Upravljanje korisničkim računima u Rails okruženju

Čivić, Monika

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

UPRAVLJANJE KORISNIČKIM RAČUNIMA U RAILS OKRUŽENJU

Diplomski rad

Monika Čivić

Osijek, 2016.

Sadržaj

1. UVOD

Osnovni zadatak ovog rada jest upoznavanje s Ruby on Rails programskim okruženjem za izradu web aplikacija i načinima na koje je unutar tog okruženja, uz pomoć Devise dodatka, moguće realizirati sustav s korisničkim računima koji imaju različite korisničke uloge. Prema tome, diplomski zadatak daje detaljan uvid u postavljanje i primjenu Devise sustava za autentikaciju korisnika. Uz Devise, objašnjen je i OmniAuth sustav koji služi za prijavu i registraciju korisnika putem korisničkih računa društvenih mreža ili drugih sustava. U sklopu navedenog, obrađen je način rada s podacima u Railsu kao i rad s Rails upraviteljem rutama te osnove programskog jezika Ruby. Kao praktični dio diplomskog zadatka, izrađena je Rails aplikacija za udrugu AMA-ETFOS. Aplikacija se sastoji od korisničkog i administratorskog dijela, a procesi prijave i registracije ostvareni su putem spomenutih Rails dodataka. Za izradu aplikacije korišteni su i drugi Rails dodaci te primijenjeni aktualni principi web programiranja te brojne web tehnologije.

U sljedećem poglavlju objašnjen je pojam web aplikacija te opisan njihov razvoj i slojevita struktura. Navedene su i ukratko pojašnjene trenutno najznačajnije web tehnologije i primjena tih tehnologija u praktičnom dijelu diplomskog zadatka. Uz to su prikazani osnovni principi programiranja primjenjivi prilikom izrade web aplikacija, primjerice MVC i REST. Osnovicu trećeg poglavlja čini opis Ruby on Rails programskog okruženja za i osnove programskog jezika Ruby. U sklopu teorije Railsa i strukture Rails aplikacija, opisan je rad s Rails dodacima, odnosno *gemovima*, te njihova instalacija i upotreba. Četvrto poglavlje obuhvaća procese vezane za autentikaciju korisnika aplikacije, kako standardnih korisnika, tako i administratora. Opisan je način na koji je autentikacija ostvarena uz pomoć Devise i OmniAuth besplatnih sustava za identifikaciju korisnika. U petom poglavlju predstavljena je aplikacija AlumniETFOS te objašnjena njena struktura i način rada. Aplikacija sadrži sve tehnologije detaljno opisane u prethodnim poglavljima i zapravo je njihova primjena u praksi.

2. WEB APLIKACIJE

U računarstvu, web aplikacija programska je aplikacija koja radi na načelu klijent - poslužitelj, s tim da je klijent neki od web preglednika. Web aplikacije proširile su se razvojem Interneta, ali i razvojem sada već sveprisutnog računarstva u oblaku (engl. *cloud computing*). Jedan od razloga popularnosti web aplikacija jest taj što ih nije potrebno instalirati na korisničkom računalu. Nadalje, uporaba web preglednika kao klijenta čini održavanje i ažuriranje aplikacije vrlo jednostavnim, bez suradnje korisnika. Također, takav pristup omogućava rad aplikacije na svim platformama, od osobnih računala pa sve do tableta i pametnih telefona. Razlika između uobičajene web stranice bilo koje vrste i web aplikacije nije jasno definirana. Web stranice koje su ujedno i web aplikacije imaju karakteristike slične desktop ili mobilnim aplikacijama. Web aplikacija, za razliku od stranice koja je informativnog karaktera, ima veći broj funkcija. Neke od njih su baza za pohranu podataka, mogućnost registracije i prijava korisnika u sustav, interaktivnost, slanje instant poruka i sl. Značajni primjeri web aplikacija danas su društvene mreže (Facebook, Twitter, LinkedIn…), razni forumi, ali i sustavi za upravljanje sadržajem (engl. *content management system - CMS*) kao što je Wordpress. Praktični dio zadatka ovog rada izrada je jedne takve web aplikacije te će u narednim poglavljima biti objašnjene tehnologije, obrasci i alati karakteristični za izradu web aplikacija.

2.1. Povijesni razvoj i struktura

U ranijim računalnim modelima, kao što je klijent - poslužitelj model, obrada zahtjeva bila je podijeljena između programskog koda koji se nalazi na poslužitelju i koda instaliranog na lokalnom računalu. Aplikacije su imale svoj vlastiti klijent u službi korisničkog sučelja koji mora biti postavljen na osobnom računalu svakog pojedinog korisnika. U tom slučaju je ažuriranje programskog koda na poslužiteljskoj strani zahtijevalo ažuriranje koda na klijentskoj strani. Točnije, na svakoj radnoj stanici zasebno. Nadalje, i poslužiteljski i klijentski dijelovi aplikacije bili su uglavnom usko vezani za računalnu arhitekturu i operacijski sustav.

Nasuprot takvim aplikacijama, web aplikacije koriste web dokumente napisane u standardnom obliku kao što su HTML ili JavaScript jezici. Gotovo svi današnji web preglednici imaju podršku za takve oblike. Web aplikacije mogu se promatrati kao specifični klijentsko-poslužiteljski program gdje je klijentski dio programa preuzet na klijentski uređaj prilikom posjećivanja određene web stranice koristeći standardne procedure kao što su HTTP zahtjevi. Ažuriranje klijentskog web programa može se dogoditi svaki put kada je stranica posjećena. Tijekom sesije,

web preglednik interpretira i prikazuje stranice te se ponaša kao jedinstveni klijent za svaku web aplikaciju.

U ranim danima weba, kako je navedeno u [1], svaka stranica dostavljena je klijentu kao statički dokument, ali slijed stranica i način na koji su povezane i dalje je mogao pružiti interaktivno korisničko iskustvo. Ipak, svaka značajna promjena na web stranici zahtijevala je povratak na poslužitelj ne bi li se stranica osvježila. Godine 1995. Netscape je uveo klijentsko-poslužiteljski skriptni jezik nazvan JavaScript koji je programerima omogućio dodavanje dinamičkih elemenata u korisničko sučelje koje se izvodilo na strani klijenta. Tako je, umjesto slanja podataka poslužitelju s ciljem generiranja čitave stranice, ugrađena skripta izvodila različite zadatke poput validacije korisničkog unosa ili skrivanja i prikaza određenih dijelova stranice.

Godine 1996. tvrtka Macromedia predstavila je Flash, program za prikaz animacija koji se postavlja u preglednike kao dodatak (engl. *plug-in*) kako bi se animacije uključile u web stranicu. Nadalje, 1999. godine koncept web aplikacija uveden je i u Java programski jezik. Kasnije, 2005. godine razvojem AJAX-a počele su nastajati brojne web aplikacije, a jedna od njih je i popularni Gmail. 2011. godine dovršen je HTML5 koji omogućuje grafičke i multimedijske sadržaje bez potrebe za dodacima na klijentskoj strani. HTML5 doprinosi važnosti kreiranja web aplikacija. Ipak, svi navedeni jezici samo djelomice poboljšavaju klijentsko – poslužiteljsku komunikaciju. Pravi razvoj web aplikacija i dinamičkih web stranica nemoguć je bez potpore nekog od programskih jezika više razine. Oni tako čine zaseban sloj u strukturi svake aplikacije na webu. Jedan od takvih jezika je Ruby, a programsko okruženje Ruby on Rails napisano u Rubyju detaljnije je objašnjeno u narednim poglavljima.

Većinom su aplikacije prema [2] podijeljene na tri logička dijela koja se nazivaju slojevima te su grafički prikazani na slici 2.1.. Svaki aplikacijski sloj ima određenu ulogu. Tradicionalne aplikacije sastojale su se od samo jednog sloja, koji se nalazio na klijentskom stroju, ali web aplikacije imaju n-slojnu strukturu od kojih je najčešća troslojna. Troslojna arhitektura web aplikacija sastoji se od prezentacijskog, aplikacijskog i podatkovnog sloja. Prezentacijski sloj najbliži je korisniku i služi za prikaz informacija. U ovom je slučaju to web preglednik. Prezentacijski je sloj najčešće ostvaren pomoću opisnih i stilskih jezika. Drugi ili aplikacijski sloj upravlja aktivnostima koje aplikacija treba izvršavati. Taj sloj zadužen je za obradu zahtjeva, informacija i cjelokupan rad aplikacije. Aplikacijski sloj koristi neku od tehnologija za kreiranje dinamičkog web sadržaja kao što su ASP, CGI, ColdFusion, Node.js, Ruby on Rails, PHP ili Phyton. Podatkovni sloj, ujedno i posljednji sloj, služi za upravljanje podacima iz baze na

udaljenom poslužitelju. Web preglednik šalje zahtjev srednjem sloju koji ga poslužuje kreiranjem upita i ažuriranjem baze podataka te se zatim generira korisničko sučelje s odgovarajućim, promjenama podložnim, sadržajem.

Sl. 2.1. *Troslojna struktura web aplikacije*

2.2. Razvojne tehnologije i alati

Sukladno opisanoj strukturi web aplikacija u potpoglavlju 2.1., prilikom razvoja web stranica i aplikacija primjenjuju se određene tehnologije. Tehnologije za izradu prezentacijskog sloja aplikacije AlumniETFOS su HTML5 i CSS3 te JavaScript. Za aplikacijski i podatkovni sloj korišten je Ruby on Rails. Osim poznavanja navedenih, potrebno je osnovno razumijevanje SQL jezika za rad s relacijskim bazama podataka.

2.2.1. Web tehnologije

HTML5 je peta i trenutna inačica HTML (engl. *HyperText Markup Language*) standarda, opisnog jezika za kreiranje i prikaz sadržaja na webu, odnosno kreiranje hipertekstualnih datoteka. Rad s HTML-om zasnovan je na korištenju oznaka (engl. *tag*) koje web pregledniku ukazuju na željeni način prezentacije sadržaja. Različite oznake određuju je li pojedini dio teksta poveznica, slika, odlomak ili nešto drugo. Oznakama se mogu pridružiti i klase ili identifikatori što pomaže u kasnijem stiliziranju prikaza pomoću CSS-a (engl. *Cascading Style Sheets*). Treba naglasiti da HTML nije programski jezik i ne može izvršiti nikakvu zadaću, čak ni osnovne matematičke operacije. HTML5, za razliku od prijašnjih verzija, podržava oznake za multimedijski sadržaj i olakšava validaciju obrazaca (engl. *form*) bez korištenja drugih programskih jezika. HTML datoteke imaju ekstenziju .html ili .htm. Ipak, HTML dokumenti

korišteni u praktičnom dijelu rada sadrže ekstenziju htlm.erb. Razlog tome je sadržaj preuzet iz baza podataka i korištenje Ruby programskog jezika u sklopu HTML-a. Na taj način, umetanjem Ruby oznaka („<%= … %>“ i „<% … %>“) kao u kodu 2.1., dobiva se dinamički web sadržaj koji se mijenja ovisno o korisniku, trenutnoj stranici i drugim uvjetima. U kontekstu Railsa, HTML dokumenti zapravo su pogledi MVC modela detaljnije opisanog u poglavlju 2.3. Također, Rails omogućuje korištenje parcijalnih dokumenata (engl. *partials*) koje je moguće umetnuti u osnovni dokument, ali i čitavih okvira (engl. *layout*) primjenjivih na određeni dio aplikacije. Cilj takvog pristupa je pojednostavniti prikaz i stiliziranje stranica, ali i izbjeći ponavljanje koda. Prikaz koda 2.1. sadrži dio HTML-a podnožja stranice koje se proteže kroz sve korisniku prikazane dokumente aplikacije AlumniETFOS, a dobar je primjer korištenja osnovnog HTML-a u kombinaciji s Ruby on Rails oznakama i pomoćnim metodama detaljnije objašnjenim u trećem poglavlju. Točke unutar koda 2.1., ali i svakog sljedećeg prikaza izvornog koda, naznaka su da je određeni dio koda izostavljen.

```
<section class="footer">
.
.
.
<div class="row">

<div class="col-xs-4 col-sm-2 col-md-2 col-lg-2 col-sm-pull-1 col-md-pull-
1 col-lg-pull-1">
<%= image_tag 'pages.svg', :class => 'hvr-grow'%>
</div>

<div class="col-xs-6 col-sm-9 col-md-9 col-lg-9 col-sm-pull-2 col-md-pull-
2 col-lg-pull-2">
<h5 id="pages-headline">STRANICE</h5>
<%= link_to "Početna", root_path %>
<%= link_to "O udruzi", about_path %>
<%= link_to "Statut", statut_path %>
<%= link_to "Vijesti", news_path %>
<%= link_to "Kontakt", contact_path %>
</div>
</div>
.
.
.
</section>
```

Kod 2.1. Dio HTML koda podnožja stranice

Unatoč oznakama, dokumenti napisani samo u HTML-u izgledaju vrlo jednostavno i nimalo privlačno korisniku jednom kada su prikazani u web pregledniku. Kako bi se to izbjeglo, osmišljen je CSS. CSS je stilski jezik kojim se uređuje izgled i raspored elemenata web stranice. CSS je moguće pisati unutar HTML datoteke, ali kako to čini kod nepreglednim, bolja je praksa

CSS pohraniti u zasebnu datoteku s ekstenzijom .css i zatim ju povezati sa željenim HTML-om. Na taj se način isto oblikovanje može primijeniti na sve stranice. Trenutno je aktualna treća inačica CSS-a, CSS3 koja osim uobičajenih promjena kao što su boje ili pozicioniranje elemenata, omogućuje i jednostavne animacije. Iako vrlo praktičan i funkcionalan pokazalo se da CSS ima i neke nedostatke. Primjerice, nemogućnost stvaranja varijabli, za pohranu vrijednosti koje se često ponavljaju, čini izmjene u kodu prilično zahtjevnima. Zbog toga se CSS često kombinira sa Sass-om (engl. *Syntetically Awesome Stylesheets*), skriptnim jezikom koji proširuje mogućnosti CSS-a i ubrzava pisanje koda. Ispis koda 2.2. prikazuje CSS stilove primijenjene na dokumentu prikazanom u kodu 2.1..

```
$dark_blue: #05376C;
$white: #ffffff;

.footer {
   margin: 0;
   height: 100%;
   width: 100%;
   padding: 6rem 0;
   background-color: $dark_blue;

   h1,h2,h3,h4,h5,h6 {
      color: $white;
      display: inline;
    }
    .
    .
    .
    a:first-of-type {
      margin-top: 3rem;
    }

    a:nth-of-type(5) {
      margin-bottom: 6rem;
    }
}
```

