sprva skeptični nad uporabo MVC arhitekture in so se rajši držali klasičnega razvoja spletnih aplikacij, ter ustaljenih stalnic in paradigem. Tako so šele po 20 leti začeli eksperimentirati z uporabo MVC arhitekture v spletnih aplikacijah. Kot rezultat teh poizkusov so nastala ogrodja kot so WebObjects, Struts in JavaServerFaces. Vsa omenjena ogrodja bazirajo na uporabi MVC arhitekture [4].

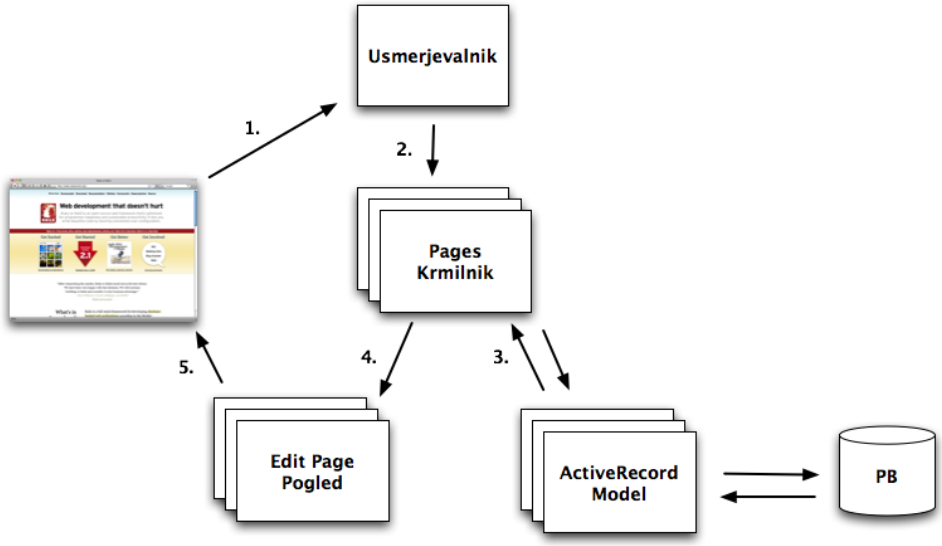
3.3 MVC arhitektura v Rails aplikacijah

Tudi ogrodje Rails uporablja MVC arhitekturo. Za vsako aplikacijo ustvari datotečno strukturo, kjer je točno določeno mesto za vsak košček kode. Za vsak model, krmilnik in pogled je določena mapa in znotraj datoteka, kamor razvijalec dodaja modele, poglede in krmilnike. Ogrodje Rails nato s pomočjo določil poveže ločene dele aplikacije v delujočo celoto, kar pomeni da razvijalcu ni potrebno pisati dodatnih nastavitvenih datotek.

V Rails aplikacijah se prihajajočo zahtevo najprej pošlje usmerjevalniku, ki ugotovi kam točno v aplikaciji naj se zahteva pošlje in kako naj se jo razčleni (pravilno razbere). Na podlagi razčlembe se določi krmilnik in metoda (ki jim pravimo akcije) znotraj krmilnika, ki naj se izvede. Akcija lahko uporabi podatke iz zahteve, lahko komunicira z modelom ali izvede nove akcije. V končni fazi pripravi podatke za pogled, ki jih nato ustrezno uredi in prikaže uporabniku.

Na sliki 3.2 je prikazan konkreten postopek kako ogrodje Rails s pomočjo MVC arhitekture obdela vhodno zahtevo in prikaže rezultat uporabniku.

Akcije s slike 3.2 so:



Slika 3.2: MVC arhitektura Rails aplikacij.

1. Brskalnik pošlje zahtevo preko URL naslova: http://www.fri.uni-lj.si/pages/9/edit.
2. Usmerjevalnik prevzame zahtevo jo razčleni, ter poišče `Pages` krmilnik.
3. Krmilnik nato preko modela `Page` iz podatkovne baze prejme podatke.
4. Krmilnik nato pokliče pogled `edit` in mu preda podatke.
5. Prikaz podatkov, njihova ustrezna postavitev in prikaz v spletnem brskalniku.

V zgornjem primeru je aplikacija predhodno prikazala seznam strani iz katere si je uporabnik eno izbral in jo želel urediti. S klikom na Uredi stran je sprožil povezavo, ki je strežniku poslala zahtevo `http://www.fri.uni-lj.si/pages/9/edit`. Zahtevo prevzame usmerjevalnik in jo nemudoma razčleni na posamezne dele. V tem primeru `pages` pomeni ime krmilnika, `9` je nam pove id strani, ki jo želimo urediti in `edit` ime akcije v krmilniku. Iz te analize usmerjevalnik ve, da mora poklicati `edit` metodo v `PagesController` z `id=9`. Krmilnik nato zahteva podatke za stran z `id=9` preko modela in jih preda pogledu, ki jih nato prikaže uporabniku. Uporabnik podatke uredi in jih zopet po isti poti pošlje usmerjevalniku, ki iz ponovne razčlembe zahteve ve katero akcijo mora poklicati, ki podatke preko modela shrani v podatkovno bazo.

3.4 Ogrodje Rails

Kot smo že omenili je MVC arhitektura v ogrodju Rails realizirana tako, da fizično loči mape in datoteke v katerih se nahajajo modeli, pogledi in krmilniki. Za predstavitev in logično povezovanje posameznih konceptov arhitekture pa poskrbi z določenimi razredi in moduli. Glavni izmed njih so:

- **ActiveRecord** je modul v katerem so razredi za upravljanje poslovne logike in komunikacij s podatkovno bazo. V ogrodju Rails igra ta modul vlogo modela.
- **ActionController** je komponenta ki upravlja z zahtevami brskalnika in komunikacijami med modelom in pogledom. Vsi krmilniki so podrazredi oz. dedujejo iz tega razreda, ki je del knjižnice **ActionPack**.
- **ActionView** je komponenta, ki skrbi za prikaz strani v brskalniku. Vsi pogledi dedujejo iz tega razreda, ki je tudi del knjižnice **ActionPack**.

3.4.1 ActiveRecord: Rails Model

V večini primerov želimo, da spletne aplikacije hranijo podatke v relacijski podatkovni bazi. V našem primeru imamo na eni strani OO (Objektno Orientiran) skriptni jezik Ruby in na drugi relacijsko podatkovno bazo. Objekti predstavljajo podatke in operacije, podatkovna baza pa zbirko vrednosti. Na splošno poznamo dva principa povezovanja objektno orientiranega sistema in relacijskih podatkovnih baz. In sicer prvi organizira aplikacijo okoli podatkovne baze, drugi pa organizira podatkovno bazo okoli aplikacije.

- Neposredno povezovanje s podatkovno bazo
- Objektno relacijsko povezovanje s podatkovno bazo

Neposredno povezovanje s podatkovno bazo

Za ta princip povezovanja relacijske podatkovne baze in objektov je značilno, da se SQL poizvedbe vgradijo v kodo aplikacije. To pomeni da se prepletata tako logika podatkovne baze, kot celotna logika aplikacije. Ta način programiranja je značilen za večino današnjih skriptnih jezikov, kot sta Perl in PHP. Tudi Ruby omogoča programiranje na ta način, vendar ga v ogrodju Rails ne bomo uporabljali.

Primer SQL poizvedbe v PHP:

```
1 <?php
2 //Connects to your Database
3 mysql_connect("your.hostaddress.com", "username", "password") or
    die(mysql_error());
4 mysql_select_db("Database_Name") or die(mysql_error());
5 $data = mysql_query("SELECT * FROM friends WHERE pet='Cat'")
6 or die(mysql_error());
7 ?>
```

Kot smo že omenili, ta način uporablja večina danes priljubljenih skriptnih jezikov in je zato množično uporabljen. Mešanje poslovne logike in logike podatkovne baze pa lahko predstavlja težavo, predvsem pri vzdrževanju aplikacije in dodajanju novih funkcionalnosti. Recimo da pride do novih zahtev in moramo popraviti vsa povpraševanja SQL, ki se tičejo spremembe. To pomeni, da moramo poiskati vsa mesta v kodi kjer se določena poizvedba pojavi in jo popraviti. Ne samo da se ponavljamo, tudi možnost napak je zelo velika pri takem načinu programiranja.

Pri klasičnem načinu programiranja smo to težavo rešili z enkapsulacijo. In sicer znotraj objekta zajamemo tako podatke, kot objekte in dobimo neko zaključeno celoto, ki se kaže kot objekt. Kar nam prinese lažje vzdrževanje ter ponovno uporabljivost kode.

Razvijalci pa so šli še korak dlje in razvili posebno plast razredov med podatkovno bazo in kodo aplikacije, ki skrbi za komunikacijo s podatkovno bazo. Tako aplikacija uporablja objekte in razrede te plasti za interakcijo s podatkovno bazo in nikoli ne komunicira z njo neposredno. Na ta način imamo ločeno shemo podatkovne baze in kodo aplikacije, ki ji ni potrebno opravljati nizkonivojska opravila za interakcijo z bazo. Govorimo o ORM ali Object-Relational Mapping, ki ga uporablja ogrodje Rails.

Objektno - relacijsko povezovanje

V tem principu knjižnice ORM povežejo table z razredi. Če ima podatkovna baza tabelo `users` bo imela naša aplikacija razred `User`. Vrstice v tej tabeli predstavljajo objekte v razredu – določen uporabnik je predstavljen, kot objekt razreda `User`. Znotraj tega objekta lahko preko atributov dostopamo do posameznih stolpcev tabele. Naš objekt `User` ima metode, ki nam vrnejo in omogočijo urejanje gesla, email naslova, itd.

