Implementacija korisničkih uloga u Ruby on Rails aplikacijama

Stančić, Dino

Undergraduate thesis / Završni rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:630410

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-20**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

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

Stručni studij

IMPLEMENTACIJA KORISNIČKIH ULOGA U RUBY ON RAILS APLIKACIJAMA

Završni rad

Dino Stančić

Osijek, 2021.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1S: Obrazac za imenovanje Povjerenstva za završni ispit na preddiplomskom stručnom studiju

Osijek, 19.09.2021.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za završni ispit na preddiplomskom stručnom studiju

Ime i prezime studenta:	Dino Stančić
Studij, smjer:	Preddiplomski stručni studij Računarstvo
Mat. br. studenta, godina upisa:	R4137, 10.10.2018.
OIB studenta:	08388227043
Mentor:	Izv. prof. dr. sc. Irena Galić
Sumentor:	Dr. sc. Hrvoje Leventić
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Časlav Livada
Član Povjerenstva 1:	Dr. sc. Hrvoje Leventić
Član Povjerenstva 2:	Dr. sc. Krešimir Romić
Naslov završnog rada:	Implementacija korisničkih uloga u Ruby on Rails aplikacijama
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rada	Zadatak završnog rada je istražiti i opisati tehnologije i načine implementacije različitih korisničkih uloga (User role) u Ruby on Rails okruženju. Opisati Devise, Warden i CanCan pakete. U praktičnom dijelu rada potrebno je izraditi web aplikaciju koja će demonstrirati implementaciju više različitih korisničih uloga korištenjem Devisea i pratećih tehnologija. Tehnologija: Ruby on Rails Tema rezervirana za: Dino Stančić Sumentor s FERIT-a: Hrvoje Leventić
Prijedlog ocjene pismenog dijela ispita (završnog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina

Datum prijedloga ocjene mentora:	19.09.2021.
Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 27.09.2021.

Ime i prezime studenta:	Dino Stančić
Studij:	Preddiplomski stručni studij Računarstvo
Mat. br. studenta, godina upisa:	R4137, 10.10.2018.
Turnitin podudaranje [%]:	8

Ovom izjavom izjavljujem da je rad pod nazivom**: Implementacija korisničkih uloga u Ruby on Rails aplikacijama**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Irena Galić

i sumentora Dr. sc. Hrvoje Leventić

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD *1*

2. WEB APLIKACIJE *2*

2.1. Povijest Interneta **2**

2.2. Povijest web aplikacija **3**

2.3. Princip rada i arhitektura web aplikacija **3**

2.3.1. MVC arhitektura 4

2.3.2. REST arhitektura i MVC 6

2.4. Verzioniranje koda web aplikacije pomoću Git-a **9**

3. PROGRAMSKI JEZIK RUBY I RUBY ON RAILS OKRUŽENJE *10*

3.1. Ruby **10**

3.2. Ruby on Rails **11**

3.2.1. Prva aplikacija u Ruby on Rails okruženju 13

3.2.2. Struktura direktorija u Ruby on Rails aplikaciji 13

3.2.3. Active Record i Active Model 14

3.2.4. Action View 15

3.2.5. Action Pack 15

3.2.6. Migracije 15

4. AUTENTIKACIJA I AUTORIZACIJA KORISNIKA *17*

4.1. Warden i Devise **17**

4.1.1. Warden 17

4.1.2. Devise i njegovo korištenje 18

4.2. CanCanCan **20**

4.2.1. Korištenje CanCanCan paketa 21

5. FINDIT APLIKACIJA *22*

5.1. Opis aplikacije **22**

5.1.1. Ograničenja unutar aplikacije 22

5.1.2. Persone 23

5.1.3. Stranice 23

5.2. Funkcionalnost i izgled aplikacije **24**

6. ZAKLJUČAK *32*

LITERATURA *33*

Sažetak *35*

Abstract *36*

1. UVOD

Cilj ovog završnog rada je upoznavanje s izradom web aplikacija u Ruby on Rails okruženju uz korištenje Devise, Warden i CanCanCan paketa, te realizacija web aplikacije koja će demonstrirati implementaciju više različitih korisničkih uloga korištenjem navedenih paketa i pratećih tehnologija. Velika pažnja u ovom završnom radu bit će na Devise paket zato što je sva autentikacija bazirana na tom paketu. Uz Devise, obrađen je način rada u Rails okruženju te osnove programskog jezika Ruby. U sklopu praktičnog dijela ovog završnog rada izrađena je web aplikacija za traženje poslova tehničke struke nazvana FindIT. Izrađena aplikacija sastoji se od korisničkog i administratorskog dijela od kojeg se korisnički dio dijeli na dva dijela. Na korisnike koji traže posao i korisnike koji postavljaju oglase.

U sljedećim poglavljima bit će objašnjeni pojmovi web aplikacija, te samog Ruby on Rails okruženja, početak njihovog korištenja i njihovog razvoja do danas. Bit će objašnjeni pojmovi kao što su MVC i REST te koja je njihova razlika. Dodaci koje Ruby on Rails koristi, tzv. *gemovi.* U 5. poglavlju bit će predstavljena FindIT aplikacija i bit će objašnjena njena struktura, ideja i način na koji funkcionira. Izrađena aplikacija sadrži sve navedene tehnologije i pakete, te je zapravo primjena svega navedenoga u praksi.

2. WEB APLIKACIJE

Razvoj Interneta kroz povijest ima veliku ulogu u razvoju web aplikacija. Web aplikacije povećavaju kvalitetu korištenja internetskih preglednika kao i samog Interneta [1]. Razlika između pojma web aplikacija i web stranica je u današnje vrijeme tanka. Po definiciji, web aplikacija je aplikacijski softver koji je, za razliku od računalnog softvera, pokrenut na web serveru, a ne na lokalnom računalu, dok je web stranica grupa globalno povezanih ostalih web stranica koji mogu dijeliti jednu domenu. Web aplikacijama i web stranicama pristupa se uz pomoć web preglednika [2]. Funkcija web preglednika je da korisnicima prikaže podatke dohvaćene sa servera, a serveru vrati podatke koje je korisnik unio. Najveća prednost ovog načina rada je što klijentska strana ne ovisi o korisnikovom operacijskom sustavu, bio on na računalu s Windows, Linux ili MacOS operacijskim sustavom ili bio on na pametnom telefonu. Iako web aplikacije rade jednako na svakom operacijskom sustavu, ne rade jednako na različitim web preglednicima i zato programeri svoje aplikacije moraju zaštiti od takvih slučajeva i omogućiti korisnicima jednostavno i intuitivno korištenje samih aplikacija.

2.1. Povijest Interneta

Internet koji danas imamo potpuno se razlikuje od Interneta u njegovim počecima [3]. Njegov početak kreće 1970. godine gdje je pristup podacima bio vrlo kompleksan jer se sve baziralo na FTP, NNTP i gopher protokolima. Ovi protokoli koriste se i danas, ali su jako pojednostavljeni. Prva prava ideja Interneta kojeg danas poznajemo pokrenuta je ranih 1990.-ih od strane britanskog znanstvenika Tim Berners-Lee-a. Njegova ideja svela se na pojednostavljenje korištenja Interneta pomoću hipertekstualnih veza. Tako je razvio prvi internetski preglednik kojeg je nazvao kao i svoj projekt *WorldWideWeb*, koji je kasnije preimenovan u *Nexus.* Cijeli sustav funkcionirao je samo na *NeXT* računalima koji su bili i pretraživač i uređivač u jednom.

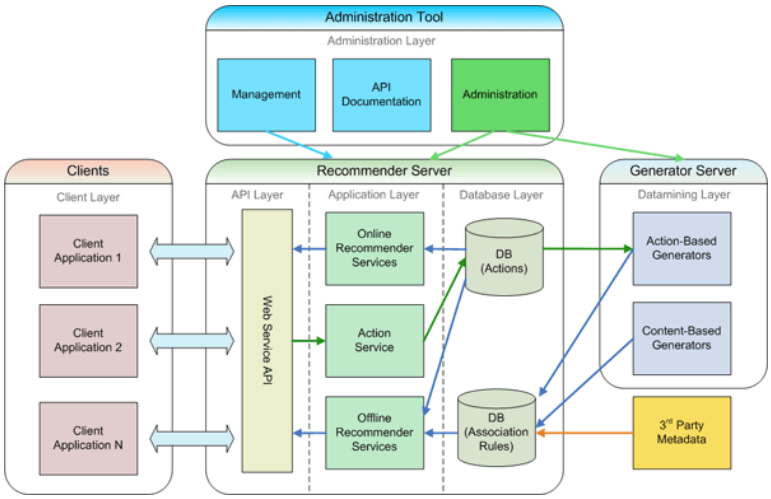
2.2. Povijest web aplikacija

Raniji modeli aplikacija kao npr. klijent – poslužitelj, funkcionirali su tako da je aplikacija jedan dio koda imala na serveru, a drugi dio koda je svaki klijent imao instaliran lokalno. Aplikacija je imala svoje zasebno sučelje i zasebne verzije za korištenje na klijentskom računalu. Svaka promjena koda na serveru, značila je i promjenu koda na klijentskoj strani što je povećavalo cijenu održavanja, a smanjivalo produktivnost samih programera. Suprotno tome imamo web aplikacije gdje se svaka aplikacija pokreće u web pregledniku. Web preglednik u pozadini tu aplikaciju dohvaća uz pomoć HTTP protokola. Prije popularizacije web aplikacija, u samim počecima Interneta, web stranice su svakom klijentu bile dohvaćane kao statički dokument, ali su i dalje imale određenu interaktivnost zbog hiperveza i forma unutar same aplikacije. Za razliku od današnjih web aplikacija, problem je bio što se za svaku promjenu, sadržaj morao mijenjati direktno na serveru.

Prva interaktivnost, te sami početak web aplikacija započeo je 1995. kada je tvrtka Netscape izdala *Javascript* [4]. Skriptni jezik koji je omogućavao programerima manipulacijom dinamičkih elemenata na klijentskoj strani, što je uvelike smanjilo potrebu za komunikaciju aplikacije sa serverom. 1996. godine, tvrtka Macromedia izdala je *Flash,* program za reprodukciju vektorske animacije koji se mogao koristiti u web preglednicima kao dodatak za korištenje animacija. Prvi koncept web aplikacije uveden je 1999. u *Servlet* specifikaciji programskog jezika Jave koji je omogućavao komuniciranje sa serverom uz korištenje HTTP zahtjeva. Iako su u to vrijeme i Javascript i XML bili razvijani, takav način komunikacije nije bio direktno moguć s njima. Zatim je 2005. godine razvijen AJAX koji je omogućio stvaranje asinkronih web stranica gdje su aplikacije mogle asinkrono slati i primati podatke sa servera bez da mijenjaju ponašanje korisničkog sučelja. 2011. godine izdana je finalizirana inačica HTML5 jezika koji pruža mogućnost korištenja multimedije bez korištenja dodataka s klijentske strane. Iako pruža velike mogućnosti, HTML5 je najvažniji pomoćni alat drugim programskim i skriptnim jezicima te njihovim okruženjima kao što su React, Vue, Django i mnogi drugi. Jedan od njih također je i Ruby on Rails koji će biti opisan u narednim poglavljima.