Kod 2.2. Dio CSS stilova za podnožje stranice

HTML5 i CSS3 dovoljni su za izradu web stranica s osnovnom funkcionalnošću i informativnom svrhom, ali ono što takvim stranicama nedostaje jest dinamički generiran sadržaj. Iz tog razloga postoje aplikacijski i podatkovni slojevi. Aplikacijski sloj, kao što je ranije rečeno, temelji se na nekim od programskih jezika više razine. Uz ovdje korišteni Ruby u sklopu Ruby on Railsa, slične mogućnosti pružaju ASP ili PHP. Ruby, kao i Ruby on Rails obrađeni su detaljnije u trećem poglavlju. Rails, osim dinamičkog sadržaja i umetanja programskog koda unutar HTML-a omogućuje komunikaciju s bazom podataka i pruža brojne pomoćne funkcije za brže pisanje HTML koda, kao što su obrasci za registraciju i slično. U programskom kodu 2.1. već su

upotrijebljene neke od njih, primjerice „link_to“ za generiranje „<a>“ oznaka, tj. poveznica. U kodu 2.3. prikazano je kako Rails omogućuje dinamičko generiranje obrazaca za unos podataka.

```
<%= form_for(resource, as: resource_name, url: session_path(resource_name))
do |f| %>
<div class="form-group">
<%= f.label :email %><br/>
<%= f.email_field :email, autofocus: true, :class => 'form-control' %>
</div>

<div class="form-group">
<%= f.label :password %><br/>
<%= f.password_field :password, autocomplete: "off", :class => 'form-
control' %>
</div>

<div class="actions">
<%= f.submit "Log in", :class => 'btn btn-login center-block' %>
</div>
<% end %>
```

Kod 2.3. HTML u kombinaciji s Rubyjem - obrasci

Budući da većina aplikacija na webu sadrži bazu podataka, važno je spomenuti SQL (engl. *Structured Query Language*) jezik koji služi za izradu, pretraživanje, ažuriranje i uklanjanje informacija iz baze. Za dohvaćanje i prikaz podataka iz baze, programski kod treba sadržavati naredbe za komunikaciju s bazom. Ranije je spomenuto kako Rails ima velik broj funkcija koje olakšavaju pisanje koda. Zbog toga za rad s bazama u Railsu nije potrebno poznavati točnu sintaksu SQL-a, već samo osnovni način rada i Rails funkcije analogne SQL naredbama. Više o povezanosti Railsa i SQL-a nalazi se odjeljku 3.2.2..

2.2.2. UNIX komandna linija

Osnovna ideja komandne linije prilično je jednostavna. Temelji se na tome da korisnik, unoseći kratke naredbe, može izvesti veliki broj operacija, kao što su kreiranje direktorija, premještanje i kopiranje datoteka ili navigacija kroz datotečni sustav. Iako korištenje komandne linije korisnicima naviknutim na grafičko sučelje, može izgledati zastarjelo ili komplicirano, treba imati na umu da izgled vara. Komandna linija zapravo je jedan od najmoćnijih programerskih alata, a za rad s Railsom na Linux baziranim operacijskim sustavima nužno je njeno osnovno poznavanje i korištenje. U tablici izrađenoj prema [3] nalazi se popis najčešće korištenih naredbi prilikom razvoja web aplikacije s Rails okruženjem na Fedora operacijskom sustavu.

Tab. 2.1. Najčešće korištene naredbe

Naredba	Opis naredbe
`ls`	Prikaz sadržaja trenutnog direktorija.
`mkdir<ime_direktorija>`	Kreiranje direktorija.
`cd <ime_direktorija>`	Promjena radnog direktorija.
`cd ..`	Premještanje u direktorij iznad trenutnog radnog direktorija.
`cd ~ ili samo cd`	Odlazak u *home* direktorij.
`cd ~/<ime_direktorija>/`	Odlazak do nekog direktorija.
`mv <izvor><meta>`	Premještanje datoteke iz jednog u drugi direktorij.
`cp <izvor><meta>`	Kopiranje datoteke iz jednog u drugi direktorij.
`rm <datoteka>`	Uklanjanje datoteke.
`rmdir <direktorij>`	Uklanjanje praznog direktorija.
`rm -rf <direktorij>`	Uklanjanje direktorija koji nije prazan.
`cat <datoteka>`	Povezivanje i ispis sadržaja datoteke.
`clear`	Brisanje prethodnih naredbi komandnog prozora.
`touch<datoteka>`	Kreiranje nove datoteke.

2.2.3. Upravljanje izvornim kodom pomoću gita

Prilikom rada na web, ali i ostalim aplikacijama, dobra je programerska praksa vršiti kontrolu izvornog koda. Sustavi za upravljanje izvornim kodom omogućuju praćenje promjena na projektu, lakšu suradnju s ostalim razvojnim programerima koji na projektu sudjeluju, ali i povratak na određenu točku razvoja ukoliko se pojavi pogreška. Poznavanje korištenja sustava za upravljanje izvornim kodom, postalo je jedna od neophodnih vještina povezanih s izradom programske podrške.

Postoji mnogo različitih upravljačkih sustava, ali Rails zajednica se na neki način već standardno opredjeljuje za git. Git je distribuirani sustav za upravljanje izvornim kodom nastao 2005. godine, a osmislio ga je Linus Trovalds za razvoj Linux jezgre. Naglasak gita je na brzini, očuvanju podataka i podršci za distribuirani rad. Korištenje takvog sustava nije preporučljivo samo zato što je to uobičajena Rails praksa, već zato što omogućuje lakše praćenje i dijeljenje izvornog koda. Neki od besplatnih, ili djelomice besplatnih, online sustava temeljenih na gitu su Bitbucket, GitHub i Gitorious. Praktično je što se rad i postavljanje koda na udaljeni git sustav

može vršiti iz komandne linije što čini pohranu i praćenje brzima i učinkovitima. Najčešće korištene naredbe prilikom rada s gitom u sklopu izrade aplikacije AlumniETFOS nalaze se u tablici 2.2. izrađenoj prema [4].

Tab. 2.2. Najznačajnije naredbe za rad s gitom

Naredba	**Opis**
`git init`	Kreiranje novog git repozitorija. Može se koristiti za pretvorbu postojećeg koda u git repozitorij ili kreiranje novog praznog repozitorija.
`git clone`	Kopira postojeći git repozitorij.
`git config`	Služi za konfiguraciju git instalacije.
`git add`	Dodavanje datoteka u git repozitorij.
`git commit`	Spremanje verzije projekta u danom trenutku.
`git push`	Pohrana svih promjena i *commit* verzija na udaljeni sustav, primjerice GitHub.
`git pull`	Preuzimanje trenutne verzije koda s udaljenog sustava na lokalno računalo.
`git log`	Prikaz svih *commit* verzija koje *postoje u gitu.*
`git diff`	Prikaz datoteka u kojima su se dogodile promjene u odnosu na prethodni *commit*.
`git status`	Prikaz svih datoteka koje će biti pohranjene ako uslijedi *git commit* te datoteka koje bi mogle biti pohranjene ako uslijedi naredba *git add*.
`git checkout`	Između ostalog, služi za provjeru trenutne grane git repozitorija.

2.3. Obrasci programske arhitekture

Obrasci programske arhitekture odnose se na skup pravila i uobičajenih metoda prilikom programiranja, kako web, tako i ostalih aplikacija. Ta se pravila odnose na način izrade programske strukture aplikacije, upravljanje podacima i kreiranje korisničkog sučelja. Iako postoji mnogo različitih pristupa, detaljnije su razrađeni obrasci koji su korišteni za izradu praktičnog dijela ovog rada, a karakteristični su za Rails razvojno okruženje.

2.3.1. MVC

Jedan od najčešće korištenih obrazaca je MVC(engl. *Model View Controller).* U doslovnom prijevodu model-pogled-upravitelj arhitektura način je kreiranja programske podrške koji promiče razlikovanje poslovne logike aplikacije od unosa i prezentacijske logike povezane s grafičkim korisničkim sučeljem. U slučaju web aplikacija, poslovna logika većinom se sastoji od

podatkovnih modela kao što su korisnici, članci ili proizvodi dok je prezentacijska logika izgled stranice u web pregledniku.

Prema [3], kada komunicira s web aplikacijom, preglednik šalje zahtjev koji web poslužitelj prima i prosljeđuje upravitelju zaduženom za daljnje akcije. U nekim slučajevima upravitelj odmah generira pogled, što je zapravo predložak koji se prebacuje u HTML jezik i šalje pregledniku. U većini slučajeva, upravitelj ipak komunicira s modelom koji je objekt i predstavlja element stranice (primjerice korisnika) te je zadužen i za komunikaciju s bazom podataka. Nakon pozivanja modela, upravitelj zatim generira pogled i vraća kompletnu web stranicu pregledniku u obliku HTML-a.

Programski kod 2.4. sadrži definiciju modela korisnika pod nazivom *Post*. Model je u ovom slučaju Ruby klasa koja komunicira s bazom podataka i dohvaća stvarni model korisnika koji se nalazi u bazi. Korisnik ima nekoliko stupaca, kao što su naslov, sadržaj i ostali podaci, ali njih nije potrebno dodatno definirati kao atribute unutar klase jer to Rails odrađuje na njemu unaprijed definiran način. Ono što se dodaje unutar klase su različite validacije za pojedine stupce, primjerice, validacija prisutnosti, validacija duljine i oblika korisničkog sadržaja, validacija tipa slike i brojne druge.

```
class Post < ActiveRecord::Base

has_attached_file :image, styles: {
        large: '2560x1680#',
        medium: '1280x840#',
        thumb:'100x100#'
        },
        default url: "/images/:style/missing.png"

validates_attachment_content_type :image, content_type: /\Aimage\/.*\Z/,
presence: true

validates :title,
        presence: true,
        length: { minimum: 2, maximum: 200 }

validates :content,
        presence: true,
        length: { minimum: 50 }

validates :author,
        presence: true,
        length: { minimum: 2, maximum: 50 }

end
```

Kod 2.4. Model objave (*Post*) stranice AlumniETFOS

Uloga upravitelja u Railsu je generiranje pogleda i manipulacija podacima. U dijelu izvornog koda 2.5. vidljivo je da upravitelj uz određene metode može sadržavati i filtere. U ovom slučaju filter *before_action* zajedno s Devise naredbom *autehenticate_user!* naznačuje da ukoliko korisnik nije prijavljen, metode koje slijede ne mogu biti izvršene niti im se može pristupiti. Na taj način ostvaruje se zaštita podataka pojedinog korisnika kako oni ne bi bili vidljivi ostalim korisnicima aplikacije ili bilo komu drugom na webu. Metoda *show_profile* pohranjuje trenutnog korisnika u instanciranu varijablu i generira HTML za pogled s podacima pohranjenim unutar varijable *@user*.

```
class UsersController < ApplicationController
        before_action :authenticate_user!
        def show_profile
                @user = current_user
                render ('users/show_profile')
        end
end
```

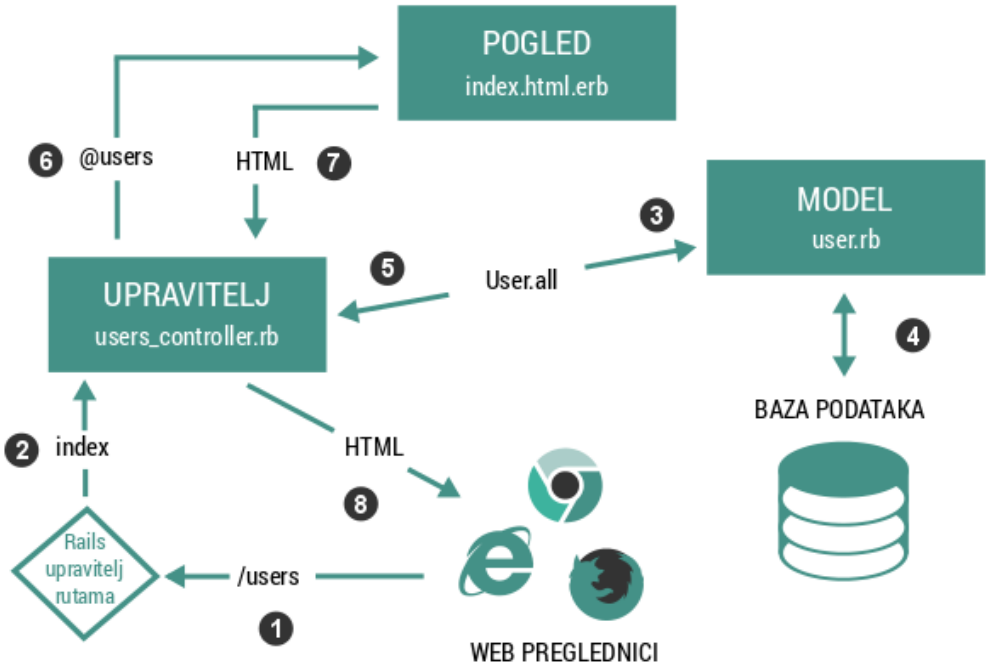
Kod 2.5. Upravitelj korisnika stranice AlumniETFOS

Generirani pogled jedinstven je za svakog korisnika i služi za prikaz određenih podataka pohranjenih u bazi. Dio koda 2.6., osim HTML jezika, sadrži i umetnuti Ruby kod za pristup pojedinim atributima instanciranog objekta *@user* koji su prikazani na stranici.

```
<%if @user.first_name != "" || @user.last_name != "" %>
<h1><%= @user.first_name + " " + @user.last_name %></h1>
<% else %>
<h1>Ime i prezime</h1>
<% end %>

<div class="col-md-6 col-md-offset-3">

<table class="table table-striped">
<tr>
        <td><b>Email:</b></td>
        <td><%= @user.email %></td>
</tr>
<tr>
        <td><b>Završetak studija:</b></td>
        <td><%= @user.graduation_year %></td>
</tr>
<tr>
        <td><b>Radno mjesto:</b></td>
        <td><%= @user.work_place %></td>
</tr>
</table>

<%= link_to "Uredi profil", edit_path, :class => 'btn btn-primary' %>

</div>
```

Kod 2.6. Pogled korisnika stranice AlumniETFOS za metodu *show_profile*

Ukratko, prema [3], model direktno upravlja podacima, logikom i pravilima aplikacije. Pogled može biti bilo kakav izlazni podatak koji je reprezentacija informacija, a upravitelj prihvaća ulazne podatke i pretvara ih u naredbe za modele ili poglede. Model pohranjuje podatke dobivene od upravitelja ili podatke koji su prikazani u pogledu. Pogled zatim generira novi izlaz vidljiv korisniku, temeljen na promjenama modela, dok upravitelj šalje naredbe modelu ne bi li došlo do ažuriranja njegova stanja. Slikovni prikaz opisanih procedura za Rails programsko okruženje nalazi se na slici 2.2.. Osim Ruby on Rails programskog okruženja za razvoj web aplikacija, postoje i druga okruženja koja su također nastala na MVC principu. Neka od njih su Django, ASP.NET i Express. Kako tehnologije napreduju, okruženja poput AngularJS-a, EmberJS-a, JavaScriptMVC-a i Backbonea kreirana su na način koji dopušta djelomično izvođenje MVC komponenti na klijentskoj strani.