Ogrodje Rails nam priskrbi metode s katerimi lahko opravljamo nizkonivojska opravila nad podatkovno bazo. Recimo da želimo iz tabele `users` prebrati podatke za uporabnika z `id=1`:

```
@user = User.find(1)
```

Kar vrne podatke za porabnika z `id=1`. Naslednja metoda:

```
@users = User.find(:all)
```

pa nam vrne zapise za vse uporabnike v tabeli `users`, kot zbirko objektov. Za nekoliko bolj kompleksna povpraševanja lahko v metodi `find` uporabimo simbol `:conditions` ki predstavlja zahteve in pogoje pri iskanju podatkov v bazi. Če nam tudi ta možnost ne zadostuje lahko uporabimo metodo `find_by_sql(sql)`, kjer kot `sql` atribut podamo vprašanje SQL.

Preko teh metod lahko tudi dostopamo in urejamo posamezne vrednosti v tabeli:

```
User.find(:all, :conditions => "name='peter'").each do |user|
  user.age = 25
  user.save
end
```

Torej ORM plast poveže tabele z razredi, vrstice z objekti in stolpce z atributi teh objektov. Metode razredov se uporabljajo za izvajanje operacije nad tabelami, metode objektov pa izvajajo operacije nad posameznimi vrsticami tabel. To pa za uporabnika ogrodja Rails pomeni, da lahko razvije še tako kompleksno aplikacijo, ne da bi moral napisati en sam SQL stavek.

ActiveRecord

Srce spletne aplikacije razvite v ogrodju Rails leži v relacijskem sistemu za upravljanje s podatkovno bazo (t.i. ORM modelu). V ta namen ogrodje zagotavlja razred `ActiveRecord`, ki skrbi za povezovanje med kodo in bazo podatkov. Zelo tesno sledi ORM standardnemu modelu: tabele povezuje z razredi, vrstice tabele z objekti in stolpce tabele z atributi objektov. Od ostalih knjižnic ORM se razlikuje predvsem na način kako se nastavlja. Z opiranjem na določila in upoštevanjem principa DRY nam ActiveRecord prihrani veliko večino nastavitev, saj jih zmanjša na minimum.

Primer: če ima razred `Page` definicijo `has_many :posts` in primerek `page`, potem bo klic `page.posts` vrnil polje z vsemi objekti razreda `Post`, ki imajo `page_id` enak `page.id`. Vse kar moramo storiti je da povemo naj razred deduje od `ActiveRecord::Base` nadrazreda.

```
class Page < ActiveRecord::Base
  has_many  :posts
end
```

ActiveRecord nam tako omogoči, da se ne ukvarjamo z samo podatkovno bazo in se rajši osredotočimo na poslovno logiko. Za nas tako opravlja sledeče naloge:

- vzpostavitev povezave s podatkovno bazo,
- branje podatkov iz podatkovne baze, in
- shranjevanje podatkov v podatkovno bazo.

Na tem mestu je omembe vredno še preverjanje podatkov, ki nam jih omogoča `ActiveRecord`. Poglejmo si primer preverjanj v modelu `User`:

```
1 class User < ActiveRecord::Base
2
3   validates_presence_of      :login, :email
4   validates_length_of        :password, :within => 4..40
5   validates_uniqueness_of    :login, :case_sensitive => false
6   validates_format_of        :email, :with => /(^([^@\s]+)@((?:[-_a
      -z0-9]+\.)+[a-z]{2,})$)|(^$)/i
7
8 end
```

V zgornjem primeru nam `validates_presence_of` preverja ali smo vnesli vrednosti v polje login in email, `validates_lenght_of` preverja dolžino gesla, `validates_uniqueness_of` preveri edinstvenost polja login, ter `validates_format_of` nam preveri format email naslova. Če preverjanje podatkov spodleti, nas Rails opozori na napake.

3.5 ActionPack: Pogled in Krmilnik

V arhitekturi MVC sta pogled in krmilnik tesno povezana. Na eni strani krmilnik priskrbi podatke pogledu na drugi strani pa prejema zahteve od pogledov. Ravno zaradi teh interakcij je podpora za poglede in krmilnike v Rails povezana v skupnem paketu imenovanem `ActionPack`.

3.5.1 Pogled

Izgled strani RoR zapisuje v posebnem jeziku za poglede, ki kombinira čisti HTML z vstavljanjem Ruby kode, celoten izgled pa nadzorujejo stilne predloge CSS. Rezultat so HTML.ERB (ERB je kratica za *Embedded Ruby*) datoteke, ki pa niso zgolj HTML datoteke z vstavljeno Ruby kodo, kot je to znano za JSP,

ASP in PHP. HTML oznakam so dodane le posebne oznake za izpis vrednosti določene kontrole, ki omogočajo izražanje pogojev in zank za urejanje izpisa, kar predloge naredi modularne. Poglejmo si primer pogleda, ki nam izpiše imena in priimek uporabnikov.

```
1  <h1>Uporabniki</h1>
2
3  <% for user in @users %>
4
5    <p><h2>Ime</h2>
6    <%= user.name %>
7    </p>
8
9    <p><h2>Priimek</h2>
10   <%= user.surname %>
11   </p>
12
13 <% end %>
```

V zgornjem primeru pogled od krmilnika prejme zbirko uporabnikov v spremenljivki `@users`. Kot lahko vidimo uporabljamo dva načina vstavljanja Ruby kode. In sicer značke `<% %>`, ter značke `<%= %>`. Razlika med obema je v tem, da v prvem primeru RoR izvede kodo med značkama vendar rezultata ne vrne predlogi, medtem ko se v drugem primeru izvrši koda med značkama, rezultat pa vrne predlogi.

3.5.2 Krmilnik

Krmilnik je v ogrodju Rails logični center vaše aplikacije. Če smo v primeru modela govorili o srcu aplikacije, potem lahko v tem primeru govorimo o možganih aplikacije. Njegova naloga je koordinacija interakcij med uporabniki, pogledi in modeli, ki jih RoR opravi v ozadju. Razvijalcu je tako prepuščeno pisanje kode na ravni aplikacijske funkcionalnosti. Kar pomeni, da je Rails krmilnike lažje za vzdrževati in razvijati. Poleg naštetih nalog pa je krmilnik zadolžen še za opravljanje sledečih nalog:

- usmerjanje - usmeri zahteve uporabnikov k ustreznim akcijam,
- upravljanjem s predpomnenjem podatkov - s tem zagotovi minimalna povpraševanja po bazi,
- upravljanjem z modulom v katerem so metode za pomoč, ki jih uporabljamo v pogledih,

- upravljanjem s sejami aplikacije.

Poglejmo si krmilnik, ki bi nam je v prejšnjem primeru vrnil zbirko objektov `User`.

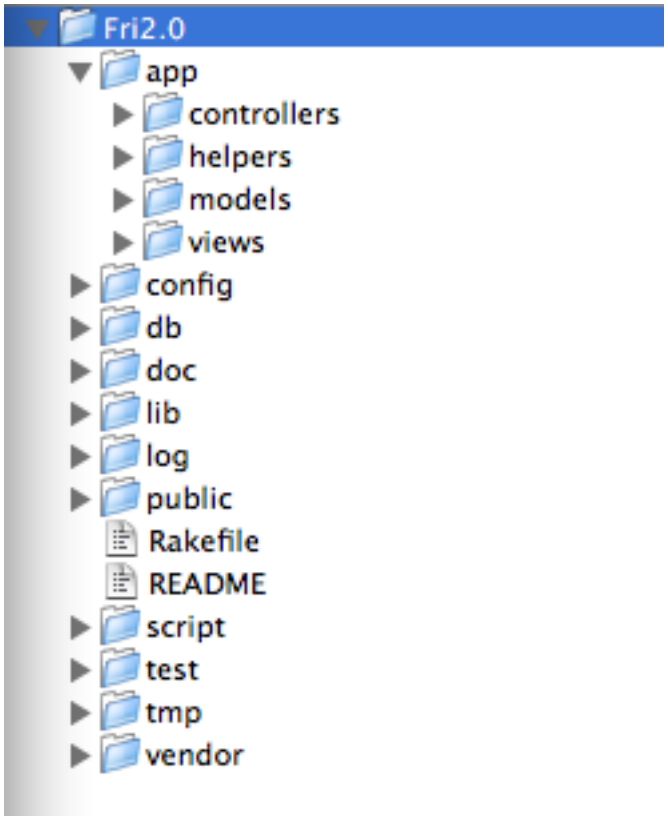
```
1 class UsersController < ApplicationController
2
3   def index
4       @users = User.find(:all)
5     end
6 end
```

3.6 Razvoj nove aplikacije

Za razvoj nove aplikacije v terminal vnesemo ukaz `rails` ter ime aplikacije, ki jo želimo razviti. RoR nam nato ustvari ogrodje v mapi z imenom, ki smo ga podali kot parameter ukaza. Kot primer kreirajmo ogrodje za CMS sistem fakultete. Najprej v terminal vnesemo ukaz `rails` ter ime aplikacije.

```
1 rails Fri2.0
```

Ogrodje Rails nam ustvari množico datotek in map, ki ji prikazuje slika 3.3.



Slika 3.3: Struktura ogrodja Rails.