2.3. Princip rada i arhitektura web aplikacija

Web aplikacije funkcioniraju tako da vrše komunikaciju sa serverom [5]. Arhitektura definira način na koji će se ta komunikacija odvijati te omogućava sigurnu međusobnu suradnju više aplikacija. Web aplikacije sastoje se od više slojeva kao što je i prikazano na slici 2.1.

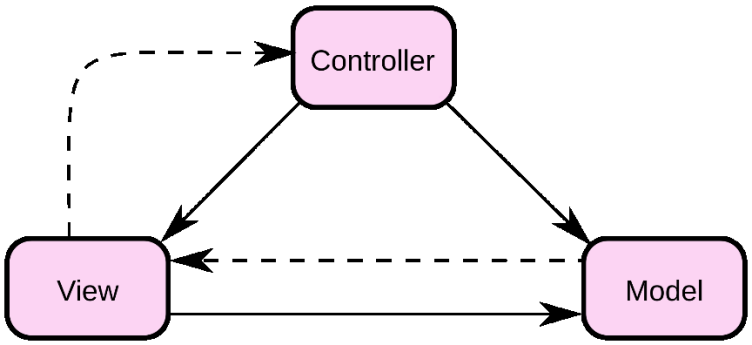


Sl. 2.1 Arhitektura web aplikacija. Izvor: [5]

Kod izrade web aplikacija, programer je taj koji odlučuje što klijentska strana radi u odnosu na serversku i obrnuto. Za izradu klijentske strane uglavnom se koriste HTML, CSS i Javascript, te razna Javascript okruženja kao React, Vue ili Angular, dok se za izradu serverske strane koristi mnoštvo jezika od kojih je jedan Ruby on Rails.

2.3.1. MVC arhitektura

MVC arhitektura prvi puta je bila prikazana 1970 godine [6]. Koristi se za odvajanje aplikacije u tri glavne komponente: model (*model)*, pogled (*view)* i upravitelj (*controller)* prikazane na slici 2.2. Model opisuje poslovnu logiku i karakteriziran je setom klasa, određuje kako se podacima rukuje ili kako se podaci mijenjaju. Pogled služi za prikazivanje korisničkog sučelja korištenjem popratnih tehnologija za izradu kao što su HTML, CSS i Javascript. Pogled također služi za prikazivanje podataka iz modela obrađene upraviteljem. Iza upravitelja stoji sva logika, on služi za procesuiranje svih zahtjeva, te obrađuje model kako bi ga što vjernije prikazao korisniku. MVC developerima pokušava olakšati izradu aplikacije tako da odvajanjem koda u dijelove kod bude čišći, kompaktniji i lakši za održavanje.



Sl. 2.2 MVC koncept. Izvor: [7]

Najjednostavniji primjer MVC arhitekture bila bi aplikacija za kalkulator u kojem bi u modelu bili brojevi, u upravitelju sve matematičke operacije koje bi aplikacija podržavala, a sami izgled kalkulatora predstavljao bi pogled. Složeniji primjer MVC arhitekture može se prikazati na primjeru FindIT aplikacije izrađene u Ruby on Rails okruženju, tako u kodu 2.1. imamo prikazan model oglasa za posao.

```
class Job < ApplicationRecord
  belongs_to :user
    has_many :comments, dependent: :destroy
    has_rich_text :description
end
```

Kod 2.1 Model oglasa za posao definiran u FindIT aplikaciji.

Definirani model svu funkcionalnost obavlja uz pomoć odgovarajućeg upravitelja prikazanog u kodu 2.2.

```
class JobsController < ApplicationController
  include RecruiterHelper
  include CommentsHelper
  before_action :authenticate_user
  before_action :set_job, only: %i[show edit update destroy]

  # . . .

  def index
    @jobs = Job.all
  end

  # . . .

end
```

Kod 2.2. Upravitelj oglasa za posao definiran u FindIT aplikaciji.

Sve mogućnosti modela definirane u upravitelju prikazuju se u odgovarajućem pogledu u kodu 2.3.

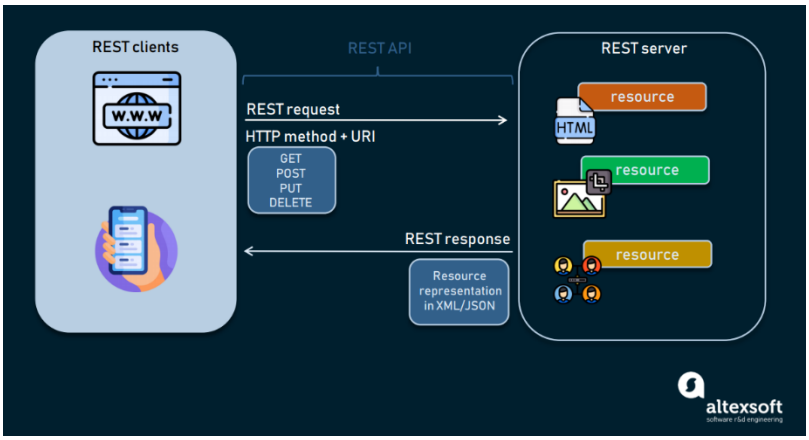
```
<section class="jobs-show-root">
  <div class="root-container">
  <h1 class="jobs-show__title">Pronađite svoj posao iz snova</h1>
    <div class="jobs-show__listing">
      <% @jobs.each do |job| %>
        <%= link_to job, class: "job__card" do %>
          <div class="job__card__header">
            <%= image_tag(find_company_logo(job.user_id), class:
'job__company-logo') %>
            <p class="job__card__header__company-details"><%=
find_company(job.user_id) %> | <%= find_recruiter(job.user_id)%> </p>
          </div>
          <p>
            <span class="job__label">Pozicija:</span>
            <%= job.title %>
          </p>
          <p>
```

```
            <span class="job__label">Senioritet:</span class="job__label">
            <%= job.position %>
          </p>
          <p>
            <span class="job__label">Plaća:</span>
            <%= job.salary_range_min.round %> kn - <%=
job.salary_range_max.round %> kn
          </p>
          <div class="job__location__grade">
            <div class="job__location">
              <span class="job__label">Lokacija:</span>
              <span><%= job.location %></span>
            </div>
            <div class="job__grade">
              <span class="job__label">Ocjena oglasa:</span>
              <span><%= ((job.grade or 0) / (job.count_of_grades == 0 ? 1 :
job.count_of_grades)).round(2) %></span>
            </div>
           </div>
        <% end %>
      <% end %>
    </div>
  </div>
</section>
```

Kod 2.3. Pogled oglasa za posao definiran u FindIT aplikaciji.

2.3.2. REST arhitektura i MVC

REST (*REpresentational State Transfer*) je arhitektura koja pruža standarde za komunikaciju računalnih sustava na mreži, te olakšava način na koji se komunikacija odvija [8]. U REST arhitekturi implementacija klijentske i serverske strane odvojene su jedna od druge kao što se vidi na slici 2.3. To podrazumijeva da promjena koda na klijentskoj strani nema utjecaja na promjenu koda na serverskoj strani i obrnuto. Sve dok obje strane znaju koji se formati poruka koriste, mogu se razvijati odvojeno. Sustavi koji prate REST paradigmu su bez stanja (*stateless*) što znači da nijedna strana ne mora znati stanje druge i obrnuto. Kod REST arhitekture komunikacija sa serverom se odvija tako da klijentska strana šalje zahtjev za dohvaćanjem ili modificiranjem resursa, a server daje odgovor.



Sl. 2.3 Način funkcioniranja REST API-a. Izvor: [9]

REST zahtjev sadrži četiri osnovna dijela: jedan od četiri HTTP zahtjeva, krajnje točke, zaglavlja i tijela [9]. Primjer REST zahtjeva možemo vidjeti na slici 2.4.

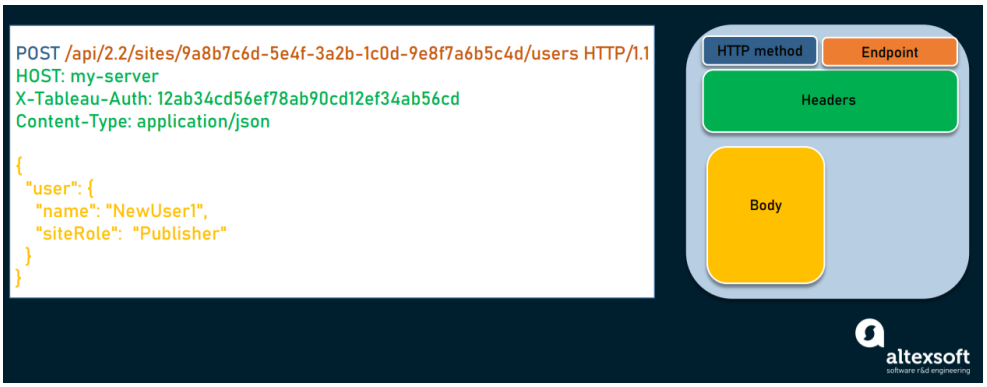
U REST arhitekturi postoje četiri HTTP zahtjeva (koji se ujedno nazivaju CRUD zahtjevi):

- GET – dohvati određeni resurs ili kolekciju resursa
- POST – stvori novi resurs
- PUT – izmjeni određeni resurs
- DELETE – obriši određeni resurs

Krajnja točka (engl. endpoint) sadrži URI (Jedinstveni Identifikator Resursa) koji nam pokazuje gdje i kako pronaći traženi resurs. Najčešći tip URI-a je URL (Jedinstveni Izvor Lokacije) koji služi kao web adresa.

Zaglavlje (engl. headers) sadrži informacije zajedničke klijentu i serveru. Zaglavlja najčešće sadrže podatke za autentikaciju kao što su: API ključevi, naziv ili IP adresa računala na kojem je server instaliran, te informacije o formatu odgovora.