Sl. 2.2. *Model – pogled – upravitelj arhitektura Rails aplikacije*

2.3.2. REST i CRUD

Prilikom čitanja i istraživanja o web aplikacijama, posebice onim izrađenim u Railsu, nemoguće je zaobići izraz REST koji je kratica engleskog izraza *REpresentational State Transfer*, što u prijevodu znači reprezentativni prijenos stanja. REST je arhitekturni stil za razvoj distribuiranih, mrežnih sustava i programskih aplikacija ako što se World Wide Web i web aplikacije. Iako je teorija REST-a prilično apstraktna, u okviru web aplikacija REST znači da je većina komponenti aplikacije modelirana kao resurs koji može biti stvoren, pročitan, ažuriran ili uklonjen. To su operacije koje odgovaraju operacijama iz teorije relacijskih baza podataka. Isto

tako, te četiri metode analogne su temeljnim HTTP zahtjevima i to: *POST, GET, PATCH* i *DELETE* kako je navedeno u [3].

Prilikom razvoja web aplikacija, REST pristup olakšava izbor koje je upravitelje i akcije potrebno napisati. Aplikacija se jednostavno kreira koristeći resurse koji mogu biti stvoreni, čitani, ažurirani ili uklonjeni. U slučaju upravljanja korisničkim računima web aplikacije, takav je pristup, na neki način, očit jer su korisnici i administratori zapravo resursi koje je moguće pohraniti, pročitati, izmijeniti ili ukloniti. Ipak, ponekad korištenje ovog principa nije tako očito pa to treba uzeti u obzir prilikom kreiranja upravitelja i ostalih metoda.

Prvi primjer u kojem REST nije primjenjiv jest stvaranje web stranica statičkog sadržaja u sklopu aplikacije. Budući da svaka dobro izrađena web aplikacija ima bazu podataka, treba naglasiti da REST slijedi osnovne operacije za rad s relacijskim bazama podataka ujedinjenim pod kraticom CRUD (engl. *create, read, update, delete*). To su operacije za rad s podacima te omogućuju stvaranje i spremanje podataka u bazu, čitanje podataka iz baze, njihovu izmjenu i, dakako, brisanje. Navedeno je primijenjeno unutar aplikacije izrađene u sklopu rada i to u upravitelju za objavu vijesti. Svaka vijest može biti napisana, objavljena na stranici, izmijenjena ili obrisana kao što je prikazano u kodu ispisanom u 2.7..

```
class PostsController < ApplicationController
before_filter :authenticate_admin!, :except => [:posts]
layout "admin_layout"

def index
        @posts = Post.all
end

def show
        @post = Post.find(params[:id])
end

def new
        @post = Post.new
end

def create
        @post = Post.new(post_params)
        if @post.save
                redirect_to(:action => 'index')
        else
                render('new')
        end
end

def edit
        @post = Post.find(params[:id])
end

def update
```

```
        @post = Post.find(params[:id])
        if @post.update_attributes(post_params)
                redirect_to(:action => 'show', :id => @post.id)
        else
                render('edit')
        end
end

def delete
        @post = Post.find(params[:id])
end

#budući da ne postoji view za destroy ne treba nam varijabla s @
def destroy
        post = Post.find(params[:id])
        post.destroy
        redirect_to(:action => 'index', :id => post.id)
end
.
.
.
end
```

Kod 2.7. Upravitelj objavama/vijestima

Osim ovih jasno definiranih i opće prihvaćenih pristupa, prilikom razvoja treba imati na umu još jedan vrlo značajan pristup koji vrijedi za bilo koju vrstu razvoja programske podrške. Često se može vidjeti kao kratica DRY (engl. *don't repeat yourself*), a jednostavno znači da ne bi trebalo ponavljati iste dijelove koda. Ukoliko se to dogodi, treba razmisliti o pisanju funkcije, definiranju parcijalnih dokumenta (engl. *partials*) ili nekom drugom rješenju. Potonje će izvorni kod učiniti preglednijim te pogodnim za održavanje i izmjene.

3. RUBY ON RAILS

Ruby on Rails programsko je okruženje za izradu web aplikacija napisano u programskom jeziku Ruby. Ruby on Rails nastao je prema REST uzorku izrade programa, a jedno je od najpopularnijih besplatnih rješenja za brz razvoj aplikacija prema MVC i REST načelima.

3.1. Programski jezik Ruby

Ruby je dinamičan, objektno orijentiran programski jezik opće namjene. Razvijen je sredinom 1990-ih godina, a za to je zaslužan Japanac Yukihiro Matsumoto. Prema riječima svog tvorca, Ruby je nastao pod utjecajem programskih jezika Perl, Smalltalk, Eiffel, Ada i Lisp. Ruby podržava brojne programske paradigme, funkcijske i objektno-orijentirane. Također, sadrži sustav dinamičkih tipova podataka i ima automatsko upravljanje memorijom.

Prvo javno izdanje Rubyja, Ruby 0.95, objavljeno je, prema [5] u prosincu 1995. godine i već se tada pojavljuju karakteristike prepoznatljive za kasnija izdanja ovog programskog jezika. Neke od njih su objektno-orijentirani dizajn, klase, rukovanje iznimkama, sakupljač smeća i slično. Nakon tog prvog izdanja, kroz godine su uslijedila i ostala. Ruby 1.8 objavljen je 2003. te je bio aktualan sve do 2013. godine. Također je bio dio nekoliko različitih industrijskih standarda. Nakon 1.8 pojavljuje se verzija 1.9 koja donosi značajne promjene u odnosu na svoju prethodnicu. Sljedeće verzije kao što su 2.0, 2.1 i 2.2 odlikuju se još većim brojem noviteta, ponajviše vezanih za sintaksu, uvođenje novih metoda i nadogradnju postojećih biblioteka, ispravljanje ranijih pogrešaka i uklanjanje zastarjelih API-ja. Ruby 2.3 verzija objavljena je u prosincu 2015. godine. Najizrazitije promjene odnose ne na rad sa *stringovima* i parovima vrijednosti i atributa, u Rubyju poznatijim kao *hash.* Također, Ruby 2.3 donio je brojna izvedbena poboljšanja, ažuriranja postojećih značajki i brojne druge. Ovo je i trenutno aktualna verzija, a najavljene su verzije 2.4 i 3.0 čija izdanja se očekuju u skorijoj budućnosti.

Zasigurno jedna od najznačajnijih karakteristika programskog jezika Ruby jest objektna orijentiranost. Svaka vrijednost u Rubyju ujedno je i objekt, uključujući klase i instance tipova podataka koje mnogi drugi programski jezici promatraju kao primitivne, primjerice *integer* i *bool* vrijednosti. Ruby varijable uvijek sadrže referencu na objekt, a svaka funkcija ujedno je i metoda objekta. Metode koje su definirane u gornjem sloju postaju metode glavne klase *Object.* Budući da je ta klasa roditeljska klasa svake druge klase, takve metode mogu biti pozvane na bilo kojem objektu. One su također vidljive u svakom dometu (engl. *scope*) i učinkovito služe kao globalne procedure. Ruby, međutim, ne podržava višestruko nasljeđivanje, ali se zato u klase mogu

umetnuti razni moduli. Važno je napomenuti kako Ruby ima jako velik broj ugrađenih funkcija koje kodiranje čini lakšim, a njihova upotreba prilično je intuitivna. Ruby podržava i proceduralno i objektno orijentirano programiranje što ga čini vrlo moćnim programskim jezikom opće namjene kako je navedeno u [6].

Sintaksa programskog jezika Ruby slična je sintaksi programskih jezika Phyton i Perl. Definicije klasa i metoda označene su ključnim riječima dok blokovi koda mogu biti definirani i ključnim riječima i vitičastim zagradama. Prelazak u novi red signalizira završetak jedne linije koda te uporaba točka-zareza nije potrebna, ali se može koristiti. Indentacija teksta nije nužna i nema drugu ulogu osim bolje preglednosti koda. U ispisu 3.1. nalazi se dio Ruby koda korišten u aplikaciji za dobivanje djelomičnog i punog naslova pojedine stranice. Iako jednostavan, prikazani kod obuhvaća osnovnu Ruby uporabu kao što je definicija klasa i metoda, ali i korištenje zadanih Ruby funkcija.

```
module ApplicationHelper

  def full_title(page_title = '')
    base_title = "ETFOS Alumni"
    if page_title.empty?
      base_title
    else
      page_title + " | " + base_title
    end
  end

  def first_50(post_text = '')
       short_text = post_text.split(" ").first(50).join(" ")
       short_text + "..."
  end

end
```

Kod 3.1. Pomoćne metode aplikacije

Jedna od razlika u usporedbi s Phytonom i Perlom je ta da Ruby drži sve instance varijabli kao privatne u određenoj klasi i čini ih dostupnima jedino korištenjem pristupnih metoda (engl. *attr_writer*, *attr_reader* i sl.). Za razliku od *get* i *set* metoda karakterističnih za programske jezike C++ i Java, pristupne metode u Rubyju mogu biti kreirane jednom linijom koda uz upotrebu principa metaprogramiranja. Ipak, tradicionalni pristup kao onaj u Javi i C++-u također je dopušten, ali nepotreban.

3.2. Rails razvojno okruženje

Web application framework ili programsko okruženje za izradu web aplikacija je programsko okruženje dizajnirano kako bi poboljšalo razvoj web aplikacija, uključujući web usluge, web resurse i web API-je. Takva okruženja trebala bi pojednostavniti i umanjiti broj aktivnosti nužnih prilikom web razvoja. Npr., brojna takva okruženja pružaju gotove biblioteke, pristup bazama podataka, upravljanje sesijama i promiču višestruku upotrebu jedinstvenog koda. Iako se većinom koriste za izradu dinamičkih web stranica, primjenjivi su i za izradu onih statičkih, kao što je navedeno u [3].

Ruby on Rails je okruženje za web razvoj napisano u programskom jeziku Ruby. To je okruženje koje se temelji na MVC principu, a sadrži zadane baze podataka, web usluge i web stranice. Potiče uporabu web standarda kao što su JSON ili XML za prijenos podataka i HTML, CSS i JavaScript za prikaz korisničkog sučelja. Kao dodatak MVC-u, Rails stavlja naglasak na upotrebu dobro poznatih inženjerskih uzoraka programiranja kao što su konvencija ispred konfiguracije ili izbjegavanje ponavljanja koda.

Prema [6], začetnik Railsa David Heinemeier Hansson izdvojio je Ruby on Rails kao dio svog rada na alatu za projektni menadžment pod nazivom Basecamp u tvrtki koja se bavi izradom web aplikacija. Hansson je najprije objavio Rails kao program otvorenog koda u srpnju 2004. godine, ali prava na doprinos nije podijelio sve do veljače 2005. godine. Od izlaska 2004., Ruby on Rails brzo je postao jedan od najmoćnijih i najpopularnijih alata za izradu dinamičkih web aplikacija. Rails koriste, ili su koristile, popularne i različite firme kao što su Airbnb, GitHub, Shopify, Twitter i brojne druge. Jedan od razloga popularnosti je i to što je potpuno besplatan i dostupan pod MIT licencom što znači da njegovo preuzimanje i upotreba ništa ne koštaju.

Također, Rails se brzo prilagođava novim načinima razvoja za web i razvoja programskih okruženja. Primjerice, Rails je jedno od prvih okruženja koje u potpunosti koristi REST arhitekturu za izradu web aplikacija. Osim toga, Rails kreatori ne oklijevaju prihvatiti i ugraditi nove principe i ideje čak i kada dolaze od konkurentskog Ruby razvojnog okruženja za web kao što je Merb.

Konačno, Rails ima veliku i raznoliku, ali i prilično entuzijastičnu zajednicu svojih korisnika. Rezultat toga su doprinosi brojnih programera Rails izvornom kodu, dobro posjećene konferencije i velik broj dodataka (engl. *gem*) kao i mnogi informativni blogovi i diskusije na forumima, osobito kad je riječ o rukovanju brojnim, ali neizbježnim, pogreškama aplikacije.

Verzija Railsa korištena za izradu praktičnog dijela ovog rada je Rails 4, točnije Rails 4.2.1. Rails je podijeljen u nekoliko različitih paketa nazvanih ActiveRecord, ActiveResource, Active Pack, Active Support i ActionMailer. Osim standardnih paketa, programeri mogu koristiti dodatke unutar Rails okruženja tzv. Ruby *gemove*. Svaki od navedenih paketa, obavlja određenu skupinu zadataka.

