Pozadinski poslovi u Ruby on Rails okruženju

Ožecky, Josip

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

Stručni studij

POZADINSKI POSLOVI U RUBY ON RAILS OKRUŽENJU

Završni rad

Josip Oužecky

Osijek, 2020.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

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

Osijek, 18.09.2020.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za završni ispit na preddiplomskom stručnom studiju

Ime i prezime studenta:	Josip Oužecky
Studij, smjer:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI 4627, 24.09.2019.
OIB studenta:	79211110363
Mentor:	Izv. prof. dr. sc. Irena Galić
Sumentor:	Dr. sc. Hrvoje Leventić
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Doc.dr.sc. Mirko Köhler
Član Povjerenstva 1:	Izv. prof. dr. sc. Irena Galić
Član Povjerenstva 2:	Dr. sc. Krešimir Romić
Naslov završnog rada:	Pozadinski poslovi u Ruby on Rails okruženju
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rada	Zadatak završnog rada je istražiti i opisati metode i načine izvršavanja pozadinskih poslova (eng. background jobs) u Ruby on Rails okruženju. Opisati ActiveJob modul i najčešće korištene sustave za izvršavanje ActiveJob poslova (Sidekiq i slični). U praktičnom dijelu izraditi web aplikaciju u Ruby on Rails okruženju, kojom će se demonstrirati i testirati primjena nekoliko različitih sustava za raspodjeljivanje (eng. Scheduling) ActiveJob poslova. Tehnologija: Ruby on Rails, Linux Tema rezervirana za: Josip Oužecky Sumentor s FERIT-a: Hrvoje Leventić Sumentor iz tvrtke: Vedran Mijatović (Bamboo Lab)
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: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	18.09.2020.

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, 24.09.2020.

Ime i prezime studenta:	Josip Oužecky
Studij:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI 4627, 24.09.2019.
Turnitin podudaranje [%]:	4

Ovom izjavom izjavljujem da je rad pod nazivom**: Pozadinski poslovi u Ruby on Rails okruženju**

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

Web programiranje podrazumijeva razvijanje internet stranica pomoću raznih tehnologija kao što su HTML (Hyper Text Markup Language), CSS (Cascading Style Sheets) i Javascript na klijentskoj strani te tehnologije kao što su PHP, Ruby on Rails, .NET na serverskoj strani. U današnjem svijetu sve se vrti oko interneta te stvari koje su se radile ručno, na primjer plaćanje računa mogu se danas platiti putem pametnih telefona i računala uz uvjet da uređaj ima pristup internetu. Korisnici od internet stranice očekuju da je brza i sigurna. Prilikom korištenja internet aplikacije, korisnik očekuje da ne mora puno čekati kako bi se određeni zadatak obavio. Dvadesetominutni video s HD rezolucijom se ne može prenijeti na internet u istom trenutku nakon što je korisnik stisao na primjer tipku "učitaj" (vrijeme trajanja prebacivanja videa ovisi o brzini interneta koju korisnik ima na raspolaganju), ali korisnik očekuje nakon pritiska te iste tipke da može normalno koristiti aplikaciju te dobiti obavijest nakon završetka posla odnosno kada se video uspiješno postavi na internet. Zbog ovakvih stvari se koriste pozadinski poslovi koji omogućuju da se nakon slanja zahtjeva za određenim zadatkom posao obavlja u pozadini dok korisnik može normalno koristiti internet aplikaciju. Ovo je samo jedan od mnogobrojnih primjera gdje se pozadinski poslovi koriste i gdje se vidi korisnost ove tehnike.

Tehnologije kao što su Ruby (programski jezik) i Ruby on Rails (mrežni okvir za izradu internet aplikacija napisan u programskom jeziku Ruby) omogućavaju razvijanje internet aplikacija puno većom brzinom nego s drugim tehnologijama i zato su ove tehnologije vrlo dobre za projekte jer mogu u vrlo kratkom vremenu osposobiti kompleksniju internet aplikaciju. Sustavi za raspoređivanje poslova služe kako bi se poslovi raspodijelili na način da se jednostavniji poslovi koji ne trebaju puno vremena za izvršavanje izvode prije nego poslovi koji su kompleksniji i zahtjevniji te znatno duže traju ili se ti isti poslovi izvršavaju po principu važnosti prikazano u [1], tj. ako je jedan proces bitniji od drugog on se prvi izvršava.

1.1. Zadatak završnog rada

Zadatak završnog rada je istražiti i opisati metode i načine izvršavanja pozadinskih poslova u Ruby on Rails okruženju te opisati Activejob modul i najčešće korištene sustave za izvršavanje ActiveJob poslova (Sidekiq, Backburner, Que, Sneakers, Delayed Job). U praktičnom dijelu izradila se internet aplikacija u Ruby on Rails mrežnom okviru u kojoj se demonstriraju i testiraju primjena sustava Sidekiq za raspodjeljivanje poslova s nerelacijskom bazom podataka Redis.

2. KORIŠTENE TEHNOLOGIJE

Za izradu internet aplikacije korištene su različite tehnologije kako bi se mogla prikazati funkcionalnost aplikacije te naravno dizajn iste aplikacije. Ruby on Rails je mrežni programski okvir koji je korišten za izradu internet aplikacija te je omogućio nesmetan i jednostavan rad u tehnologijama kao što su HTML i CSS. Ovaj mrežni okvir je osigurao međusobnu povezanost baze podataka s preostalim dijelom aplikacije preko SQLite baze. Isto tako korištena je SASS odnosno nova sintaksa jezika CSS-a kako bi se neke stvari brže i jednostavnije napisale. Mrežni programski okvir Boostrap je korišten s ciljem postizanja ljepšeg izgleda internet aplikacije, a korištena je i tehnologija Git za verzioniranje projekta putem servisa GitHub. Kako bi se aplikacija razvila u produkciji odnosno na internetu korišten je Heroku (oblak platforma koja služi kao servis za podizanje internet aplikacija na internet) kao što je objašnjeno u [2]. Active Job modul služi za kreiranje te stavljanje poslova u red te izvršavanje istih poslova u pozadini kao što je prikazano u [3]. Sidekiq i Redis su korišteni kako bi se pokazalo sučelje sidekiq-a te kako bi se vidjele detaljnije informacije o poslovima koji se trebaju izvršiti. Sidekiq i Redis su najučestalije tehnologije koje se koriste za pozadinske poslove u Ruby on Rails aplikacijama kao što je objašnjeno u [4].

2.1. Ruby

Ruby je objektno - orijentirani te dinamički programski jezik koji upotrebljava sintaksu kao što je programski jezik Perl. U Japanu ga je razvio Yukihiro Matsumoto, a u ostalim zemljama programski jezik Ruby doživljava svoj vrhunac tek u 2000 - toj godini s pojavom knjige na stranom (engleskom) jeziku pod nazivom „Programming Ruby“. Program Ruby danas je najrašireniji programski jezik. Riječ je o programskom jeziku opće namjene koji se koristi na više platformi te je otvorenog koda što znači da svi mogu dobiti uvid u kod. Jezik se primjenjuje u područjima sistemske administracije i izrade internetskih aplikacija, a nakon 2006 - e godine programeri ga primjenjuju u drugačije svrhe poput programiranja analognog hardwarea, programiranja računalnih igara i stvaranja elektroničke glazbe. Tvorac je htio izraditi programski jezik koji će programerima pomoći da budu produktivniji, uživaju u svome poslu te budu sretniji. Načelo kojim se vodio bilo je dizajniranje korisničkih sučelja te je više računa vodio o ljudskim nego kompjuterskim potrebama. Ruby je programski jezik koji ima veliku zajednicu i za gotovo sve probleme postoje rješenja na internetu. Ruby ima puno sličnosti s ostalim programskim jezicima po pitanju klasa, polimorfizma i nasljeđivanja, ali ključna razlika je u tome da kod Ruby-a bilo to varijabla, polje ili metoda je zapravo objekt. Ovaj

programski jezik se može brzo savladati i naučiti jer je vrlo jednostavan za početnike no nešto je kompliciraniji ukoliko ga se želi u potpunosti svladati. Što se tiče sintakse i pogrešaka vrlo je rijetko da će ih biti puno jer ima jako malo pravila o sintaksi kojih se treba pridržavati. Na slici 2.1. je prikazana sintaksa programskog jezika Ruby. Objašnjeno je u [2,5].

```
# tipični „hello world“ program

puts "Hello World!"

_____________________________________________________________________________


# stvaranje i poziv funkcije

def hello(name)
  puts "Hello #{name}"
end

hello("world")

_____________________________________________________________________________


# definicija klase

class Person

  def initialize(name)
    @name = name
  end

  def say(text)
    puts "#{@name} says: " + text
  end

# stvaranje objekta i korištenje metode say

frank = Person.new("Frank")
frank.say("Hello everyone")

_____________________________________________________________________________
```

Sl. 2.1. Prikaz sintakse programskog jezika Ruby

2.2. Ruby on Rails

Mrežni programski okvir Ruby on Rails služi kako bi se napravila internet aplikacija po MVC arhitekturi (Model – View - Controller) s bazom podataka. Ovaj mrežni programski okvir sadrži sve što je neophodno kako bi se ista aplikacija razvila što brže. Razumijevanje

MVC arhitekture je ključno za razumijevanje Ruby on Railsa. MVC arhitektura raspodijeljuje aplikaciju u tri sloja: model, pogled i upravljač te svaki sloj ima svoju funkcionalnost. Sloj "model" reprezentira modele domene kao što je korisnički račun ili na primjer nekakav produkt na internet stranici koja ima ulogu rasprodaje određenih proizvoda. Svaki model domene sadrži poslovnu logiku specifičnu za pojedinu aplikaciju. Upravljač prihvaća razne ulazne podatke, obrađuje ih te pretvara u naloge modelu ili pogledu. Upravljač je odgovoran za upravljanje HTTP zahtjeva koji dolaze te za osiguravanje odgovarajućeg odgovora, a u većini slučajeva to bi značilo vraćanje HTML-a. Rails upravljač može generirati XML, JSON i mnoge druge formate kao odgovor. Pogled ili "view" je bilo koji grafički prikaz podataka kao što je tablica, formular, dijagram ili na primjer statična internet stranica. Pogledi u Rails okruženju su najčešće u .erb formatu koji je zapravo HTML s ugrađenim Ruby kodom. Ruby on Rails sadrži alate koji web programerima olakšavaju tipične zadatke poput "scaffoldin" alata koji omogućuje automatsko konstruiranje modela i pogleda koji su potrebni za osnovnu internet stranicu. Internet aplikacija u Rails-u se tipično objavljuje u produkciji s bazom podataka kao što je MySQL ili PostgreSQL te je ovo prikazano i objašnjeno u [2,6,7,8] .

Rails koristi razne mrežne programske okvire te biblioteke koje služe za odrađivanje specifičnih poslova u aplikaciji. DRY (eng. Don't Repeat Yourself) princip je jedan od principa koje Rails želi naglasiti. Ovaj princip govori kako se kod ne bi trebao ponavljati. Primjer poštivanja ovakvog principa bi bile metode ili funkcije jer se one mogu definirati samo jednom, a pozvati koliko god puta želimo te je samo na jednom mjestu napisana funkcionalnost. Ukoliko se nešto želi promijeniti samo se promijeni kod u metodi. CoC (eng. Convetion over Configuration) je princip kod kojeg programer jedino treba specificirati različite segmente aplikacije. Na primjer ako postoji klasa Car kao model, odgovarajuća tablica koja predstavlja podatak o modelu Car automatski će u bazi podataka dobiti ime cars. Poštivanjem ovih principa izrada internet aplikacija je učinkovita, brza i jednostavna kao što je objašnjeno u [1].

Klasa ActiveRecord označava u MVC (Model – View - Controller) arhitekturni model. Ova klasa pojednostavljuje kreiranje i uporabu objekata čiji podatci podrazumijevaju dugotrajnu pohranu u bazi podataka. ActiveRecord zapravo je objektno relacijsko mapiranje (eng. ORM – Object Rational Mapping) za programski jezik Ruby. Objektno relacijsko mapiranje je metoda u programiranju za transformiranje podataka između nekompatibilnih ili

neusklađenih tipova podataka u objektno orijentiranim programskim jezicima. Ova metoda ili tehnika kreira neku vrstu "virtualne objektne baze podataka" koja se koristi unutar programskog jezika, ako je potrebno. ActiveRecord ima nekoliko svojstava, a to su da karakterizira modele i njihove podatke, označava korelaciju između tih modela, karakterizira hijerarhiju nasljeđivanja putem povezanih modela, ispituje podatke prije spremanja u bazu podataka te na objektno orijentirani način provodi operacije nad bazom podataka. Action View u Rails aplikaciji omogućava osvježavanje postojećih pogleda ili prikazivanje novih. Svaki upravljač unutar internet aplikacije ima svoj poddirektorij unutar direktorija ***app/views*** gdje se nalaze svi predlošci koji služe za generiranje pogleda za taj upravljač. Kreiranje početne internet aplikacije u Ruby on Rails okruženju je vrlo jednostavno uporabom svega nekoliko komandi (programski jezik Ruby mora biti instaliran te Ruby on Rails mrežni okvir) kao što je objašnjeno u [2,9,10]:

1. Napraviti novu Rails aplikaciju (myapp je naziv aplikacije)

```
$ rails new myapp
```

2. Promjeniti direktorij u myapp - u te pokrenuti internet server

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

3. Otvoriti internet preglednik te upisati adresu: http://localhost:3000. Zatim bi trebali vidjeti početnu stranicu aplikacije u Ruby on Railsu te će se vidjeti rečenica "Yay! You're on Rails! "

Ruby on Rails se pokreće na intepretatoru Matz koji je jako kritiziran zbog problema sa skalabilnošću. Ova osuda se spominje zbog pogrešaka u okviru Twittera tijekom 2007. i 2008. godine. Zbog toga je Twitter prešao na programski jezik Scala, ali samo djelomično (Java virtualna mašina pokreće ovaj programski jezik). Ruby on Rails je pogonio sučelje Twittera do 2011. godine nakon čega je zamijenjen zbog perfomansi. Iako je Twitter kritizirao Rails, Gartner Research je došao do zaključka da mnoge tvrtke koje se bave internet programiranjem upotrebljavaju upravo Rails za velike i agilne skalabilne aplikacije. Neki od većih sajtova su GitHub, Shopify, BaseCamp i mnogi drugi. Zanimljiva činjenica je da od siječnja 2016. godine preko 1.2 milijuna internet stranica koristi Ruby on Rails. Što se sigurnosti tiče, u 2013. godini otkriven je sigurnosni propust u Ruby on Rails okruženju to jest problem je bio što je cijela hash sesija bila smještena unutar sesije kolačića dopuštajući bilo koju autentificiranu sesiju da se ulogira kao ciljani korisnik kada god bi želio u budućnosti. Daniel Jakson i Joseph Near su

istraživači koji su napravili program „Space“ koji je otkrio mnogo novih sigurnosnih propusta. Danas su ove greške ispravljene u Ruby on Rails okruženju kao što je prikazano u [6].

Ruby on Rails je usklađen s REST arhitekturom. REST (eng. Representational–State-Transfer) je zapravo stil programske arhitekture koji se upotrebljava u razvoju distribuiranih i mrežnih sustava te za razvoj internet aplikacija. REST arhitekturom se definira kako klijent i poslužitelj komuniciraju pri korištenju različitih resursa pomoću HTTP protokola. Klijent pošalje zahtjev poslužitelju, poslužitelj primljeni zahtjev obradi i stvara odgovarajući odgovor te ga zatim vraća nazad klijentu. Podatak odnosno resurs može biti npr. slika, tekst ili video te on sadrži specifičnu adresu lokacije korištenjem univerzalne sintakse koja se upotrebljava u hipermedijskim poveznicama. Možemo zaključiti da RESTful internet servisi upotrebljavaju uobičajeno HTTP protokol za transmisiju, a za adresiranje se koristi URL mehanizam. Podaci se jednoznačno identificiraju i određuju s URI, a za sve manipulacije nad resursom ili podatkom koristimo HTTP metode odnosno CRUD. Ako želimo dohvatiti nekakav resurs koristimo GET metodu, a ukoliko neki resurs želimo obrisati koristimo DELETE metodu. PUT metoda nam služi za stvaranje novog resursa, a POST metoda za ažuriranje resursa. Pojmovi su objašnjeni i prikazani u [2, 11, 12, 13].