Tijelo (engl. body) se koristi kako bi postojećem zahtjevu dodali dodatne informacije i to poslali na server. To su najčešće podaci koje želimo zamijeniti ili dodati u nekoj bazi podataka.



Sl. 2.4. Primjer REST zahtjeva. Izvor: [9]

REST arhitekturu često vidimo kao alat u MVC arhitekturi, tako da postoji mala razlika, a razlika između MVC arhitekture i REST arhitekture je ta što se MVC brine o načinu na kojem aplikacija funkcionira, dok se REST arhitektura brine o načinu na kojem aplikacija komunicira s drugom aplikacijom. Ruby on Rails kao razvojno okruženje smatran je da prati MVC arhitekturu, ali također prati i REST arhitekturu načinom na koji definira rute. Kreiranjem određenog upravitelja, kreiraju se i određene rute koje su prikazane u tablici 2.1. Kreirane rute služe nam za korištenje odgovarajućih metoda unutar samog upravitelja, ako pokušamo pristupiti nepostojećoj ruti ili ako pokušamo pristupiti ruti koja nema odgovarajuću metodu, Ruby on Rails nam pokaže odgovarajuću poruku pogreške. Korištenje REST API-a unutar Ruby on Rails okruženja možemo objasniti u nekoliko koraka:

1. Model korisnika postavimo takvog da nam odgovara zahtjevu.
2. Unutar upravitelja definiramo metode koje nam odgovaraju određenim rutama.
3. Unutar pogleda kreiramo način na koji ćemo napraviti određeni zahtjev ili prikazati odgovor na određeni zahtjev.
4. Nakon poslanog zahtjeva dobijemo određeni odgovor koji možemo ili prikazati ili koristiti unutar pogleda.

Tablica 2.1 Primjeri Ruby on Rails kreiranih ruta

Prefiks	**HTTP Zahtjev**	**URI**	**Upravitelj#Metoda**
`comments`	`GET`	`/comments(.:format)`	`comments#index`
	`POST`	`/comments(.:format)`	`comments#create`
`new_comment`	`GET`	`/comments/new(.:format)`	`comments#new`
`edit_comment`	`GET`	`/comments/:id/edit(.:format)`	`comments#edit`

2.4. Verzioniranje koda web aplikacije pomoću Git-a

Prilikom izrade aplikacija dobra je praksa koristiti sustave za verzioniranje koda. Sustav za verzioniranje koda je sustav koji prati izmjene u kodu u posebnom obliku baze podataka. Sustavi za verzioniranje jako su dobri i pouzdani jer ako se dogodi neka pogreška, vrlo lako se sustav vrati na prethodnu valjanu verziju [10]. Današnji najpoznatiji sustav za verzioniranje koda je Git. Git je distribuirani sustav za verzioniranje koda napravljen od strane Linusa Torvaldsa 2005. godine kada je BitKeeper, tadašnji najpopularniji sustav za verzioniranje koda, zabranio svoju besplatnu inačicu [11]. Sustav za verzioniranje, Git, može se koristiti na dva načina, pomoću grafičkog sučelja ili pomoću tekstualnog sučelja. Prilikom izrade web aplikacije FIND IT korišteno je Git tekstualno sučelje, te su neke od korištenih naredbi opisane u tablici 2.2.

Tablica 2.1 Git naredbe

NAREDBA	OPIS
`git config`	Koristi se kako bi se konfigurirale određene postavke Git-a. Najčešća primjena je kod postavljanja identiteta korisnika.
`git init`	Inicijalizacija lokalnog repozitorija.
`git clone <url>`	Dohvaćanje repozitorija s nekog sustava.
`git add`	Postavljanje promijenjenih datoteka u indeks kako bi se mogle postaviti na udaljeni repozitorij.
`git commit`	Postavljanje poruke za datoteke u indeksu kako bi se znalo do kakve je promjene došlo.
`git pull`	Dohvaćanje promjena postavljenih na udaljeni repozitorij.
`git pull --rebase <branch-name>`	Akcija koja uz dohvaćanje promjena postavljenih na udaljeni repozitorij primijeni commit po commit na trenutni branch.
`git merge <branch-name>`	Spajanje svih promjena s odabranog branch-a na trenutni branch.
`git push`	Postavljanje promijenjenih datoteka iz indeksa na udaljeni repozitorij

Danas postoji velik broj besplatnih sustava baziranih na Git-u koji se koriste za verzioniranje koda. Za izradu ove aplikacije korišten je GitLab, ali tu još postoje GitHub, Bitbucket i mnoštvo drugih.

3. PROGRAMSKI JEZIK RUBY I RUBY ON RAILS OKRUŽENJE

3.1. Ruby

Ruby je programski jezik visoke razine i generalne primjene dizajniran i razvijan sredinom 1990.-ih godina od strane japanskog programera Yukihiro 'Matz' Matsumotoa [12]. Ruby je dinamično pisani jezik koji koristi sakupljač smeća i dinamično prevođenje. Također podržava više programerskih paradigmi kao što su proceduralno, objektno orijentirano i funkcijsko programiranje. Matsumoto je pri izradi Ruby-a kao inspiraciju uzimao druge programske jezike kao što su Perl, Python, Smalltalk, Eiffel, Ada, BASIC i Lisp. Pošto su znanstvenici u to vrijeme brinuli da programski jezik bude računalima razumljiviji i brži za korištenje, Matsumotov cilj je bio da dizajnira programski jezik koji će povećati produktivnost i zabavu pri samom korištenju, te koji će prvotno biti razumljiv čovjeku, a tek onda računalu. Prilikom izrade, Matsumotu je cilj bio da Ruby bude moćniji od Perl-a, a više objektno orijentiran od Python-a.

U Ruby programskom jeziku, sve je objekt. Svaki dio informacije ili koda može imati svoja svojstva ili akcije. U objektno orijentiranom programiranju svojstva se nazivaju varijable instance, dok se akcije nazivaju metodama. Objektno orijentirana svojstva Ruby programskog jezika najbolje se može pokazati na primjeru koda gdje se akcija primjenjuje na broj. U mnogim programskim jezicima, brojevi i ostali primitivni tipovi nisu objekti. Ruby, poput Smalltalk programskog jezika pridodaje metode svakom tipu.

```
5.times { print "Ruby je super!" }
```

Kod 3.1 Primjer objektno orijentiranog pristupa Ruby programskog jezika

Ruby je jako fleksibilan jer svojim korisnicima, odnosno programerima, dozvoljava da mijenjaju njegove već napravljene klase i metode unutar tih klasa. Glavni dijelovi se mogu micati ili redefinirati kako korisnik to želi. Tako imamo primjer gdje zbrajanje dva broja možemo raditi klasičnim putem tako da napišemo kao u primjeru koda 3.2.

```
a = 5 + 6
# a je sada jednako 11
```

Kod 3.2 Primjer klasičnog zbrajanja dva broja

Ako želimo da nam se dva broja zbroje tako da koristimo neku ključnu riječ kao npr. *plus*, to možemo napraviti tako da *plus* metodu dodamo u Ruby-evu već napravljenu *Numeric* klasu kao u primjeru koda 3.3.

```
class Numeric
  def plus(b)
    self.+(b)
  end
end

a = 5.plus 6
# a je sada jednako 11
```

Kod 3.3 Primjer zbrajanja izmjenom Numeric klase

Ruby pri imenovanju varijabli preferira engleske riječi, ali koristi određene simbole kako bi deklarirao varijable s obzirom na njihov doseg [13]. To su simboli „@“ i „$“ koji se u primjerima koriste:

- `var` – lokalna varijabla
- `@var` – varijabla instance
- `$var` – globalna varijabla

Također ima i mnoštvo drugih osobina kao što su:

- Rukovanje s iznimkama kao Python i Java.
- Sakupljač smeća za svaki objekt bez korištenja dodatnih biblioteka.
- Zato što je napisan u C-u, dodaci se mogu pisati vrlo jednostavno, jednostavnije nego u Perl-u i Python-u te se tako može koristiti kao skriptni jezik.
- Bez obzira na operacijski sustav, omogućava višenitnost za sve platforme koje pokreću Ruby.

Ruby je izvan Japana postao popularan tek u 2000. godini pojavom knjige „Programming Ruby“, a najveću popularnost je dobio dolaskom Ruby on Rails okruženja za web aplikacije [14].

3.2. Ruby on Rails

Ruby on Rails (u nastavku samo „Rails“) je besplatno razvojno okruženje napisano u Ruby programskom jeziku izdano 2004. godine od strane danskog programera Davida Heinemeier Hanssona [15]. Već pri prvom izdanju postao je jedan od najpopularnijih alata za izradu dinamičnih web aplikacija. Rails je korišten u mnogim tvrtkama kao što su *Airbnb, SoundCloud, Disney, Hulu, Github* i *Shopify*. Također, Rails je korišten od strane mnogih startup-a kao i samostalnih programera. Iako postoji velik broj razvojnih okruženja, Rails se izdvaja od mnogih

svojom jednostavnošću, elegancijom i snagom pri izradi web aplikacija. Rails je jednostavan za korištenje samim početnicima jer zbog njegove MVC arhitekture, korisnici ne moraju posezati za drugim alatima. Rails-ova psihologija obuhvaća dva načela [16]:

- **DRY –** Ne ponavljaj se **(**Don't Repeat Yourself) je jedno od načela programiranja koje govori da svaki dio koda mora imati jednu autoritativnu reprezentaciju u programu. Što bi značilo da bi se ponavljanje pisanja jednakih dijelova koda trebalo izbjegavati kako bi kod bio lakši za korištenje i održavanje, te kako bi bio manje podležan pogreškama.
- **Konvencija ispred konfiguracije –** U Railsu postoje mišljenja na koji način je najbolje napraviti određene stvari u web aplikaciji, te je zato postavljen prema tim konvencijama kako bi se izbjegla promjena raznih detalja kroz mnoštvo konfiguracijskih datoteka.