3.2.1. Struktura direktorija novokreirane Rails aplikacije

Prilikom instalacije i pokretanja Railsa te kreiranja prve aplikacije moguće je vidjeti od čega se Rails zapravo sastoji te kako izgleda njegov datotečni sustav na koji će se kasnije nadograđivati kod aplikacije koju je potrebno izraditi**.** Osnovno stablo direktorija objašnjeno je u tablici 3.1. izrađenoj prema [3]. Nakon odabiranja željenog direktorija moguće je kreirati novu Rails aplikaciju. Osnovna Rails aplikacija kreira se prilično jednostavno iz komandne linije naredbom:

```
$ rails new ime_aplikacije
```

Rails dolazi s postavljenim lokalnim poslužiteljem na kojemu je moguće testirati aplikaciju, rute i sl. Za pokretanje Rails poslužitelja u komandnu liniju upisuje se sljedeća naredba:

```
$ rails server
```

Novokreiranoj Rails aplikaciji moguće je zatim pristupiti upisivanjem adrese *http://localhost:3000.*

Tab. 3.1. Rails stablo direktorija

Datoteka / Direktorij	**Svrha**
`app/`	Glavni kod aplikacije uključujući modele, poglede i upravitelje.
`app/assets`	Aplikacijske karakteristike kao što su CSS, JavaScript i slike.
`bin/`	Binarne izvedbene datoteke.
`config/`	Aplikacijska konfiguracija.
`db/`	Datoteke za rad s bazom podataka.
`doc/`	Dokumentacija aplikacije.
`lib/`	Moduli biblioteka.

`lib/assets`	Karakteristike biblioteka kao što su CSS, JavaScript i slike.
`log/`	Aplikacijske *log* datoteke.
`public/`	Podaci dostupni javno, primjerice putem web preglednika. Neki od njih su stranice s pogreškama.
`bin/rails`	Program za generiranje koda, otvaranje konzolnih sesija i pokretanje lokalnog poslužitelja.
`test/`	Aplikacijski testovi.
`tmp/`	Privremene datoteke.
`vendor/`	Kod koji dolazi od treće strane kao što su dodaci ili *gemovi*.
`vendor/assets`	Karakteristike koda treće strane kao što su CSS, JavaScript i slike.
`README.rdoc`	Kratak opis aplikacije.
`Rakefile`	Korisni zadaci dostupni putem *rake* naredbe.
`Gemfile`	*Gem* zahtjevi aplikacije.
`Gemfile.lock`	Popis *gemova* korištenih ne bi li se osiguralo da sve kopije aplikacije koriste iste verzije *gemova*.
`config.ru`	Konfiguracijska datoteka za Rack posrednički program.
`.gitignore`	Uzorci za datoteke koje bi git trebao zanemariti.

3.3. Rails upravitelj rutama i rad s podacima

Za izradu kvalitetnih Rails aplikacija, razvojni programer mora dobro poznavati način na koji radi Rails upravitelj rutama te paket pod nazivom ActiveRecord za upravljanje podacima i komunikaciju s bazom podataka. Rad s navedenim Rails alatima treba savladati već na samom početku.

3.3.1. Rails upravitelj rutama

Rails upravitelj rutama (engl. *Rails router*) značajan je dio Rails okruženja kojemu je zadaća prepoznati dobiveni URL i proslijediti ga nekoj od metoda upravitelja (engl. *controller*) iz MVC modela. Rails upravitelj rutama može samostalno, iz MVC upravitelja , generirati puteve i URL-ove bez potrebe za kodiranjem istih u pogledima. Kada zaprimi HTTP zahtjev, preglednik mora znati koja metoda MVC upravitelja treba biti pozvana. Primjerice, treba li biti pozvana metoda za kreiranje novog korisnika ili prikazan novi kontakt obrazac.

Rails upravitelj rutama može se opisati kao usluga koja pronalazi podudarnost između URL-ova i metoda iz upravitelja MVC modela. Rails upravitelj rutama provjerava vrstu HTTP zahtjeva (*GET, POST, PUT, DELETE*) te zahtijevani URL i povezuje ga s pripadajućom funkcijom iz MVC upravitelja. Ukoliko Rails upravitelj ne uspije pronaći podudarnost između traženog URL-a i neke funkcije, aplikacija će pokazivati grešku. Nadalje, kada dođe do HTTP zahtjeva, Rails uzima parametre koji dolaze s tim zahtjevom i čini ih dostupnima u posebnoj varijabli nazvanoj „params" koju je kasnije moguće korstiti u MVC upraviteljima. To je osobito korisno kod podnošenja obrazaca, primjerice onih za prijavu i registraciju, jer je podatke iz obrasca tada moguće koristiti za stvaranje ili izmjenu objekata. Datoteka u kojoj se mijenjaju i postavljaju rute unutar Rails aplikacije nalazi se u direktoriju *config*. Za ispis svih trenutnih ruta koje postoje u aplikaciji, koristi se naredba:

```
$ rake routes
```

koja se upisuje u komandnu liniju. Kao rezultat dobiva se popis svih ruta dostupnih u aplikaciji. Najvažnija i najjednostavnija ruta svake aplikacije je osnovni ili korijenski URL na kojeg će korisnici biti usmjereni kada u preglednik upišu domenu stranice ili *http://localhost:3000* u slučaju rada na lokalnom poslužitelju. Sve što je potrebno za postavljanje korijenske rute jest dodati liniju koda u *config/routes.rb* i naznačiti koju metodu unutar upravitelja Rails mora pozvati. U kodu 3.2. nalazi se dio *routes.rb* datoteke aplikacije AlumniETFOS. U drugoj liniji koda, *static_pages* označava da se radi o upravitelju pod nazivom *static_pages*, a željena akcija je *home*. Točnije, *home* je samo metoda koja se nalazi u *static_pages* upravitelju.

```
Rails.application.routes.draw do


  root to:             'static_pages#home'

  get  'o-udruzi'  =>  'static_pages#about',    as: :about
  get  'statut'    =>  'static_pages#statut',   as: :statut
  get  'kontakt'   =>  'contacts#new',  as: :contact
  get  'vijesti'   =>  'static_pages#posts',    as: :news
  get  'vijest/:id'=>  'static pages#single post'

  .
  .
  .
```

Kod 3.2. Dio sadržaja *routes.rb* datoteke

3.3.2. Rute koje slijede REST

U drugom poglavlju koje se bavi web aplikacijama opisan je REST način rada prema kojemu postoje četiri, odnosno sedam, osnovnih funkcija za rad s resursima(modelima ili objektima) kao što su korisnici, objave i dr. Na primjeru objave kao resursa navedene funkcije su:

1) prikaz svih objava (zahtjev *GET*, metoda *index*)
2) prikaz pojedinačne objave (zahtjev *GET*, metoda *show*)
3) prikaz stranice koja omogućuje kreiranje nove objave (zahtjev *GET*, metoda *new*)
4) podnošenje ispunjenih podataka potrebnih za stvaranje nove objave i njihovo slanje poslužitelju (zahtjev *POST*, metoda *create*)
5) prikaz stranice koja omogućuje uređivanje postojeće objave (zahtjev *GET*, metoda *edit*)
6) slanje novih, uređenih podataka poslužitelju i njihova pohrana (zahtjev *POST*, metoda *update*)
7) funkcija za brisanje pojedinačnih objava (zahtjev *DELETE*, metoda *destroy*).

Index, *show*, *new*, *create*, *edit*, *update* i *destoy* ključne su riječi koje odgovaraju standardiziranim nazivima metoda unutar Rails upravitelja, a isto je navedeno u [7]. Svaka od tih akcija odgovara određenoj ruti te se rute definirane na taj način nazivaju *RESTful* rute. Jedan od način definiranja takvih ruta koje odgovaraju pojedinim akcijama unutar Railsa prikazan je u kodu 3.3. te je za svaku akciju definirana ruta kojom se akciji pristupa iz preglednika.

```
Rails.application.routes.draw do
  .
  .
  .
  get  'posts'          =>  'posts#index'
  get  'posts/:id' =>  'posts#show'
  get  'posts/new'      =>  'posts#new'
  post 'posts'          =>  'posts#create'
  get  'posts/:id/edit' =>  'posts#edit'
  put  'posts/:id'      =>  'posts#update'
  delete 'posts/:id'    =>  'posts#destroy'
  .
  .
  .
```

Kod 3.3. Definiranje *RESTful* ruta – prvi način

Očito je da neke od prikazanih ruta imaju identičan URL, ali koriste drugačiji HTTP zahtjev tako da Rails može pozvati drugačiju akciju upravitelja. Nadalje, polje „id” sadrži dvotočku što znači

da sve što se nalazi u tom dijelu URL-a treba pohraniti kao identifikator. Tako je omogućeno podnošenje *GET* zahtjeva za prvu ili petu objavu koristeći istu rutu, a drugačiji identifikator.

Drugi način za definiranje *RESTful* ruta u Rails okruženju mnogo je brži. Budući da je Rails programsko okruženje zapravo utemeljeno na REST arhitekturi, zamišljeno je da takav pristup koriste i razvojni programeri kreirajući vlastite Rails aplikacije. Stoga Rails može automatski prepoznati kada je potrebno koristiti svih sedam navedenih putanja. Naime, postoji pomoćna metoda koja će u jednoj liniji koda učiniti sve što je napisano u sedam linija prethodnog koda. Kod 3.4. zamjenjuje sedam linija koda iz prikaza koda 3.3..

```
Rails.application.routes.draw do
  .
  .
  .
  resources :posts
  .
  .
  .
```

Kod 3.4. Definiranje *RESTful* ruta – drugi način

3.3.3. Pomoćne metode za rad s rutama

Upisivanjem naredbe *rake routes* u komandnu liniju, dobiva se prikaz svih ruta koje su dostupne u aplikaciji. Prilog 1 sadrži tablicu sa svim rutama aplikacije ALumniETFOS. U srednjem stupcu prikaza nalazi se URL dok je u krajnjem desnom stupcu prikazana akcija MVC upravitelja. Prikaz je sličan sadržaju koji se nalazi u *config/routes.rb* datoteci. U krajnjem lijevom stupcu nalaze se tzv. pomoćni nazivi pojedine rute i vrsta HTTP zahtjeva. Ti se nazivi koriste kada je u kodu potrebno pristupiti određenoj ruti. Primjerice, kada stranica ili aplikacija sadrži izbornik s poveznicama na neku drugu stranicu aplikacije. U tom slučaju nije dobro definirati statične URL-ove kao vrijednost *href* atributa „<a>“ oznake, već se koriste definirani pomoćni nazivi. Rails sadrži pomoćnu metodu *link_to* koja služi za generiranje „<a>“ oznaka s poveznicom na određeni dio aplikacije. Pomoćne metode mogu se koristiti unutar HTML datoteka s ekstenzijom .html.erb, a jedan takav primjer korištenja vidljiv je u kodu 2.1..

Ukoliko programer aplikacije ne odredi pomoćna imena, Rails ih generira automatski, a ona odgovaraju uzorku „ime_rute_path”. Dakle, nazivu rute dodan je sufiks “_path”. Svaka ruta koja zahtijeva identifikator ili neki drugi parametar zahtijevat će da ti parametri budu predani kao parametri pomoćnog imena, odnosno pomoćne metode za dobivanje rute. Ukoliko programer želi koristiti metodu *resources :post* iz koda 3.4. ne bi li skratio vrijeme kodiranja, ali ne želi

koristiti sve metode unutar željenog upravitelja, moguće je koristiti ključne riječi *:only* i *:except* kako bi se filtriralo koje će rute postojati u sustavu, a koje ne. Prema primjeru 3.5. postojat će putanje samo do metode za prikaz svih objava i metode za prikaz jedne objave. Što se tiče korisničkih putanja (resurs *Users*), definirane će biti sve rute, osim one koja prikazuje sve korisnike.

```
Rails.application.routes.draw do
  .
  .
  .
  resources :posts, only => [:index, show]
  .
  .
  .
  Resources :users, except => [:index]
```

Kod 3.5. Definiranje *RESTful* ruta – drugi način

Naravno, treba imati na umu da, iako je REST način definiranja ruta preporučljiv, nije i obavezan pa je sasvim prihvatljivo dodati vlastita imena za rute koje odgovaraju određenoj metodi upravitelja u slučajevima kada je to potrebno.

3.3.4. Rad s podacima

Podaci su osnova svake web aplikacije, od jednostavnog bloga do masivnih aplikacija kao što je Facebook. Iz tog je razloga važno prije početka rada na aplikaciji, dizajnirati odgovarajući podatkovni model. Budući da su podaci tako važni za rad svake aplikacije, u Railsu također postoji način koji olakšava upravljanje podacima. ActiveRecord naziv je Rails sučelja između baze podataka i aplikacije kako je navedeno u [8]. ActiveRecord omogućava kreiranje podatkovnih modela za korisnike, objave, komentare i dr., i to na vrlo jednostavan način sličan razgovornom jeziku. ActiveRecord uzima podatke spremljene u bazi koje je moguće dohvatiti ili izmijeniti pisanjem SQL naredbi te omogućuje postupanje s njima kao s običnim Ruby objektima. Ukoliko je potrebno dohvatiti niz koji sadrži sve korisnike, umjesto pisanja koda za povezivanje s bazom podataka te zatim pisanja SQL naredbe „SELECT * FROM posts" i pretvaranja rezultata u niz, dovoljno je upisati *Post.all* i ActiveRecord će vratiti niz popunjen objektima klase *Post*. Primjer korištenja nalazi se u *index* metodi koda 2.7.. Također, nije važno koja vrsta baze podataka je korištena za izradu aplikacije dokle god se u *config/database.yml* datoteci nalaze valjane postavke. Zadana baza korištena prilikom razvoja u Railsu je sqlite3. Da bi razvojni programer u Railsu imao pristup podacima iz baze prije pisanja samog koda, može koristiti Rails konzolu koja se pokreće upisivanjem naredbe:

```
$ rails console
```

Zatim se otvara sučelje za rad s bazom. Upisivanjem određenih naredbi moguće je ispisati pojedine retke i stupce iz baze ili u bazu dodati nove korisnike. Npr., aplikacija AlumniETFOS ne pruža mogućnost registracije administratora, već su administratorski računi uneseni ručno preko konzole.

3.3.5. Migracije i asocijacije

Kada se radi s bazama podataka u Rails okruženju, nemoguće je izbjeći pojam migracije. Migracija je zapravo skripta koja govori Railsu na koji način postaviti ili izmijeniti bazu podataka. To je dio ActiveRecorda koji omogućuje izbjegavanje pisanja SQL koda za kreiranje tablica. Migracije se pokreću naredbom:

```
$ rake db:migrate
```

koja izvršava sve ranije neizvršene migracije. Pokretanjem te naredbe Rails generira odgovarajući SQL kod za kreiranje tablice u bazi. Takav je način vrlo koristan jer omogućuje postavljanje baze korištenjem puno pristupačnijeg Ruby, nego SQL koda, ali ne samo to, nego i olakšava postavljanje baze u produkcijskom načinu rada. Štoviše, prema [8], u slučaju pogreške uvijek je moguće izbrisati bazu i ponovno pokrenuti migracije. Ukoliko je, pak, zabunom izvršena neka migracija, moguće ju je poništiti naredbom *rake db:rollback* jer je jedna od glavnih karakteristika migracija reverzibilnost. Za svaku metodu napisanu unutar migracije, dobro je definirati suprotnu metodu. Suprotno od kreiranja tablice jest brisanje tablice i slično. Neke od suprotnih funkcija mogu biti izvršene automatski, a neke je potrebno zasebno definirati.