2.3. HTML I CSS

HTML je skraćenica od HyperText Markup Language, a koristi se kao prezentacijski jezik za izrađivanje internet stranica. Pomoću HTML-a kreiraju se sadržaj i hiperveze među različitim dokumentima. HTML je vrlo lagan jezik te se brzo savladava i uči. Zbog ove činjenice je opće prihvaćen i korišten. Od samog početka HTML je bio besplatan, svima dostupan i jednostavan te ja upravo radi toga danas toliko raširen. Putem internet preglednika se može vidjeti HTML svake internet stranice preko takozvanog "DevTools-a". Bitno je naglasiti da HTML nije programski jezik te da se programeri ne koriste ovim jezikom kako bi izvršili nekakvu zadaću. Naime, pomoću ovog jezika se ne mogu izvršiti ni najjednostavnije matematičke operacije kao što su zbrajanje i oduzimanje brojeva. HTML je čista tekstualna datoteka koja dolazi s ekstenzijom .html ili .htm. Temeljni građevni elementi svake internet stranice čine znakovi odnosno elementi (eng. tags) koji služe za opisivanje prikaza pojedinog segmenta za internet stranici. Poveznice odnosno linkovi u HTML-u povezuju različite dokumente, stvaraju hijerarhijsku strukturu i samim time definiraju kako zapravo posjetitelj odnosno korisnik doživljava internet stranicu. Na slici 2.2. prikazan je izgled HTML sintakse te su ostali pojmovi objašnjeni u [14, 2].

```
<!DOCTYPE html>
<html>
  <head>
    <title> Naziv stranice </title>
  </head>
  <body>
    <p> Ovdje se unosi sam sadržaj stranice. </p>
  </body>
</html>
```

Sl. 2.2. Prikaz sintakse HTML-a

CSS je skraćenica od Cascading Style Sheets. Ovaj jezik se koristi kako bi se vizualno prezentirao HTML. Korištenjem CSS-a formulira se HTML tj. definira se kako će se pojedini element u HTML-u prikazati na internet pregledniku. CSS ima nekoliko pravila koja se odnose na sintakse, a svako pravilo obuhvaća selektor i dekleracijski blok. Selektor predstavlja dio HTML-a na koji će se aplicirati određeni stil. Selektori su na primjer znakovi odnosno elementi HTML-a, klase ili identifikatori. CSS ima pseudoklase koje omogućavaju interpretiranje informacija odnosno podataka koje nisu raspoložive u DOM-u (Document Object Model) kao što je npr. pseudoklasa ***:hover*** koji identificira sadržaj samo ako korisnik pređe mišem preko tog sadržaja. Nakon selektora idu vitičaste zagrade koje predstavljaju deklaracijski blok te se unutar tog bloka definiraju različita svojstva te njihove vrijednosti. Pritom je važno naglasiti da se nakon svake deklaracije mora staviti znak točka zarez. Deklaracija je napisana u obliku: ***svojstvo: vrijednost;***. Na slici 2.3. prikazana je sintaksa CSS jezika a ostali pojmovi su objašnjeni u [14, 2].

```
h1 {
  font-size: 20px;
  color: red;
}

/* h1 označava selektor dok su font-size i color svojstva kojima se
pridružuju vrijednosti unutar deklaracijskog bloka */
```

Sl. 2.3. Prikaz sintakse CSS-a

2.4. Bootstrap

Bootstrap je mrežni programski okvir koji posjeduje različit skup alata koji su kreirani u CSS-u i Javascriptu te omogućava brzu i kvalitetnu izradu front-end internet stranica. Razvio ga je Twitter. Boostrapove prednosti su da olakšava i ubrzava proces izrade internet stranice odnosno posjeduje alate koji omogućavaju brzi razvoj korisničkog sučelja, izradu internet

formi te se aplikacija razvijena Boostrapom prikazuje na svim preglednicima jednako, tj. Boostrap podržava sve internet preglednike. Boostrap koristi pristup "mobile-first" što bi značilo da je razvijen kako bi se prvo radila internet stranica za mobilne uređaje, a tek nakon toga za stolna računala. Glavni nedostak Boostrapa je što se učitava svaki put kada se internet stranica učita kod korisnika. Internet aplikacija je sporija jer sadrži u sebi veliki broj stilova, a to zapravo ne pogoduje SEO (eng. Search Engine Optimization) pravilima. Još jedna mana Boostrapa je da svaka internet stranica razvijena s ovim mrežnim programskim okvirom podsjeća jedna na drugu. Glavni selektor kod Boostrapa su klase u kojima su opisani različiti stilovi koji se mogu koristiti. Na slici 2.4. je prikazano kako se Boostrap uključuje u HTML datoteku i kako se koristi a ostali navedeni pojmovi su objašnjeni u [2, 15].

```
# boostrap se uvijek treba uključiti u HTML ovo je način uključivanje pomoću
CDN-a

<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.cs
s" integrity="sha384-
9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZl5MYYxFfc+NcPb1dKGj7Sk"
crossorigin="anonymous">

# klasa text-danger služi kako bi se tekst prikazao crvenom bojom
<h1 class="text-danger"> Ovo je tekst prikazan crvenom bojom </h1>

# klasa text-primary služi kako bi se tekst prikazao plavom bojom
<h1 class="text-primary"> Ovo je tekst prikazan plavom bojom </h1>
```

Sl. 2.4. Primjer uporabe Boostrap klasa za različite stilove

2.5. Git i GitHub

Git je sustav kontrole verzije koju je pokrenuo Linus Trovalds. Linus Trovalds je stvorio i Linux. Git je jako sličan ostalim sustavima za kontrolu verzije – Subversion, CVS i Mercurial. Kada programeri razvijaju i stvaraju aplikacije, kodovi se stalno mijenjaju te se objavljuju nove inačice do i nakon prvog službenog izdanja. Sustavi za nadzor verzije zadržavaju izmjene te ih čuvaju u spremištu. To omogućuje programerima da lagano surađuju jer mogu preuzeti verziju aplikacije, napraviti izmjene i staviti najnoviju verziju aplikacije. Svaki programer koji radi na projektu ove izmjene može vidjeti i izmijeniti.

GitHub je internetsko mjesto koje služi kao skladište odnosno mjesto gdje su pohranjenje sve datoteke za određeni projekt. Svaki projekt ima vlastiti repozitorij te mu se može pristupiti jedinstvenim URL-om. Naravno GitHub koristi Git sustav za kontrolu verzije. Na slici 2.5.

prikazano je kako se može napraviti repozitorij te njegovo stavljanje na Github, ali pod uvjetom da imamo već napravljen repozitorij na Githubu. Svi navedeni pojmovi objašnjeni i prikazani su u [2, 16] .

```
# Postupak izrade tekstualne datoteke i stavljanja teksta u nju

$ touch datoteka.txt
$ echo "Ovaj tekst ide u datoteku" >> datoteka.txt

# inicijalizacija git sustava za kontrolu verzije i njegovo korištenje

$ git init
$ echo "Ovo je izmjenjen tekst" >> datoteka.txt
$ git add datoteka.txt
$ git commit -m "Izmjenjen sadržaj datoteke datoteka.txt"
$ git push origin master
```

Sl. 2.5. Primjer kreiranja jednostavnog sustava za kontrolu verzije pomoću Git komandi i stavljanje na GitHub

2.6. Heroku

Heroku je platforma na koju se hostaju internet aplikacije. Heroku se razlikuje od drugih platformi koje omogućavaju servis hostanja zbog toga što već posjeduje potrebne konfiguracije kao što je na primjer podrška za Git čime je proces postavljanja aplikacije na internet iznimno lagan. Ovaj servis na početku je podržavao samo programski jezik Ruby, a danas podržava različite programske jezike kao što su Go, PHP, Scalu, Javu, Node.js i mnoge druge. Heroku omogućava jednostavno razvijanje i skaliranje aplikacija. Koristi Postgres bazu podataka, omogućava zaštitu i veliku pristupačnost baze kao što je prikazano u [2, 17, 18].

2.7. Sass

Sass predstavlja novu sintaksu za CSS koja ima za cilj pojednostaviti nadzor i upravljanje nad velikim CSS datotekama. Navedeno se postiže upotrebom preprocesora koji zapravo prevodi novu unapređenu sintaksu (.sass ili .scss) u klasični CSS koji internet preglednik jedino razumije. Sass je napravljen jer internet stranice brzo rastu i postaju kompleksnije, a prilikom pisanja velike količine CSS-a kod postaje slabije razumljiv i čitljiv. Naspram starog CSS-a, Sass je jezik koji sadrži stvari kao što su: ugnježđivanje, funkcije, miksini, varijable, petlje te uslovnu logiku. DRY princip je osnovni princip na kojem je zasnovan Sass jer ponavljanjem koda dobivamo jako nečitljiv CSS. Velika prednost je da se CSS selektori mogu ugnijezditi na isti način kao i HTML. Na slikama 2.6. i 2.7. prikazana je usporedba

obične CSS sintakse i napredne SCSS sintakse. Svi navedeni pojmovi su prikazani i objašnjeni u [2, 19, 20].

```
# CSS sintaksa

nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}

nav li {
  display: inline-block
}

nav a {
  display: block;
  padding: 6px 12px;
  text-decoration: none;
}
```

Sl. 2.6. Primjer CSS sintakse

```
# SCSS sintaksa – primjena ugnježđivanja

nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }
  li {
    display: inline-block;
  }
  a {
    display: block;
    padding: 6px 12px;
    text-decoration: none;
  }
}
```

Sl. 2.7. Primjer SCSS sintakse

2.8. ActiveJob

Active Job je modul za kreiranje poslova i pripremanje poslova koji se izvršavaju na različitim sustavima za raspoređivanje poslova. Sve što se može podijeliti na male jedinice rada i pokrenuti paralelno predstavlja posao kao što je npr. slanje emaila. Glavna svrha je osigurati da sve Rails aplikacije imaju infrastrukturu poslova na jednom mjestu. Zatim možemo uključiti u gemfile različite sustave za raspoređivanje poslova kao što su npr.

Delayed Job, Resque i Sidekiq. Prebacivanje sa sustava na sustav je jednostavano jer se poslovi koji su napisani ne moraju prepravljati. Rails dolazi s načinom obavljanja poslova koji ima problem, a to je ako se uređaj resetira svi poslovi koji su u redu izvršavanje (eng. queue) se gube. Ovakav uobičajen način je dobar za male poslove i prilikom razvijanja aplikacije, ali u produkciji se treba koristiti jedan od sustava za raspoređivanje poslova kao što je prikazano u [3].

Active Job sadrži generator poslova koji automatski kreira posao u ***apps/jobs*** direktoriju te u direktoriju ***test/jobs*** kreira fajl za kreiranje testova za iste poslove. Isto tako generator omogućava da se kreira posao koji će se izvršavati u specifičnom redu za izvršavanje (eng. queue). Na slici 2.8. prikazana je naredba za automatsko generiranje datoteka kako bi se napravio jednostavan posao kao što je prikazano u. Prema Rails dokumentaciji [3] je napravljen postupak za generiranje i isprobavanje određenog posla.

```
# naredba za automatsko generiranje posla
$ rails generate job guests_cleanup

# naredba za generiranje posla koji će se izvršavati u specifičnom redu
$ rails generate job guests_cleanup –queue urgent
```

Sl. 2.8. Primjer naredbe za automasko generiranje poslova

Naravno posao se može i implementirati bez korištenja Railsovog generatora. U direktoriju ***app/jobs*** napravi se datoteka. Treba voditi računa da klasa treba naslijediti sve iz klase ***ApplicationJob***. Posao bi trebao izgledati kao što je prikazano na slici 2.9.

```
class GuestsCleanupJob < ApplicationJob
  queue_as :default

  def perform(*guests)
    # Napravi nešto
  end
end
```

Sl. 2.9. Primjer koda ako želimo ručno implementirati posao

Treba naglasiti da metoda ***perform*** može imati proizvoljni broj argumenata. Posao se može postaviti u red na različite načine, a na slici 2.10. prikazani su neki od najčešćih korištenih.

```
# Posao će se obaviti kada sustav bude prvom prilikom slobodan
  GuestsCleanupJob.perform_later guest
# Posao će se obaviti sljedeći dan u podne
  GuestsCleanupJob.set(wait_until: Date.tomorrow.noon).perform_later(guest)
# Posao će se obaviti točno za sedam dana
  GuestsCleanupJob.set(wait: 1.week).perform_later(guest)
```

Sl. 2.10. Primjer koda za postavljanje poslova u red

Metode kao što su ***perform_now*** i ***perform_later*** će se obaviti u pozadini tako da se metodi može predati onoliko argumenata koliko je definirano.

Kako bi se poslovi postavili u red i izvršili treba izabrati sustav za raspoređivanje poslova i postaviti ga. Na slici 2.11. prikazan je kod za odabir sustava za obradu poslova.

```
# ove linije koda treba staviti u direktorij config/application.rb
# pri tom treba voditi računa da adapter koji se koristi je prisutan u
gemfile

module TvojaAplikacija
  class Application < Rails::Application
    config.active_job.queue_adapter = :sidekiq # Sidekiq se koristi
  end
end
```

Sl. 2.11. Primjer koda za odabir sustava za obradu pozadinskih poslova

Može se konfigurirati i da određeni posao koristi određeni sustav za raspoređivanje poslova kao što je prikazano na slici 2.12.

```
# ovaj posao koristi resque adapter koji overajda konfiguraciju aplikacije

class GuestsCleanupJob < ApplicationJob
  self.queue_adapter = :resque
end
```

Sl. 2.12. Primjer koda za odabir sustava za pojedini posao

Većina adaptera podržava više redova. S Active Job - om može se napraviti da se određeni poslovi izvršavaju prije ili poslije, ovisno o važnosti. Na slici 2.13. prikazan je kod koji znači da se posao postavlja u red s najmanjim prioritetom.

```
class GuestsCleanupJob < ApplicationJob
  queue_as :low_priority # ovo bi značilo da će ovaj posao imati najmanji
                           prioritet i izvršavati se u redu low_priority
end
```

Sl. 2.13. Primjer koda za postavljanje posla u red s najmanjim prioritetom

Naredba s kojom se može odrediti u kojem će se redu izvršiti određeni posao prikazana je na slici 2.14.

```
MyJob.set(queue: :another_queue).perform_later(record)
```

Sl. 2.14. Primjer koda za određivanje reda u kojem će se specifični posao izvršiti

Treba voditi računa da se adapter konfigurira kako bi "oslušavao" određeni red izvršavanja (eng. queue). Neki sustavi za raspoređivanje poslova traže da se specifiraju ovakve stvari dok drugi ne [3].

Active Job mrežni programerski okvir ima mogućnost da izvrši određenu logiku u životnom ciklusu posla. Na slici 2.15. prikazan je kod za izvršavanje određene logike prije i nakon izvršenog posla.

```
class GuestsCleanupJob < ApplicationJob
  queue_as :default

  around_perform :around_cleanup

  def perform
    # Napravi nešto kasnije
  end

  private
    def around_cleanup
      # Napravi nešto prije izvršenja posla
      yield
      # Napravi nešto poslije izvršenog posla
    end
end
```

Sl. 2.15. Primjer koda za izvršavanje određene logike prije i nakon izvršenog posla

Neki od mogućih poziva u životnom ciklusu posla su: **`before_enqueue`** (prije postavljanja u red izvršavanja), **`around_enqueue`** (prije i poslije postavljanja u red

izvršavanja), **`after_enqueue`** (poslije postavljanja u red izvršavanja), **`before_perform`** (prije izvršavanja posla), **`around_perform`** (prije i poslije izvršavanja posla) te **`after_perform`** (nakon izvršavanja posla).

Posao koji je najzastupljeniji u modernim internet aplikacijama je slanje elektroničke pošte. Active Job je integriran za Action Mailerom koji dozvoljava da se elektronička pošta šalje preko aplikacije koristeći mailer klase i poglede. U Railsu se elektronička pošta može poslati i bez Active Job okvira , ali se u praksi koristi s Active Jobom zbog brzine aplikacije.

Na slici 2.16. prikazan je kod za korištenje opcije slanja elektroničke pošte s Action Jobom i bez njega [3].

```
# Ako se želi elektronička pošta poslati sada koristi se #deliver_now
UserMailer.welcome(@user).deliver_now

# Ako se želi elektronička pošta poslati preko Active Job-a koristi se
  #deliver_later
UserMailer.welcome(@user).deliver_later
```

Sl. 2.16. Primjer koda za korištenje opcije slanja elektroničke pošte s Action Job-om i bez njega

Active Job podržava sljedeće tipove podataka kao argumente: osnovni tipovi podataka (stringovi, cijeli i decimalni brojevi, boolean), simboli, datum (Date), vrijeme (Time), datum i vrijeme (DateTime), hashevi te polja. Isto tako Active Job pruža način kako da se hvataju iznimke prilikom izvršavanja određenog posla. Na slici 2.17. prikazan je kod za hvatanje iznimke.

```
class GuestsCleanupJob < ApplicationJob
  queue_as :default

  rescue_from(ActiveRecord::RecordNotFound) do |exception|
    # Napravi nešto ako se pronađe iznimka
  end

  def perform
    # Napravi nešto kasnije
  end
end
```

Sl. 2.17. Primjer koda za hvatanje iznimaka

Postoji mogućnost i ponovnog izvršavanja posla ukoliko dođe do iznimke kao i njegovo ubijanje. Na slici 2.18. prikazan je kod za ovakvu funkcionalnost.

```
class RemoteServiceJob < ApplicationJob
  retry_on CustomAppException # 3s se čeka, 5 pokušaja

  discard_on ActiveJob::DeserializationError

  def perform(*args)
    # Kod koji bi mogao podići iznimku
  end
end
```

Sl. 2.18. Primjer koda izvršavanje poslova ako dođe do iznimke kao i njegovo ubijanje

Ako koristimo generator za kreiranje poslova odmah će se kreirati i test s kojim možemo testirati naše poslove. Testovi se koriste kako bi se prilikom pisanja aplikacija odmah testirale aplikacije, ali ne ručno već programski. Kako bi bilo na jednoj velikoj internet stranici kao što je "24 sata" provjeriti da li svaki link vodi na dobru stranicu. Bio bi to jako dug i iscrpljujuć posao. Upravo zato mi imamo testove koje napišemo te simuliramo zapravo naše ručno testiranje. Na slici 2.19. primjer je jednostavnog testa koji provjerava i tvrdi da li je posao napravio šta je trebao.

```
require 'test_helper'

class BillingJobTest < ActiveJob::TestCase
  test 'that account is charged' do
    BillingJob.perform_now(account, product)
    assert account.reload.charged_for?(product)
  end
end
```

Sl. 2.19. Primjer jednostavnog testa za provjeru posla

Posao na slici 2.19. provjerava da li je korisnički račun naplaćen ili nije.

2.9. Sustavi za raspoređivanje poslova

Sustavi za raspoređivanje poslova koji postoje su : BackBurner, Delayed Job, Que, queue_classic, Resque, Sidekiq, Sneakers, Sucker Punch, Active Job Async Job, Active Job Inline [21].

Backburner je sustav za raspoređivanje poslova koji radi s beanstalk - om te može obraditi veliki volumen poslova. Beanstalk je brzi red koji je implementiran u C jeziku. Backburner najviše se koristi sa Sinatrom, Padrinom i Railsom. Backburner sprema sve poslove kao JSON poruku. Beanstalk može obraditi oko tisuću poslova po sekundi. Kako bi se ovaj sustav koristio potrebna je instalacija beanstalka pa zatim backburner - a te ubacivanje gema

u gemfile u Rails aplikaciji [22].

Delayed Job je sustav za raspoređivanje poslova koji enkapsulira uobičajene obrasce za asinkrono izvršavanje dužih zadataka u pozadini. Direktno je izvučen iz Shopify - a gdje je tablica poslova odgovorna za mnoštvo zadataka među kojima su : promjena veličine slike, http preuzimanja, slanje mnoštvo članaka i sl kao što je prikazano u [23].