Rails obuhvaća sve što je potrebno kako bi se napravila web aplikacija potpuno podržana bazom podataka. Kako bi se razumjela srž Rails-a, potrebno je dobro razumjeti MVC arhitekturu pojašnjenu u **2.3.1.** U Rails-u, klase modela izvedene su iz *ActiveRecord::Base* ili *ActiveModel* paketa koji će biti objašnjeni u narednim poglavljima. Pogled je sastavljen od predložaka koji su odgovorni za adekvatnu reprezentaciju podataka. Predlošci mogu biti raznih formata, ali su u Rails-u najčešće u .erb formatu što je format za ugrađeni Ruby kod unutar HTML-a. Svi pogledi su generirani uz pomoć *ActionView* paketa koji će biti objašnjen u narednim poglavljima. Rails-ov upraviteljski sloj odgovoran je za rukovanje HTTP zahtjevima i pružanju odgovarajućeg odgovora. Inače bi to značilo vraćanje HTML-a, ali u Railsu upravitelji mogu generirati XML, JSON, PDF i još mnoge ostale formate. Klase upravitelja izvedene su iz *ActionController::Base* paketa koji će također biti objašnjen u narednim poglavljima.

Kako bi započeli s Rails programiranjem potrebno je instalirati određene programe kao što su:

- Ruby
- SQLite3
- Node.js
- Yarn

3.2.1. Prva aplikacija u Ruby on Rails okruženju

Za izradu prve aplikacije potrebno je instalirati Rails gem pokretanjem sljedeće naredbe [17]:

```
$ gem install rails
```

Kod 3.4 Instaliranje Rails gem-a

U Ruby programskom jeziku sve nadogradnje nazivaju se gem-ovi. Kako se koriste i što su bit će pojašnjeno u jednom od narednih poglavlja. Prvu aplikaciju kreiramo naredbom:

```
$ rails new myapp
```

Kod 3.5 Kreiranje prve Rails aplikacije

Gdje nam „myapp“ predstavlja naziv naše aplikacije. Pokretanje naše aplikacije postižemo sljedećim naredbama:

```
$ cd myapp
$ rails server
```

Kod 3.6 Pokretanje Rails aplikacije

Aplikaciji možemo pristupiti putem adrese 'http://localhost:3000'.

3.2.2. Struktura direktorija u Ruby on Rails aplikaciji

Struktura direktorija u Ruby on Rails aplikaciji generirana je pri pokretanju naredbe 3.5, te nije preporučeno da se ta struktura mijenja.

Tablica 2.1 Datoteke i direktoriji kreirani prilikom pokretanja naredbe u 3.5

Datoteka/Direktorij	**Namjena**
`app/`	Sadrži kontrolere, modele, poglede, pomoćne klase i metode, metode za slanje mailova, kanale, poslove, te dodatne datoteke kao što su slike i fontovi.
`bin/`	Sadrži *rails* skriptu koja pokreće aplikaciju i može sadržavati ostale skripte za postavljanje, izmjenu, postavljanje na server.
`config/`	Sadrži informacije o rutama aplikacije, bazi podataka itd.
`config.ru`	*Rack* konfiguracija za *Rack* bazirane servere koji služe za pokretanje aplikacije.
`db/`	Sadrži shemu trenutne baze podataka, te migracije na toj bazi.
`Gemfile` `Gemfile.lock`	Ove datoteke dozvoljavaju specifikaciju koji gem dodaci su potrebni Rails aplikaciji.
`lib/`	Sadrži pakete koji proširuju web aplikaciju.

`log/`	Sadrži datoteke u koje se zapisuju razne informacije o aplikaciji.
`package.json`	Datoteka koja sadrži popis svih paketa koji se trenutno koriste u aplikaciji. Ova datoteka potrebna je *yarn* alatu za upravljanje paketima.
`public/`	Sadrži statične datoteke i prevedene datoteke.
`Rakefile`	Datoteka koja locira i učitava zadatke koji se mogu pokretati iz komandne linije. Zadaci su definirani kroz Rails komponente.
`README.md`	Datoteka koja sadrži osnovne informacije o projektu.
`storage/`	Sadrži *ActiveStorage* datoteke.
`tests/`	Sadrži Unit testove.
`tmp/`	Sadrži privremene datoteke kao što su cache datoteke.
`vendor/`	Sadrži kod iz nepoznatih izvora.
`.gitignore`	Datoteka koja govori git-u koje datoteke da ignorira prilikom verzioniranja.
`.ruby-version`	Datoteka koja sadržava osnovne informacije o verziji Ruby-a koja se koristi.

3.2.3. Active Record i Active Model

Active Record predstavlja M u MVC arhitekturi Ruby on Rails okruženja. Active Record povezuje klase s tablicama u relacijskoj bazi podataka kako bi se uspostavio sloj s vrlo malo konfiguracije za aplikaciju [18]. Paket pruža osnovnu klasu koja, kada se razdvoji, pruža određene mogućnosti između kreirane klase i postojeće baze podataka. Novo kreirane klase nazivaju se modelima i one se mogu povezivati s drugim modelima tj. klasama i to je moguće definiranjem asocijacija. Active Record pokušava pružiti koherentan omotač kao rješenje za objektno-relacijsko mapiranje. Osnovna direktiva za pružanje koherentnog omotača je ta da se smanji količina koda za stvaranje modela iz stvarnog svijeta.

```
rails generate model Article title:string body:text
```

Kod 3.7 Naredba za kreiranje modela

```
class Article < ActiveRecord::Base
end
```

Kod 3.8 Izgled klase modela

Active Model pruža set sučelja koje pružaju određenu funkcionalnost klasama modela [19]. Omogućuju pomoćnim funkcijama da komuniciraju s modelima koji nisu tipa Active Record. Također, pružaju mogućnost stvaranja novih objektno-relacijskih mapiranja izvan samog Rails okruženja.

3.2.4. Action View

Action View predstavlja V u MVC arhitekturi Ruby on Rails okruženja. Action View je okruženje zaduženo za rukovanje pogledima, odnosno pretraživanje i prikazivanje predložaka, te pruža pomoćne poglede koje pripomažu stvaranju HTML elemenata [20]. Action View predlošci napisani su korištenjem ugrađenih Ruby elemenata unutar HTML-a [21]. Action View za svaki kontroler kreira direktorij koji sadrži predložak tog kontrolera. Predlošci služe za prikaz rezultata bilo koje akcije izvršene u kontroleru.

3.2.5. Action Pack

Action Pack predstavlja C u MVC arhitekturi Ruby on Rails okruženja [22]. Action Pack je okruženje zaduženo za rukovanje i odgovaranje zahtjevima. Pruža mehanizam za usmjeravanje tako da mapira URL zahtjeve u akcije definirajući upravitelje koji implementiraju tu akciju. Sastoji se od više dijelova:

- **Action Dispatch** – služi za obradu informacije o zahtjevu, rukuje usmjeravanjem definiranim od strane korisnika, te obavlja napredne obrade povezane s HTTP zahtjevima kao što su: MIME tip pregovora, dekodiranje parametara u POST, PATCH i PUT tijelu zahtjeva, rukovanje kolačićima i sesijama, itd.
- **Action Controller** – pruža osnovnu klasu upravitelja koja se može podijeliti kako bi se lakše primijenili filtri i akcije za rukovanje HTTP zahtjevima. Rezultat akcije je uglavnom sadržaj generiran iz pogleda.

U Ruby on Rails okruženju, korisnici direktno djeluju s Action Controller paketom dok se Action Dispatch automatski aktivira, a Action View prikazuje sadržaj implicitnom aktivacijom od strane Action Controllera.

3.2.6. Migracije

Migracije se u Ruby on Rails okruženju pišu u Ruby programskom jeziku i koriste kako bi izmijenile strukturu baze podataka aplikacije [16]. Primjer migracije prikazan je u kodu 3.9.

```
class CreateArticles < ActiveRecord::Migration[6.0]
  def change
    create_table :articles do |t|
      t.string :title
      t.text :body

      t.timestamps
    end
  end
end
```

Kod 3.9 Primjer migracije

Poziv na *create_table* metodu određuje oblik *articles* tablice koja će biti konstruirana. c*reate_table* automatski stvara *id* stupac u tablici koji predstavlja primarni ključ. U tijelu metode *create_table* definirana su dva stupca *title* i *body* koji su dodani u migraciju pokretanjem naredbe 3.7. Na kraju same metode imamo definiran *timestamps* koji nam još nadodaje dva stupca *created_at* i *updated_at* koji predstavljaju informacije o tome kada smo stvorili model i kada smo ga izmijenili. Kreiranu migraciju pokrećemo naredbom definiranom u kodu 3.10, te kao rezultat migracije dobijemo model s kojim možemo manipulirati bazom podataka.

```
rails db:migrate

# komandna linija ispisat će sljedeće ako je migracija uspješna

==  CreateArticles: migrating ====================================
-- create_table(:articles)
   -> 0.0018s
==  CreateArticles: migrated (0.0018s) ===========================
```

Kod 3.10 Naredba za pokretanje migracije

4. AUTENTIKACIJA I AUTORIZACIJA KORISNIKA

Jedne od najbitnijih značajki web aplikacija koje zahtijevaju kreiranje korisničkog računa su autentikacija i autorizacija. Za autentikaciju korisnika unutar Ruby on Rails aplikacija može se koristiti **has_secure_password** metoda koja već ugrađena unutar samog Ruby on Rails okruženja, ali korištenjem te metode programeri sami moraju pisati svoje validacije lozinki što vrlo brzo postane komplicirano. Ruby on Rails programeri za autentikaciju korisnika najčešće koriste Devise paket koji nam rješava navedeni problem. Pisanje autorizacije predstavlja identičan problem kao i pisanje autentikacije, zato Ruby on Rails programeri koriste CanCanCan paket koji pojednostavljuje pisanje autorizacije. Devise i CanCanCan paketi bit će pojašnjeni u sljedećim poglavljima.

4.1. Warden i Devise

4.1.1. Warden

Warden je rješenje bazirano na Rack-u[1], dizajnirano da pruža mehanizam s mnogobrojnim opcijama za autentikaciju korisnika u Ruby web aplikacijama [23]. Warden je dizajniran tako da ako ga ne koristimo neće promijeniti ništa u aplikaciji, ali ako ga koristimo pružit će nam mnogobrojne mogućnosti za autentikaciju u bilo kojoj aplikaciji baziranoj na Rack-u.

Korištenjem Rack baziranih aplikacija, dozvoljavamo korištenje više aplikacija unutar jedne, tako da Warden omogućava svim posredničkim programima i krajnjim točkama koje se koriste unutar aplikacije da dijele zajednički mehanizam za autentikaciju. Svaka aplikacija zatim može ili pristupiti autenticiranom korisniku ili zatražiti autentikaciju.