Kako je baza podataka sqlite3 zapravo relacijska baza podataka u kojoj su dvije tablice povezane jedna s drugom korištenjem primarnog i stranog ključa, ActiveRecord omogućuje ostvariti takve veze uz svega jednu liniju koda. Ako u sustavu postoje korisnici (npr. administratori) koji mogu objavljivati vijesti (objave), to će značiti da jedan korisnik može imati više objava te da svaka objava pripada nekom od korisnika. Da bi Rails prepoznao željenu funkcionalnost potrebno je u modelima upisati pomoćne metode „has_many" ili „belongs_to". Još jedna značajna asocijacija jest „has_and_belongs_to many" te obuhvaća obje prethodne asocijacije.

3.4. Ruby Gems

Gem je programski dodatak ili biblioteka koja ima određenu funkcionalnost, a cilj joj je ubrzati i olakšati razvoj aplikacija. *Gemovi* se instaliraju u svrhu ispunjavanja nekog određenog zahtjeva kako bi se izbjegla potreba za pisanjem vlastitog koda otpočetka, pogotovo ako je riječ o često korištenoj funkcionalnosti koja se ponavlja iz aplikacije u aplikaciju. Svaki *gem* sastoji se od imena, verzije i platforme. Primjerice *gem* pod nazivom *uglifier* ima verziju 1.3.0, a platforma je Ruby što znači da će takav *gem* raditi na bilo kojoj platformi na kojoj postoji instaliran Ruby. Platforme su bazirane na CPU arhitekturi, tipu operacijskog sustava, a ponekad i verziji operacijskog sustava. Prema [9], unutar svakog *gema* nalaze se sljedeće komponente:

- kod (uključujući testove i korisne dodatke)
- dokumentacija i
- gemspec datoteka.

Svi *gemovi* slijede isti standard organizacije direktorija i poddirektorija, a njihov popis i objašnjenje nalaze se u tablici 3.2.

Tab. 3.2. *Gem* struktura

Datoteka/ Direktorij	**Sadržaj**
`bin`	Izvršna datoteka koja će biti učitana u korisničku putanju nakon što je *gem* instaliran.
`lib`	Izvorni kod *gema*.
`test ili spec`	Testovi ovisno o razvojnom okruženju koje programer koristi.
`Rakefile`	Datoteka koju *rake* program koristi za automatizaciju testova, generiranje koda i druge zadatke.
`README`	Sadrži dokumentaciju programa.
`.gemspec`	Informacije o *gemu*, informacije o testovima, platforma, verzija, ime autora i dr.

Postoje dva načina korištenja *gemova*. Neki od njih su samostalni Ruby programi koji se izvode s određenim ciljem. Primjer takovog *gema* je cjelokupno Rails okruženje. Naredbom *rails new ime_projekta* iz komandne linije generiran je novi Rails projekt. S druge strane, tu su *gemovi* koji se koriste samo unutar vlastitih projekata. Samostalno nisu od velike koristi, ali u sklopu cjelokupnog koda imaju značajnu svrhu.

Gemovi se mogu koristiti za stvaranje Ruby programa izvan Rails okruženja, ali i unutar Railsa. Neki od najpoznatijih i najčešće preuzetih *gemova* su *gemovi* za autentikaciju korisnika tj.

kreiranje sustava za prijavu i registraciju, zatim *gemovi* koji olakšavaju pisanje testova ili oni za obavljanje zadataka koji se izvršavaju u pozadini. Nakon kreiranja Rails aplikacije sljedeći je korak korištenje *bundlera* koji služi za instaliranje i uključivanje svih potrebnih *gemova* koje aplikacija zahtijeva. *Bundler* se pokreće automatski naredbom iz komandne linije:

```
$ bundle install
```

Željeni *gemovi* definirani su u datoteci *Gemfile*, čiji se sadržaj nalazi u kodu 3.6., a preuzeti su sa stranice RubyGems. RubyGems je naziv programa, odnosno web aplikacije, koja omogućuje lako preuzimanje, instalaciju i upotrebu Ruby programskih paketa na vlastitom sustavu. Na stranici poput te korisnici mogu postavljati vlastite *gemove* ili preuzeti *gemove* koje su izradili drugi programeri. *Gemovi* se mogu koristiti ne bi li izmijenili funkcionalnost Ruby aplikacija. Ovisno o onome što se želi ostvariti, potrebno je potražiti odgovarajući *gem*. Iako na RubyGems stranicama postoji velik broj *gemova*, neki od njih su zastarjeli, ostali nisu dovoljno dobro dokumentirani dok neki imaju preveliki broj pogrešaka. Iz tog razloga je prije instaliranja i odabiranja potrebno detaljno istražiti statistiku vezanu za pojedini *gem*, kada je posljednji put ažuriran, koliko ga je korisnika preuzelo i slične informacije. U tom istraživanju od pomoći je stranica RubyToolbox, ali i informacije koje se mogu pronaći na Git stranicama projekta.

```
source 'https://rubygems.org'

ruby  '2.1.2'

gem 'rails',                   '4.2.2'
gem 'sass-rails',              '~> 5.0'

gem 'devise'
gem 'omniauth-facebook'

gem 'figaro'

gem 'bootstrap-sass',  '3.2.0.0'

gem 'will_paginate',    '3.0.7'
gem 'bootstrap-will_paginate', '0.0.10'

gem 'paperclip', '~> 4.2.0'

gem 'mail_form', '~> 1.5.1'

gem 'uglifier',                '>= 1.3.0'
gem 'coffee-rails',    '~> 4.1.0'

gem 'jquery-rails'
gem 'turbolinks'
gem 'jbuilder',                '~> 2.0'
gem 'sdoc',                    '~> 0.4.0', group: :doc
```

```
group :development, :test do
  gem 'annotate'
  gem 'sqlite3', '1.3.11'
  gem 'byebug',  '9.0.0', platform: :mri
  gem 'spring'
end

#Heroku deployment
group :production do
  gem 'pg';
  gem 'rails_12factor'
end
```

Kod 3.6. *Gemfile* aplikacije AlumniETFOS

Ukoliko nije određena verzija *gema* koju se želi instairati, *bundler* će automatski instalirati najnoviju. To je slučaj kada je upisan samo naziv *gema* bez dodatnih parametara. Međutim, postoje dva uobičajena načina na koji se može instalirati točno određena verzija *gema*. Primjer koji slijedi instalirat će *gem* pod nazivom *uglifier* i to njegovu posljednju verziju sve dok nije veća od 1.3.0. Primjerice:

```
gem 'uglifier', '>= 1.3.0'
```

Sljedeći primjer instalirat će gem pod nazivom *bcrypt* dokle god je noviji od verzije 3.0.0, ali nije noviji od verzije 3.1. Ukratko „>=" uvijek instalira najnoviji *gem* dok „~>" instalira najnoviji *gem* nižeg stupnja (sa 3.0.0 nad 3.0.5), ali ne i glavnog stupnja (sa 3.0 na 3.1).

```
gem 'bcrypt', '~> 3.0.0'
```

Kontrola verzija *gemova* jako je važna jer je praksa pokazala da čak i razlike u verzijama istog stupnja mogu dovesti do pogrešaka u aplikaciji. *Gemovi* se unutar *Gemfilea* mogu podijeliti u skupine pa će neki od njih biti instalirani samo u razvojnom ili samo u testnom okruženju. U praktičnom dijelu rada, izradi aplikacije, korišteno je nekoliko *gemova* od kojih je najvažniji Devise sustav za provjeru vjerodostojnosti korisnika. Osim Devisea, upotrijebljeni su i neki manje složeni gemovi kao što je Paperclip koji korisnicima omogućuje postavljanje slike u sustav ili Will-paginate za lakši pregled velikog broja podataka.

4. PROVJERA IDENTITETA KORISNIKA

Za provjeru identiteta korisnika aplikacije AlumniETFOS korišten je vrlo popularan Ruby *gem* pod nazivom Devise. Devise pruža jednostavan i modularan pristup kreiranju sustava za autentikaciju, a ujedno predstavlja sigurno rješenje koje ubrzava razvoj aplikacije. Uz Devise je jednostavno ostvariti različite korisničke uloge u sustavu, primjerice, osnovnog korisnika i administratora. Osim putem Devisea, korisnike je moguće autenticirati preko postojećih korisničkih računa za neke druge sustave, npr., preko korisničkih računa društvenih mreža. Za taj tip autentikacije implementiran je OmniAuth sustav.

4.1. Devise

Devise je fleksibilno rješenje za identifikaciju korisnika u Railsu. Devise je zapravo *gem* koji donosi nove funkcionalnosti u Rails aplikacije i olakšava kreiranje sustava prijave i registracije korisnika, validacije obrazaca, generiranja pogleda i slično. Devise je temeljen na Racku i u potpunosti je MVC orijentirano Rails rješenje. Nadalje, Devise omogućuje višestruke modele prijavljene istovremeno. Još jedna značajna karakteristika Devisea je to da je zasnovan na modularnom konceptu. Prema tome, moguće je upotrijebiti samo one module koji su potrebni i odgovaraju zahtjevima aplikacije. Devise se, kao što je navedeno u [10], sastoji od ukupno deset modula:

1) *Database Authenticatable* - služi za kriptiranje i pohranu lozinke u bazu podataka ne bi li se potvrdila autentičnost korisnika dok se prijavljuje u sustav. Identifikacija se vrši ili putem *POST* zahtjeva ili putem osnovne HTTP autentikacije.
2) *Omniauthable* - dodaje podršku za OmniAuth što je još jedan sustav za identifikaciju koji ima mogućnosti prijave korisnika s korisničkim računima društvenih i drugih mreža.
3) *Confirmable* - slanje emaila s uputama za potvrdu i verifikacija potvrde korisničkog računa.
4) *Recoverable* - postavljanje nove lozinke i slanje uputa za promjenu iste.
5) *Registerable*- rukovanje procesom registracije korisnika, također im omogućavajući izmjene računa ili uništenje računa.
6) *Remmemberable*- upravljanje generiranjem i uklanjanjem tokena za pamćenje korisnika iz spremljenog kolačića (engl. *cookie*).
7) *Trackable* - praćenje koliko je drugo vremena korisnik prijavljen u sustav te IP adrese s koje s prijavljuje.
8) *Timeoutable* - završetak sesija koje neko određeno vrijeme nisu bile aktivne.

9) *Validatable* - pruža mogućnosti potvrde emaila i lozinke. Opcionalno i može se izmijeniti prema vlastitim potrebama. Programer može dodati vlastite validacije.
10) *Lockable* - zaključavanje računa nakon određenog broja neuspjelih pokušaja prijave u sustav. Račun se može otključati nakon zadanog vremenskog perioda i to putem emaila.

4.1.1. Instalacija Devisea

Već je u ranijim poglavljima spomenuto kako su svi gemovi potrebni za rad aplikacije definirani u *Gemfile* datoteci. Za instalaciju Devisea u datoteku je potrebno upisati sljedeću liniju koda:

```
gem 'devise'
```

Za instalaciju je potrebno koristiti naredbu *bundle* i instalirati sve *gemove*. Zatim, nakon instalacije Devisea potrebno je pokrenuti generator iz komandne linije. Generator će instalirati pokretač koji opisuje sve konfiguracijske opcije koje ima Devise.

```
$ rails generate devise:install
```

Nakon instaliranja generatora i pregleda konfiguracijskih opcija, Devise se može dodati u bilo koji od postojećih modela koristeći generator.

```
$ rails generate devise ime_modela
```

Ime modela je potrebno zamijeniti imenom klase korištene za korisnike aplikacije (najčešće je to *User*, ali može biti i *Admin*). Ova akcija kreira model i postavlja unaprijed zadane Devise module. Generator također uvodi promjene u *config/routes.rb* datoteku koja sadrži sve putanje koje postoje unutar aplikacije. Sada ta datoteka ukazuje i na Devise upravitelj. Sljedeći korak je provjeriti model za slučaj da su mu potrebne neke dodatne konfiguracijske mogućnosti kao npr., mogućnost potvrde i zaključavanja. U slučaju dodavanja opcije potrebno je ispitati migracijsku datoteku i ukloniti komentare s pojedinih dijelova koda. Primjerice, ako se postavi da model ima mogućnost potvrde potrebno je ukloniti komentar sa željenog dijela koda i pozvati naredbu *rake db:migrate*.

4.1.2. Devise rute

Devise pruža mogućnost unošenja različitih korisničkih modela čija je autentikacija potrebna prije korištenja određenih dijelova aplikacije, primjerice, korisnika, administratora i dr. Velika prednost koju Devise pruža je generiranje upravitelja, a opcionalno i pogleda, za svaki Devise

model. Upravitelji su podijeljeni u kategorije ovisno o funkcijama koje obavljaju te tako postoje upravitelji vezani za sesije, registracije, ali i upravitelji koji omogućuju pregled svih korisnika ili izmjene korisničkog računa. Dakako, broj upravitelja ovisi o omogućenim Devise modulima. Za pristup Devise funkcijama aplikacije korisnik mora pristupiti određenom URL-u ili ruti. Rute koje se pojavljuju nakon instalacije Devise gema pozivaju metode iz pojedinih upravitelja. Metode, kao i rute, moguće je prilagoditi vlastitim potrebama. Zadane rute koje dolaze sa svakim novokreiranim Devise modelom navedene su u nastavku.