3.6.1 Organizacija ogrodja Rails

app

Vsebuje vso kodo, ki je specifična za posamezno aplikacijo, v našem primeru sistem za upravljanje s spletnimi stranmi fakultete.

app/controllers

V tej mapi so zbrani vsi krmilniki aplikacije. Poimenovani naj bi bili na način `users_controller.rb` zaradi avtomatskih URL preslikav. Vsi krmilniki so podrazredi razreda `ApplicationController::Base`.

app/models

Vsebuje modele, ki naj bi bili poimenovani `user.rb`. Večina modelov izhaja iz razreda `ActiveRecord::Base`.

app/views

Vsebuje predloge za pogled, ki naj bi bil poimenovan `users/index.html.erb` za `UsersController#index` akcijo. Vsi pogledi uporabljajo eRuby sintakso zato `.erb` končnica datoteke.

app/views/layouts

Vsebuje predloge za razporeditve, ki so uporabljene v pogledih. Privzeta razporeditev za vse poglede je shranjena v datoteki application.html.erb. Izjema je le kadar v pogled ročno vključimo drugo razporeditev ali če je v tej mapi razporeditev z istim imenom kot ime datoteke v katerem je pogled. Recimo imamo poglede v datoteki `users/` v kolikor bomo imeli razporeditev `user.html.erb` bo ta privzeta in ne `application.html.erb`.

app/helpers

Vsebuje metode za pomoč v pogledih, ki naj bi bili poimenovani `users_helper.rb`. Metode za pomoč se samodejno ustvarijo ko za ustvarjanje krmilnikov uporabimo ukaz `script/generate`. Uporabljamo jih, da zajamemo funkcionalnosti pogledov v obliki metod. V njih imamo lahko tako zelo kompleksne metode katerih rezultat nam je v pogledu dosegljiv z klicem te metode. To pa nam omogoči, da v pogledih čim manj uporabljamo Ruby kodo.

config

Nastavitvene datoteke za okolje Rails, podatkovno bazo, usmerjevalne preslikave in ostale odvisnosti.

db

Vsebuje shemo podatkovne baze v datoteki `schema.rb`. Mapa `db/migrate` pa vsebuje vsa zaporedja migracij podatkovne baze za našo shemo.

doc

V tej mapi je shranjena dokumentacija aplikacije, ki jo ustvarimo z ukazom `rake doc:Fri2.0`.

lib

Specifične knjižnice za posamezno aplikacijo. Praktično vsa uporabniško napisana koda, ki ne spada v mape controllers, models ali helpers.

public

Mapa dosegljiva spletnemu strežniku. Vsebuje mape za slike, slogovne datoteke in java skripte. Vsebuje tudi Rails *"dispatcher"* in privzete HTML datoteke. Ta mapa naj bi bila nastavljena kot `DOCUMENT_ROOT` vašega spletnega strežnika.

script

Skripte za ustvarjanje komponent ogrodja in poganjanje ukazov.

test

Vsebuje datotek za funkcijsko testiranje in testiranje posameznih enot. Pri uporabi ukaza `script/generate` nam ogrodje Rails ustvari predloge za testiranje in shrani v to mapo.

vendor

Zunanje knjižnice, ki jih aplikacija uporablja. Vsebuje tudi mape za dodatke (angl. *plugin*), ki smo jih v aplikaciji uporabili.

Ko imamo narejeno osnovno ogrodje moramo poskrbeti za povezavo med podatkovno bazo in aplikacijo. To enostavno storimo tako, da odpremo datoteko `config/databse.yml` in vnesemo potrebne nastavitve za aplikacijo. Od tipa in imena podatkovne baze do uporabniškega imena in gesla za dostop do podatkovne baze. Na tem mestu naj še omenimo da trenutna verzija ogrodja Rails prihaja z SQLite 3 podatkovno bazo, ki je privzeta vsaki novi aplikaciji. Seveda Rails podpira večino drugih podatkovnih baz za spletne aplikacije kot so MySQL, PostgreSQL, Oracle, IBM DB2, MS SQL, itd.

Za delo s podatkovnimi bazami nam je v ogrodju Rails na voljo množica ukazov preko terminala. Za ustvarjanje nove podatkovne baze uporabimo ukaz rake `db:create:all`.

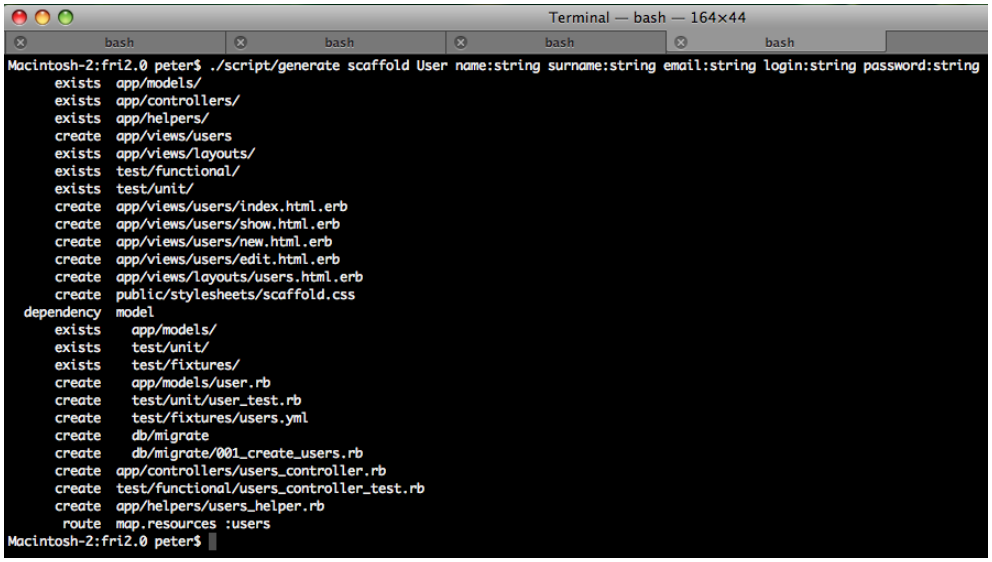
3.6.2 Ukaz *scaffold*

`Scaffold` je v ogrodju Rails magična beseda, ki nam v bistvu zgradi celotno aplikacijo za dodajanje, urejanje, prikazovanje in brisanje primerkov `ActiveRecord` razreda. Za primer v našem spletnem sistemu želimo uporabnike, ki bodo lahko dodajali in urejali določene vsebine. Poglejmo si kako bi s pomočjo ukaza `scaffold` ustvarili vso potrebno kodo za dodajanje, urejanje, prikazovanje in brisanje uporabnikov sistema.

```
1 script/generate scaffold User name:string surname:string email:
    string login:string password:string
```

Kot lahko opazimo v zgornjem primeru kličemo ukaz `generate` iz mape `script`. Če želimo ustvariti krmilnik kličemo metodo `controller`, če želimo model kličemo metodo `model` v našem primeru pa smo klicali metodo `scaffold`. Ta pa za nas kreira tako krmilnik, model kot poglede in vse pripadajoče datoteke. Klicu metode `scaffold` povemo ime modela, ter atribute ki jih želimo v našem model. Se pravi po izvršitvi zgornjega ukaza bo ogrodje Rails za nas ustvarilo model `User`, ki bo imel atribut `name` tipa string, atribut `surname` tipa string itd. Kaj točno vse nam ustvari ogrodje Rails ob uporabi ukaza `scaffold` lahko vidimo na sliki 3.4.

Sedaj imamo že skoraj vse pripravljeno za izvajanje naše aplikacije. Poskrbeti moramo samo še da se naš model `User` ustrezno preslika v podatkovno bazo in nato zagnati strežnik. Na sliki 3.4 lahko vidimo da je ogrodje Rails ustvarilo datoteko `db/migrate/001_create_users.rb`. Vsebina te datoteke ni nič drugega kot koda za ustvaritev tabele `users` v naši podatkovni bazi. Z ukazom `rake db:migrate` poskrbimo da Rails izvede zadnje ustvarjeno datoteko v tej mapi (v našem primeru 001_create_users.rb) in tako ustvari tabelo.



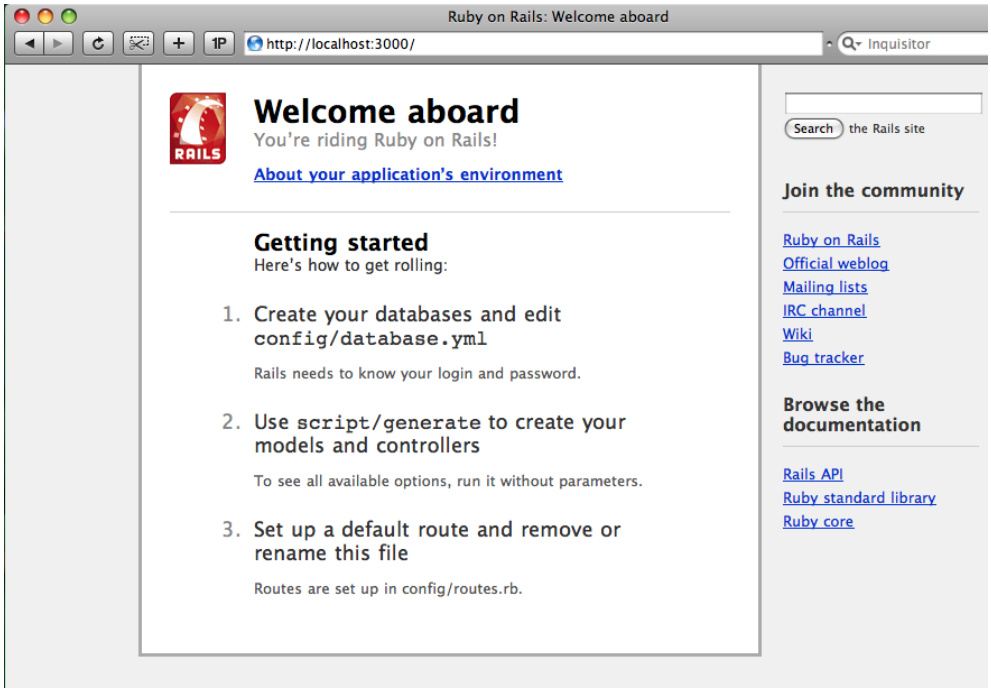
Slika 3.4: Uporaba scaffold ukaza.