Warden funkcionira tako da koristi koncept kaskadnog spajanja strategija kako bi zaključilo treba li zahtjev biti autenticiran. Konceptualno, strategija nam predstavlja logiku za autentikaciju korisnika. Warden će koristiti strategije i prolazit će kroz svaku sve dok:

- Jedna strategija uspije
- Jedna strategija ne uspije
- Ne pronađe relevantne strategije

[1] Rack – Modularno sučelje između web servera i web aplikacije razvijeno u Ruby programskom jeziku.

```
# definiranje strategije

Warden::Strategies.add(:password) do
  def valid?
    params['username'] || params['password']
  end

  def authenticate!
    u = User.authenticate(params['username'], params['password'])
    u.nil? ? fail!(„Could not log in“) : success!(u)
  end
end

# korištenje strategije

env['warden'].authenticate(:password)

# korištenje više strategija

env['warden'].authenticate(:password, :basic)
```

Kod 4.1 Primjer za definiranje i korištenje Warden strategije.

Kod 4.1. pokazuje nam kako definirati strategiju s različitim metodama i kako ju koristiti. U ovom slučaju imamo dvije metode **valid?** i **authenticate!**.

- **valid?** – služi nam kao metoda koja štiti strategiju. Opcionalna je, i ako se ne deklarira, strategija će se izvršiti svaki put, a ako se deklarira, strategija će se izvršiti samo ako **valid?** vrati pozitivan rezultat.
- **authenticate!** – služi nam kao glavna metoda unutar strategije. Sva logika se nalazi unutar te metode i strategija nam se bazira na toj metodi.

Korištenje Warden strategije je jednostavno. Pozovemo ju onako kako smo ju nazvali, a ako želimo postaviti još jednu strategiju koja će se izvršiti, samo ju pozovemo nakon neke strategije.

4.1.2. Devise i njegovo korištenje

Devise je fleksibilno rješenje za autentikaciju u Ruby on Rails okruženju bazirano na Warden paketu [24] opisano u prethodnom poglavlju. Devise je modularno rješenje koje donosi nove funkcionalnosti u Rails aplikacije, omogućuje istovremeno prijavljene višestruke modele, te korištenje različitih modula ovisno o potrebi. Module koje Devise donosi su:

1. **Database Authenticatable –** zaslužan je za hashiranje i pohranu korisničkih lozinki u bazu podataka kako bi se omogućilo autenticiranje korisnika prilikom prijave. Autentikacija je moguća pomoću POST zahtjeva ili osnovne HTTP autentikacije.
2. **Omniauthable –** pruža OmniAuth podršku.

3. **Confirmable –** služi za slanje e-mail adresa s instrukcijama za potvrdu i potvrđuje je li korisnik već potvrdio svoju e-mail adresu prilikom prijave.
4. **Recoverable –** ponovno postavlja korisnikovu lozinku i šalje set instrukcija za postavljanje iste.
5. **Registerable –** rukuje registracijom korisnika kroz proces, također im dozvoljava da unište i urede svoj korisnički račun.
6. **Rememberable –** upravlja generiranjem i brisanjem tokena za pamćenje korisnika iz spremljenog kolačića.
7. **Trackable –** prati broj prijava, vremensku oznaku i IP adresu korisnika.
8. **Timeoutable –** briše sesiju korisnika čija nije bila aktivna određen period vremena.
9. **Validatable –** opcionalan modul koji pruža validaciju za e-mail i lozinku.
10. **Lockable -** zaključava korisnički račun nakon određenog broja neuspjelih prijava.

Prije samog korištenja potrebno je instalirati Devise paket. On se može instalirati tako što se u *Gemfile* upiše linija ***gem 'devise'***, te se nakon toga pokrene naredba ***bundle install*** kako bi se paket instalirao. Nakon toga potrebno je u konzoli napisati naredbu iz koda 4.2

```
rails generate devise:install
```

Kod 4.2 Naredba za generiranje Devise potrebnih datoteka

Nakon pokretanja ove naredbe pokazat će se instrukcije koje je potrebno pratiti kako bi Devise mogao funkcionirati kako treba. Pokrenuti generator instalirat će sve što je potrebno da opiše sve konfiguracijske postavke Devise-a. Generiranje modela pomoću Devise-a pokreće se naredbom iz 4.3. s tim da MODEL u naredbi možemo zamijeniti s bilo kojim nazivom modela.

```
rails generate devise MODEL
```

Kod 4.3 Naredba za generiranje modela

Ova naredba kreira nam migraciju koju, ako želimo, možemo izmijeniti, nadodati dodatne opcije za model ako su nam potrebne i slično. Zatim pokrećemo migraciju s naredbom opisanom u kodu 3.10 kako bi nam se kreirana migracija upisala u bazu podataka i stvorila potrebne entitete.

Devise uz to što kreira modele, akcije i upravitelje, kreira i dodatne pomoćne funkcije koje možemo koristiti za autentikaciju korisnika, provjeru je li korisnik prijavljen, provjeru trenutnog korisnika, pristup korisničkoj sesiji i slično. Nakon što se korisnik prijavi u aplikaciju, Devise traži rutu povezanu s modelom. U ovom primjeru tražit će *user_root_path* i koristiti ga ako

postoji, ako ne postoji koristit će zadanu korijensku rutu, te je iz tog razloga potrebno postaviti zadanu korijensku rutu koja je opisana u kodu 4.5.

```
before_action :authenticate_user! # prije akcija definiranih u kontroleru potrebna je prijava korisnika

user_signed_in? # provjera je li korisnik prijavljen

current_user # pristup objektu trenutnog korisnika

user_session # pristup objektu korisničke sesije
```

Kod 4.4 Primjeri pomoćnih naredbi Devise paketa

```
root to: 'home#index' # index metoda u home kontroleru
```

Kod 4.5 Primjer zadane rute

Devise kreira i rute potrebne za pozivanje određenih akcija unutar upravitelja. Rute se postavljaju automatski i poziv svake od njih aktiviraju određenu akciju. Primjere nekih od kreiranih ruta možemo vidjeti u tablici 4.1.

Tablica 4.1 Primjeri nekih od Devise kreiranih ruta

Prefiks	**HTTP Zahtjev**	**URI**	**Upravitelj#Metoda**
new_user_session	GET	/users/sign_in(.:format)	devise/sessions#new
user_session	POST	/users/sign_in(.:format)	devise/sessions#create
user_registration	POST	/users(.:format)	devise/registrations#create

4.2. CanCanCan

Za projektni zadatak ovog završnog rada, uz Devise, korišten je i CanCanCan paket za autorizaciju napisan u Ruby programskom jeziku [25]. CanCanCan paket olakšava pružanje fleksibilnosti samoj aplikaciji. Korištenjem CanCanCan paketa možemo lakše definirati uloge i dopuštenja korisnika kako bi postavili još jedan dodatni sloj sigurnosti. Služi za restrikciju određenih resursa određenim korisnicima. Sve restrikcije ili mogućnosti definiraju se u jednom ili više *ability* datoteka koje se ne smiju duplicirati u više upravitelja, pogleda i upita baze podataka. Sastoji se od dva glavna dijela:

1. **Autorizacijske biblioteke** – dozvoljava definiranje pravila za pristup različitim objektima i pruža pomoćne metode za provjeru definiranih pravila.
2. **Rails pomoćne metode** – kako bi pojednostavili kod u Rails kontrolerima tako što obavljaju učitavanje i provjeru definiranih pravila i također smanjuju dupliciranje koda.

CanCanCan paket nastao je iz CanCan paketa kreiranog od strane Ryan Bates-a, velika većina problema s kompatibilnosti modernih paketa bila je popravljena, ali originalni dizajn i način funkcioniranja originalnog CanCan paketa ostali su isti.

4.2.1. Korištenje CanCanCan paketa

Korištenje CanCanCan paketa ne razlikuje se puno od korištenja Devise paketa. Prvo je potrebno u *Gemfile* napisati liniju opisanu u kodu 5.1, te zatim pokrenuti naredbu u kodu 5.2. Nakon toga opisane naredbe stvaraju *ability* datoteku u kojoj su opisana pravila što određeni model može raditi. Primjer za *ability* datoteku opisan je u kodu 5.3.

```
gem 'cancan'
```

Kod 4.2 Naredba za instalaciju CanCan paketa

Ability datoteka je ono što ovaj paket čini najbitnijim. Definirana je Ability klasom u kojoj su definirane sve korisnikove mogućnosti. Glavni dio metoda deklariranih unutar Ability klase je **can** metoda koja se koristi za definiranje svih dopuštenja i zahtjeva dva argumenta. Prvi argument je akcija za koju se dopuštenje postavlja, a drugi dio je klasa na koju se dopuštenje postavlja. Postoje četiri glavne akcije, a to su: **:read**, **:create**, **:update**, **:destroy**.

```
rails g cancan:ability
```

Kod 4.3 Naredba za kreiranje *ability* datoteke

```
class Ability <
  include CanCan::Ability

  def initialize (user)
    can :read, Post, public: true

    if user.present?
      can :read, Post, user_id: user.id

      if user.admin?
        can :read, Post
      end
    end
  end
end
```

Kod 4.4 Primjer *ability* datoteke

5. FINDIT APLIKACIJA

Za zadatak praktičnog dijela završnog rada bilo je potrebno izraditi web aplikaciju koja će demonstrirati implementaciju različitih korisničkih uloga korištenjem Devise i CanCan paketa te pratećih tehnologija. Za samo planiranje aplikacije korištena su znanja ostvarena na kolegiju Programsko inženjerstvo, te će pomoću tih znanja biti opisana.

5.1. Opis aplikacije

FindIT je platforma za traženje poslova fokusirana na IT sektor. Uz traženje poslova, omogućeno je postavljanje poslova za koje korisnici moraju napraviti korisnički račun. Svaki registrirani korisnik može ostaviti ili ocjenu ili komentar na oglas za posao. Postavljanje oglasa ograničeno je na naslov, raspon plaće (u kunama), opis posla, mjesto posla. Aplikacija ima dvije vrste korisnika: **admini** i **korisnici**.

- **Admini –** Admini imaju pristup svim sadržajima i imaju read/write pristup u sve sadržaje.
- **Korisnici –** Korisnici se registriraju u aplikaciju kroz sučelje za registraciju te pritom mogu birati jesu li poslodavac (recruiter) ili posloprimac (finder). Jedina razlika prilikom registracije poslodavca i posloprimca je ta što poslodavac pri registraciji mora napisati naziv tvrtke u kojoj radi i poziciju u toj tvrtki. Svaki korisnik koji je poslodavac ima mogućnost postavljanja oglasa, te uređivanja tog istog oglasa tj. ponovno postavljanje drugih. Posloprimci koji pristupe tom oglasu imaju mogućnost ostavljanja ocjene i komentara na taj oglas.