Rute vezane za sesije modela *User*:

- *GET /user/sign_in*
- *POST /user/sign_in*
- *DELETE /user/sign_out*

Rute vezane za registraciju i izmjene računa modela *User*:

- *GET /user/sign_up*
- *POST /user*
- *GET /user/edit*
- *PUT /user*
- *DELETE /user*

Na ovaj se način automatski dobivaju standardizirani putevi do određenih dijelova aplikacije. Ukoliko je model nazvan *User* stranica za prijavu korisnika imat će rutu *users/sign_in*, stranica za registraciju *users/sign_up*, stranica za prikaz svih korisnika *users* i tako dalje. Ukoliko je naziv modela *Admin* rute će biti generirane analogno prethodnom primjeru. Iako su rute unaprijed definirane, moguće ih je prilagoditi. Jedan od primjera gdje je to potrebno jest određivanje stranice na koju će korisnici biti preusmjereni nakon prijave. Unaprijed je određeno da je to *root*, korijenska ruta, ali ponekad je praktičnije preusmjeriti korisnika na njegov profil. Rute koje će Devise generirati također ovise o omogućenim Devise modulima. Ako je u aplikaciji omogućeno dobiti novu lozinku u slučaju zaboravljanja, postojat će ruta do te funkcije, a u protivnom neće. U Deviseu postoje i kratice, tj. pomoćne metode koje vode do određenih ruta, a to su: *new_user_session, user_session, destroy_user_session, new_user_registration* i druge. Navedene kratice moguće je koristiti za stvaranje poveznica na te rute u HTML dokumentima aplikacije.

4.1.3. Pomoćne funkcije i osnovna upotreba

Devise automatski stvara neke od pomoćnih funkcija koje se mogu koristiti unutar upravitelja i pogleda. Za postavljanje upravitelja s korisničkom identifikacijom potrebno je pozvati naredbu, pod pretpostavkom da je model nazvan *User*:

```
before_action :authenticate_user!
```

Za provjeru je li korisnik prijavljen u sustav koristi se sljedeća pomoćna funkcija:

```
user_signed_in?
```

Za dobivanje informacija o trenutno prijavljenom korisniku dostupna je pomoćna funkcija:

```
current_user
```

Postoji i pomoćna varijabla za pristup sesiji:

```
user_session
```

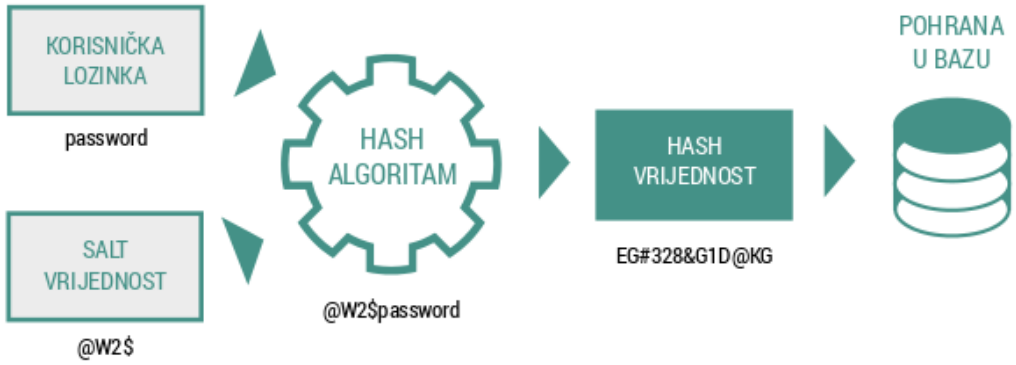
Nakon prijave korisnika u sustav, potvrde računa ili promjene lozinke, Devise traži određenu korijensku putanju na koju će preusmjeriti istog. Primjerice, korištenjem resursa *:user* bit će korištena putanja *user_root_path* ukoliko postoji. Ukoliko ta putanja ne postoji korišten je unaprijed zadani *root_path*. Što znači, ako je potrebno postaviti korijensku putanju unutar datoteke s putanjama, treba definirati:

```
root to: 'home#index'
```

Također, treba imati na umu da ako korisnik nije nazvan *User*, nego primjerice *Admin*, tada će sve prethodno opisane naredbe poprimiti sljedeće oblike: *admin_signed_in?*, *admin_sassion* i tako dalje. Devise metoda u modelima također prihvaća neke opcije za postavljanje modela. Npr., moguće je odabrati trošak algoritma za kriptiranje lozinke te definirati brojne druge opcije.

Uobičajena praksa, koju slijedi i Devise, je da se korisnička lozinka u bazu podataka ne pohranjuje u svom osnovnom obliku jer bi to uvelike ugrozilo sigurnost sustava i sigurnost korisnika aplikacije. Umjesto toga, prije pohrane, na lozinku se primjenjuje neka od brojnih vrsta *hash* funkcija koje služe za enkripciju podataka. Takve funkcije određeni ulazni podatak pretvaraju u izlazni podatak konstantne duljine. *Hash* je jednosmjerna funkcija što znači da ne postoji nikakva funkcija koja bi dobiveni izlazni podatak pretvorila natrag u ulazni podatak. Kriptografska *hash* funkcija zapravo je matematički algoritam koji je nemoguće invertirati.

Prilikom primjene *hash* funkcije na lozinku, lozinka se uglavnom ulančava s nekom nasumičnom *salt* vrijednosti, a tek se onda na njih primjenjuje kriptografska funkcija te se novonastali izlazni rezultat pohranjuje u bazu podataka kao što je prikazano na slici 4.1..



Sl. 4.1. Način rada *hash* funkcije

4.2. OmniAuth

OmniAuth je biblioteka koja omogućuje standardiziranu autentikaciju korisnika za različite pružatelje usluga, a namijenjena je korištenju u web i drugim aplikacijama. Glavne prednosti OmniAutha su fleksibilnost i velik broj mogućnosti primjene te jednostavno korištenje. Svaki razvojni programer može odabrati vlastiti način na koji će OmniAuth autenticirati korisnike putem različitih sustava. Naravno, postoje i široko primjenjivi, najčešće korišteni načini primjene za gotovo svaki sustav. Za korištenje OmniAutha u vlastitoj aplikaciji moguće je odabrati jedan ili više načina autentikacije, odnosno strategiju primjene. Načine ili strategije potrebno je instalirati kao Ruby *gemove*. Strategija odgovara korisničkom računu neke društvene mreže ili nekog drugog sustava što je detaljnije opisano u [11].

OmniAuth može se koristiti i integrirati zasebno, ali i s vlastitim sustavom za prijavu i registraciju. Budući da aplikacija AlumniETFOS koristi Devise sustav, OmniAuth je korišten u sklopu Devisea što znači da modul pod nazivom *Omniauthable* mora biti omogućen prilikom instalacije. OmniAuth ne zahtijeva mnogo dodatnih izmjena od strane programera. Njegova je namjena biti crna kutija kojoj će korisnici pristupiti kada je potrebna autentikacija i dobiti odgovarajuće izlazne informacije kao rezultat. OmniAuth je dizajniran tako da se ne povezuje automatski s modelom korisnika, niti stvara pretpostavke o tome što činiti s prikupljenim podacima. Definiranje tih aktivnosti prepušteno je programeru što čini OmniAuth prilagodljivim potrebama aplikacije. Za upotrebu OmniAutha potrebno je preusmjeriti korisnike na */auth/:provider*, gdje je *:provider* ime strategije, primjerice Facebook ili Twitter. Nakon tog

koraka, OmniAuth će obaviti potrebne zadatke i provesti korisnika kroz potrebne korake za autentikaciju s odabranom strategijom.

4.2.1. OmniAuth i Facebook

Iako se OmniAuth može koristiti za autentikaciju putem brojnih korisničkih računa, ne samo s društvenih mreža, već i drugih usluga, za potrebe aplikacije AlumniETFOS omogućena je prijava i registracija putem Facbooka. Da bi se to ostvarilo, nije dovoljno samo postaviti OmniAuth. Potrebno je imati vlastiti Facebook račun i preko njega registrirati aplikaciju na stranici „Facebook for Developers“. Registracija aplikacije vrlo je jednostavna zahvaljujući intuitivnom sučelju. Zatim na listu vlastitih aplikacija treba dodati i imenovati željenu aplikaciju kojoj će Facebook dodijeliti aplikacijski ključ (engl. *app key*) i aplikacijsku tajnu (engl. *app secret*) za pristup uslugama. Osim toga, treba odrediti o kojem se tipu aplikacije radi, u ovom slučaju web aplikacija, te na kraju postaviti domenu aplikacije. Ukoliko se radi o lokalnom Rails okruženju tada je to *http://localhost:3000*. Ključ i tajna koriste se u programskom kodu prilikom postavljanja OmniAutha. Za veću sigurnost, preporučljivo ih je pohraniti kao varijable okruženja.

5. APLIKACIJA AlumniETFOS

Kao praktični zadatak u sklopu ovog diplomskog rada izrađena je web aplikacija AlumniETFOS. Aplikacija se sastoji od dva osnovna dijela. Prvi je korisnički, a drugi administratorski dio. Korisnički dio aplikacije zapravo je web stranica čija je glavna svrha promocija udruge AlumniETFOS te prikupljanje informacija o prijavljenim bivšim studentima dok je administratorski dio namijenjen upravljanju korisnicima i sadržajem stranice.

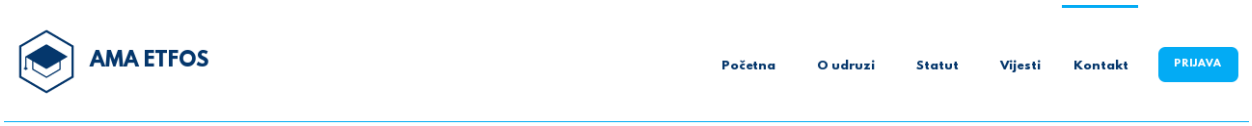
5.1. Opis aplikacije

Korisnički dio aplikacije sastoji se od nekoliko statički definiranih stranica, dijela za prijavu i registraciju korisnika te dijela za pregled novosti vezanih za udrugu. Jedna od statičkih stranica je početna stranica koja sadrži osnovne informacije i poveznice na stranice za registraciju i prijavu korisnika. Registracija i prijava ostvarene su pomoću Devise *gema* čija je uporaba detaljno opisana u prethodnom poglavlju i potkrijepljena primjerima iz aplikacije. Ostale stranice sa statičkim sadržajem su stranice "O nama" i "Statut" čiji sadržaj odgovara njihovim nazivima. Te su stranice definirane odgovarajućim *static_pages* upraviteljem. Upravitelj sadrži metode za prikaz pojedine stranice, a sadržaj upravitelja statičkih stranica i stranice s vijestima prikazan je u kodu 5.1..

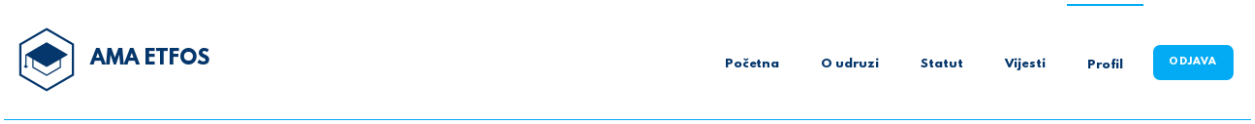
```
class StaticPagesController < ApplicationController

  def home
  end

  def about
  end

  def statut
  end

  def posts
@posts = Post.all.paginate(:page => params[:page], :per_page => 5)
  end

  def single_post
    @post = Post.find(params[:id])
  end

end
```

Kod 5.1. Upravitelj za prikaz stranica aplikacije

Svi korisnici, prijavljeni ili ne, imaju pristup ranije spomenutim stranicama aplikacije kao i stranici na kojoj su prikazane vijesti. Vijesti su promjenjivog sadržaja te njihov sadržaj uređuju i objavljuju administratori aplikacije. Ukoliko žele imati mogućnost prijave, korisnici se najprije moraju registrirati. U slučaju uspješne registracije, korisnici imaju mogućnost pogledati vlastiti profil i unijeti osobne podatke kao *alumni* članovi. Promjena izgleda navigacijske trake ukazuje korisnicima jesu li trenutno prijavljeni u sustav. Na slici 5.1. prikazane je navigacijska traka za prijavljene, a na slici 5.2. navigacijska traka za neprijavljene korisnike. Prijavljeni korisnici jednostavno se odjavljuju klikom na „Odjavi“, a pozadinski dio opisanih akcija obavlja Devise. Korisnici, također, umjesto uobičajene registracije, imaju mogućnost registracije putem korisničkog računa društvene mreže Facebook.

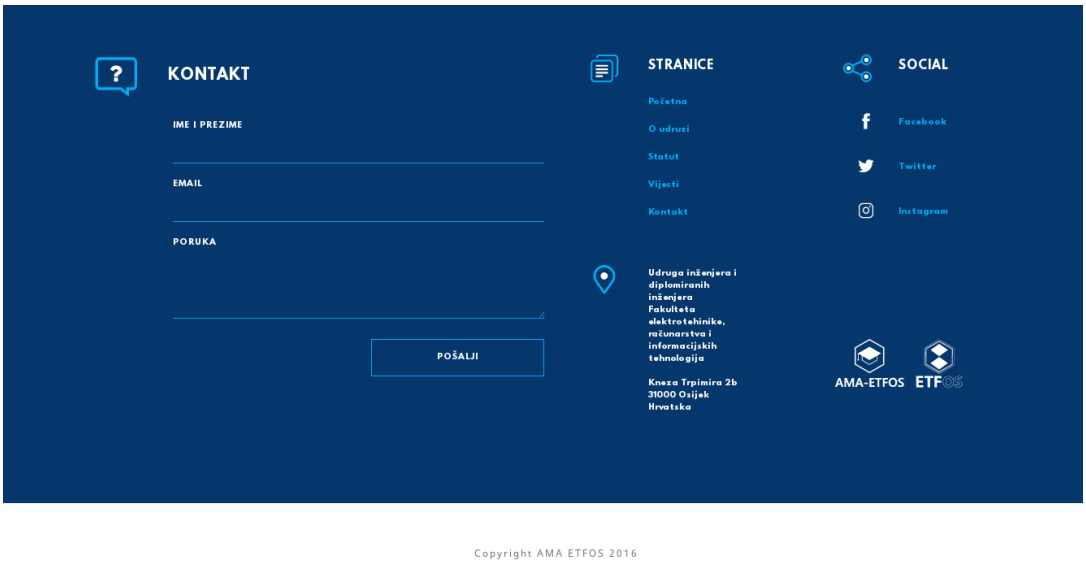


Sl. 5.1. *Izgled navigacijske trake neprijavljenog korisnika*



Sl. 5.2. *Izgled navigacijske trake prijavljenog korisnika*

Svaka od stranica aplikacije ima jednako podnožje s kontakt podacima i kontakt obrascem pomoću kojeg je moguće poslati upit, a prikazano je na slici 5.3.. Upit je dostavljen na kodom definiranu email adresu nekog od administratora. Međutim, funkcija za slanje poruka putem kontakt obrasca nije moguća u lokalnom razvojnom okruženju, već tek kada je aplikacija u produkciji tj., postavljena na udaljeni poslužitelj.