Que je sustav za raspoređivanje poslova koji poboljšava pouzdanost u aplikaciji tako što štiti poslove prema ACID standardu. ACID (atomicity, consistency, isolation, durability) je set svojstava za bazu podataka namijenjen da garantira valjanost podataka usprkos erorima, nestanku napona napajanja i sličnih stvari. Que je sustav koji se koristi za Ruby i PostgreSQL. Que sustav koristi sigurnosne brave koji daju određene prednosti naspram ostalih sustava za manipulaciju relacijskih baza podataka kao što je na primjer sigurnost gdje ako Ruby proces ugine, posao koji se izvršava neće biti izgubljen ili ostavljen u zaključanom stanju nego posao postane dostupan za drugog *workera*. Izraz *worker* predstavlja zapravo jedan dio sustava kao što je procesor i služi za obradu i izvršavanje poslova u aplikaciji kao što je prikazano u [24].

Jedan od sustava za raspoređivanje poslova je i queue_classic sustav. Ovaj sustav koristi značajke poput asinkronog povlačenja poslova iz spremnika. Baza podataka održava brave i nema ovisnosti (eng. dependency). Ovaj sustav samo zahtjeva PostgresSQL bazu podataka. Velika prednost sustava je sposobnost da se posao stavi u red u fazi transakcije, osiguravajući time da se stvari naprave tek kada se izmjene izvrše kao što je objašnjeno[25].

Resque je sustav za raspoređivanje poslova koji radi s Redisom. Pozadinski posao može biti Ruby klasa ili modul koji odgovora na metodu perform. Resque je inspiriran Delayed Job sustavom i sadrži tri dijela: Ruby biblioteku za kreiranje i procesiranje poslova, Rails zadatak koji pokreće *workera* i procesira posao te Rails aplikaciju za nadgledanje redova, poslova i *workera* kao što je objašnjeno u [26].

Sneakers je visoko performatni sustav za raspoređivanje pozadinskih poslova koji je baziran na RabbitMQ. RabbitMQ je softver koji predstavlja posrednika u razmjeni poruka te on omogućava lakše povezivanje i skaliranje aplikacija. Sneakers koristi hibridni procesno - nitni model gdje su mnogi procesi kreirani i mnogo je niti korišteno za procese kao što je prikazano u [27].

Sucker Punch je jednoprocesna biblioteka za asinkronu obradu poslova koja koristi celluloid mrežni programerski okvir. Celluloid se koristi za kreiranje asinkronih i višenitnih Ruby programa koristeći objektno orijentirane principe. Sucker Punch je preporučen za poslove koji su brzi i jednostavni za izvršiti kao što je slanje elektroničke pošte kao što je objašnjeno u [28].

Active Job Async adapter je zadani adapter koji se koristi za raspoređivanje poslova u Rails-u. Ovaj adapter je dobro prilagođen i koristi se tijekom razvijanja aplikacije te testiranja iste zbog razloga što ne treba vanjsku infrastrukturu. Za produkciju nije pogodan te se preporučuje neki od gore navedenih sustava. Kako bi se koristio, ovaj adapter koristi ključnu riječ ***async***kao što je prikazano u [21].

Active Job Inline je adapter koji se koristi ako želimo da se posao izvrši u istom trenutku nakon zahtjeva. Kako bi se koristio, ovaj adapter koristi ključnu riječ ***inline*** kao što je prikazano u [21].

Sidekiq je sustav za raspoređivanje poslova koji je efikasan za Ruby poslove. Sidekiq koristi niti kako bi rukovao s više poslova u isto vrijeme u istom procesu. Nije potreban Rails, ali je integriran sa Railsom kako bi učinio izradu pozadinskih poslova vrlo jednostavnim. Sidekiq je najbrži sustav za raspoređivanje poslova u Rubyu te je čak do dvadeset puta brži od konkurencijem tj. gore navedenih sustava. Sidekiq koristi Redis za čitanje poslove iz njega te koristi FIFO (eng. First In First Out) model za podatke s ciljem procesiranja posla kao što je prikazano u [30,32]. Najnovija verzija 6.0.2 može procesirati sto tisuća poslova za samo 14 sekundi uz kašnjenje od 3 milisekunde. To bi značilo da se po sekundi može obaviti oko 7100 poslova. Smeće napravljeno za kreiranje deset tisuća poslova je oko 150 megabajta, a u starijim verzijama je bilo i do 1257 megabajta. Na tablici 2.1. prikazane su sve perfomanse starijih verzija.

Tablica. 2.1. Tablica performansi Sidekiq-a Izvor [32]

Version	Latency	Garbage created for 10k jobs	Time to process 100k jobs	Throughput	Ruby
Sidekiq 6.0.2	3 ms	156 MB	14.0 sec	**7100 jobs/sec**	MRI 2.6.3
Sidekiq 6.0.0	3 ms	156 MB	19 sec	5200 jobs/sec	MRI 2.6.3
Sidekiq 4.0.0	10 ms	151 MB	22 sec	4500 jobs/sec	
Sidekiq 3.5.1	22 ms	1257 MB	125 sec	800 jobs/sec	
Resque 1.25.2	-	-	420 sec	240 jobs/sec	
DelayedJob 4.1.1	-	-	465 sec	215 jobs/sec	

U usporedbi sa Sidekiqom Resque i Delayed Job sustavi nisu ni približno brzi kao što je prikazano u [32].

Na tablici 2.2. prikazana je usporedba sustava za raspoređivanje poslova.

Tablica. 2.2. Tablica usporedbe različitih sustava za raspoređivanje poslova - Izvor [21]

	Async	Queues	Delayed	Priorities	Timeout	Retries
Backburner	Yes	Yes	Yes	Yes	Job	Global
Delayed Job	Yes	Yes	Yes	Job	Global	Global
Que	Yes	Yes	Yes	Job	No	Job
queue_classic	Yes	Yes	Yes*	No	No	No
Resque	Yes	Yes	Yes (Gem)	Queue	Global	Yes
Sidekiq	Yes	Yes	Yes	Queue	No	Job
Sneakers	Yes	Yes	No	Queue	Queue	No
Sucker Punch	Yes	Yes	Yes	No	No	No
Active Job Async	Yes	Yes	Yes	No	No	No
Active Job Inline	No	Yes	N/A	N/A	N/A	N/A

Kao što možemo vidjeti na tablici 2.2. svi sustavi osim Active Job Inline podržavaju asinkronost to bi značilo da Active Job Inline izvršava poslove u istom procesu dok ostali imaju mogućnost izvršavanja poslova na način da se ne blokiraju. Svi sustavi mogu postaviti

poslove u kojem će se redu izvršavati pomoću ***queue_as*** naredbe. Active Job Inline adapater ne koristi redove u izvršavanju poslova, Sneakers ne može koristiti metodu ***perform_later*** dok ostali sustavi mogu. To bi značilo da neki sustavi mogu izvršavati poslove kasnije, a neki ne. Što se tiče prioriteta neki sustavi imaju mogućnost da postave prioritet na razini poslova a neki imaju da postave prioritet na razini redova. Backburner sustav može postaviti prioritet i na jednoj i na drugoj razini dok sustavi kao što su Sucker Punch, queue_classic i Active Job Async nemaju mogućnost postavljanja prioriteta. Neki sustavi imaju mogućnost da zaustave posao nakon određenog vremena bilo to na razini posla ili reda. Delayed Job i Resque imaju mogućnost da postave ovaj način rada i na razini posla i na razini reda dok sustav Backburner ima samo mogućnost da postavi ovaj način rada na razini posla, a sustav Sucker Punch na razini reda. Ostali sustavi nemaju ovu mogućnost. Sustavi imaju opciju da se posao ponovno pokuša izvršiti ako je došlo do problema. BackBurner, Delayed job i Resque mogu postaviti ovu opciju i na razini reda i na razini posla dok Que i Sidekiq mogu postaviti samo na razini posla kao što je prikazano u [21].

2.10. Redis

Redis (eng. Remote Dictionary Server) je nerelacijska baza podataka koja je veoma brza i koja sprema ključne vrijednost u pet potencijalnih struktura podataka, a te strukture podataka su: stringovi, liste , skupovi, poredani skupovi i hashevi. Salvatore Sanfilippo je razvio Redis u najvećoj mjeri. Redis je napisan u ANSI C jeziku i koristi se u mnogim POSIX sistemima kao što je Linux. Naziv POSIX predstavlja zapravo skupina IEEE standarda te ti standardi definiraju sučelje za programiranje određenog softvera s različitim verzijama UNIX operacijskih sustava. Linux i OS X su dva operacijska sustava gdje je Redis najviše napravljen i testiran te se preporučuje preporuka korištenje Linux operacijskog sustava za implementaciju. Podaci se spremaju u paru vrijednost (eng. value) i ključ (eng. key). Treba naglasiti da je ova baza podataka otvorena za sve programere te je besplatna. Bitna svojstva u kojima se Redis ističe su brzina unosa i dohvaćanje podataka, brza i sigurna promjena unesenih podataka, te naravno podržavanje velikog broja programskih jezika. Nedostaci Redisa su sljedeći: komunikacija s Redis serverom ide liniju po liniju, nemogućnost zahtjeva nad vrijednostima objekata (to bi značilo da se ne može prići n-torki gdje je na primjer vrijednost jednaka zadanom broju). Zbog ovakvih problema Redis se povezuje putem raznih programskih jezika i platformi s ciljem dostavljanja mnogobrojnih upita prema serveru u istom trenutku. Redis koriste velike internet stranice kao štu su: Twitter, Instagram, GitHub, StackOverflow i mnoge druge. Redisom ove internet stranice spremaju podatke o

korisnicima odnosno o njihovim sesijama i upravljaju predmemorijom aplikacije kao što je prikazano u [31, 33, 34]. Na slici 2.22. je prikazana instalacija Redisa.

```
$ sudo apt update
$ sudo apt install redis-server
$ sudo code /etc/redis/redis.conf
```

Sl. 2.20. Ažuriranje sustava i instaliranje redis - a

Na slici 2.20. u prvom redu smo komandom ažurirali ***apt*** paket dok smo drugom komandom instalirali Redis. Trećom komandom ili naredbom otvaramo određenim tekst editorom konfiguraciju Redisa u ovom slučaju ***redis.conf*** datoteka će biti otvoren editorom Visual Studio Code [31].

Prema Redis dokmentaciji [31], budući da se koristi Ubuntu operacijski sustav, u konfiguracijskog datoteci potrebno je liniju koda:

```
supervised no
```

Zamjeniti s ispod navedenom linjom koda:

```
supervised systemd
```

Zatim se restartira Redisov servis kako bi se primjenile promjene u konfiguraciji koje smo napravili:

```
$ sudo systemctl restart redis.service
```

U samo par koraka Redis je uspješno instaliran i konfiguriran te se može vrtiti na računalu. Ako je novi softver tek instaliran bilo bi dobro napraviti testove pomoću kojih se možemo uvjeriti u funkcioniranje Redisa.

Ovom komandom se provjerava da li se Redis servis izvršava:

```
$ sudo systemctl status redis
```

Ovom komandom se provjerava da li Redis normalno funkcionira te se povezuje s terminalom Redis - a:

```
$ redis-cli
```

Pomoću korištenja ping komande provjerava se konekcija:

```
ip-address> ping
```

Ip adresa se može očitati iz izlaza koja je dala ova naredba:

```
$ sudo systemctl status redis
```

Ako dobijemo izlaz PONG to znači da je serverska konekcija živa. Sljedećim komandama se provjerava da li se može postaviti ključ:

```
ip-address> set test "Ispravno radi"
```

Ako dobijemo izlaz "OK" znači da je sve u redu. Zatim se radi provjera tako da dobijemo vrijednost koju smo postavili pomoću komande:

```
ip-address> get test
```

Ako sve normalno radi dobit ćemo izlaz "Ispravno radi". Ako se dohvati vrijednost treba izaći iz Redisa te se vratiti natrag u ljusku (eng. bash) naredbom:

```
ip-address> exit
```

Zatim se restartira Redis. Kako bi provjerili da li sve normalno radi treba se opet povezati s Redisovim terminalom i probati dohvatiti vrijednost. Ako sve uspiješno radi Redis normalno funkcionira.

Ako kojim slučajem postoji nemogućnost povezivanja s lokalnim hostom otvori se ***redis.conf*** datoteka te se locira linija u kojoj piše ključna riječ ***bind*** te povezati svoju lokalnu ip adresu s Redisom:

```
bind ip-address
```

Ne treba zaboraviti da nakon promjene konfiguracije Redis se uvijek treba restartirati kako bi se promjene uspiješno primijenile.

Bitne stvari koje Redis još pruža su mogućnost konfiguracije naredba koje su jako opasne za korištenje te postavljenje nove lozinke. Ove dvije stavke spadaju u sigurnosne stavke Redis sustava te čine sustav manje ranjivim na napade od strane hakera [31].

3. PRAKTIČNI DIO

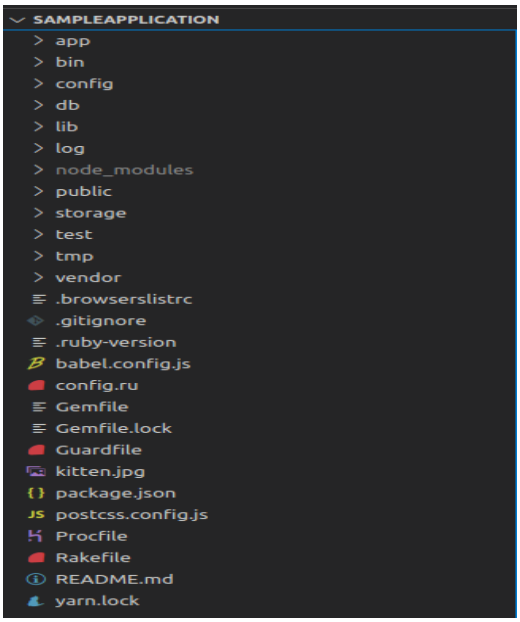
U praktičnom djelu je napravljena aplikacija koja ima slične funkcije Twitterum, a napravljena je u skladu s udžbenikom "Ruby on Rails Tutorial Sixth Edition" koji je napisao Michael Hartl [2]. U drugom djelu aplikacije je na temelju napravljene aplikacije integriran Active Job modul te je pokazan kako se koristi sustav za raspoređivanje poslova Sidekiq s Redisom. Za ovu aplikaciju je korišten Heroku servis kako bi se aplikacija mogla postaviti u produkciju kao što je objašnjeno u.

3.1. Gemfile i Bundler

Bitni pojmovi koji se trebaju znati su Gemfile i Bundler. Gemfile je tekstualna datoteka u kojoj se definiraju svi dodatni servisi i mogućnosti koje su već napravljene od strane drugih programera te koje mogu biti korištene u bilo kojoj aplikaciji. Pomoću bundlera se ovi gotovi servisi instaliraju te se mogu razvrstati koji servisi se koriste u produkciji, koji u izradi aplikacije, a koji pri testiranju aplikacije.

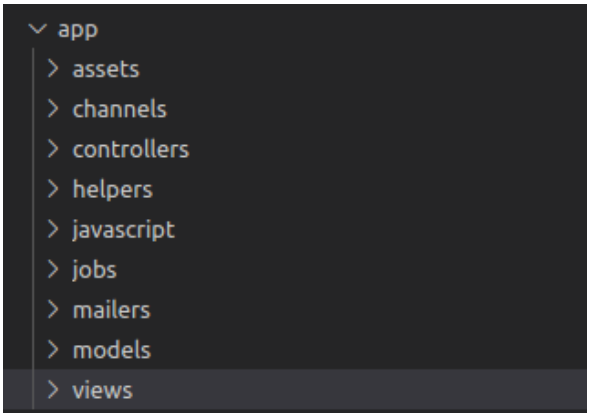
3.2. Struktura projekta

Na slici 3.1. je prikazana struktura datoteka i direktorija kada se kreira Ruby on Rails aplikacija. Bitni direktoriji su ***app*** koji služi za aplikaciju to jest tamo se definiraju pogledi, modeli i kontroleri i sve vezane funkcionalnosti koje se direktno tiču aplikacije. Direktorij ***db*** je zaslužan za bazu podataka te manipulairanje istim podacima dok je direktorij ***test*** zaslužan za razne testove koji se mogu napisati za Ruby on Rails aplikaciju kao što je na primjer provjera ruta vode li gdje bi trebale.



Sl. 3.1. Prikaz strukture Ruby on Rails aplikacije

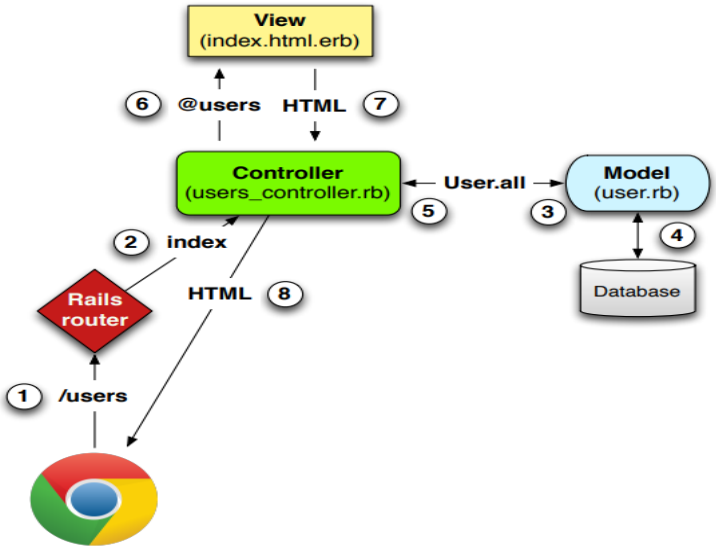
Na slici 3.2. prikazan je sadržaj ***app*** direktorija gdje se nalaze bitni direktoriji kao što su ***views***, ***models***, ***assets***, ***javascript***, ***controllers*** i ***jobs***. Svi modeli se nalaze u direktoriju ***models*** dok se svi pogledi nalaze u direktoriju ***views*** koji su napisani u HTML-u s ugrađenim Ruby programskim jezikom, a kontroleri se nalaze u direktoriju ***controllers***. U ***assets*** direktoriju se nalaze sve vanjske stvari kao što su na primjer slike, videozapisi i stylesheet u kojemu se nalaze .scss i .css datoteke koje se koriste za dizajn stranice. ***Javascript*** direktorij je odgovoran za javascript kod te sve fron-end stvari koje se mogu raditi s navedenim programskim jezikom. Direktorij ***jobs*** je zaslužan za poslove koji se definiraju unutar navedenog direktorija, a u ***helpers*** direktoriju se nalaze stvari kao što su metode s određenom funkcionalnostima koje koristimo u kontrolerima.