5.1.1. Ograničenja unutar aplikacije

- Anonimni korisnici (ne-ulogirani) nemaju pristup oglasima za posao, vide samo početnu stranicu.
- Mogućnosti registriranih korisnika ovise o tome je li korisnik poslodavac ili posloprimac.
- Admini mogu uređivati sve i vide sve. Registracija Admina se vrši unutar rails_admin sučelja, te je unutar admin sučelja moguće brisanje i uređivanje svega (korisnika, oglasa, komentara), dok je ostale mogućnosti potrebno omogućiti direktno u sučelju aplikacije.
- Poslodavci osim što imaju mogućnost postavljanja oglasa, imaju mogućnost postavljanja edukativnih materijala za početnike.

5.1.2. Persone

- **Admin –** Korisnički račun s admin privilegijama. Imaju pravo pristupa u rails_admin sučelje i mogu mijenjati stvari na razini baze. Specijalni slučaj Admin persone je SuperAdmin, koji predstavlja korisnički račun kreiran pomoću rails generate devise user naredbe, kako bi se moglo pristupiti rails_admin sučelju. SuperAdmin može sve što i Admin i sve funkcionalnosti koje može izvršiti Admin, može i SuperAdmin.
- **Poslodavac –** Korisnički račun koji je kreirao korisnik koji će predstavljati određenu tvrtku. Njegove ovlasti nadilaze ovlasti običnih korisnika.
- **Korisnik** (**Posloprimac**) **–** Korisnički račun koji je kreirao korisnik kako bi ostavio određeni komentar na oglas.
- **Anon –** Korisnik koji nema registrirani račun već samo pristupa stranici.

Svi korisnički računi kreirani su pomoću Devise gem-a. Admini imaju postavljene privilegije unutar rails_admin sučelja. Privilegije ostalih korisnika ovise o tome koji dio stranice gledaju.

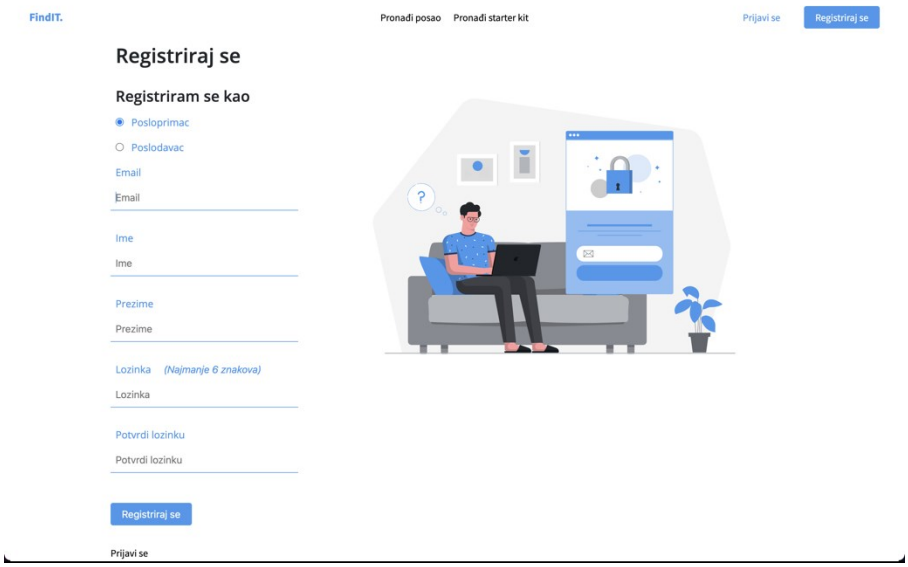
Primjer: Poslodavci mogu uređivati svoj oglas, mijenjati ga kako žele, ali ne mogu uređivati tuđe oglase, tada su njegove mogućnosti kao i od posloprimca.

5.1.3. Stranice

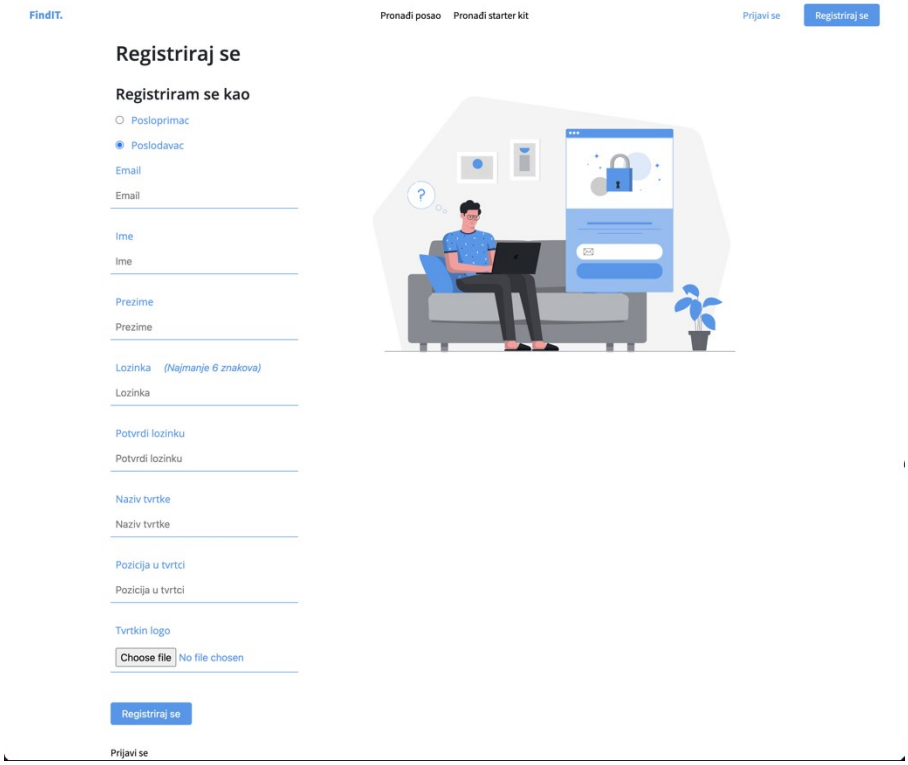
- **HOMEPAGE –** početna stranica. Sadrži sve oglase i određene funkcionalnosti ovisno o tome je li korisnik logiran i koja mu je rola.
- **JOBS –** stranica sa svim objavljenim oglasima za posao.
- **JOB –** stranica određenog oglasa. Sadrži sav opis posla, mogućnost komentiranja i ostavljanja ocjene na oglas.
- **STARTER_KIT_LIST –** stranica sa svim edukativnim materijalima. Sadrži postavljene edukativne materijale/starter kitove od svake tvrtke/recruiter-a.
- **STARTER_KIT –** stranica svakog zasebnog starter kit-a.
- **USER_PROFILE –** stranica u kojoj pojedini korisnik ima pristup svojim podacima i može ih uređivati.
- **ADMIN_DASH** - stranica rails_admin sučelja. Sadrži sve što je potrebno kako bi Admin mogao kreirati/brisati/uređivati sve u bazi.

5.2. Funkcionalnost i izgled aplikacije

Korisnici kako bi koristili sve mogućnosti aplikacije moraju napraviti korisnički račun. Aplikacija ima više korisničkih uloga, te zato postoji različita forma ovisno o tome registrira li se korisnik kao poslodavac ili kao posloprimac, što je i prikazano na slikama 6.1 i 6.2. Registracija administratora vrši se pri inicijalizaciji same baze i nije moguća kroz sučelje aplikacije.



Sl. 6.1 Registracija posloprimca



Sl. 6.2 Registracija poslodavca

Nakon registracije i odabrane korisničke uloge, korisnik ima pristup ostalim mogućnostima aplikacije. Neke od mogućnosti korisnika definirane su uz pomoć CanCanCan paketa unutar *ability.rb* datoteke, prikazano u kodu 6.1. Razlikuju se ovisno o tome kakvu ulogu korisnik ima, a za određene provjere u kreiranim pogledima aplikacije korištene su provjere uloga definirane u shemi korisnika unutar baze podataka, prikazano u kodu 6.2. Shema korisnika i njegove mogućnosti kreirane su uz pomoć Devise paketa opisanog u poglavlju **4.1.2**, ali nove uloge dodane su migracijom prikazanoj u kodu 6.3. Svaki registrirani korisnik ima mogućnost čitanja svih postavljenih oglasa i starter kits-a, dok mogućnost za uređivanje oglasa za posao i starter kits-a imaju samo administratori i poslodavci.

```
class Ability
  include CanCan::Ability

  def initialize(user)

    user ||= User.new # guest user (not logged in)
    if user.superadmin_role?
      can :manage, :all
      can :access, :rails_admin
    end

    if user.recruiter_role?
      can :manage, Job
      can :manage, Comment
      can :manage, StarterKit
      can :read, :all
    end

    if user.finder_role?
      can :manage, Job
      can :manage, Comment
      can :read, :all
    end
  end
end
```

Kod 6.1 Ability datoteka u kojoj su definirane korisničke mogućnosti.

U kodu 6.1 vidljivo je da korisnici koji su poslodavci (recruiter_role) i korisnici koji su posloprimci (finder_role) imaju istu **:manage** ulogu za poslove. Razlog tome je to što posloprimci imaju mogućnost ostavljanja ocjene na određeni oglas za posao, oni ne mogu uređivati oglase što je ograničeno uvjetom u pogledu oglasa za posao.

```
ActiveRecord::Schema.define(version: 20_210_822_160_718) do
  enable_extension 'plpgsql
.
.
.
create_table 'users', force: :cascade do |t|
  t.string 'email', default: '', null: false
  t.string 'encrypted_password', default: '', null: false
  t.string 'reset_password_token'
  t.datetime 'reset_password_sent_at'
  t.datetime 'remember_created_at'
  t.datetime 'created_at', precision: 6, null: false
  t.datetime 'updated_at', precision: 6, null: false
  t.boolean 'superadmin_role', default: false
  t.boolean 'recruiter_role', default: false
  t.boolean 'finder_role', default: true
  t.string 'company_name'
  t.string 'role_at_company'
  t.string 'first_name'
  t.string 'last_name'
  t.index ['email'], name: 'index_users_on_email', unique: true
  t.index ['reset_password_token'], name:
'index_users_on_reset_password_token', unique: true
end
.
.
.
```

Kod 6.2 Dio schema.rb datoteke u kojem je definiran korisnik.