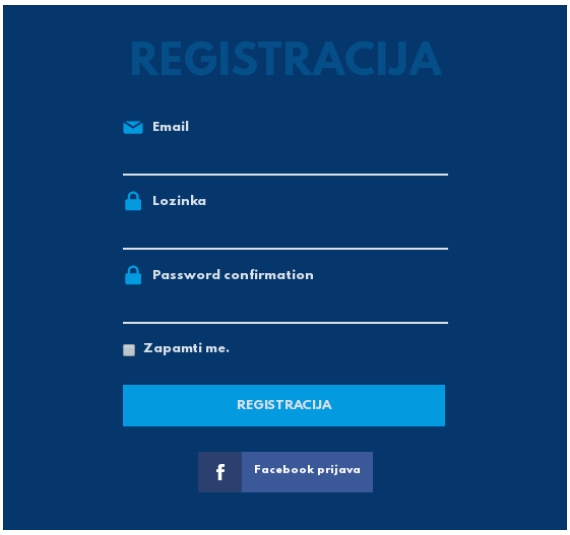
Sl. 5.3. *Podnožje stranice s kontakt obrascem*

5.1.1. Različite korisničke uloge

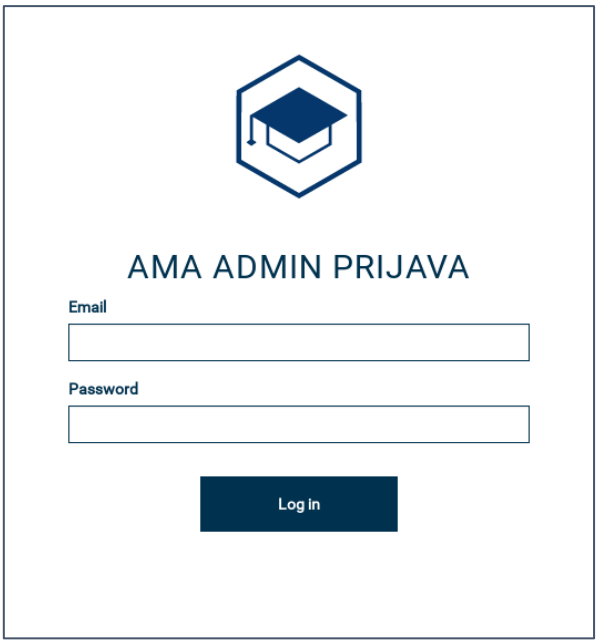
U sustavima čiji korisnici imaju različite uloge, primjerice korisnik i administrator, potrebno je razdvojiti autentikaciju pojedinih vrsta korisnika. Devise olakšava taj postupak te postoje dva načina na koji su mogu realizirati različite uloge u sustavu. Prvi način je korištenjem samo jednog korisničkog modela. I administratori i korisnici tada su objekti istog modela. Međutim, takvi objekti moraju imati dodatni atribut ili stupac u bazi. Taj stupac označava vrstu korisnika, prema uputama navedenim u [12]. Primjerice, u stupcu podatkovnog tipa *bool* administratori će imati oznaku „1“, a korisnici oznaku „0“ pa ih se prema tome identificira.

Drugi način realizacije je kreiranje dvaju zasebnih korisničkih modela različitog naziva. Takav način korišten je za aplikaciju izrađenu kao praktični dio ovog rada. Postoje dva zasebna korisnička modela: *User* i *Admin*. Devise za svaki od njih generira zasebne upravitelje, poglede i rute. U ovom slučaju, definiran je i drugačiji izgled korisničkog i administratorskog dijela aplikacije. Svaki model ima mogućnost obavljanja različitih aktivnosti i vidljive su mu različite stranice. Na slikama 5.4. i 5.5. prikazana su sučelja za prijavu i registraciju korisnika, a slika 5.6. prikazuje jednostavnije sučelje za prijavu administratora. Oba sučelja zapravo su prilagođeni pogledi koje Devise kreira prilikom definiranja modela.

Sl. 5.4. *Obrazac za prijavu korisnika*



Sl. 5.5. *Obrazac za registraciju korisnika*

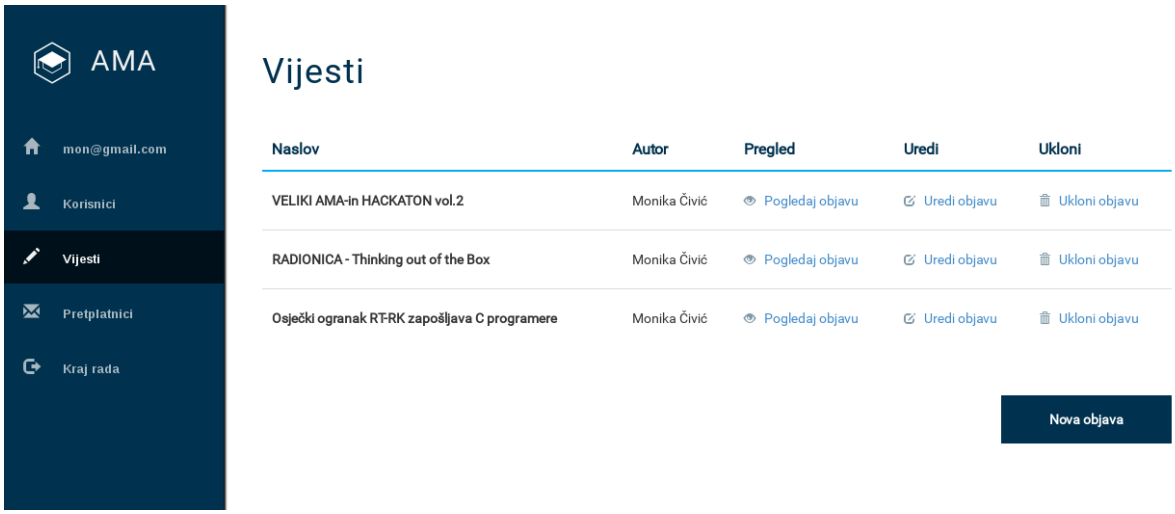


Sl. 5.6.*Obrazac za prijavu administratora*

5.1.2. Administratorski dio aplikacije

Administratorski dio aplikacije AlumniETFOS sastoji se od pojednostavljenog CMS-a (engl. *Content Management System*). Naziv CMS odnosi se na računalne aplikacije koje omogućuju kreiranje i izmjenu digitalnog sadržaja pomoću jednostavnog korisničkog sučelja. CMS je vrlo širok pojam, no većina CMS sustava zasnovana je na objavi i formatiranju sadržaja na webu. Takva sučelja omogućuju odvajanje prezentacije od sadržaja web stranica.

CMS sustavi za upravljanje web sadržajem dizajnirani su ne bi li osobe, koje možda nisu programeri i ne poznaju HTML, CSS ili, u ovom slučaju, Rails, mogle uređivati sadržaj web stranice. Web sadržaji obuhvaćaju tekst, slike, fotografije, video i dr. sadržaje. Svaki sustav takvog tipa sastoji se od dva dijela: aplikacije pomoću koje korisnik s minimalnim znanjem o web aplikacijama može uređivati sadržaj web stranice te aplikacije koja dostavlja postavljeni sadržaj, odnosno, obrađuje ga i osvježava web stranicu ovisno o unesenim promjenama. CMS stranice AlumniETFOS omogućuje korisnicima, koji imaju status administratora, objavu pojedinačnih vijesti. Svaka objava vijesti sastoji se od naslova, autora, teksta vijesti i odgovarajuće fotografije. Kada administrator objavi određenu vijest, ona postaje vidljiva svim posjetiteljima web stranice i nalazi se u odjeljku s vijestima. Svaki administrator ima uvid u sve do tada objavljene vijesti te ih može pregledavati, uređivati ili trajno ukloniti sa stranice i iz baze podataka. Sučelje stranice AlumniETFOS koje predstavlja jednostavan CMS prikazano je na slici 5.7..

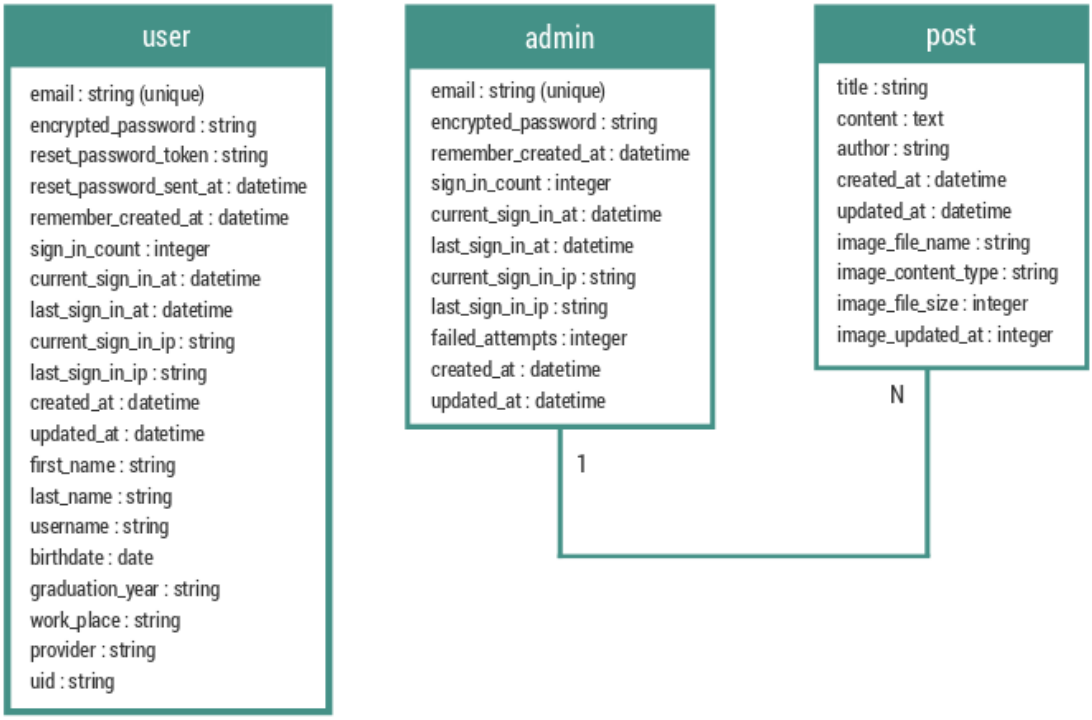


Sl. 5.7. *Jednostavan CMS stranice AlumniETFOS*

Osim izmjene sadržaja web stranice, administratori imaju uvid u sve prijavljene korisnike, *alumni* studente, te informacije koje su isti postavili na svom korisničkom profilu. Ukoliko ustanovi da određeni korisnik nije vjerodostojan, administrator može jednostavno ukloniti korisnika. Budući da su ovlasti koje administrator ima prilično značajne, administratorskim stranicama ne mogu pristupiti bilo koji korisnici. Za autentikaciju administratora, baš kao i korisnika, korišten je Devise *gem* s tim da administratori nemaju mogućnost registracije već ih razvojni programer unaprijed, ručno unosi u sustav. Način realizacije administratorskog računa objašnjen je u 5.1.1..

5.2. Podatkovni model i rute

Baza podataka aplikacije AlumniETFOS sastoji se od tri tablice. Tablice *User* i *Admin* kreirane u bazi prilikom generiranja Devise modela. Iako su i jedan i drugi model Devise modeli, stupci koje sadrže se razlikuju. Razlog tomu su različiti Devise moduli koji su omogućeni za administratora i korisnika. Način na koji su povezani Rails modeli i tablice iz baze podataka detaljnije su razjašnjeni u poglavlju o Rails paketu pod nazivom ActiveRecord. Treća tablica *Post* kreirana je kako bi se ostvarila mogućnost objave vijesti na stranici s vijestima. Vijesti kreiraju administratori te svaka vijest pripada jednom administratoru. Jedan administrator, pak, može imati više objavljenih vijesti. Grafički prikaz podatkovne sheme vidljiv je na slici 5.8., dok je u kodu 2.4. prikazan način na koji je definiran svaki model čija preslika postoji u bazi. Očito je da je svaki takav model zapravo klasa koja nasljeđuje osnovnu klasu iz ActiveRecorda.



Sl. 5.8. *Podatkovni model aplikacije*

Za pristup određenim dijelovima aplikacije, korisnici u preglednik upisuju URL kojim se tada zahtijeva određena metoda upravitelja kako je to opisano u poglavlju 3.3.1. koje govori o Rails upravitelju rutama. Većina ruta aplikacije AlumniETFOS odgovara Devise rutama generiranim prilikom kreiranja Devise modela korisnika i administratora. Ipak, postoje i zasebno definirane rute kao i dodatne rute koje vode na istu akciju kao neka postojeća ruta. One su definirane kako bi se korisniku olakšala navigacija stranicom u slučaju samostalnog upisivanja adrese. Na

primjer, upisavanjem URL-a */users/sign_in* pokreće se ista akcija kao upisivanjem korisniku logičnijeg URL-a */prijava*. Sve rute koje postoje u aplikaciji ispisane su u tablici iz priloga 1.

5.3. Prezentacijski sloj aplikacije

Aplikacija AlumniETFOS trebala bi biti korištena od strane članova udruge AMA ETFOS te postavljena kao njihova web stranica. Iz tog razloga, osim funkcijskog dijela, posvećena je pozornost i dizajnu aplikacije te njenom prezentacijskom dijelu (engl. *front-end*). Prezentacijski dio izrađen je pomoću popularnog HTML, CSS i JavaScript okruženja pod nazivom Bootstrap. Bootstrap je idealno rješenje za brz i učinkovit razvoj responzivnih web stranica čiji se izgled može prilagoditi svakom uređaju neovisno o veličini zaslona. Bootstrap je jednostavno instalirati kao *gem* unutar Rails okruženja. Nakon instalacije, moguće je koristiti Bootstrap klase bilo gdje unutar .html.erb datoteka. Također, moguće je prilagoditi izgled i ponašanje klasa vlastitim zahtjevima definiranjem novih CSS datoteka za čiju je primjenu u aplikaciji odgovoran dio Railsa pod nazivom Asset Pipeline. Uz pomoć ovih alata definiran je različit izgled administratorskog i korisničkog dijela aplikacije.