Vpišemo ukaz `script/server` in že nam na vratih 3000 teče spletni strežnik Mongrel. V brskalnik vnesemo naslov `http://localhost:3000` in prikaže se nam pozdravno okno ogrodja Rails 3.5.

Pozdravno okno je signal, da smo na pravi poti do končne aplikacije. Ker pa smo malo prej ustvarili scaffold model `User` bi morali imeti na razpolago tudi metode za prikaz, dodajanje, urejanje in brisanje uporabnikov. URL naslovu dodamo ime krmilnika to je v našem primeru `users`. V kolikor ne dodamo nobene akcije se bo izvedla `index` akcija in nam prikazala seznam vseh uporabnikov. Zaslonsko masko lahko vidimo na sliki 3.6.

1. Povezava za dodajanje novega uporabnika.
2. Povezava za prikaz uporabnika.
3. Povezava za urejanje uporabnika.
4. Povezava za brisanje uporabnika.

Metoda scaffold nam omogoča da na zelo hiter način naredimo delujočo aplikacijo. Primerna je predvsem v začetnih fazah razvoja aplikacije ko na ta način lahko zelo hitro naročniku pokažemo prototip aplikacije. Seveda pa



Slika 3.5: Pozdravno okno ogrodja Rails.

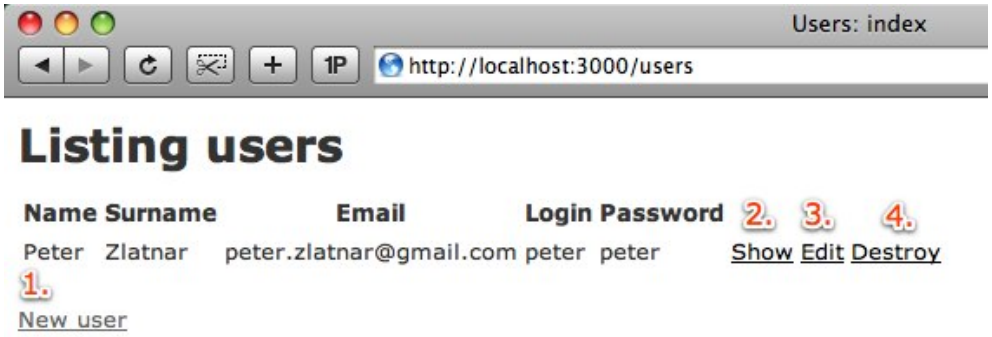
moramo v kasnejših fazah razvoja metodo scaffold dopolniti oz. nadomestiti z svojimi metodami glede na potrebe in zahteve naročnika.

3.6.3 Spletni strežniki

Za razvojne potrebe nam je v samem ogrodju Rails na voljo Mongrel (mongrel.rubyforge.org) in WEBrick spletni strežnik. Ob klicu `script/server` bo Rails privzeto poizkušal zaganti Mongrel ali lighttpd spletni strežnik. V nasprotnem primeru se bo zagnal WEBrick. To pa nam omogoča, da lahko tudi brez produkcijskega okolja zelo hitro preizkušamo in razvijamo aplikacijo.

Mongrel je Ruby spletni strežnik z C komponentami in je primeren predvsem za razvoj Rails aplikacij. Če Mongrel ni dosegljiv bo ogrodje Rails skušalo poiskati lighttpd. V primerjavi z Mongrelom in WEBrickom je znatno hitrejši in tako primeren tudi za produkcijsko okolje. Vendar pa moramo spletni strežnik lighttpd naknadno namestiti, primeren pa je predvsem za unix okolja.

Če smo do sedaj govorili predvsem o razvojnem okolju in z njim povezanimi



Slika 3.6: Seznam uporabnikov preko scaffold metode.

spletnimi strežniki, omenimo še produkcijska okolja. Ogrodje Rails lahko poganjamo na vsaki platformi, ki podpira FastCGI protokol (www.fastcgi.com). Med bolj znanimi so spletni strežniki kot so Apache, Lighttp, Microsoft IIS, Nginx, itd.

3.7 Podpora tehnologiji Ajax

Ogrodje Rails prihaja z vgrajenim Prototype JavaScript ogrodjem (prototypejs.org) in Scriptaculous JavaScript knjižnico (script.aculo.us) za kontrolo in vizualne efekte. V kolikor ju želimo uporabljati v naši aplikaciji moramo v HEAD značko html strani dodati naslednjo vrstico kode `<%= javascript_include_tag :defaults %>`.

3.7.1 Prototype

Prototype je JavaScript knjižnica, ki omogoča DOM manipulacijo, Ajax funkcionalnost in bolj tradicionalno objektno - orientirane pripomočke za JavaScript. Prototype modul v ogrodju Rails nam priskrbi množico metod za pomoč, ki nam olajšajo klice Prototype funkcij. Nudi nam tudi možnost klicev oddaljenih (angl. *remote*) metod z uporabo Ajaxa. Klic oddaljene metode pomeni zahtevo v ozadju po Rails akciji. To pa pomeni, da lahko kličemo akcije krmilnikov ne da bi morali ponovno naložiti spletno stran, ampak osvežimo sam del strani s pomočjo vbrizgov v DOM. Tipičen primer rabe Protoype metod v ogrodju Rails je recimo dodajanje komentarjev določeni novici ne da bi morali ponovno naložiti celotno spletno stran ali posodobitev zneska nakupovalne

košarice pri dodajanju novih artiklov.

3.7.2 Scriptaculous

Modul za Scriptaculous JavaScript knjižnico nam v ogrodju Rails nudi množico metod za klice funkcij omenjene knjižnice vključno s tistimi, ki nam omogočajo Ajax funkcionalnosti in vizualne efekte. Prototype in Scriptaculous knjižnici sta zelo tesno povezani [3]. Saj je knjižnica Scriptaculous zgrajena na osnovi knjižnice Prototype, ki jo tudi razširja. Bistvena razlika med njima je v tem da Prototype služi predvsem za Ajax klice in DOM interakcije, Scriptaculous pa deluje na višjem nivoju bližje aplikaciji in uporabniškemu vmesniku. Komponente knjižnice Prototype razširja z elementi, ki omogočajo vizualne efekte in kontrolo uporabniškega vmesnika.

Med šest osnovnih vizualnih efektov štejemo:

- Highlight - elementu spremeni barvo ozadja za kratek čas.
- Morph - spremeni CSS lastnosti elementa.
- Move - element premakne na drugo pozicijo.
- Opacity - zamegli element.
- Scale - spremeni dimenzije elementa.
- Parallel - omogoča sočasno uporabo več efektov.

Vsi ostali so izpeljani iz osnovnih efektov. Naj naštejemo še nekatere elemente kontrole uporabniškega vmesnika:

- InPlaceEditor - na mestu urejanje tekstovnih elementov.
- InPlaceCollectionEditor - na mestu urejanje izbirnih elementov.
- Autocompletion - samodejno dokončanje tekstovnih nizov.
- Slider - premikanje elementa po horizontalni osi.

Poglavje 4

Možnosti prenove spletnega sistema FRI z Ruby on Rails

4.1 Kaj je CMS

CMS (angl. *Content Managment System*) je aplikacija za dodajanje, urejanje, upravljanje in objavljanje vsebin. V našem primeru gre za spletno verzijo. CMS sisteme delimo na uporabniški del in administrativni del. Uporabniški del je namenjen končnim uporabnikom, ki pregledujejo vsebino. Med tem ko nam administrativni del omogoča dodajanje, urejanje, upravljanje in objavljanje vsebin.

Na trgu je danes cela množica CMS sistemov. Med njimi je zelo veliko odprtokodnih, ki so zelo priljubljeni tako med razvijalci kot uporabniki. Vendar gre v večini primerov za zelo kompleksne sisteme, ki terjajo od nas veliko prilagoditev in nastavitev za težavo ki jo rešujemo. Na drugi strani pa imamo CMS sisteme, ki jih podjetja razvijejo za lastne oz. naročnikove potrebe.

4.2 Pregled obstoječega spletnega sistema FRI za urejanje vsebin

Poglejmo si spletno stran Fakultete za Računalništvo in Informatiko v Ljubljani, natančneje kar glavno navigacijo strani:

- novice in dogodki
- izobraževanje

Slika 4.1: Spletna stran FRI.

- raziskave
- katedre in laboratoriji
- osebje
- o fakulteti

Če malo pobrskamo po spletni strani lahko kaj hitro opazimo, da sistem omogoča uporabnikom dodajati in urejati vsebino, kot so novice, projekti, obvestila, laboratoriji in osebni profil. Seveda pa morajo imeti dostop do administrativnega dela sistema z ustreznimi uporabniškimi pravicami. V večini primerov gre za enostavne spletne strani, ki vsebujejo besedilo in povezave na druge strani sistema. Sam uporabnik sistema ima možnost urejanje osebnega profila oz. osebne predstavitvene strani, kje lahko objavi svojo bibliografijo. Omembe vredno je še povezava med osebami in projekti, ki se ali so se odvijali na FRI.

Vse te funkcionalnosti smo skušali zajeti in jih v nekaterih vidikih tudi izboljšati v tej diplomski nalogi.