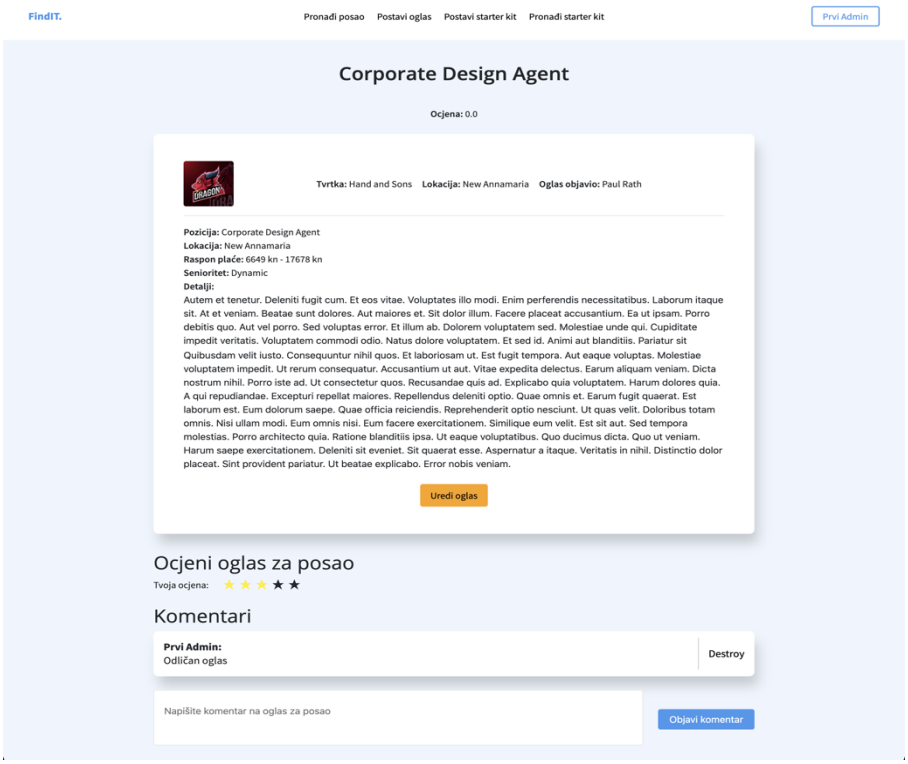
```
class AddRolesToUsers < ActiveRecord::Migration[6.1]
  def change
    add_column :users, :superadmin_role, :boolean, default: false
    add_column :users, :recruiter_role, :boolean, default: false
    add_column :users, :finder_role, :boolean, default: true
  end
end
```

Kod 6.3 Migracija u kojoj su dodane korisničke uloge k.

Uređivati oglase za posao mogu samo oni korisnici koji su postavili oglas i korisnici koji su administratori, što je prikazano u kodu 6.4, a na slici 6.3 vidimo oglas za posao s mogućnošću uređivanja oglasa, davanja ocjene i ostavljanja komentara.

```
.
.
.
<% if current_user.superadmin_role? or job.user_id === current_user.id %>
  <div class="job__edit">
    <%= link_to "Uredi oglas", edit_job_path, class: 'button button--edit'%>
  </div>
<% end %>
.
.
.
```

Kod 6.4 Dio koda u kojem je ograničena mogućnost korisnika za uređivanje oglasa za posao.



Sl. 6.3 Oglas za posao s mogućnošću uređivanja oglasa.

Korisnik klikom na jednu od zvjezdica vidljivih na slici 6.3 daje ocjenu od 1 do 5 oglasu za posao. Funkcionalnost davanja ocjena i mijenjanje boja zvjezdica napravljena je uz pomoć Javascripta, prikazano u kodu 6.5. Za korištenje Javascripta u Rails aplikacijama potrebno je kreirati **.js** datoteku unutar ***app/javascript/packs*** direktorija, zatim je tu datoteku potrebno pozvati unutar nekog pogleda u kojem je napisani Javascript kod potreban, kako je i prikazano u kodu 6.6. Uobičajena je praksa pozivati javascript datoteku na kraju kako bi se ostatak stranice mogao normalno učitati.

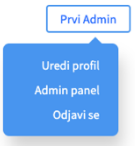
```
.
.
stars.forEach((star, index) => {
  star.addEventListener("mouseenter", () => {
    for (let i = 0; i < index + 1; i++) {
      stars[i].className = "material-icons star-hovered"; }
  });
  star.addEventListener("mouseleave", () => {
    for (let i = index; i < stars.length; i++) {
      stars[i].className = "material-icons"; }
  });
  star.addEventListener("click", () => {
    gradeInput.value = String(Number(gradeInput.value) + index + 1);
    countOfGrades.value = String(Number(countOfGrades.value) + 1);
    submitButton.click();
  });
});
.
.
```

Kod 6.5 Dio koda iz javascript datoteke za funkcionalnost davanja ocjene oglasu.

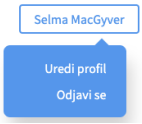
```
<article class="job-view-root">
  <h2 class="example-job-add-title"><%= @job.title %></h2>
.
.
.
</article>
<%= javascript_pack_tag 'jobs', 'data-turbolinks-track': 'reload' %>
```

Kod 6.6 Dio koda iz datoteke pogleda određenog posla gdje se poziva javascript datoteka.

Razlika u mogućnostima administratora i korisnika vidljiva je na slikama 6.4 i 6.5 gdje administratori na padajućem izborniku navigacijske trake imaju dodatnu opciju za pristup admin stranici.

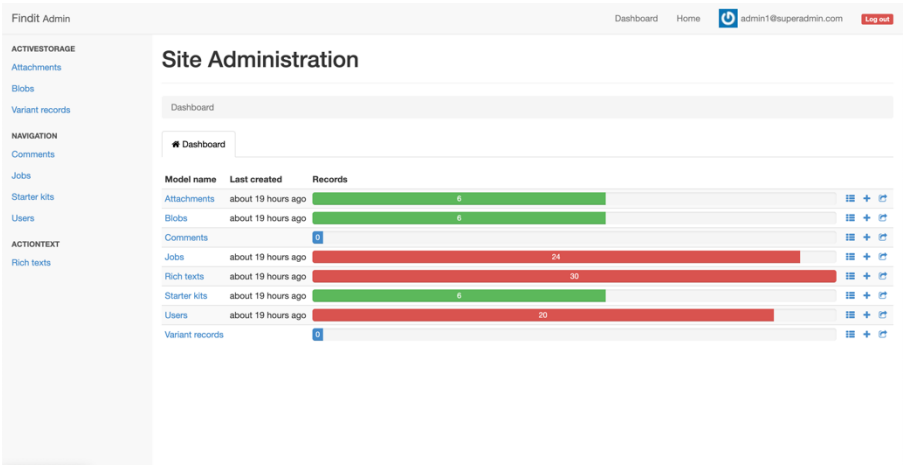


Sl. 6.4 Padajući izbornik navigacije za korisnika koji je administrator.



Sl. 6.5 Padajući izbornik navigacije za korisnika koji nije administrator.

Administratori na admin stranici prikazanoj na slici 6.6 mogu vidjeti i uređivati sve registrirane korisnike, postavljene oglase i starter kits-e. Pristup admin stranici ograničen je provjerom korisnikovog statusa u *_user_dropdown.rb* datoteci prikazanom u kodu 6.7.

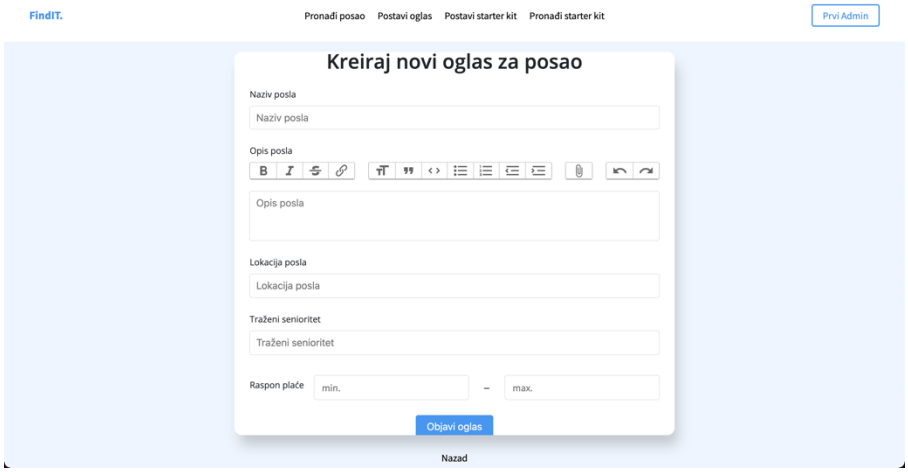


Sl. 6.6 Sučelje admin stranice.

```
<div class="user-dropdown">
  <span class="button  button--secondary--profile">
    <%= current_user.first_name %> <%= current_user.last_name %>
  </span>
  <div class="user-dropdown__popup">
    <%= link_to "Uredi profil", edit_user_registration_path, class: 'user-
dropdown__popup__link' %>
    <% if current_user.superadmin_role %>
      <%= link_to "Admin panel", '/admin', class: 'user-
dropdown__popup__link' %>
    <% end %>
    <%= link_to "Odjavi se", destroy_user_session_path, class: 'user-
dropdown__popup__link', method: :delete %>
  </div>
</div>
```

Kod 6.7 Sadržaj datoteke padajućeg izbornika.

Administratori i poslodavci oglase za posao i starter kits-e postavljaju kroz odgovarajuće sučelje koje koristi *trix* uređivač teksta prikazan na slici 6.7. *Trix* uređivač teksta je tzv. ***wysiwyg*** odnosno „What you see is what you get“ (Što vidiš, to i dobiješ) uređivač koji koristi Rails-ov Active Storage modul, koji korisnicima pruža razne mogućnosti dodavanja i uređivanja teksta, s *Trix* uređivačem teksta moguće je podebljati ili nagnuti slova, postaviti određeni dio teksta kao naslov, dodati datoteku, stvoriti listu …



Sl. 6.7 Sučelje za kreiranje oglasa za posao koje koristi *Trix* uređivač teksta.