Sl. 3.2. Prikaz direktorija u "app" direktoriju

3.3. MVC arhitektura u Rails okruženju

Na slici 3.3. prikazana je jednostavna MVC arhitektura u Rails okruženju. Naime, prvi korak je da internetski preglednik izdaje zahtjev za ***/users*** URL. Rails ruter preusmjerava ***/users*** na indeks akciju u ***User*** kontroleru. Indeks akcija pita model User da se prikupe svi korisnici (***Users.all***). Model ***User*** izvlači sve korisnike iz baze podataka. Zatim baza podataka vraća sve korisnike kontroleru. Kontroler zatim stavlja sve korisnike u varijablu koja može biti ,na primjer ***@users*** i zatim daje tu varijablnu indeks pogledu. Pogled koristi ugrađeni Ruby kako bi generirao stranicu u HTML-u. Zadnji korak je da kontroler vraća HTML natrag internet pregledniku. I tako zapravo MVC arhitektura radi u Rails okruženju.



Sl. 3.3. Prikaz MVC arhitekture u Rails okruženju Izvor[2]

3.4. REST arhitektura u Rails okruženju

Na slici 3.4. prikazane su rute koje su u skladu s REST arhitekturom. Ovo je prikaz ruta kojemu je resurs zapravo ***Users***. REST arhitektura znači da komponente u aplikaciji koje su modelirane kao resurs moraju sadržavati funkcionalnost kao što su kreiranje i brisanje resursa te naravno njihovo dohvaćanje i ažuriranje. Ove funkcionalnosti su opisane HTTP zahtjevima GET, POST, PATCH i DELETE. Jedinstveni naziv za ove četiri operacije je CRUD (Create, Read, Update, Delete).

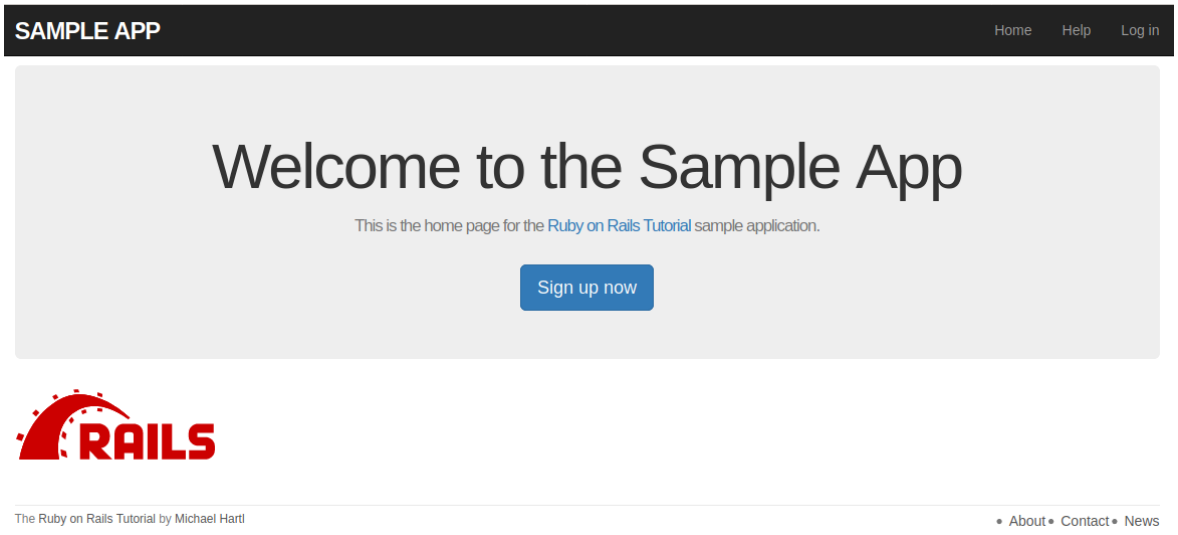
HTTP request	URL	Action	Purpose
GET	/users	**index**	page to list all users
GET	/users/1	**show**	page to show user with id **1**
GET	/users/new	**new**	page to make a new user
POST	/users	**create**	create a new user
GET	/users/1/edit	**edit**	page to edit user with id **1**
PATCH	/users/1	**update**	update user with id **1**
DELETE	/users/1	**destroy**	delete user with id **1**

Sl. 3.4. Prikaz ruta koje su u skladu s REST arhitekturom Izvor[2]

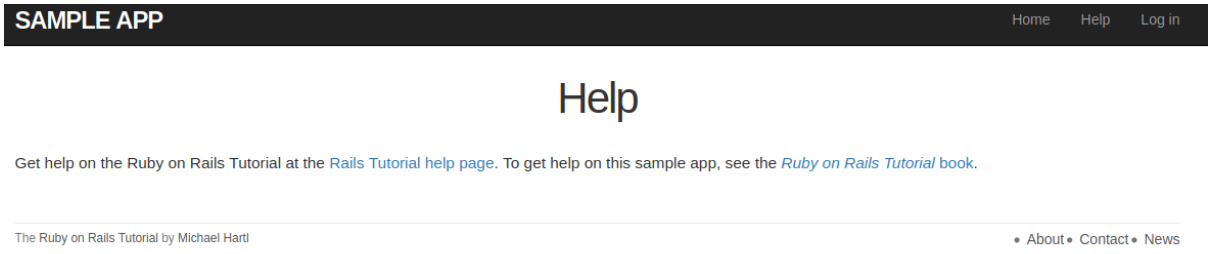
3.5. Opis aplikacije

Početna stranica aplikacije je prikazana na slici 3.5. S početne stranice logira se klikom na tipku "Log in" ako je korisnički račun kreiran ili na tipku "Sign up now" ako korisnički račun nije kreiran. Prikazane su statične stranice kao što su Help, About i Contact te do njih se

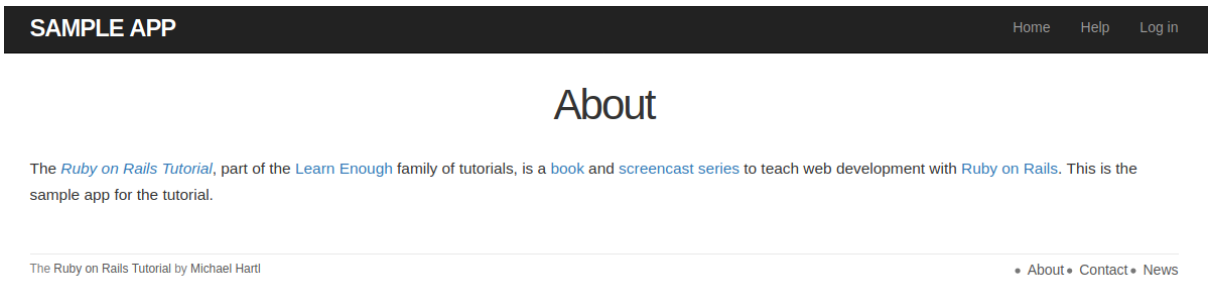
dolazi klikom na gornju ili donju navigaciju u ovisnosti gdje se nalazi link do njih. Na slikama 3.6., 3.7., i 3.8. prikazane su ostale statične stranice. Za ove stranice je zaslužan poseban kontroler nazvan ***static_pages_controller*** te posebni pogledi u ***views/static_pages*** direktoriju.



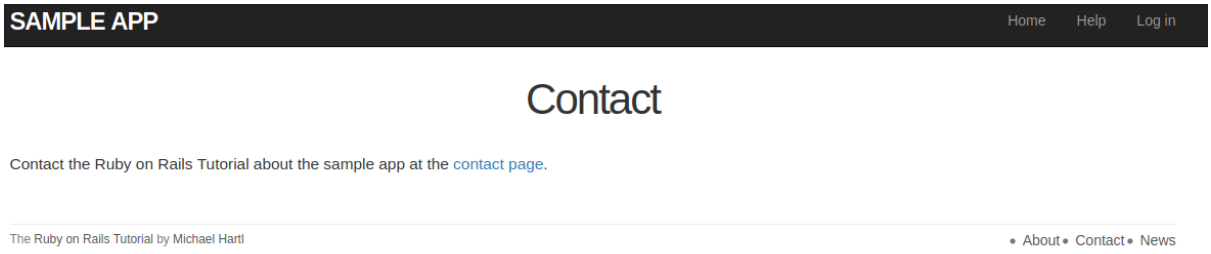
Sl. 3.5. Početna stranica aplikacije



Sl. 3.6. Help stranica

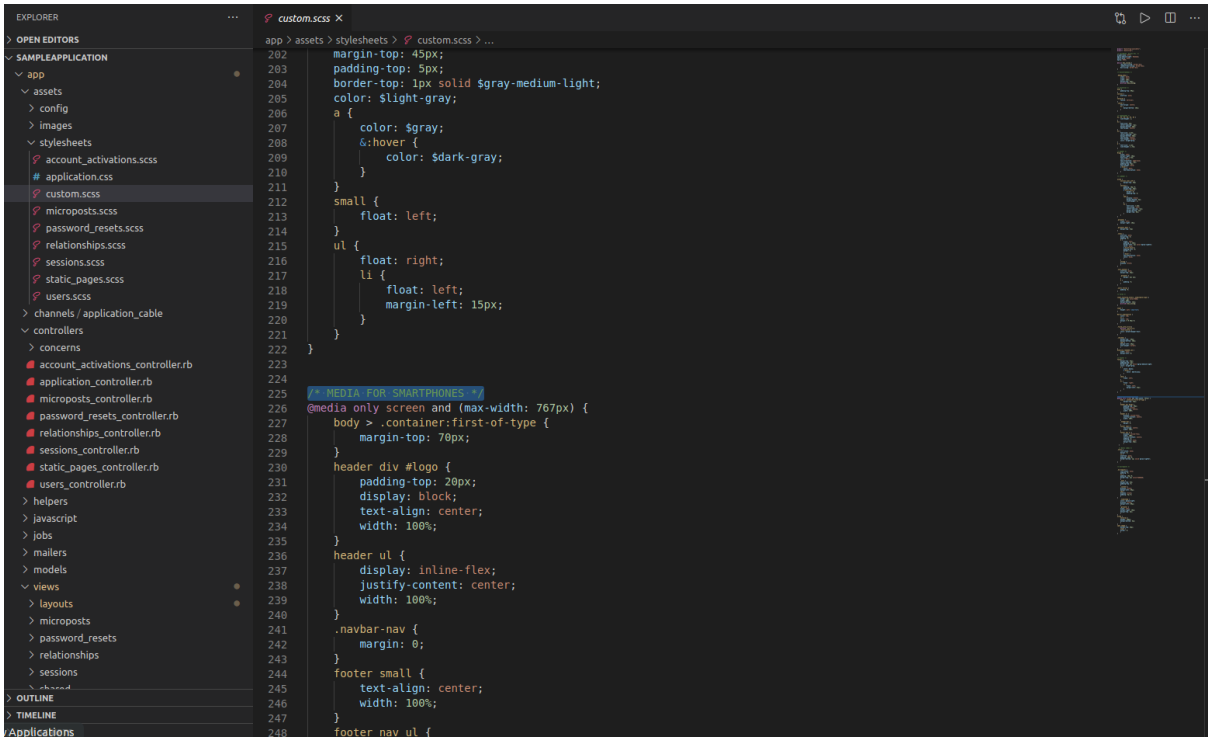


Sl. 3.7. About stranica



Sl. 3.8. Contact stranica

CSS odnosno SCSS sintaksa je prikazana na slici 3.9. te se u custom.scss nalaze svi definirani stilovi i dizajn u aplikaciji. Internet stranice je responzivna to znači da se prikazuje jednako dobro i na mobilnim i na stolnim uređajima.



Sl. 3.9. Prikaz custom.scss datoteke i primjena responzivnog dizajna pomoću media queria

Jedna bitna komponenta u Ruby on Rails aplikaciji je ruter koji se nalazi u direktoriju ***config/routes.rb***. Rails ruter je modul koji prepoznaje internetski URL i otprema ih kontroleru odnosno njegovim akcijama. Na slici 3.10. prikazana je ***routes.rb*** datoteka.

routes.rb

config > routes.rb

```
1   Rails.application.routes.draw do
2     get 'password_resets/new'
3     get 'password_resets/edit'
4     root "static_pages#home"
5     get '/help', to:'static_pages#help'
6     get '/about', to:'static_pages#about'
7     get '/contact', to:'static_pages#contact'
8     get '/signup', to:'users#new'
9     get '/login', to: 'sessions#new'
10    post 'login', to: 'sessions#create'
11    delete '/logout', to: 'sessions#destroy'
12    resources :users do
13      member do
14      get :following, :followers
15      end
16    end
17    resources :account_activations, only: [:edit]
18    resources :password_resets, only: [:new, :create, :edit, :update]
19    resources :microposts, only: [:create, :destroy]
20    resources :relationships, only: [:create, :destroy]
21  end
22
```

Sl. 3.10. Prikat routes.rb datoteke

Najbitni model koji se koristi u aplikaciju je ***User*** model odnosno model korisnika. Na slici 3.11. nalazi se shema kako izgleda tablica u bazi podataka te koji su osnovni podaci kako bi se korisnik mogao kreirati.

users	
id	integer
name	string
email	string
created_at	datetime
updated_at	datetime
password_digest	string

Sl. 3.11. Prikaz modela User Izvor[2]

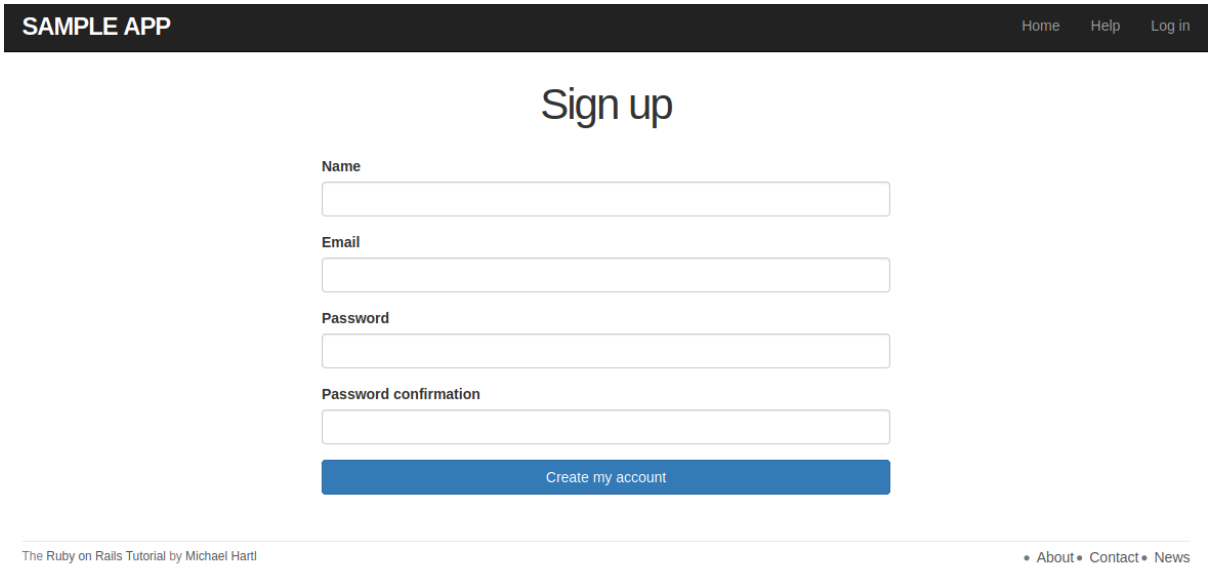
Kako bi se korisnik uspješno kreirao i kako se ne bi na primjer dogodilo da dva korisnika s istim emailom mogli registrirati, potrebno je napraviti validaciju korisnika. Na slici 3.12. prikazan je kod kojim su izvršene sve funkcionalnosti za validaciju. Email se prije pohrane u bazu podataka pohranjuje u formatu tako da email bude napisan s malim slovima. Ime korisnika ne smije biti prazno te može biti maksimalno do pedeset znakova kao i što email korisnika ne smije biti prazan te maksimalna duljina emaila je 255 znakova. Format emaila je definiran u varijabli ***VALID_EMAIL_REGEX*** te svi emailovi koji ne poštuju ovaj format nisu ispravni. Naravno email mora biti jedinstven i ne smije više korisnika imati isti email. Kako se u bazu ne bi direktno spremala lozinka koju je korisnik unio prilikom registracije, koristi se hash

lozinka. Lozinka isto tako ne smije biti prazna te mora biti minimalno sastavljena od šest slova. Gem koji se koristi za pretvaranje obične unesene lozinke u hash je "bycrypt".