4.3 Ključni gradniki sistema v Ruby on Rails

Ključni gradniki naše aplikacije so uporabniki, strani, objave in projekti. Najprej si bomo pogledali osnovna gradnika to sta stran in objava. Posamezno stran v sistemu smo modelirali z modelom `Page`, tako imamo v podatkovni bazi tabelo `pages`. Dve bistveni stvari pri modelu Page sta sledeči definiciji:

```
1 class Page < ActiveRecord::Base
2   acts_as_tree :order => "title"
3   has_many  :posts
4   acts_as_textiled  :body
5 end
```

Prvo bomo pustili za malo kasneje, ko bomo govorili o urejanju vsebine. Če si kar pogledamo prvo, definicija `acts_as_tree :order => "title"` pove modelu naj se obnaša kot drevesna struktura. Seveda predno lahko kličemo omenjeno definicijo moramo predhodno naložiti istoimenski dodatek in potem lahko v aplikaciji uporabljamo drevesno strukturo. V našem primeru so recimo povezave v glavni navigaciji sistema strani brez starša, medtem ko vse podstrani otroci določene strani. To nam omogoča dodajanje podstrani, kategorij po posameznih vejah drevesa.

Druga definicija `has_many :posts` pa pomeni, da lahko posamezna stran oz. podstran vsebuje več objav oz. prispevkov. Če bomo pogledali v model `Post` bomo videli da vsebuje definicijo `belongs_to :page`, kar pomeni da objava pripada strani. Tem določilom v ogrodju Rails pravimo asociacije z njimi pa določamo relacije med posameznimi modeli. Recimo da želimo poiskati vse objave na prvi strani, kar bi storili z naslednjim klicem `@objave = Page(1).posts`.

Z modelom `User` smo modelirali uporabnike sistema, ki imajo določene pravice do uporabe administrativnega dela sistema. Če ne drugače si lahko vsak uporabnik uredi svojo osebno stran ter dodaja in ureja projekte na katerih je sodeloval. Tako smo prišli do modela `Project` s katerim smo modelirali projekte. Uporabnik, ki kreira projekt se samodejno doda projektu, medtem ko je ostale sodelavce na projektu potrebno ročno dodati.

Na kratko smo predstavili kjučne gradnike sistema, sedaj pa si bomo pogledali, kako smo rešili domeno urejanja vsebin.

4.4 Administrativni del sistema

4.4.1 Avtentikacija uporabnikov

Ogrodje Rails nam omogoča vrsto načinov avtentikacije uporabnikov, ki so nam danes poznane v svetu spletnih aplikacij. In sicer smo v diplomski nalogi uporabili naslednja dva:

- avtentikacija z uporabniškim imenom in geslom
- avtentikacija z uporabo servisa OpenID

Uporabniško ime in geslo

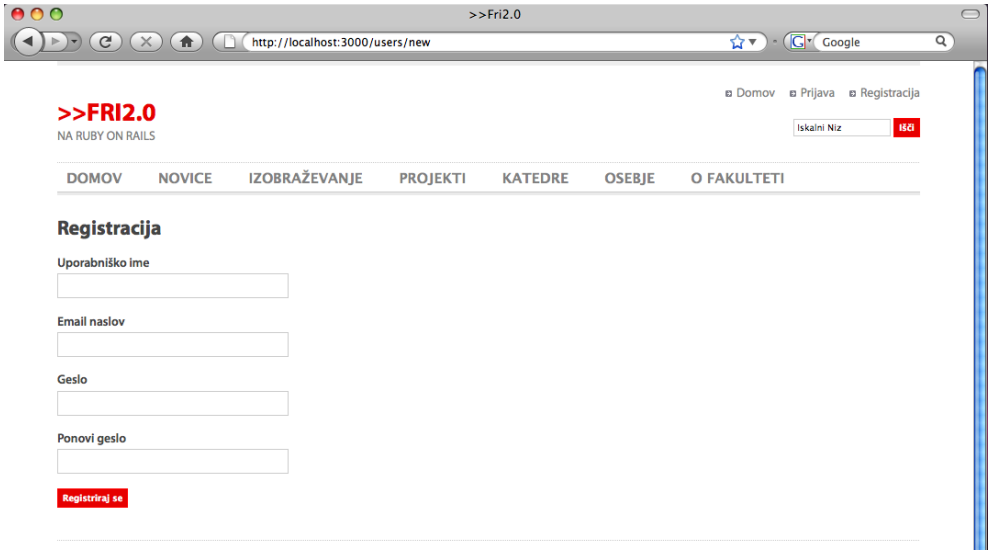
Verjetno najbolj razširjen in poznan način avtentikacije uporabnikov. Seveda predno se uporabnik lahko prijavi v sistem mora opraviti registracijo. V našem primeru smo za avtentikacijo uporabnikov uporabili Rails dodatek `restful_authentication` ki nam omogoča:

- prijavo/odjavo
- varno rokovanje z gesli
- aktivacijo računa preko email sporočila
- odobritev/zavrnitev računa s strani administratorja

Uporabnik, ki se želi vpisati v sistem izpolni spletni obrazec v katerega vnese željen podatke, slika 4.2. Sistem mu nato pošlje email sporočilo preko katerega aktivira svoj račun. Po uspešno opravljeni aktivaciji se lahko prijavi v sistem z izbranim uporabniškim imenom in geslom.

OpenID

V sistem se lahko prijavimo tudi z OpenID (openid.net). OpenID omogoča uporabnikom, da se v različne spletne aplikacije prijavijo z edinstvenim spletnim naslovom, ki ga dobijo pri OpenID ponudniku. Obstaja že cela kopica OpenID ponudnikov med njimi tudi MojID (www.mojid.com) iz Slovenije. Pri izbranem ponudniku si ustvarimo profil, kjer nato dobimo svoj spletni naslov, ki je oblike `http://mojid.com/imeprofila`.



Slika 4.2: Registracija v sistem.

Uporabnik se lahko nato s svojim spletnim naslovom prijavi na vsa spletna mesta, ki omogočajo prijavo z OpenID. Pri prvi prijavi ga spletno mesto preusmeri na OpenID račun kjer z geslom potrdi svojo identiteto. Ko opravi ta proces se naslednjič lahko prijavi samo z spletnim naslovom.

Uporaba OpenID spletnega naslova pomeni, da nam ni potrebno hraniti cele kopice takšnih ali drugačnih uporabniških imen in gesel, vse kar potrebujemo je spletni naslov OpenID.

Za uporabo servisa OpenID v Ruby on Rails obstaja dodatek `open_id_authentication`, ki smo uporabili z že omenjenim `restful_authentication` dodatkom.

4.4.2 Izboljšave sistema

Glavna slabost obstoječega sistema je ločen administracijski del sistema od uporabniškega dela. In sicer v praksi to pomeni, da mora uporabnik ki hoče urediti določene vsebine, zapustiti trenutno zaslonsko masko in se ločeno prijaviti v administracijski del sistema. Tam pa mora ponovno poiskati želeno vsebino in jo urediti. Na tak način deluje večina današnjih sistemov za urejanje vsebin, kar pa je zelo neučinkovito in nepriročno.

V diplomski nalogi se uporabnik prijavi v sistem kadarkoli potrebuje pravice za urejanje vsebin, ne glede na to kje se nahaja. Sitem ga po uspešni prijavi vrne na mesto kjer je zahteval prijavo. Z uporabniškega vidika je sistem tako

prijaznejši do uporabnika in mu prihrani vso odvečno delo, ki ga mora opraviti v obstoječem sistemu. Iz samega tehničnega vidika pa vidimo izboljšavo z združitvijo kode uporabniškega dela in administracijskega dela. Namreč v večini današnjih sistemov za urejanje vsebin sta uporabniški in administracijski del ločena, tudi iz vidika programske kode. Ker pa vsaka dodatna vrstica kode predstavlja grožnjo za morebitne napake in hrošče se nam zdi ključnega pomena, aplikacija s čim manj vrstic programske kode. Ne samo, da tako drastično zmanjšamo možnosti za pojavitev hroščev, tudi vzdrževanje in nadgrajevanje aplikacije je mnogo lažje.

4.4.3 Urejanje vsebin

Kot v večini spletnih spletnih sistemov za urejanje vsebin tudi v spletnem sistemu FRI poteka urejanje vsebin s pomočjo WYSIWYG (angl. *What You See Is What You Get*) urejevalnika. Na ta način vsebini na licu mesta določimo končni izgled s pomočjo vizualnih ukazov. Primer urejanje vsebine z WYSIWYG urejevalnikom je prikazan na sliki 4.3.

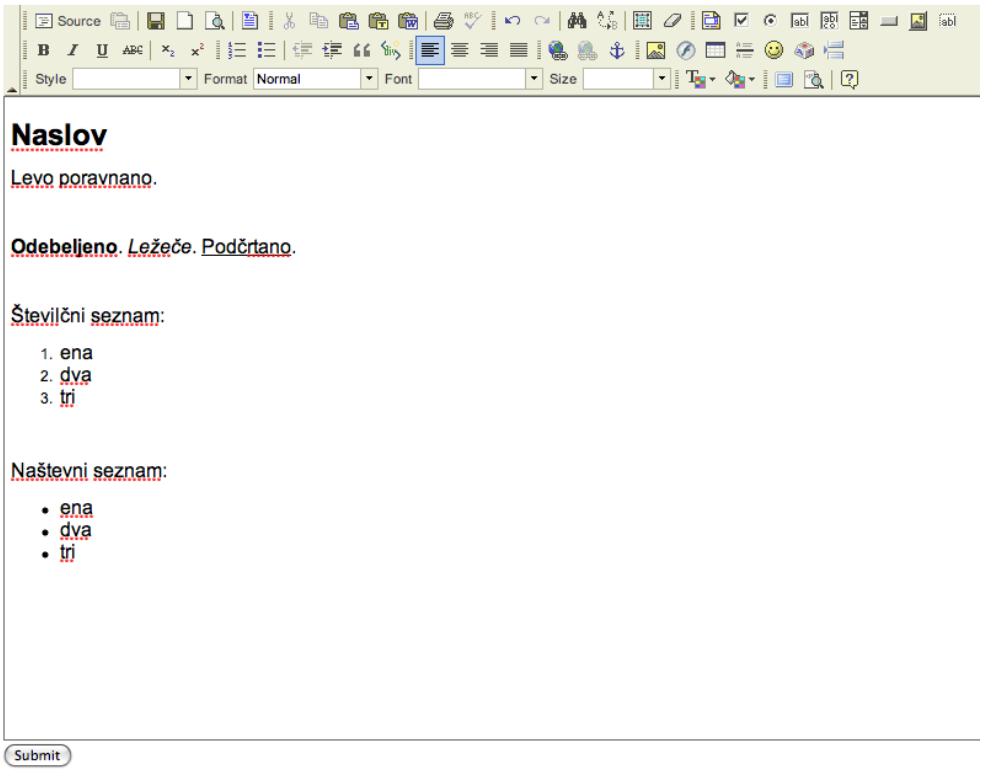
Seveda ima urejanje vsebine z metodo WYSIWYG tako svoje prednosti, kot slabosti. Med prednosti z gotovostjo lahko vključimo enostavnost uporabe tovrstnih urejevalnikov. Saj uporabniki hitro postanejo vešči uporabe vizualnih urejevalnikov besedil, kot jih lahko imenujemo.