Dodatni resursi aplikacije (engl. *assets*) su datoteke koje preglednik poziva nakon primitka osnovnog HTML koda. Oni obuhvaćaju CSS stilove, JavaScript skripte, slike, video i ostale slične datoteke. Prilikom razvoja aplikacije, poželjno je organizirati kod u mnogo različitih datoteka kako bi se olakšalo praćenje izmjena, ali i samo programiranje i korištenje resursa. Ipak, ako preglednik mora dohvatiti nekolicinu različitih CSS datoteka, svaki od tih zahtjeva usporit će odziv aplikacije.

Sličan organizacijski problem nastaje prilikom pohrane slika. Jednostavnije ih je koristiti ako su u odvojenim direktorijima. Rails rješenje je takvo da nastoji sve dodatne resurse aplikacije organizirati u veliki zajednički direktorij ovisno o tipu datoteka. Za proces kojim se to obavlja zadužen je Ralisov Asset Pipeline. Za CSS stilove to će značiti da Rails odabire sve individualne datoteke s .css ekstenzijom i slaže ih u jedinstvenu datoteku. Zatim se pokreću *uglifier* i *minifier* programi čija je zadaća ukloniti nepotrebne razmake unutar datoteka ne bi li one bile što manje prije nego im se pristupi iz preglednika. Isti način rada primjenljiv je i na JavaScript datotekama jer kada se radi o HTTP zahtjevima bolje je imati jednu veću datoteku nego nekoliko zasebnih HTTP zahtjeva.

Rails mora znati koje je datoteke potrebno uključiti u spomenutu jedinstvenu datoteku pa se za to koristi manifest datoteka. Primjerice, manifest za JavaScript datoteke nalazi se u

app/assets/javascripts/application.js. Na prvi pogled čini se kako je sadržaj manifest datoteke stavljen u komentare, ali linije koda koje počinju s „//=” zapravo naznačuju Railsu koje je datoteke potrebno uključiti. *Require_tree* pomoćna metoda dohvaća sve što se nalazi u trenutnom stablu direktorija. Manifest za stilove nalazi se u *app/assets/stylesheets/application.css.scss* te je njegov sadržaj prikazan u programskom kodu 5.2.. Iz sadržaja se može uočiti da su datoteke čiji će se stilovi primijeniti na cijelu stranicu sve datoteke unutar direktorija, ali i datoteke iz Bootstrap okruženja.

```
.
.
.
//= require jquery
//= require jquery_ujs
//= require turbolinks
//= require_tree .
//= require bootstrap
```

Kod 5.2. Sadržaj manifest datoteke za stilove

5.3.1. Asset Pipeline primjena i pomoćne funkcije

Kako bi preglednik mogao pristupiti jedinstvenim datotekama koje sadrže stilove ili JavaScript, potrebno je unutar *layouta* aplikacije definirati sljedeću linije koda koje sadrži pomoćne funkcije za uključivanje stilova i JavaScript skripti:

```
<%= stylesheet_link_tag 'application' %>
<%= javascript_include_tag 'application' %>
```

Budući da Rails sažima sve stilove u jednu datoteku, problemi mogu nastati kada postoji mnogo različitih stranica te je potrebno da ista klasa izgleda drugačije na nekoj od stranica. Npr., boja navigacijske trake na početnoj stranici razlikuje se od boje na ostalim stranicama. U slučaju kada se želi izbjeći kopiranje koda postojeće klase u novu uz izmjenu boje, koristi se ugnježđivanje. CSSS koji se primjenjuje za početnu stranicu moguće je ugnijezditi unutar *div* ili *section* elementa s klasom .home te zatim promjene koje će nastati za stranicu s vijestima ugnijezditi u element s klasom .news i na taj način promijeniti vrijednosti samo nekih atributa bez kopiranja čitavog koda. Takav pristup predložen je u [13]. Primjer korištenja prikazan je u kodu 5.3..

```
.
.
.header {
    background-color: $light_blue;
    padding: 6rem;
    margin: 3rem;
}
.section-news {
```

```
    .header {
       background-color: $dark_blue;
    }
}
.section-home {
    .header {
       background-color: $white;
       padding: 2rem;
    }
}
```

Kod 5.3. CSS stilovi prilagođeni Asset Pipeline pravilima

Važno je napomenuti kako Asset Pipeline radi malo drugačije u razvojnom i u produkcijskom okruženju pa je prilikom postavljanja aplikacije na udaljeni poslužitelj potrebno voditi računa o izmjenama unutar konfiguracijskih datoteka u Railsu. Na početku poglavlja spomenuto je kako Asset Pipeline nije namijenjen samo za manipulaciju JavaScript i CSS datotekama, nego i slikama. Za jednostavan pristup slikama potrebno ih je smjestiti unutar samostalno kreiranog direktorija pod nazivom *images* koji se nalazi u */assets* direktoriju. Zatim je za umetanje željene slike unutar HTML-a dovoljno koristiti sljedeću liniju koda s pomoćnom metodom *image_tag* koja zamjenjuje korištenje „<img>“ oznaka:

```
<%= image_tag 'logo.png'%>
```

Rails ima i ugrađene pretprocesore pa je tako osim osnovnog CSS-a i JavaScripta dozvoljeno pisanje Sass ili Coffeescript naredbi dok god su korištene odgovarajuće ekstenzije prilikom pohrane datoteka. Konačni vanjski izgled aplikacije nalazi se na slikama iz priloga 2 koji sadrži direktorij sa slikama zaslona svake pojedine stranice u aplikaciji.

6. ZAKLJUČAK

Teorijski dio diplomskog zadatka obuhvatio je web tehnologije i dodatke potrebne za ostvarivanje sustava koji omogućava korisničke račune s različitim ulogama, točnije korisničkom i administratorskom ulogom. Detaljno je opisano Ruby on Rails okruženje za razvoj web aplikacija te MVC i REST arhitekturni stilovi programiranja preporučljivi prilikom rada s Railsom. Isti ti principi primijenjeni su praktično prilikom izrade aplikacije AlumniETFOS. Kad je riječ o Deviseu i OmniAuthu, obuhvaćene su osnove korištenja i jednog i drugog sustava te obrazložene na primjerima preuzetim iz izvornog koda aplikacije. Na isti način prikazan je rad važnih dijelova Rails okruženja, kao što su ActiveRecord, Rails upravitelj rutama, ali i Asset Pipeline namijenjen za rad s dodatnim resursima poput stilova i slika. Aplikacija AlumniETFOS izrađena je tako da obuhvaća sve teorijski opisane tehnologije i sustave.

Konačna aplikacija ispunjava sve zahtjeve diplomskog zadatka, kao što su postojanje korisničkih računa s različitim ulogama, mogućnost prijave i registracije korisnika uz Devise i OmniAuth, ali i uređivanje podataka na korisničkom profilu. Nadalje, aplikacija omogućuje prijavu administratora koji mogu uklanjati korisnike i uređivati sadržaj osnovne web stranice putem jednostavnog CMS sustava. Iako su osnovni zahtjevi ispunjeni, aplikacija ima još mnogo prostora za nadogradnju prije stvarne upotrebe od strane udruge AMA. Mogućnosti nadogradnje odnose se ne na uvođenje dodatnih Devise modula kao što su oni za email potvrdu registracije ili zaključavanje računa. Osim toga, ostavljen je prostor za implementaciju pretplate korisnika na *newsletter* udruge i procedure za zaboravljenu lozinku. Također, aplikaciju bi, prije objave, trebalo testirati na udaljenom poslužitelju, npr. na Heroku platformi, jer Rails zahtijeva dodatne postavke za okruženje koje nije razvojno. Tek tada bi aplikacija bila u potpunosti spremna za javno korištenje od strane stvarnih korisnika.

Literatura

[1] Wikipedia ,Web application (https://en.wikipedia.org/wiki/Web_application), pristup ostvaren 05.09.2016.

[2] M. Z. Sušac, Web aplikacije (http://www.mathos.unios.hr/wp/wp2009-10/P14_Web_aplikacije.pdf), pristup ostvaren 05.09.2016.

[3] M. Hartl, Ruby on Rails Tutorial: Learn Web Development with Rails, Person Education, New Jersey, SAD, 2015.

[4] SiteGround, Major git commands with example (https://www.siteground.com/tutorials/git/commands.htm), pristup ostvaren 07.09.2016.

[5] Wikipedia, Ruby (https://en.wikipedia.org/wiki/Ruby_(programming_language)), pristup ostvaren 03.09.2016.

[6] Wikipedia, Ruby on Rails (https://en.wikipedia.org/wiki/Ruby_on_Rails), pristup ostvaren 04.09.2016.

[7] The Odin Project, Routing (http://www.theodinproject.com/ruby-on-rails/routing), pristup ostvaren 16.09.2016.

[8] The Odin Project, ActiveRecord (http://www.theodinproject.com/ruby-on-rails/active-record-basics), pristup ostvaren 17.09.2016.

[9] EnvatoTutsPlus, Working with Gems (https://code.tutsplus.com/articles/ruby-for-newbies-working-with-gems--net-18977), pristup ostvaren 04.09.2016.

[10] GitHub, Devise (http://www.rubydoc.info/github/plataformatec/devise/), pristup ostvaren 02.09.2016.

[11] GitHub, OmniAuth (https://github.com/omniauth/omniauth), pristup ostvaren 11.09.2016.

[12] GitHub, How to Add an Admin Role? (https://github.com/plataformatec/devise/wiki/How-To:-Add-an-Admin-Role), pristup ostvaren 03.09.2016.

[13] The Odin Project, The Asset Pipeline (http://www.theodinproject.com/ruby-on-rails/the-asset-pipeline), pristup ostvaren 18.09.2016.

Sažetak

Okosnicu ovog diplomskog rada čini opis programskog okruženja za izradu web aplikacija Ruby on Rails te postupak instalacije i implementacije Devise *gema* za autentikaciju korisnika s različitim korisničkim ulogama. Također, objašnjen je način na koji se korisnike može autenticirati koristeći OmniAuth sustav za registraciju i prijavu putem korisničkih računa društvenih mreža i drugih sustava. U sklopu navedenog, opisani su temeljni pristupi izradi web aplikacija kao što su MVC i REST. Detaljnije je razjašnjen rad s podacima u Railsu i način rada Rails upravitelja rutama. Osim opisnog dijela, za praktični je dio izrađena aplikacija koja sadržava sve prethodno navedene tehnologije. Aplikacija je nazvana AlumniETFOS te se sastoji od korisničkog i administratorskog sučelja, svako za pojedinu vrstu korisnika. Korisnici imaju uvid u statičke stranice s informacijama o udruzi za koju je aplikacija izrađena, stranice s vijestima vezanim za udrugu, ali i mogućnost prijave i registracije u sustav. Korisnici se mogu registrirati unosom traženih podataka, ali i putem Facebook korisničkog računa. Za uobičajenu prijavu i registraciju korisnika zadužen je Devise, dok je Facebook prijava ostvarena pomoću OmniAutha. Prijavljeni korisnici mogu urediti i pohraniti svoje podatke kako bi administratori imali uvid u članove. Administratori nemaju mogućnost registracije, već su u sustav uneseni od strane programera. Prijavljeni administratori imaju pregled svih korisnika te mogućnost uređivanja stranice s vijestima pomoću jednostavnog CMS-a. Osim programskog jezika Ruby i Rails okruženja, za izradu aplikacije primijenjene su HTML, CSS, Sass i JavaScript web tehnologije. Uz Devise i OmniAuth za Facebook korišteni su i drugi *gemovi,* od koji su najvažniji namijenjeni boljoj organizaciji sadržaja i radu sa slikama.

KLJUČNE RIJEČI: web aplikacija, programsko okruženje, model, upravitelj, pogled, *gem*, autentikacija, baza podataka, Rails, Devise, OmniAuth

Abstract

The main subject of this final thesis is the indepth description of Ruby on Rails web application framework along with the Devise gem installation and implementation process. Devise gem is a system which is used to authenticate standard users and users with different user roles within Rails applications. Besides that, the way of authenticating users with social network and other system accounts using OmniAuth is also described. What is more, thesis gives insight into basic principles of buliding web applications, such as MVC and REST architecture. Data management with Rails and purpose of Rails router are explained in detail as well. As for the practical part of the thesis, application called AlumniETFOS was built in a way that it contains all the above mentioned technologies. Application consists of two main parts, part that is used by ordinary users and the second part, used by admins. Users can browse static pages that hold information about AMA association or check news pages with posts related to it. Moreover, users have a possibility to register and log into the system. Users are able to register by submitting a required form data or with their Facebook accounts. For usual registration process, application uses Devise while OmniAuth is in charge of Facebook authentication. Logged in users can edit their personal information so admins can have insight into all current members. Admins are also allowed to delete users and edit the news page using simple CMS built into the application. Besides Ruby programming language and Ruby on Rails framework, technologies such as HTML, CSS, Sass and JavaScript make this application complete. When it comes to Ruby gems, Devise and OmniAuth for Facebook are the most significant ones, but gems for easier content representation and image manipulation, are also important part of this application.

KEY WORDS: web application, framework, model, controller, view, gem, authentication, database, Ruby, Rails, Devise, OmniAuth

Životopis

Monika Čivić rođena je 24. lipnja 1992. godine u Slavonskom Brodu. Nakon završetka osnovnoškolskog obrazovanja upisuje prvi razred II. gimnazije Osijek. 2011. godine upisuje preddiplomski studij Računarstvo na Elektrotehničkom fakultetu u Osijeku, a zatim i diplomski studij. 2013. godine dobila je nagradu tvrtke Siemens PLC za akademska postignuća. Uz studiranje, aktivan je član studentske udruge IAESTE preko koje je odradila dvije stručne prakse: 2015. godine u Kini te 2016. u Njemačkoj.

Monika Čivić

_____

(Potpis studentice)

Prilozi

Prilog 1 – tablica s rutama aplikacije

Prilog 2 – slike zaslona sa svim stranicama aplikacije

Prilog 3 – izvorni kod aplikacije