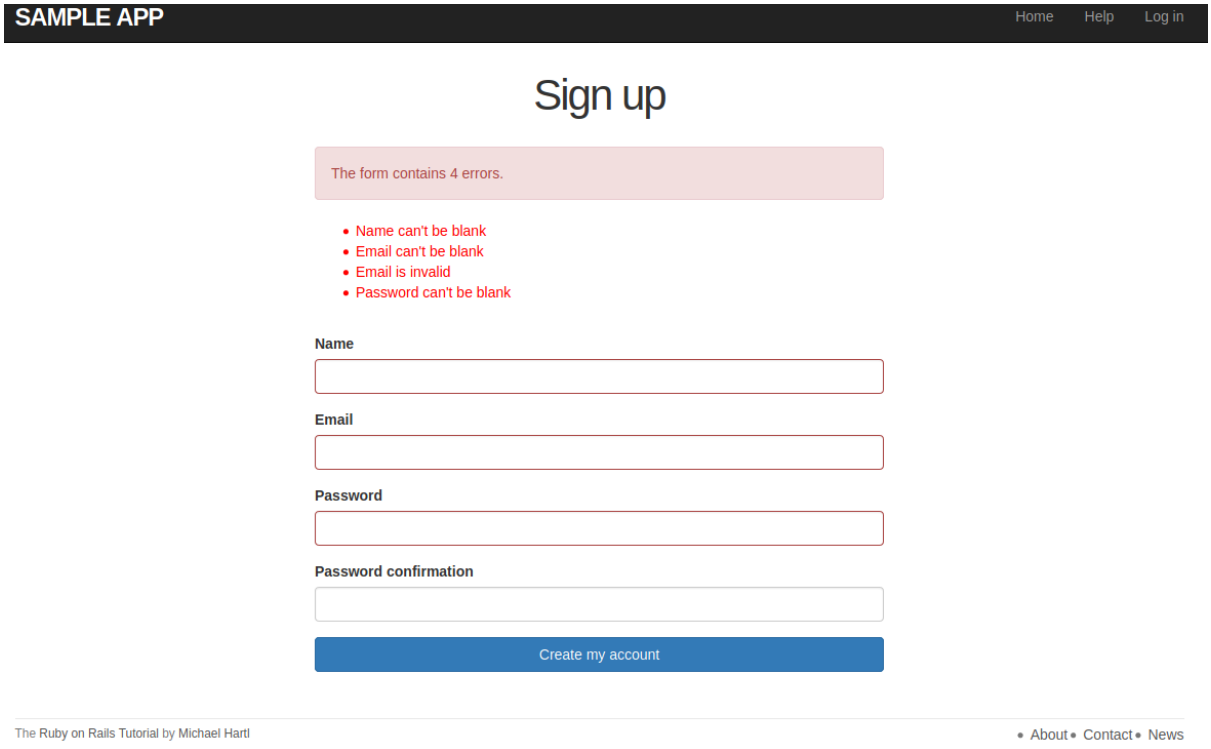
```
class User < ApplicationRecord
    before_save {email.downcase!}
    validates :name, presence:true, length:{maximum:50}
    VALID_EMAIL_REGEX = /\A[\w+\-.]+@[a-z\d\-]+(\.[a-z\d\-]+)*\.[a-z]+\z/i
    validates :email, presence:true, length:{ maximum:255}, format: {with: VALID_EMAIL_REGEX}, uniqueness:true
    has_secure_password
    validates :password, presence:true, length:{minimum:6}
end
```

Sl. 3.12. Kod za validaciju korisnika

Na slici 3.13. prikazano je kako izgleda forma "Sign up" kada se korisnik želi registrirati. Na slici 3.14. je pak prikazano kada korisnik ne unese dobro podatke to jest kada podaci nisu u skladu s validacijskim pravilima te se dobije poruka u crvenoj boji što nije dobro napisano i što treba ispraviti kako bi registracija korisnika bila validna.

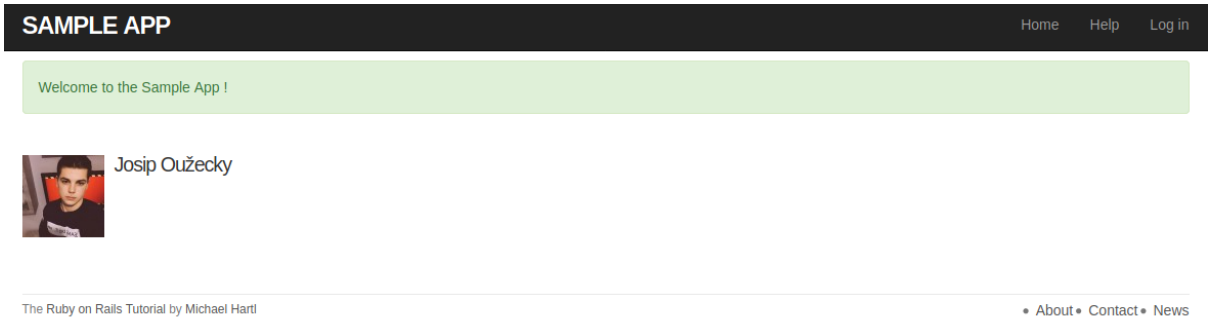


Sl. 3.13. Izgled registracijskog obrasca



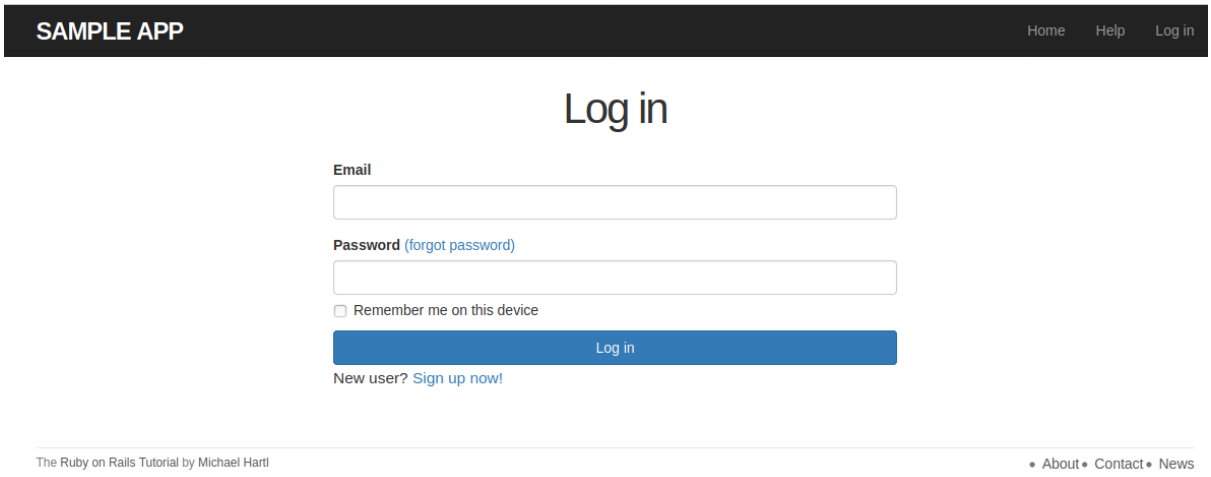
Sl. 3.14. Prikaz grešaka prilikom registracije korisnika

Ako je korisnik dobro ispunio podatke prilikom pritiska na gumb "Create my account" korisnik se automatski prebacuje na njegov profil te pomoću Gravatara učitava sliku ako korisnik s navednim emailom ima postavljenu sliku na njegovom orginalnom emailu. Ako nema postavlja se zadana slika od Gravatara. Na slici 3.15. prikazan je pogled nakon uspješnog registriranja.

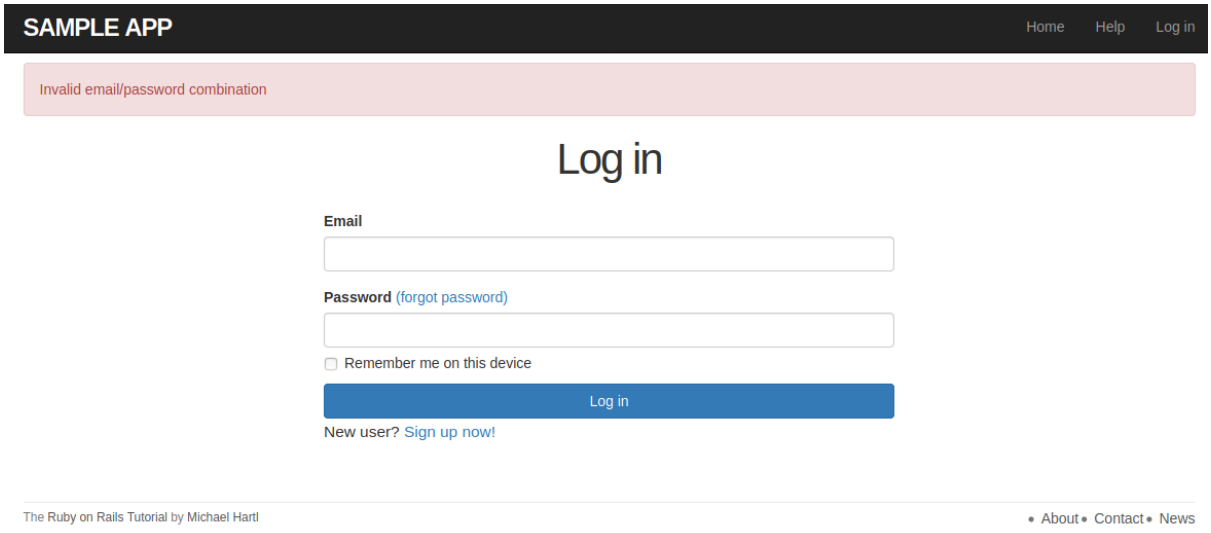


Sl. 3.15. Prikaz stranice nakon uspješnog registriranja

Ako korisnik već ima kreiran račun može se logirati pritiskom na link u navigaciji "Log in". Na slici 3.16. prikazan je izgled "Log in" forme. Ukoliko je korisnik kreiran i postoji u bazi podataka nakon unesenih podataka aplikacija će preusmjeriti korisnika na njegov profil. Ako korisnik krivo unese podatke ili korisnik ne postoji u bazi podataka, izbacit će se greška koja je prikazana na slici 3.17.



Sl. 3.16. Izgled forme koja služi za prijavu postojećih korisnika

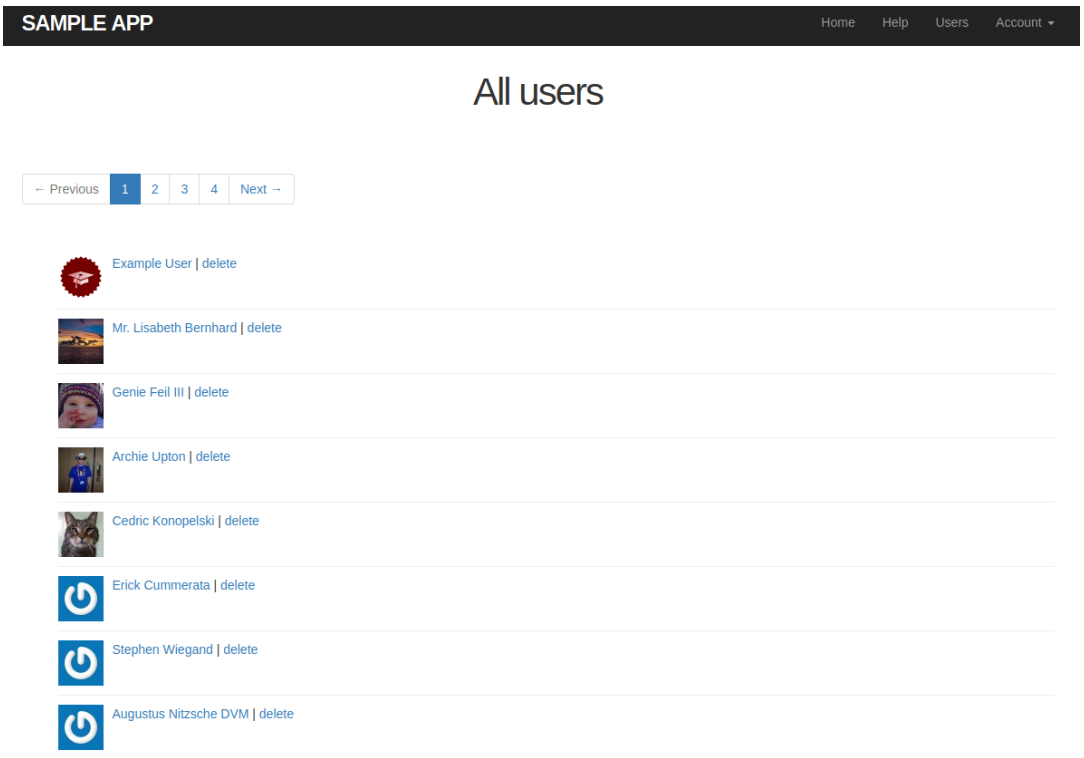


Sl. 3.17. Izbacivanje greške ako korisnik ne unese dobru kombinaciju email-a i lozinke

Dodatna stavka koju "Log in" forma ima je da se korisnik zapamti prilikom logiranja na određenom uređaju. Ako se označi polje "Remeber me on this device" korisnik se neće morati svaki put logirati u aplikaciju što pozitivno utječe na korisničko iskustvo. Naravno prilikom promjene uređaja korisnik će morati ponovno upisati svoje podatke. Prilikom otvaranja internet stranice provjerava se da li je korisnik prijavljen ili nije te se na temelju toga zaključuje na koju će internet stranicu biti preusmjeren. U Railsu, kako bi se implementirala sesija, se koriste kolačići jer njima uspijevamo dobiti funkcionalnost da se korisnik zapamti prilikom ulogiranja. Kolačići su zapravo tekstualne datoteke spremljene na računalu te se one čuvaju u svrhu praćenja bitnih informacija. Na primjer serverska skripta šalje kolačiće internetskom pregledniku. To mogu biti informacije kao što su ime, godine ili identifikacijski broj te zatim

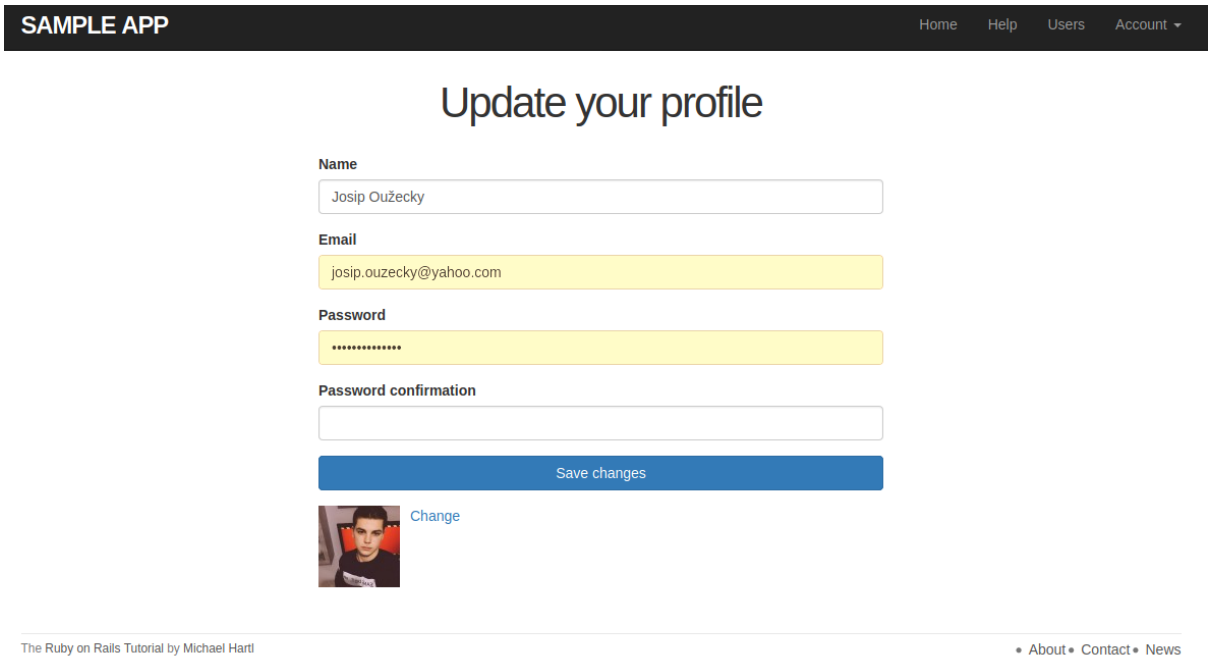
internetski preglednik sprema te informacije na lokalnu mašinu za buduću uporabu. Kada sljedeći put internetski preglednik šalje zahtjev serveru, odmah šalje kolačiće serveru i server koristi te informacije kako bi identificirao korisnika. Sesija zapravo kreira datoteku u privremenom direktoriju na serveru gdje se nalaze njezine varijable i ostale vrijednosti koje su tamo pohranjene. Ti podaci su dostupni svim stranica na aplikaciji tijekom tog posjeta. Sesija završava kada korisnik zatvori internetski preglednik ili kada napusti internet stranicu. Server najčešće izbriše sesiju nakon određenog perioda koji je najčešće trideset minuta.

Korisnik ima sve potrebne akcije koje poštuju REST arhitekturu. Na slici 3.18. prikazan je izgled stranice kada se stisne na link "Users" te se dohvaćaju svi korisnici koji su kreirani u aplikaciji. U CRUD-u je to zapravo R odnosno "Read" jer iščitavamo korisnike. Gem koji se koristi za lažne korisnike je "faker" te omogućava izradu velikog broja korisnika i ispunjvanje baze podataka vrlo jednostavno. Za obilježavanje stranice (engl. pagination) se koristi gem "will_paginate" koji daje mogućnost dohvaćanje iz baze na primjer po trideset korisnika te svaka stranica sadrži novih trideset korisnika. Time se zapravo postiže prividna brzina aplikacije jer se šalje zahtjev da se dohvati prvih trideset korisnika, a ne sve korisnike iz baze. Ako postoji oko deset tisuća korisnika i da se moraju dohvatiti svi korisnici u jednom zahtjevu to bi dosta trajalo i usporilo aplikaciju, a to naravno nije cilj.



Sl. 3.18. Dohvaćanje svih korisnika iz aplikacije

Korisnik ima mogućnost i da ažurira svoje podatke o profilu. U CRUD-u je to U odnosno "Update". Na slici 3.19. prikazana je forma s kojom ažuriraju osnovni podatci o korisniku.



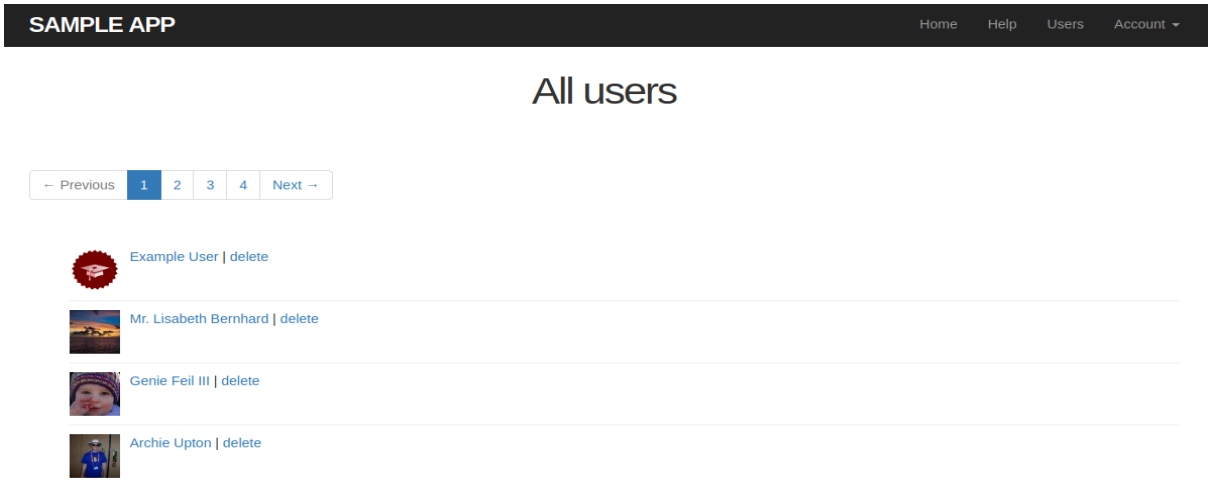
Sl. 3.19. Forma za ažuriranje informacija korisnika na linku "Account -> Settings"

Zadnja operacija koja nam ne dostaje u CRUD-u je zapravo D odnosno "DELETE" koja daje mogućnost brisanja određenih korisnika iz baze. Ova operacija je zapravo vrlo opasna i ne treba biti cilj da se ovakva privilegija daje svakog korisniku nego samo određenim korisnicima. Administratorska prava su prikladna za ovaj problem jer samo određeni korisnici s privilegijama mogu obrisati ostale korisnike. Na slici 3.20. prikazano je kako dodati adminstratorska prava preko rails konzole nakon implementiranja funkcionalnosti u aplikaciji.