Na drugi strani pa imajo nevešči uporabni preveč proste roke pri urejanju vsebin. To pa lahko privede do vizualno popolnoma neskladnih vsebin, saj vsak uporabnik lahko ureja vsebine po svojem okusu. Do neskladij lahko pride tudi pri kopiranju vsebin iz namiznih urejevalnikov besedil, saj z kopiranjem prenesemo tudi sloge. Poleg naštetih slabosti nas vizualni urejevalniki besedil z vsemi svojimi opcijami mnogokrat tudi upočasnjujejo pri delu.

Glede na ciljno skupino uporabnikov spletnega sistema fakultete smo v diplomski nalogi podprli urejanje vsebin s pomočjo označevalnega jezika `Textile` (textism.com/tools/textile).

Podpora označevalnemu jeziku Textile

Kaj je označevalni jezik Textile? Verjetno se je že vsakdo srečal z Wikipedijo v takšni ali drugačni obliki. Tisti ki ste že poizkusili urejati vsebine v Wikipediji ste se že spoznali z urejanjem vsebin s pomočjo označevalnega jezika. In sicer

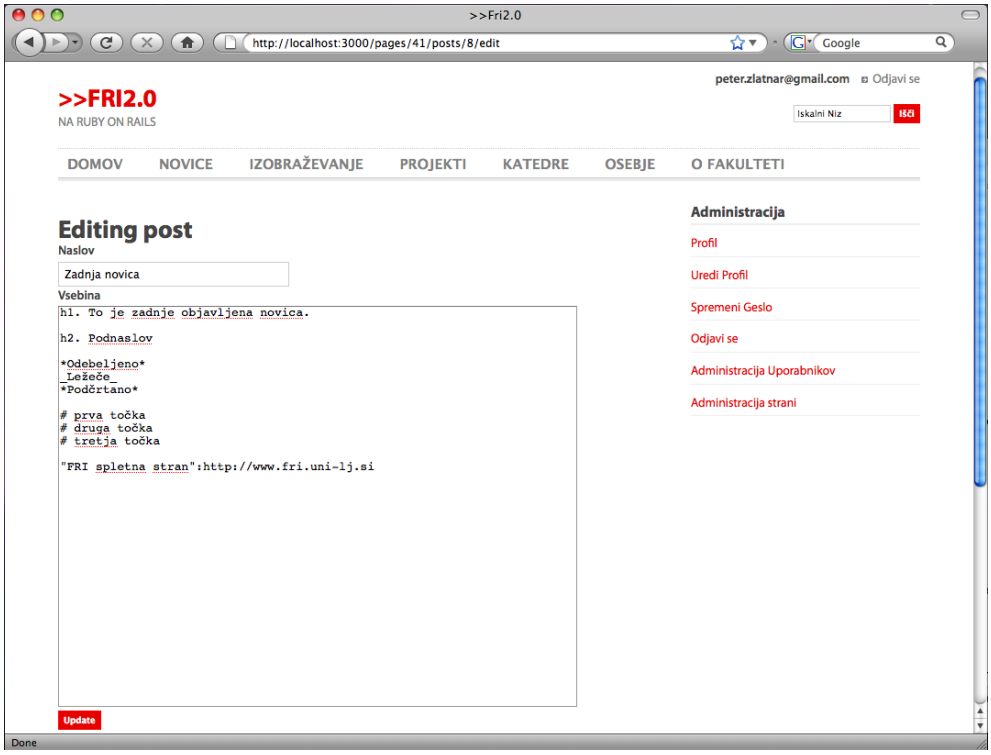


Slika 4.3: WYSIWYG urejevalnik.

besedilu glede na želeno obliko dodamo označevalne značke. Označevalni jezik nato značke pretvori v HTML značke, ki pa jih brskalnik brez težav prikaže.

V diplomski nalogi smo uporabi označevalni jezik `Textile`, ki je dosegljiv v obliki dodatka za ogrodje Rails in se imenuje `RedCloth`. Poglejmo si primer urejanja vsebin z jezikom Textile na sliki 4.4. Končni izgled besedila lahko vidimo na sliki 4.5.

Označevalni jezik Textile nam nudi poenoteno celostno podobo besedil. Tako nam ni potrebno skrbeti ali smo uporabili pravilno pisavo, velikost, barvo, itd. Dokumenti so zelo lepo strukturirani in hkrati pregledi. Sama uporaba jezika je zelo enostavna in hitro učljiva. Vseeno pa se je potrebno te značke naučiti in jih ročno vnašati v samo besedilo, kar bi lahko šteli kot minus v primerjavi z vizualnimi urejevalniki. Vendar glede na uporabnike spletnega sistema FRI menimo, da je uporaba jezika Textile smotrnejša od vizualnega



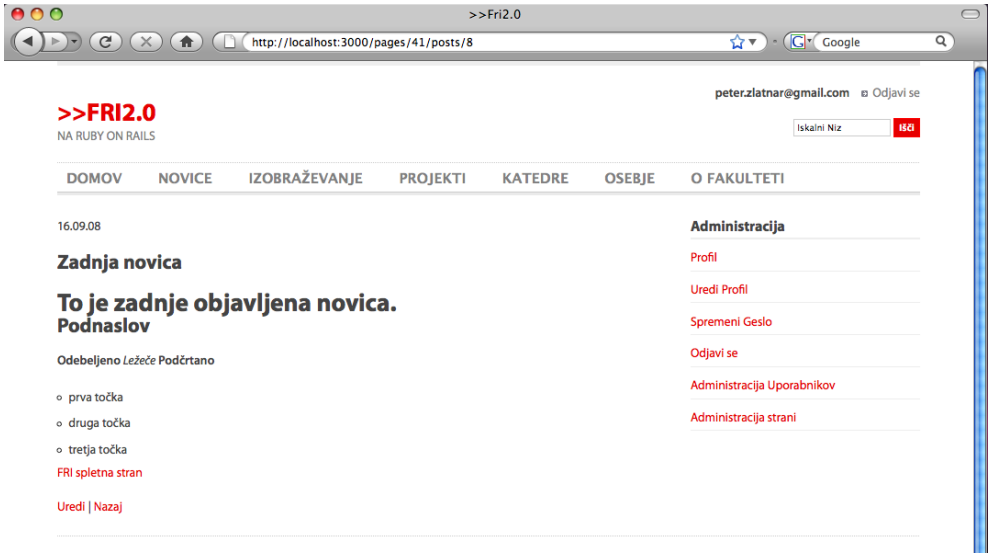
Slika 4.4: Urejanje besedila z jezikom Textile.

urejanja.

4.5 Uporaba tehnologije AJAX

4.5.1 Prijava v sistem in urejanje vsebin

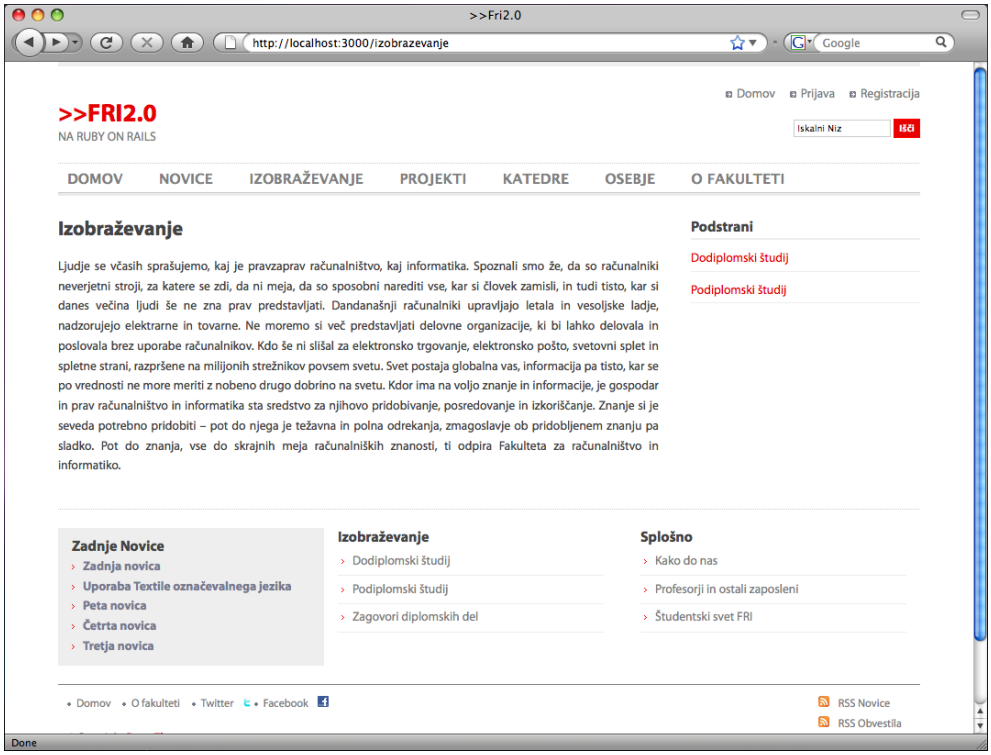
S pomočjo tehnologije AJAX (angl. *asynchronous JavaScript and XML*) smo uporabnikom omogočili prijavo v sistem, ne da bi morali zapustiti trenutno spletno stran. V zgornjem desnem kotu kliknemo na povezavo Prijava in pojavijo se polja za prijavo v sistem. Vpišemo podatke, kliknemo prijava in že so smo prijavljeni v sistem. Prikažejo se nam tudi povezave za administracijo in urejanje osebnega profila. Kar pa je bistvenega pomena, brskalnik ob uporabniških zahtevah ne nalaga spletne strani vedno znova, ampak osveži samo tisti del strani s prijavo. To pomeni, da ima uporabnik vsebino ki jo hoče urediti vseskozi pred očmi. Ko smo prijavljeni v sistem lahko z ustreznimi uporabniškimi pravicami urejamo vsebino. Tudi urejanje vsebine smo