Za korištenje *Trix* uređivača teksta u pogledu određenog modela, potrebno je u modelu naznačiti da njegovo polje u bazi podataka koristi ***wysiwyg*** uređivač teksta što je i prikazano u kodu 6.8, dok je njegovo korištenje unutar forme prikazano u kodu 6.9.

```
class Job < ApplicationRecord
  belongs_to :user
  has_many :comments, dependent: :destroy
  has_rich_text :description
end
```

Kod 6.8 Model oglasa za posao koji koristi ***wysiwyg*** uređivač teksta.

```
<%= form_with(model: job, class: 'job__form') do |form| %>
.
.
.
  <div class="job__form__description">
    <%= form.label :description, "Opis posla", class: 'job__input__label
        job__input__label--description' %>
    <%= form.rich_text_area :description, placeholder: "Opis posla",
        required: true %>
  </div>
.
.
.
```

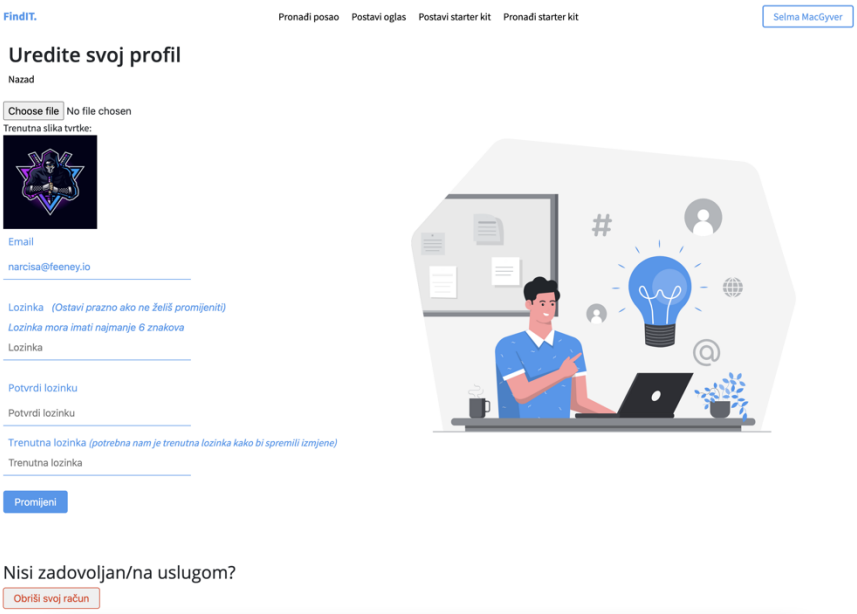
Kod 6.9 Korištenje ***wysiwyg*** uređivača teksta unutar forme oglasa za posao.

Svaki korisnik može urediti svoj profil kroz određeno sučelje, ali samo administratori i poslodavci mogu mijenjati svoju sliku profila, odnosno, logo tvrtke koju predstavljaju. Mijenjanje slike profila ograničeno je provjerom korisničke uloge u pogledu za uređivanje korisničkog računa u kodu 6.10. Izmijenjeni pogled kreiran je uz pomoć Devise paketa i dodatno je uređen kako bi korisniku na ljepši i lakši način omogućio promjenu svojih podataka. Korisnici također mogu obrisati svoj račun ako nisu zadovoljni s aplikacijom što je i prikazano na slici 6.8.

```
.
.
.
<% if current_user.superadmin_role? or current_user.recruiter_role? %>
  <div class="field  field--image">
    <%= f.file_field :company_logo %>
    <% if current_user.company_logo.attached? %>
      <span>Trenutna slika tvrtke:</span>
      <%= image_tag(current_user.company_logo, class: 'company-logo')  if
current_user.company_logo.attached? %>
    <% else %>
      <span>Vaša tvrtka nema sliku profila</span>
    <% end %>

  </div>
<% end %>
.
.
```

Kod 6.10 Dio koda iz pogleda za uređivanje korisnika kreiranog od strane Devise paketa.



Sl. 6.8 Sučelje za promjenu podataka korisnika koji je poslodavac ili admin.

6. ZAKLJUČAK

S razvojem interneta i pratećih tehnologija dolazi do sve većeg broja aplikacija i tehnologija za razvoj istih. Tehnologije koje su se razvijale i koje se još uvijek razvijaju, postoje kako bi korisnicima olakšale korištenje, a programerima razvijanje. Prilikom izrade aplikacija, programeri moraju odabrati koje tehnologije će se koristiti na klijentskoj, a koje na serverskoj strani. U današnje vrijeme je ta podjela na frontend tehnologije, kao što su to Vue, Angular ili React, i podjela na backend tehnologije kao što su NodeJS, Python C#.NET, itd.

Model, View, Controller arhitektura olakšava stvaranje aplikacije jer omogućava stvaranje i klijentske i serverske strane odjednom. Programerima olakšava pisanje čišćeg i jasnijeg koda odvajajući svaki dio u svoju cjelinu.

Ruby on Rails razvojno okruženje prati paradigmu MVC arhitekture i tako predstavlja robusno i jednostavno za korištenje rješenje pri izradi kompleksnijih aplikacija. Ruby on Rails ima odličan CLI i jako puno stvari rješava sam. Također, ima jako dobru potporu zajednice koja stvara različite pakete za različite stvari, tako imamo Devise i CanCan pakete koji se brinu o stvaranju i autorizaciji korisnika, imamo i ActiveStorage koje predstavlja Rails-ovo rješenje za prijenos datoteka i slika.

Ruby on Rails je razvojno okruženje koje postoji već duži period i postojat će zbog svoje robusnosti i jednostavnog načina korištenja. Omogućuje izradu različitih aplikacija i uvijek će predstavljati odlično rješenje za izradu aplikacija koje zahtijevaju MVC arhitekturu.

LITERATURA

[1] https://www.guru99.com/difference-web-application-website.html - "Difference between Website and Web Application", pristupljeno 29.6.2021.

[2] https://www.devsaran.com/blog/history-web-application-development - "From History of Web Application Development" - Pradeep Saran, 8.7.2016., pristupljeno 29.6.2021.

[3] https://blog.keepsite.com/a-brief-history-of-the-web-809509ba23df - "A Brief History of the Web" - John Flynn-York, 22.10.2015., pristupljeno 29.6.2021.

[4] https://sites.google.com/site/webapplicationshistory/ - "Web applications history", pristupljeno 29.6.2021.

[5] https://stackify.com/web-application-architecture - "What is Web Application Architecture? How It Works, Trends, Best Practices and More" - Angela Stringfellow 21.9.2017., pristupljeno 29.6.2021.

[6] https://yourstory.com/mystory/mvp-vs-mvc-vs-mvvm/amp - "MVP vs MVC vs MVVM - Choosing Web Architecture for your Project" - Ronak Patel 6.7.2019., pristupljeno 29.6.2021.

[7] https://hr.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller – „Model-view-controller“, pristupljeno 29.6.2021.

[8] https://www.codecademy.com/articles/what-is-rest - codeacademy, pristupljeno 30.6.2021.

[9] https://www.altexsoft.com/blog/rest-api-design/ - „REST API: Key Concepts, Best Practices, and Benefits“, pristupljeno 30.6.2021.

[10] https://www.atlassian.com/git/tutorials/what-is-version-control - Atlassian, BitBucket, pristupljeno 30.6.2021.

[11] https://en.wikipedia.org/wiki/Git, pristupljeno 30.6.2021.

[12] https://en.wikipedia.org/wiki/Ruby_(programming_language), pristupljeno 1.7.2021.

[13] https://www.ruby-lang.org/en/about/, pristupljeno 1.7.2021.

[14] https://hr.wikipedia.org/wiki/Ruby_(programski_jezik), pristupljeno 1.7.2021.

[15] The Ruby on Rails Tutorial, sixth edition - Michael Hartl, 2020, pristupljeno 1.7.2021.

[16] Getting Started with Rails - https://guides.rubyonrails.org/getting_started.html, pristupljeno 1.7.2021.

[17] Rails repozitorij - https://github.com/rails/rails, pristupljeno 1.7.2021.

[18] Rails repozitorij, active record readme - https://github.com/rails/rails/blob/main/activerecord/README.rdoc, pristupljeno 1.7.2021.

[19] Rails repozitorij, active model readme - https://github.com/rails/rails/blob/main/activemodel/README.rdoc, pristupljeno 1.7.2021.

[20] Rails repozitorij, action view readme - https://github.com/rails/rails/blob/main/actionview/README.rdoc, pristupljeno 1.7.2021.

[21] Action View Overview, https://guides.rubyonrails.org/action_view_overview.html, pristupljeno 1.7.2021.

[22] Rails repozitorij, action pack readme - https://github.com/rails/rails/blob/main/actionpack/README.rdoc, pristupljeno 1.7.2021.

[23] Warden repozitorij - https://github.com/wardencommunity/warden/wiki, pristupljeno 10.9.2021.

[24] Devise repozitorij - https://github.com/heartcombo/devise, pristupljeno 2.7.2021.

[25] CanCan repozitorij - https://github.com/CanCanCommunity/cancancan, pristupljeno 5.7. 2021.

Sažetak

U ovom završnom radu opisana je povijest web aplikacija, te povijest interneta i način na koji je on bio razvijan. Opisan je način na koji web aplikacije funkcioniraju i opisane su različite arhitekture aplikacija kao MVC i REST arhitektura. Velika pažnja posvećena je na verzioniranje koda pomoću Git-a, zato što je verzioniranje jedno od najvažnijih stavki prilikom izrade bilo kakvog softvera. Velika pažnja također je posvećena Ruby programskom jeziku i Ruby on Rails okruženju pomoću kojeg je izrađen praktični dio završnog rada, web aplikacija za traženje poslova koja prikazuje implementaciju različitih korisničkih uloga pomoću Devise i CanCan paketa. Na samom kraju rada, opisana je izrađena aplikacija.

Ključne riječi: Internet, web, aplikacija, MVC, REST, Git, Ruby, Rails, Devise, CanCan

Abstract

In this final thesis, web application history is described. History of how the Internet was developed and the way it was developed. There are numerous explanations of how web applications work and how MVC and REST architecture function. Great attention is on Git code versioning because code versioning is one of the most important things in software development. Ruby and Ruby on Rails are explained thoroughly because the practical part of this final thesis is made with Ruby on Rails. For the practical part, a job-finding web application is made which shows different user roles with help of Devise and CanCan packets. At the end of this final thesis, the created application is described.

Key words: Internet, Web, application, MVC, REST, Git, Ruby, Rails, Devise, CanCan

Životopis

Dino Stančić, rođen 13.7.1998. u Sisku. Nakon završetka osnovne škole upisuje Tehničku školu Kutina. Tamo pokazuje veliko zanimanje za programiranje i nakon toga upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Krajem 2018. godine počinje se baviti web developmentom koje ga počinje zanimati sve više i više, te se još uvijek bavi istim i radi kao developer u PROTOTYP d.o.o.

_____

Potpis autora