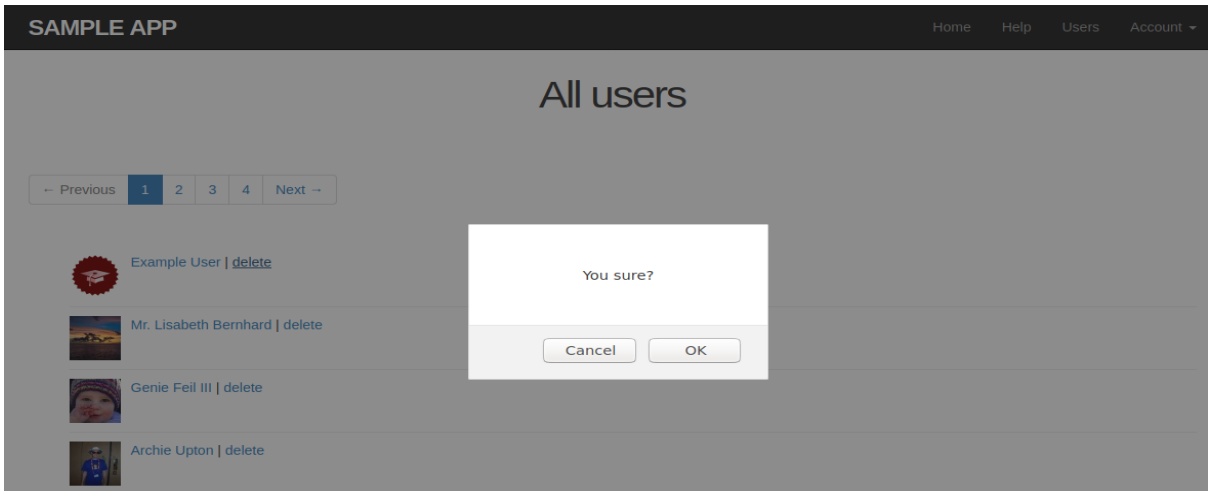
```
$ rails console --sandbox
>> user = User.first
>> user.admin?
=> false
>> user.toggle!(:admin)
=> true
>> user.admin?
=> true
```

Sl. 3.20. Dodavanje prvom korisniku u bazi podataka administratorska prava Izvor[2]

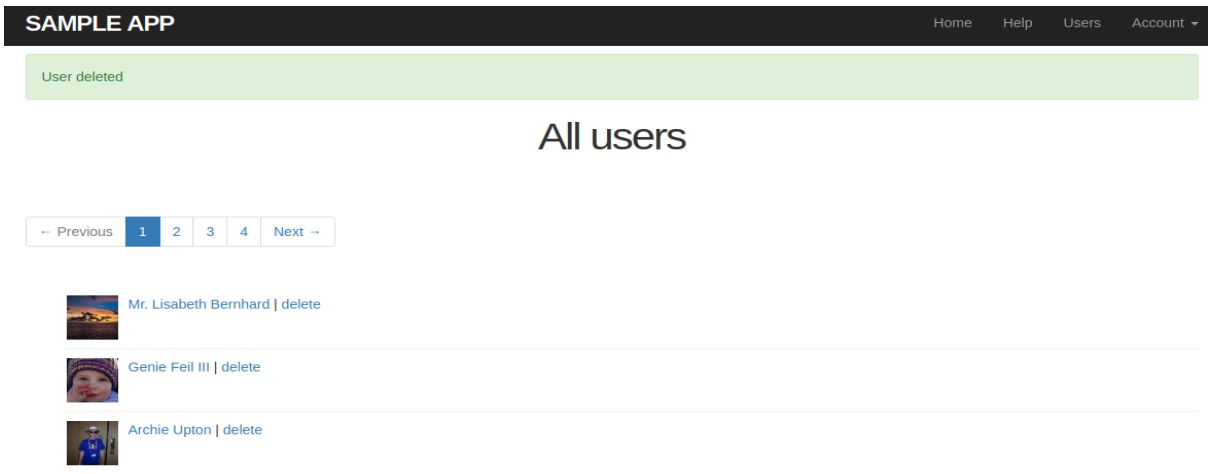
Kada korisnik ima administratorska prava korisnik pritiskom na "Users" link u navigaciji može obrisati svakog korisnika u aplikaciji. Na slikama 3.21., 3.22.. i 3.23. je prikazana ova funkcionalnost.



Sl. 3.21. Prikaz dodatne opcije "delete" pokraj svakog korisnika



Sl. 3.22. Potvrda nakon pritiska na "delete"



Sl. 3.23. Poruka nakon uspješnog obavljenog zadatka

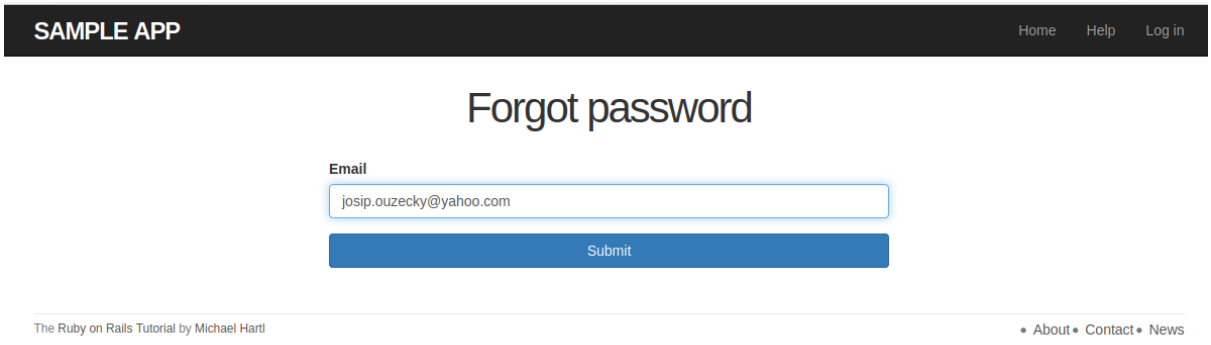
Za funkcionalnost da se korisnik zapamti prilikom logiranja te da se omogući da određeni korisnici imaju administratorska prava, u modelu korisnika su potrebna dva nova atributa, a to

su ***rebemer_digest*** i ***admin***. Na slici 3.24. prikazan je model korisnika s tim atributa te još četiri dodatna atributa s kojom se omogućava aktivacija korisničkog računa te resetiranje lozinke.

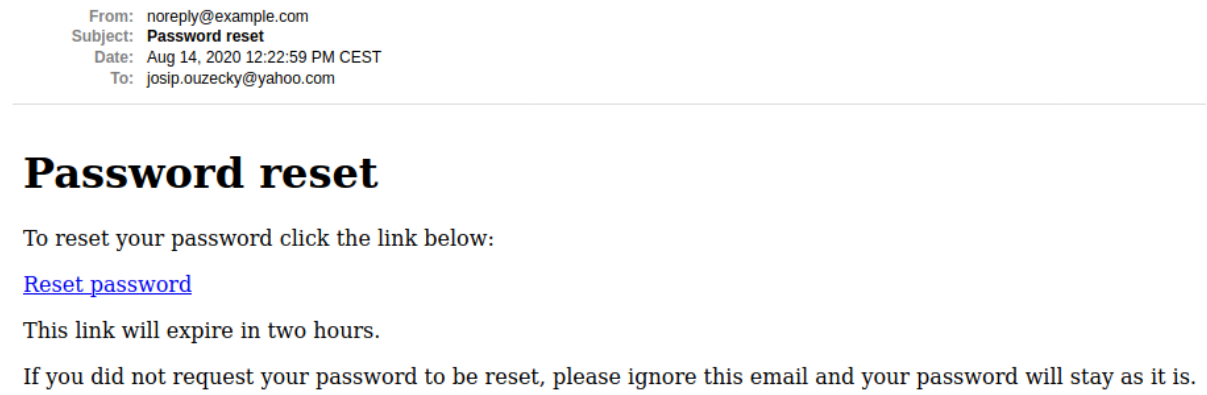
users	
id	integer
name	string
email	string
created_at	datetime
updated_at	datetime
password_digest	string
remember_digest	string
admin	boolean
activation_digest	string
activated	boolean
activated_at	datetime
reset_digest	string
reset_sent_at	datetime

Sl. 3.24. Prikaz krajnjeg modela korisnika Izvor[2]

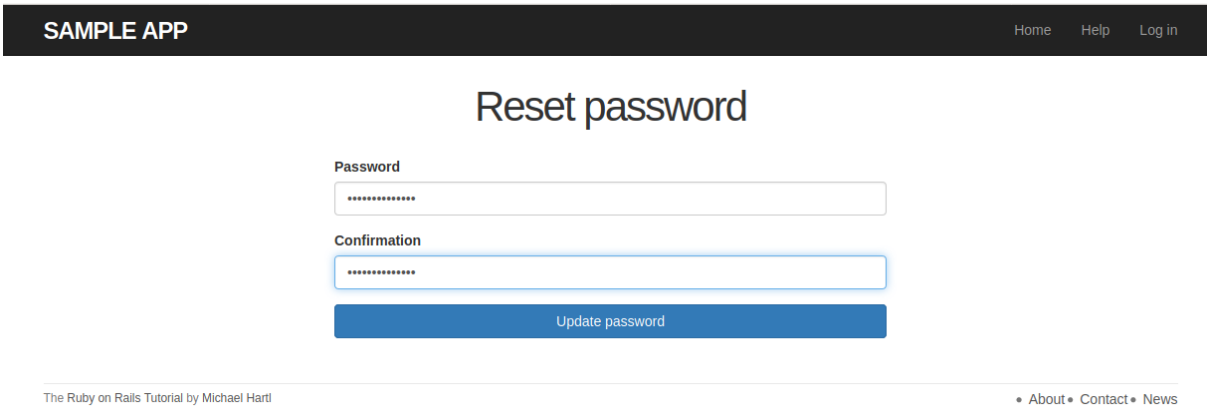
Ruby on Rails ima ugrađeni Mailer s kojim se šalje elektronička pošta putem aplikacije. Te slanje elektroničke pošte je jedan od najčešćih poslova u modernim internet aplikacijama. Ako je korisnik zaboravio kojim slučajem lozinku prilikom logiranja pristiskom na link "forgot password" će mu omogućiti da dobije na email link s kojim će biti u mogućnosti resetirati lozinku. Na slikama 3.25., 3.26., 3.27. i 3.28. prikazan je postupak resetiranja lozinke.



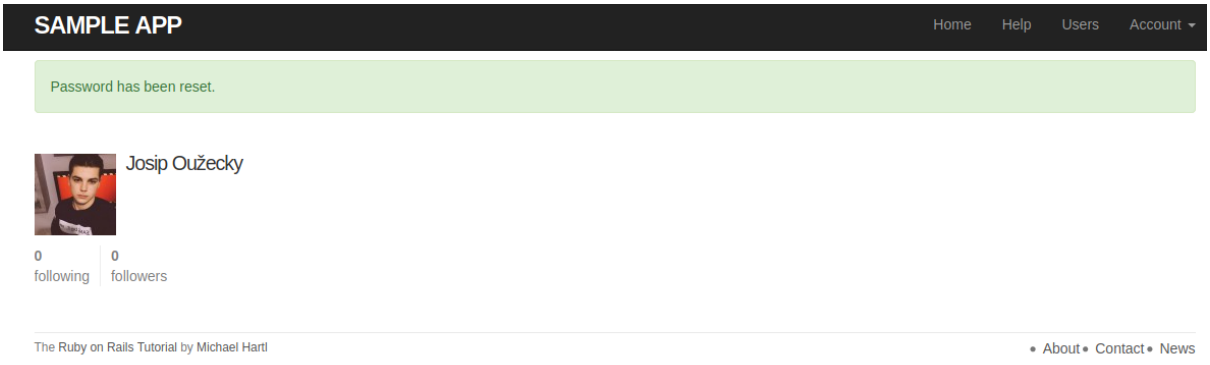
Sl. 3.25. Obrazac nakon pritiska linka "forgot password"



Sl. 3.26. Predložak koji je poslan na email



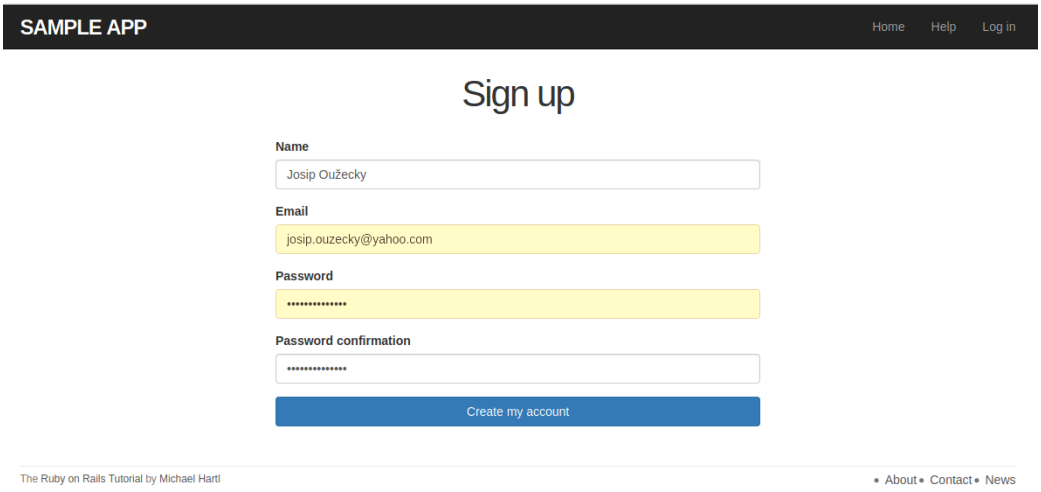
Sl. 3.27. Obrazan nakon pritiska na link "Reset password"



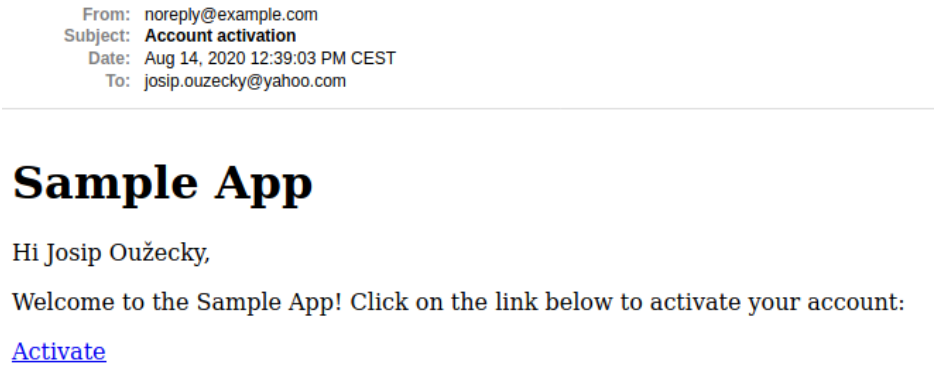
Sl. 3.28. Poruka da je lozinka uspiješno promijenjena nakon pritiska na tipku "Update password"

U modelu ***reset_digest*** je token koji se zapravo generira nasumično za svakog korisnika te se ovaj token priključuje linku s kojim korisnik resetira svoju lozinku dok ***reset_at*** atribut služi kako bi se odredilo točno vrijeme resetiranja lozinke.

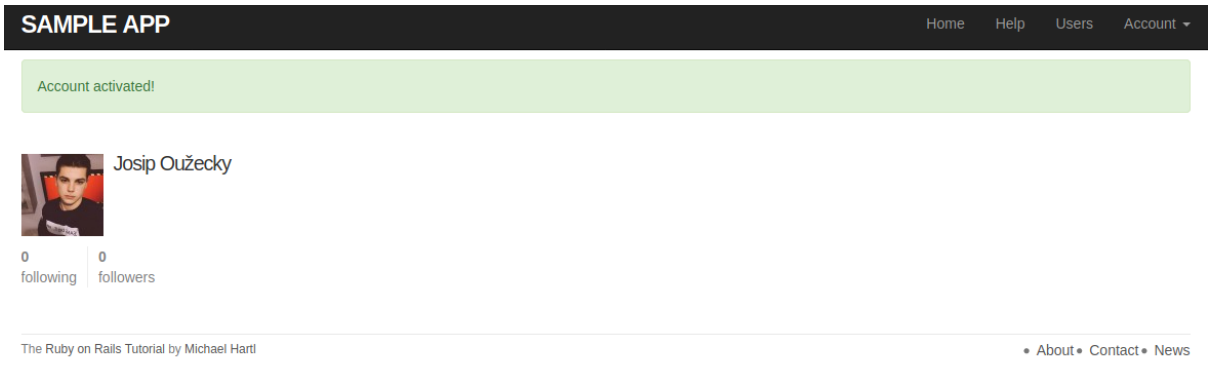
Dodatna mogućnost koja je ugrađena naknadno u aplikaciju je aktivacija emaila prilikom kreiranja korisničkog računa. Naime, kada korisnik napravi korisnički račun on ne može pristupiti početnoj stranici prije nego što aktivira svoj račun. Na slikama 3.29., 3.30. i 3.31. je prikazan postupak modificiranog kreiranja korisničkog računa.