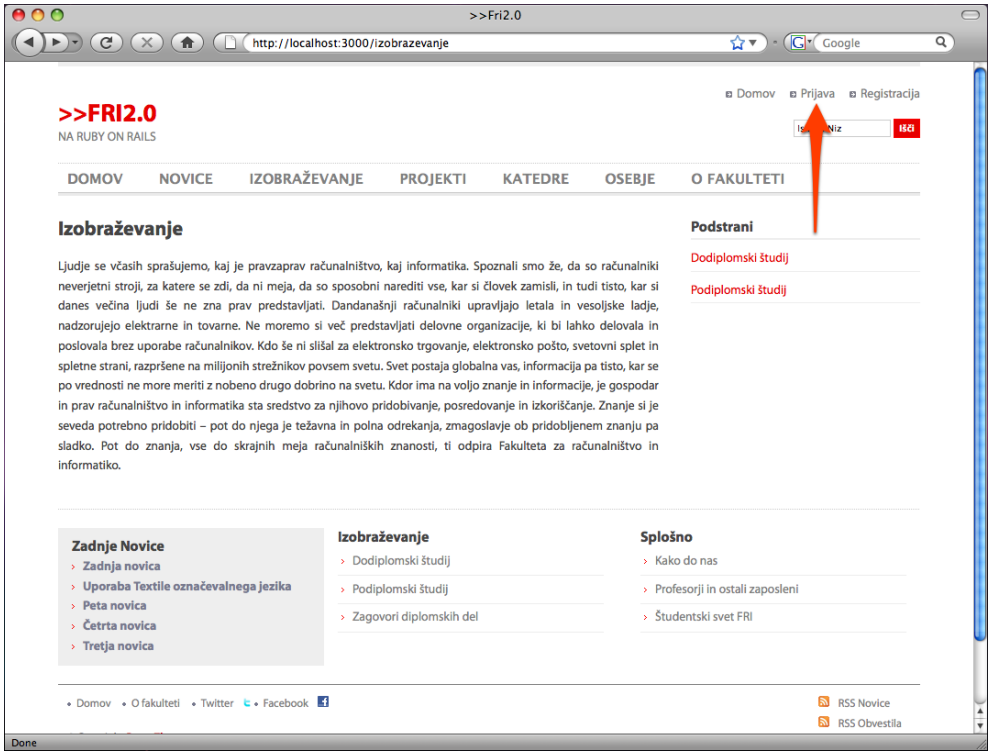
Slika 4.5: Urejeno besedilo z jezikom Textile.

implementirali z uporabo tehnologije AJAX. Uporabnik enostavno klikne na vsebino, ki jo hoče urediti in prikaže se mu obrazec za urejanje. Ponovno se osveži samo obrazec za urejanje vsebine.

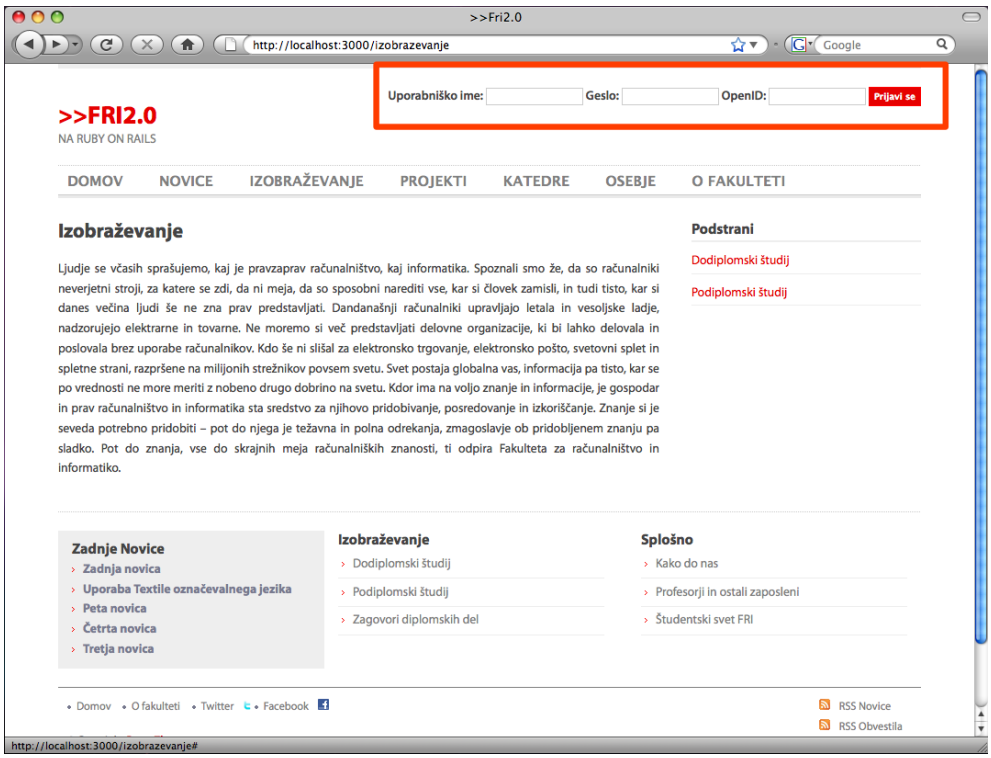
Poglejmo si primer, ko je uporabnik na strani **Izobraževanje** (slika 4.6) in hoče urediti vsebino. Klikne povezavo Prijava (slika 4.7), prikažejo se mu polja za prijavo v sistem (slika 4.8). Po uspešni prijavi se v desnem zgornjem kotu prikaže email naslov prijavljenega uporabnika in povezava za odjavo. Prikažejo se tudi administracijske povezave, tako za urejanje uporabniških nastavitev, kot same strani (slika 4.9). S klikom na besedilo strani se mu prikaže obrazec za urejanje vsebine (slika 4.10).



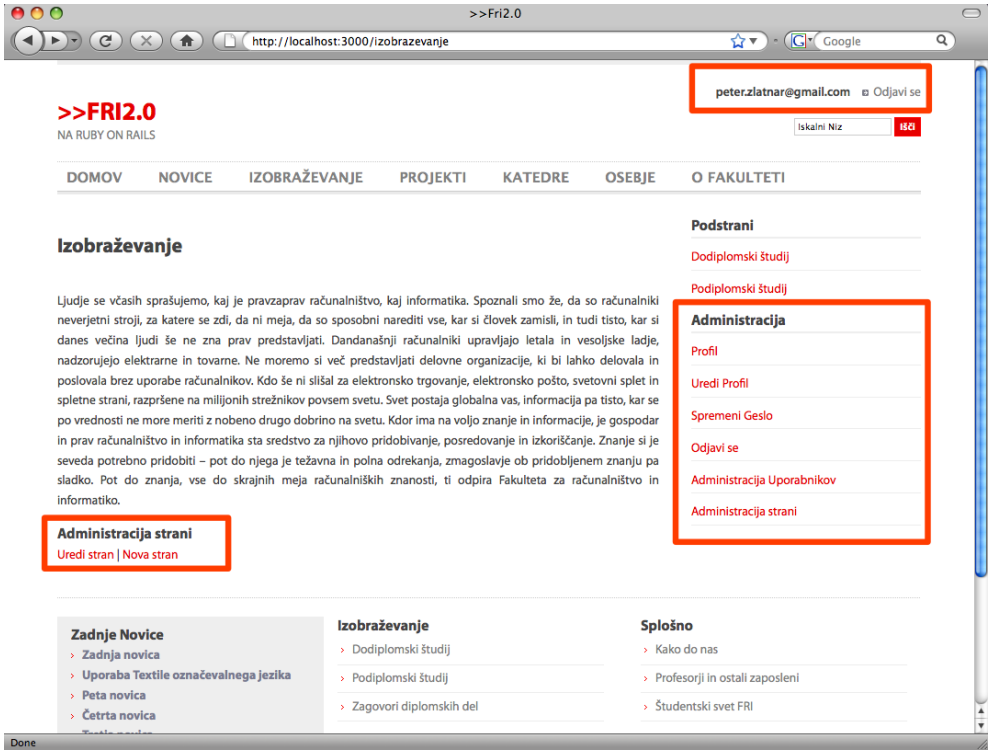
Slika 4.6: Izobraževanje.