Sl. 3.29. Prikaz obrasca za kreiranje korisničkog računa



Sl. 3.30. Prikaz predloška koji se dobije na email



Sl. 3.31. Prikaz poruke nakon pritiska na link "Activate" u emailu

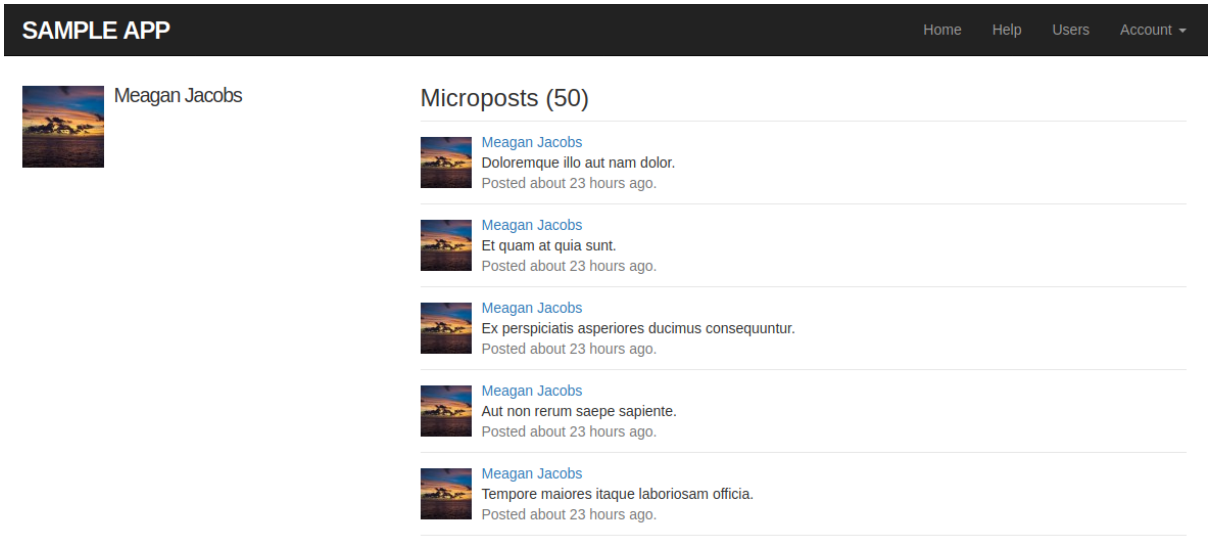
U modelu atribut ***activated*** služi kako bi utvrdio stanje da li je korisnički račun aktiviran ili nije kako bi dao odgovarajuću poruku korisniku kada pristupi aplikaciji. Ako korisnik nije aktiviran dobije poruku da mora prvo aktivirati račun kako bi se mogao uspješno logirati. Ako je korisnik aktiviran onda se korisnik bez ikakvih problema može prijaviti.***Activated_at*** atribut nam govori kada je korisnički račun aktiviran to jest u kojem točno vremenu. ***Activate_digest*** je token koji se zapravo generira nasumično za svakog korisnika te se ovaj token priključuje linku s kojim korisnik aktivira svoj račun.

Drugi najbitniji model je ***Micropost*** koji zapravo označava status svakog korisnika. Na slici 3.32. prikazan je model i njegovi atributi. ***Id*** atribut služi kako bi svaki status bio jedinstven dok ***content*** atribut služi kako bi sadržao tekst statusa. Kako bi se status povezao s određenim korisnikom postoji atribut ***user_id*** . ***Created_at*** i ***updated_at*** atributi nam služe da se vidi vrijeme kada je objavljen i ažuriran status.

microposts	
id	integer
content	text
user_id	integer
created_at	datetime
updated_at	datetime

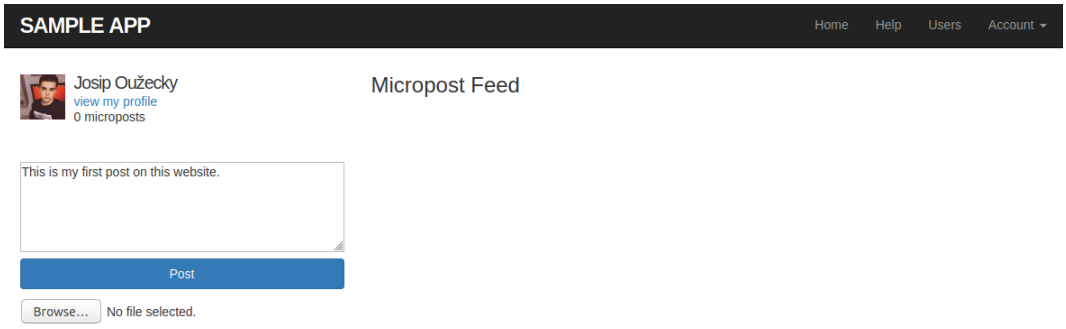
Sl. 3.32. Prikaz "Micropost" modela Izvor[2]

Bitne metode su ***has_many*** i ***belongs_to*** koje se koriste kako bi se prikazala veza između dva modela. Validacija što se tiče mikroposta je da ne smije biti prazan te da može imati maksimalno 140 znakova. Mikropostovi su poredani po datumu objavljivanja tako što su postovi koji su svježe postavljeni prikazani prvi, a koji su napravljeni prije su prikazani ispod. Gem "faker" je korišten kako bi svakom korisniku dodao statuse kako bi mogli testirati lakše aplikaciju. Na slici 3.33. prikazan je izgled mikroposta kod svakog korisnika.

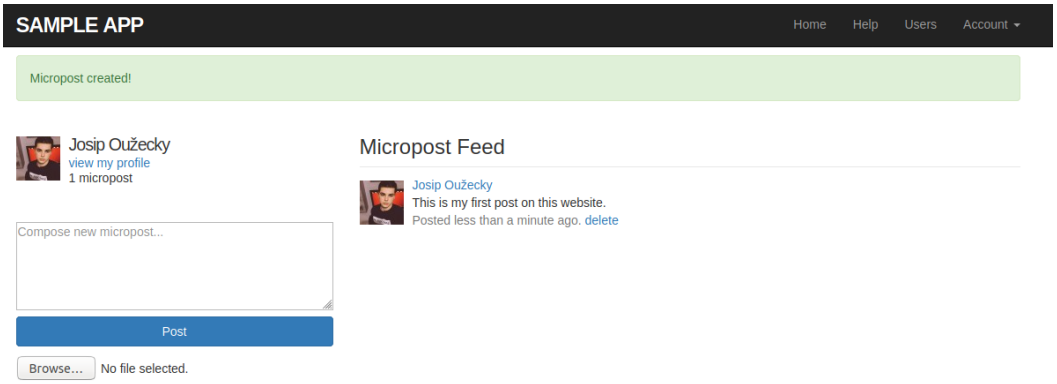


Sl. 3.33. Izgled mikroposta

Mikropost mora imati dvije bitne operacije u CRUD–u, a to su da se mikropost može kreirati kao i da se kreirani mikropost može obrisati. Na slikama 3.34. i 3.35. prikazan je postupak kreiranja vlastitog mikroposta. Prvo treba otići na link "Home", a zatim korisnik upisuje željeni status i klikne na dugme "Post".

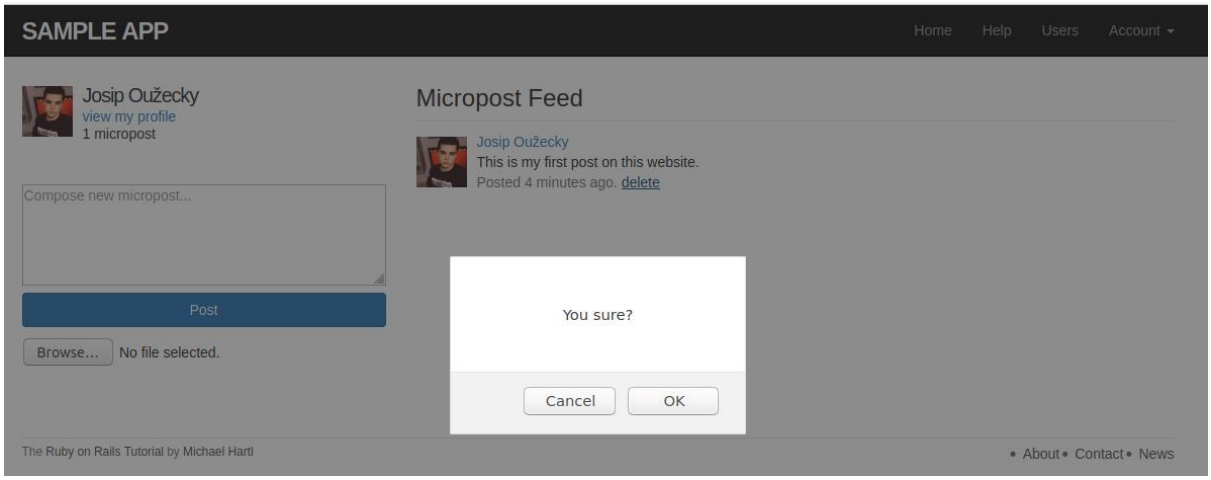


Sl. 3.34. Izgled forme za postavljanje mikroposta

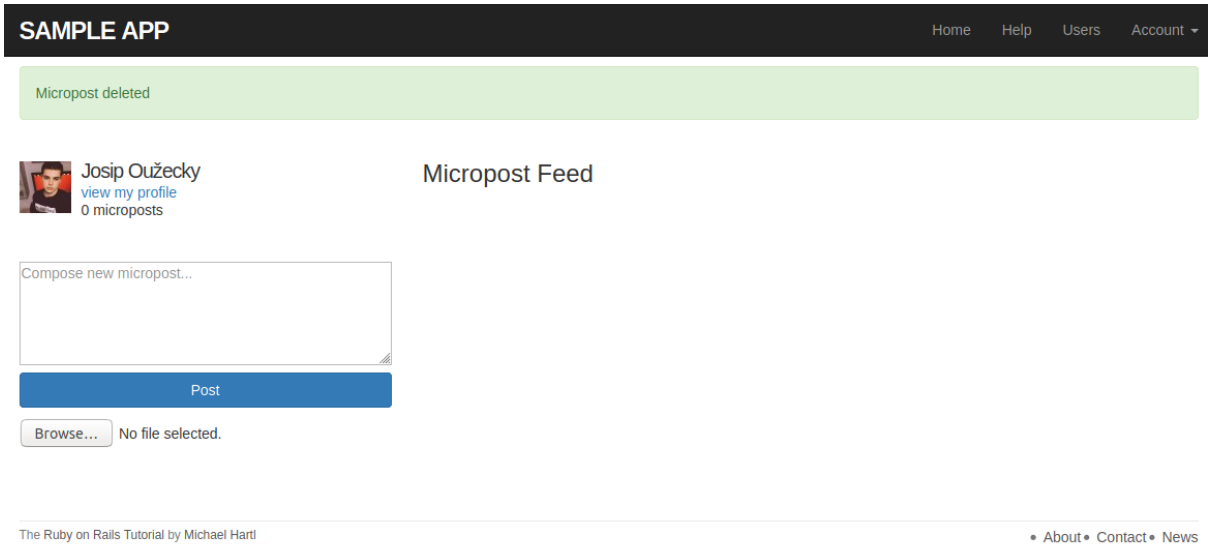


Sl. 3.35. Poruka za uspješni kreiran mikropost i prikaz mikroposta u sekciji "Micropost Feed"

Kada korisnik napravi mikropost on ga može i obrisati pritiskom na "delete" link pored vremena kada je mikropost objavljen. Na slikama 3.36. i 3.37. prikazan je postupak brisanja mikroposta.

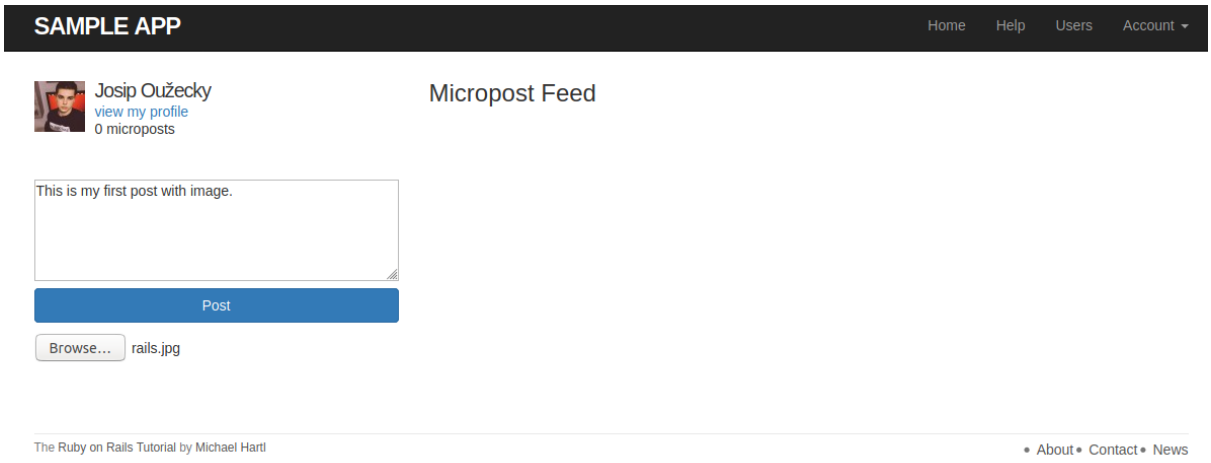


Sl. 3.36. Popup poruka nakon pritiska na "delete" link

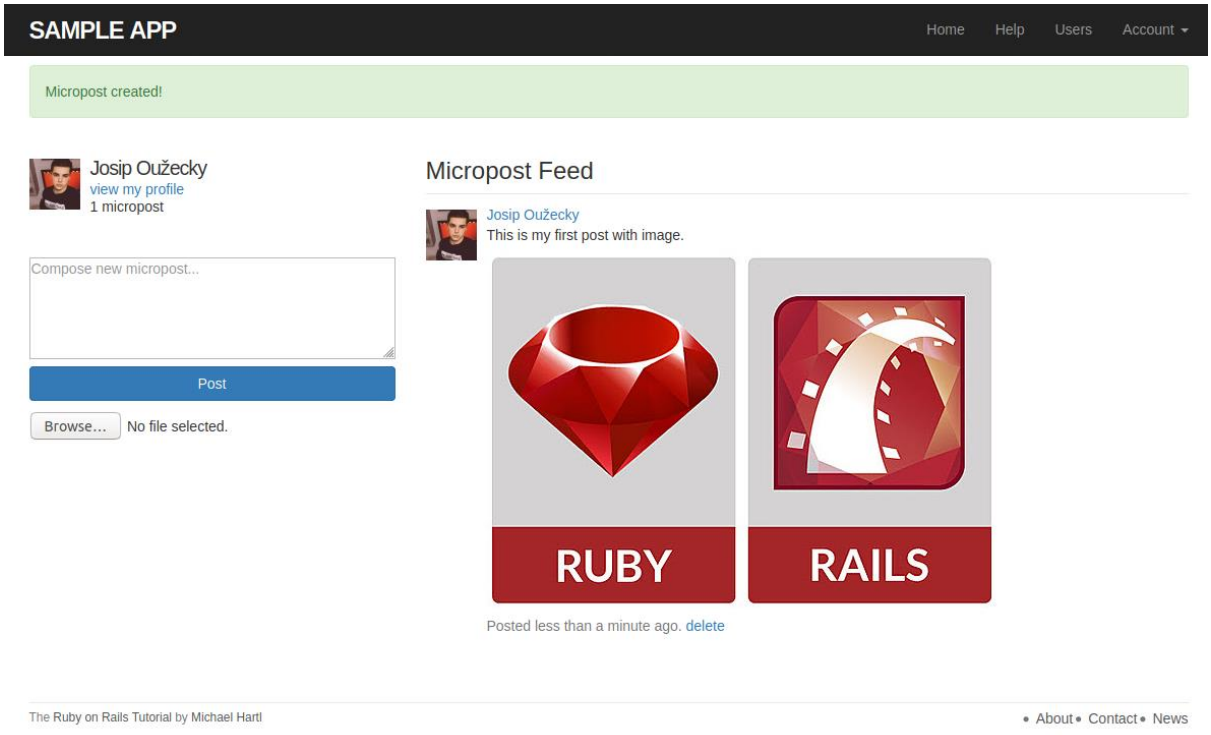


Sl. 3.37. Poruka da je posao brisanja uspješno obavljen i mikroposta više nema u sekciji "Micropost Feed"

Naime, postoji još jedna mogućnost prilikom objavljivanja mikroposta, a to je da u mikropost se uključi slika. ***Active_storage*** se koristi kako bi se uspješno slika mogla pohraniti i prikazati na internet aplikaciji. Kako slike imaju mnogo rezolucija i kako iste te slike mogu lagano pokidati izgled stranice koristi se i gem ***mini_magick*** i ***image_processing*** kako bi se sve slike uvijek pokazivale u zadanoj rezoluciji. Gem koji se koristi za validaciju slike je ***active_storage_validations*** te je zadano da slika ne smije biti veća od 5 megabajta i da prima samo 3 formata, a to su .gif, .jpeg, i .png. Na slikama 3.38. i 3.39. prikazan je postupak objavljivanja mikroposta sa slikom.



Sl. 3.38. Mikropost gdje je korištena slika rails.jpg



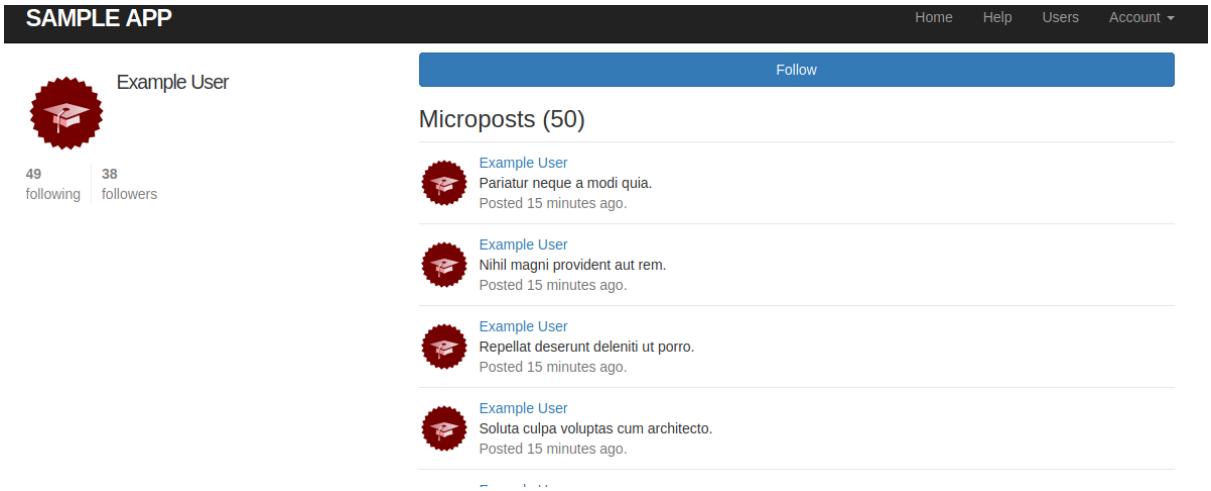
Sl. 3.39. Prikaz posta nakon pritiska na dugme "Post "

Zadnja mogućnost aplikacije je da korisnik može pratiti druge korisnike te da na svojoj početnoj stranici može vidjeti i statuse od ljudi koje prati. Zadnji model koji je potreban je prikazan na slici 3.40. ***Follower_id*** označava nam korisnika, a ***followed_id*** nam označava tko prati korisnika s id-om koji je sadržan u atributu ***follower_id***.