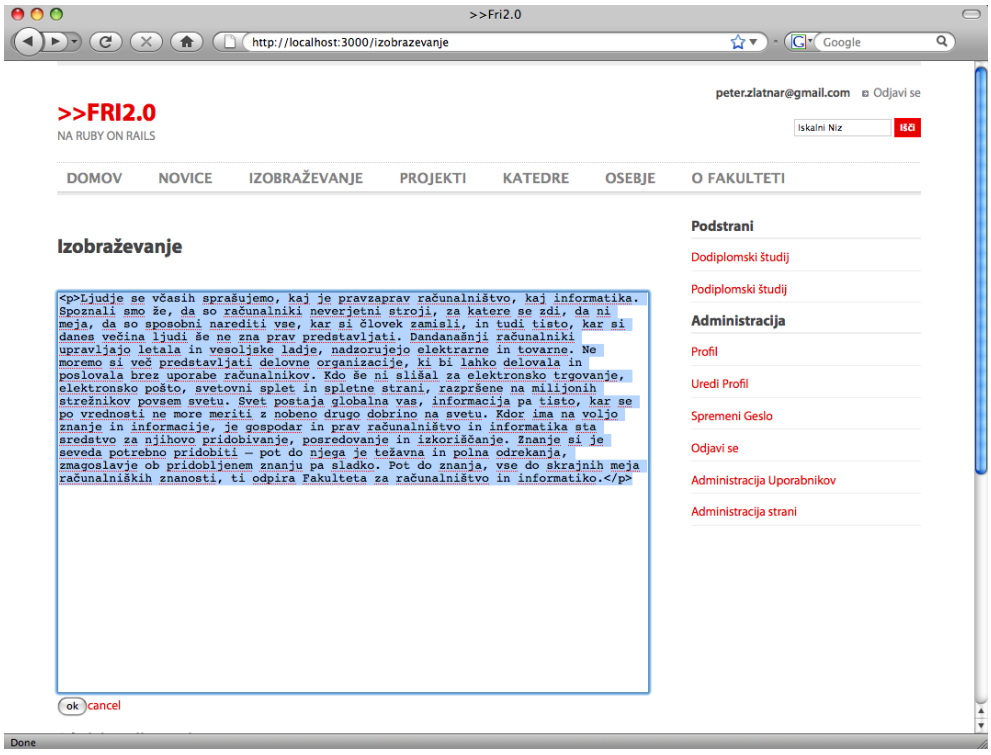
Slika 4.7: Prijava.



Slika 4.8: Prijavna polja.



Slika 4.9: Prijavljen uporabnik.



Slika 4.10: Urejanje vsebine.

Poglavje 5

Zaključek

V diplomski nalogi smo najprej predstavili osnove programskega jezika Ruby, ker se nam zdi pomembno razumevanje konceptov tega jezika za nadaljnje razumevanje dela. Nato smo se poglobili v ogrodje za razvoj spletnih aplikacij Ruby on Rails. Omenili smo nekatere koncepte in principe, ki jih omenjeno ogrodje zagovarja. Predvsem smo se osredotočili na MVC arhitekturo ogrodja. Pogledali smo si ključne razrede s katerimi Rails ogrodje zagotavlja MVC arhitekturo aplikacij. Na enostavnem primeru smo tudi prikazali, kako naredimo novo aplikacijo, ter predstavili uporabo ukaza scaffold. Za konec pregleda ogrodja Rails smo omenili še podporo tehnologiji Ajax.

V nadaljevanju naloge smo preučili možnosti prenove spletnega sistema FRI s tehnologijo Ruby on Rails. Povedali smo kaj je to CMS in kaj so glavne pomanjkljivosti obstoječih sistemov. Prikazali smo nekatere uporabe sistema, predvsem tisti del, ki smo ga v diplomski nalogi poizkušali izboljšati. To pa je enostavnejše urejanje vsebin z uporabo označevalnega jezika Textile in podporo tehnologije Ajax. Omogočili smo na mestu urejanje vsebin in prijavo v sistem, ne da bi moral uporabnik zapustiti trenutno spletno stran.

Na eni strani eleganca programskega jezika Ruby in na drugi strani fleksibilnost ogrodja Rails iz dneva v dan navdušuje vse več razvijalcev spletnih aplikacij. Ogrodje Rails nam omogoča izredno hitro razvijanje fleksibilnih spletnih aplikacij, predvsem na podlagi uporabe že napisanih razredov in dodatkov, ter avtomatizacije določenih delov aplikacije. Ogrodje Rails od nas zahteva, da se držimo določenih pravil in nas tako usmerja skozi celoten proces razvoja aplikacije. Dokler se držimo teh smernic razvoj poteka zelo hitro in pregledno. Seveda pa lahko uberemo tudi svojo pot, ki pa je ponavadi daljša in ne tako udobna. Predvsem pa ogrodje Rails ne povzroča prevelikih glavobolov razvijalcem, nasprotno večina izmed njih je navdušena nad razvojem spletnih aplikacij

v ogrodju Rails. Z gotovostjo lahko trdimo, da ogrodje Rails ni samo trend v svetu razvoja spletnih aplikacij ampak še kako močno kroji razvoj spletnih aplikacij, ter postavlja nove smernice in trende.

Tako kot je ogrodje Rails prineslo svež veter v spletne vode, ga je pred časom tudi tehnologija Ajax, ki je doživela svoj veliki pok s pojavom v spletnem servisu Gmail. Če je Rails pustil močan pečat na razvijalcih potem ga je Ajax predvsem na uporabnikih saj jim zagotavlja boljšo uporabniško izkušnjo. Ker pa so se tega zavedali tudi ustvarjalci ogrodja Rails so na zelo eleganten način vgradili podporo tehnologiji Ajax, ki nam je tako na dosegu roke v nekaj vrsticah kode. Zato bi lahko rekli da nas ogrodje Rails vzpodbuja k uporabi tehnologije Ajax in tako k razvoju ustvarjalnejših aplikacij, za kar so nam vsekakor najbolj hvaležni uporabniki.

V diplomski nalogi smo tako videli, kako lahko na zelo enostaven način ter z uporabo pravih orodij in tehnologij naredimo aplikacijo, ki je prijaznejša do njenih uporabnikov. Aplikacije morajo biti enostavne za uporabo in kar se da prijazne do uporabnikov. Zato smo v diplomski nalogi sledili principu KISS (angl. *Keep It Simple, Stupid*) in idejam avtorjev knjige Getting Real (gettingreal.37signals.com), ki zagovarjajo agilen razvoj spletnih aplikacij. Čeprav je na začetku knjige omenjeno, da vsebina ni usmerjena samo v razvoj spletnih aplikacij in določenih tehnologij je na vsakem koraku čutiti prisotnost filozofije ogrodja Rails, ki omogoča vse kar avtorji v delu navajajo.

Na način, kot danes deluje večina CMS sistemov se spletnih aplikacij ne gradi več. Enostavno ne moremo spregledati dejstva ter zanemariti množice tehnologij in principov razvoja sodobnih spletnih aplikacij. Diplomska naloga s stališča tehnologij ne ponuja nič novega, kot inovacijo s stališča trenutne implementacije spletnega sistema pa združuje način urejanja vsebin, kot ga poznamo iz Wiki sistemov, z dodatkom ščepca čarovnije Ajax. Rezultat je spletni sistem za urejanje vsebin, ki je mnogo prijaznejši do uporabnikov.

V zvezi z zastavljenimi cilji in delom opravljenem v tej diplomski nalogi bi bilo potrebno raziskati in preučiti še nekatere vidike aplikacije ter odgovoriti na določena vprašanja. Tako bi bilo v zvezi z jezikom Textile potrebno preučiti dodatne možnosti njegove razširitve in uporabe v Rails aplikacijah. Sedaj nam jezik omogoča urejanje enostavnih besedil, potrebno pa bi bilo raziskati, kako bi lahko ta označevalni jezik povezali s podatki v podatkovni bazi, tako da bi lahko z določeno oznako vključili na stran podatke iz izbranega modela. Potrebno bi bilo analizirati podatkovno shemo obstoječega sistema in poiskati poti, ki bi nam omogočile enostaven način prenosa podatkov v Rails aplikacijo. Spet na drugi strani bi bilo potrebno postaviti in preizkusiti različna produkcijska okolja za tehnologijo Rails. In nenazadnje, potrebno bi

bilo preučiti uporabniško izkušnjo iz vidika grafične podobe strani, ter izdelati predloge CSS, ki bi to uporabniško izkušnjo dvignile na višjo raven.

Literatura

[1] David Flanagan, Yukihiro Masumoto,, "The Ruby Programming Language," *O'REILLY*, Sebastopol, CA, 2008.

[2] Obie Fernandez, "The Rails Way," *Addison-Wesley*, Upper Saddler River, NY, 2007.

[3] Scott Raymond, "Ajax on Rails," *O'Reilly*, Sebastopol, CA, 2006.

[4] Dave Thomas, David Heinemeier Hansson, "Agile Web Development with Rails: Second Edition," *The Pragmatic Bookshelf*, North Carolina, Dallas, 2007.

[5] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, Juhani Warsta, "Agile software development methods: review and analysis," *VTT Technical Research Centre of Finland*, Espoo, Finland, 2002.

Izjava

Izjavljam, da sem diplomsko nalogo izdelal samostojno pod vodstvom mentorja izr. prof. dr. Blaža Zupana. Izkazano pomoč drugih sodelavcev sem v celoti navedel v zahvali.

Ljubljana, 26.9.2008 Peter Zlatnar