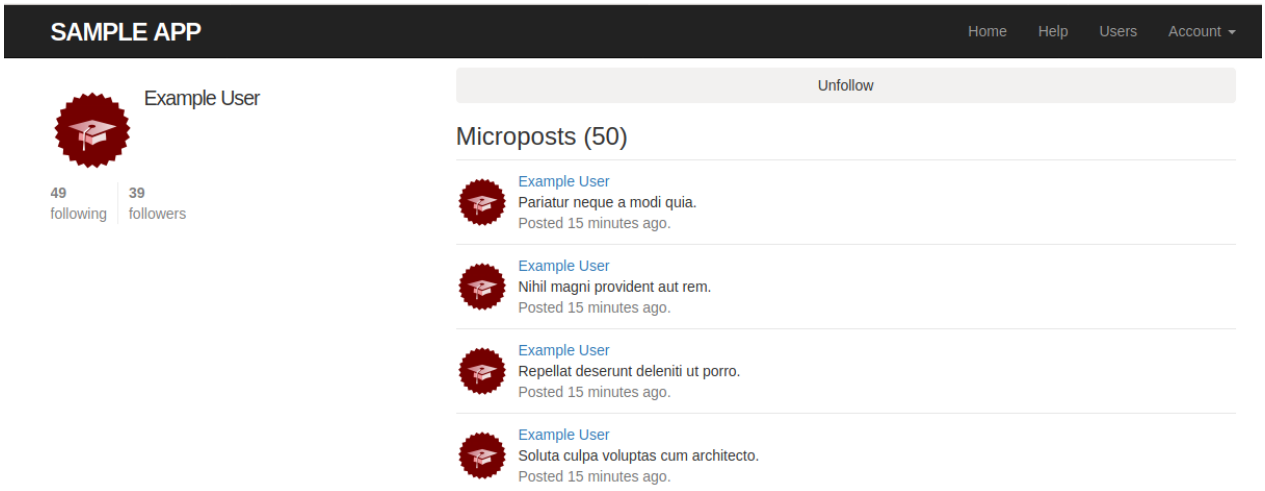
relationships	
id	integer
follower_id	integer
followed_id	integer
created_at	datetime
updated_at	datetime

Sl. 3.40. Prikaz modela koji omogućuje funkcionalnost praćenja korisnika Izvor[2]

S ovim modelom se zapravo stvara veza između korisnika kako bi se znalo koji korisnik koga prati te kako bi se moglo generirati na početnoj stranici svakog korisnika statusi odnosno vlastiti statusi korisnika te statusi od korisnika kojeg pratimo. Na slikama 3.41. i 3.42. prikazan je izgled kako se prati i otprati korisnik te kako izgleda sučelje gdje piše koliko korisnik ima pratitelja i koliko on prati korisnika.

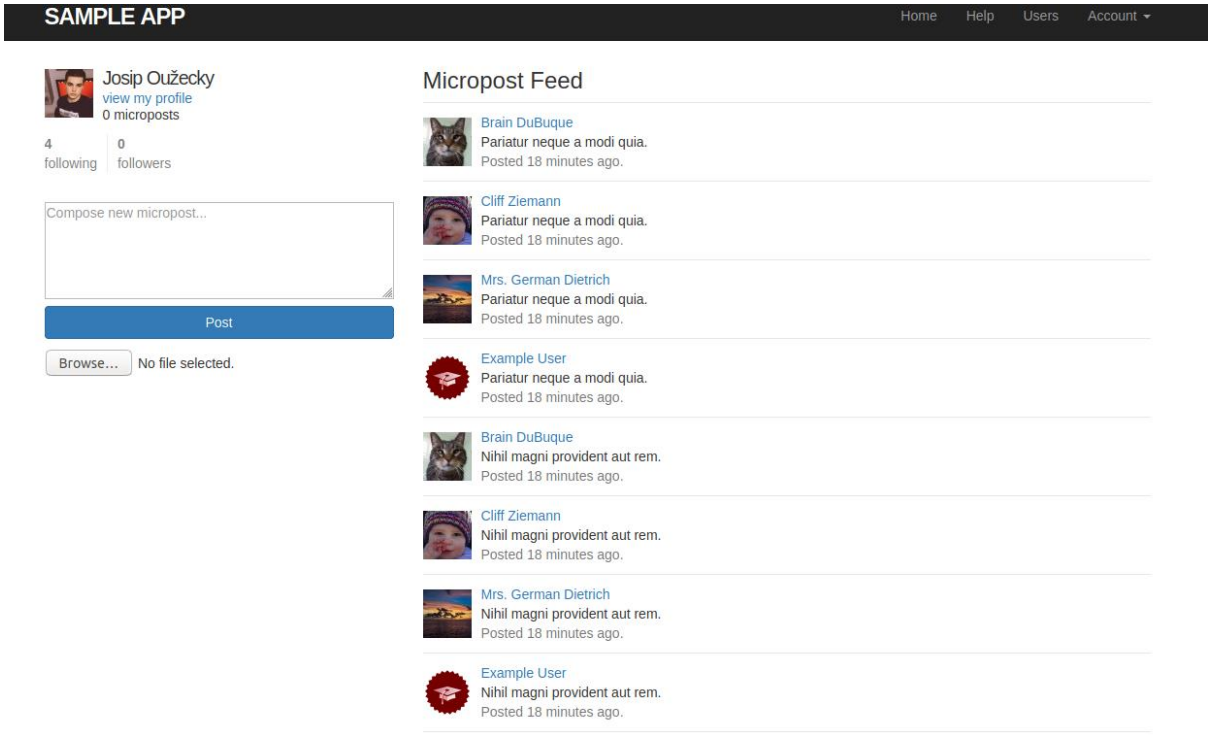


Sl. 3.41. Prikaz sučelja gdje možemo pratiti korisnika te vidjeti informaciju koliko korisnik ima pratitelja i koliko on prati



Sl. 3.42. Prikaz izgleda nakon pritiska na dugme "Follow"

Kada korisnik zaprati drugog korisnika pritiskom na link "Home" u navigaciji bi trebao vidjeti statuse od korisnika koje prati. Na slici 3.43. prikazan je izgled stranice nakon nekoliko zapraćenih korisnika.



Sl. 3.43. Izgled početne stranice nakon nekoliko zapraćenih korisnik

3.6. Integracija Active Job-a u aplikaciju

U aplikaciju je uključen jednostavni posao koji predstavlja običnu komandu ***sleep*** koja predstavlja spavanje na određeno vrijeme te se tako simulira jednostavan posao u Ruby on Rails okruženju. Komanda sleep simulira jednostavan posao jer izlazi iz aplikacije i u bashu ili komandnoj liniji izvodi operaciju što zapravo blokira aplikaciju na određeno vrijeme i tako se simulira posao na efektivan način. Postupak generiranje ovog posla započinje pisanjem komande koja je prikazana na slici 3.44. kao što je objašnjeno u [3].

```
zegac@zegac-VirtualBox:~/enviroment/SampleApplication$ rails generate job TestWorkloadJob --queue urgent
The dependency tzinfo-data (>= 0) will be unused by any of the platforms Bundler is installing for. Bund
r is installing for ruby but the dependency is only for x86-mingw32, x86-mswin32, x64-mingw32, java. To
d those platforms to the bundle, run `bundle lock --add-platform x86-mingw32 x86-mswin32 x64-mingw32 jav
.
/home/zegac/.rvm/gems/ruby-2.7.0/gems/actionpack-6.0.2.2/lib/action_dispatch/middleware/stack.rb:37: war
ng: Using the last argument as keyword parameters is deprecated; maybe ** should be added to the call
/home/zegac/.rvm/gems/ruby-2.7.0/gems/actionpack-6.0.2.2/lib/action_dispatch/middleware/static.rb:110: w
ning: The called method `initialize' is defined here
Running via Spring preloader in process 3880
      invoke  test_unit
      create    test/jobs/test_workload_job_test.rb
      create  app/jobs/test_workload_job.rb
```

Sl. 3.44. Komanda za izradu posla te njezin rezultat

Nakon što se generira posao u datoteci ***apps/jobs/test_workload_job.rb*** opisuje se u metodi ***perform*** šta naš posao treba raditi. Na slici 3.45. prikazan je kod koji opisuje da posao izvršava sleep komandu na određeno vrijeme.

```ruby
class TestWorkloadJob < ApplicationJob
  queue_as :urgent

  def perform(user: nil, duration:20)
    if user.nil?
      puts "TODO: bacanje greske ako nema usera"
    end

    job = user.my_jobs.create(title: "Task od #{duration} sekundi", start: DateTime.now)

    #Do something later
    'sleep #{duration}'

    job.end = DateTime.now
    job.save
  end
end
```

Sl. 3.45. Funkcionalnost posla koji na određeno vrijeme spava

Nakon funkcionalnosti kreira se model koji će biti korišten za bazu podataka te koje će se vrijednosti u tablici upisivati za određeni posao. Na slici 3.46. prikazana je komanda s kojom se generira model ***my_job***.

```
zegac@zegac-VirtualBox:~/environment/SampleApplication$ rails generate model MyJobs title:string start:date
time end:datetime
The dependency tzinfo-data (>= 0) will be unused by any of the platforms Bundler is installing for. Bundle
r is installing for ruby but the dependency is only for x86-mingw32, x86-mswin32, x64-mingw32, java. To ad
d those platforms to the bundle, run `bundle lock --add-platform x86-mingw32 x86-mswin32 x64-mingw32 java`
.
Running via Spring preloader in process 4268
[WARNING] The model name 'MyJobs' was recognized as a plural, using the singular 'MyJob' instead. Override
 with --force-plural or setup custom inflection rules for this noun before running the generator.
      invoke  active_record
      create    db/migrate/20200901083607_create_my_jobs.rb
      create    app/models/my_job.rb
      invoke    test_unit
      create      test/models/my_job_test.rb
      create      test/fixtures/my_jobs.yml
```

Sl. 3.46. Komanda za kreiranje modela te njezin rezultat

Nakon kreiranje treba se definirati da ovaj model pripada korisniku te je na slici 3.47. prikazan kod za ovu funkcionanost.

app > models > my_job.rb

```ruby
1  class MyJob < ApplicationRecord
2      belongs_to :user
3  end
4
```

Sl. 3.47. Kod za funkcionalost da se *my_job* model poveže sa korisnikom

Naravno nakon povezivanja ***my_job*** modela s korisnikom treba se u modelu korisnika naglasiti da svaki korisnik ima više poslova te da ako se korisnik izbriše izbrisat će se i svi poslovi u bazi podataka. Kod za ovu funkcionalost je prikazan na slici 3.48.

app > models > user.rb

```ruby
class User < ApplicationRecord
    has_many :microposts, dependent: :destroy
    attr_accessor :remember_token, :activation_token, :reset_token
    has_many :active_relationships, class_name: "Relationship",foreign_key: "follower_id",dependent: :destroy
    has_many :passive_relationships, class_name: "Relationship",foreign_key: "followed_id",dependent: :destroy
    has_many :following, through: :active_relationships, source: :followed
    has_many :followers, through: :passive_relationships, source: :follower

    has_many :my_jobs, dependent: :destroy

    before_save :downcase_email
    before_create :create_activation_digest
    validates :name, presence:true, length:{maximum:50}
    VALID_EMAIL_REGEX = /\A[\w+\-.]+@[a-z\d\-]+(\.[a-z\d\-]+)*\.[a-z]+\z/i
    validates :email, presence:true, length:{ maximum:255}, format: {with: VALID_EMAIL_REGEX}, uniqueness:true
    has_secure_password
    validates :password, presence: true, length: { minimum: 6 }, allow_nil: true
```

Sl. 3.48. Kod za funkcionalost da se my_job model poveže sa korisnikom 2

U datoteci ***db/schema.rb*** trebamo kreirati tablicu, a kod za ovu funkcionalost je prikazan na slici 3.49.

db > schema.rb

```ruby
    t.datetime "activated_at"
    t.string "reset_digest"
    t.datetime "reset_sent_at"
    t.index ["email"], name: "index_users_on_email", unique: true
  end

  create table "my_jobs", force: :cascade do |t|
    t.string "title"
    t.datetime "start"
    t.datetime "end"
    t.datetime "created_at", precision: 6, null:false
    t.datetime "updated_at", precision: 6, null:false
    t.integer "user_id", null: false
    t.index ["user_id"], name: "index_my_jobs_on_user_id"
  end

  add_foreign_key "my_jobs", "users"
  add_foreign_key "active_storage_attachments", "active_storage_blobs", column: "blob_id"
  add_foreign_key "microposts", "users"
end
```

Sl. 3.49. Kod za kreiranje tablice *my_jobs* te povezivanje tablica *my_jobs* i *users*

Kako bi dali referencu da svaki korisnik ima posao koristi se rails komanda koja je prikazana na slici 3.50.

```
zegac@zegac-VirtualBox:~/enviroment/SampleApplication$ rails g migration AddUserIdToMyJobs user:references
The dependency tzinfo-data (>= 0) will be unused by any of the platforms Bundler is installing for. Bundle
r is installing for ruby but the dependency is only for x86-mingw32, x86-mswin32, x64-mingw32, java. To ad
d those platforms to the bundle, run `bundle lock --add-platform x86-mingw32 x86-mswin32 x64-mingw32 java`
.
Running via Spring preloader in process 4663
      invoke  active_record
      create    db/migrate/20200901085921_add_user_id_to_my_jobs.rb
zegac@zegac-VirtualBox:~/enviroment/SampleApplication$
```

Sl. 3.50. Komanda koja generira migraciju te daje referencu izmedu korisnika i posla

Kod koji je generiran pomoću ove komande djelotvorno radi za baze podataka kao što je Postgres i MySql, ali kod koji je prikazan na slici 3.51. ne radi dobro što se tiče SQLite baze podataka koja se koristi u razvijanju rails aplikacije. Stoga je na slici 3.52. prikazan kod koji je poboljšan i radi za sve baze podataka. Bez obzira na to postoje li u tablici postojeći retci ili ne, SQLite neće dopustiti da dodate NULL stupce bez zadanih vrijednosti. Zato je dobro dodati prvo kolonu ili referencu i onda dodati ograničenje (eng. constraint) kao što je prikazano u [35].

db > migrate > 20200901085921_add_user_id_to_my_jobs.rb

```ruby
class AddUserIdToMyJobs < ActiveRecord::Migration[6.0]
  def change
    add_reference :my_jobs, :user, null: false, foreign_key: true
  end
end
```

Sl. 3.51. Kod koji je generiran rails-ovom komandom

db > migrate > 20200901085921_add_user_id_to_my_jobs.rb

```ruby
class AddUserIdToMyJobs < ActiveRecord::Migration[6.0]
  def change
    add_reference: :my_jobs, :user, foreign_key:true
    change_column_null: :my_jobs, :user:id, false
  end
end
```

Sl. 3.52. Poboljšanja koda kako bi radio u razvijanju aplikacije i u produkciji

Predzadnji korak je pokrenuti komandu rails ***db:migrate*** kako bi se sve promjene primjenile na aplikaciju. Zadnji korak u izradi ovog posla je dodavanje linka ***StartJob*** u navigaciju te u kontroleru omogućiti funkcionalnost posla te naravno u ***routes.rb*** datoteci dodati rutu s kojem se pokreće akcija u kontroleru korisnika. Na slikama 3.53., 3.54. i 3.55. je prikazan kod kako bi se ove funkcionalosti ostvarile.

app > controllers > users_controller.rb

```ruby
class UsersController < ApplicationController
  before_action :logged_in_user, only: [:index, :edit, :update, :destroy, :following, :followers, :start_job]
  before_action :correct_user, only: [:edit, :update]
  before_action :admin_user, only: :destroy

  def start_job
    duration = params[:duration].present? ? params[:duration] : 10
    TestWorkloadJob.perform_later user: current_user, duration: duration
    flash[:success] = "Job is enqueued"
    redirect_to root_path
  end

  def index
    @users = User.paginate(page: params[:page])
  end

  def show
```

Sl. 3.53. Kod u kontroleru korisnika kako bi se posao pokrenuo

app > views > layouts > _header.html.erb

```erb
<header class="navbar navbar-fixed-top navbar-inverse">
  <div class="container">
    <%= link_to "sample app", root_path , id: "logo" %>
    <nav>
      <ul class="nav navbar-nav navbar-right">
        <li><%= link_to "StartJob", start_job_path %></li>
        <li><%= link_to "Home", root_path %></li>
        <li><%= link_to "Help", help_path %></li>
        <% if logged_in? %>
          <li><%= link_to "Users", users_path %></li>
          <li class="dropdown">
            <a href="#" class="dropdown-toggle" data-toggle="dropdown">
          Account <b class="caret"></b>
            </a>
          <ul class="dropdown-menu">
            <li><%= link_to "Profile", current_user %></li>
            <li><%= link_to "Settings", edit_user_path(current_user) %></li>
            <li class="divider"></li>
            <li>
            <%= link_to "Log out", logout_path, method: :delete %>
            </li>
          </ul>
        </li>
        <% else %>
          <li><%= link_to "Log in", login_path %></li>
        <% end %>
        </ul>
      </nav>
  </div>
</header>
```

Sl. 3.54. Dodavanje linka koji pokreće posao klikom na njega

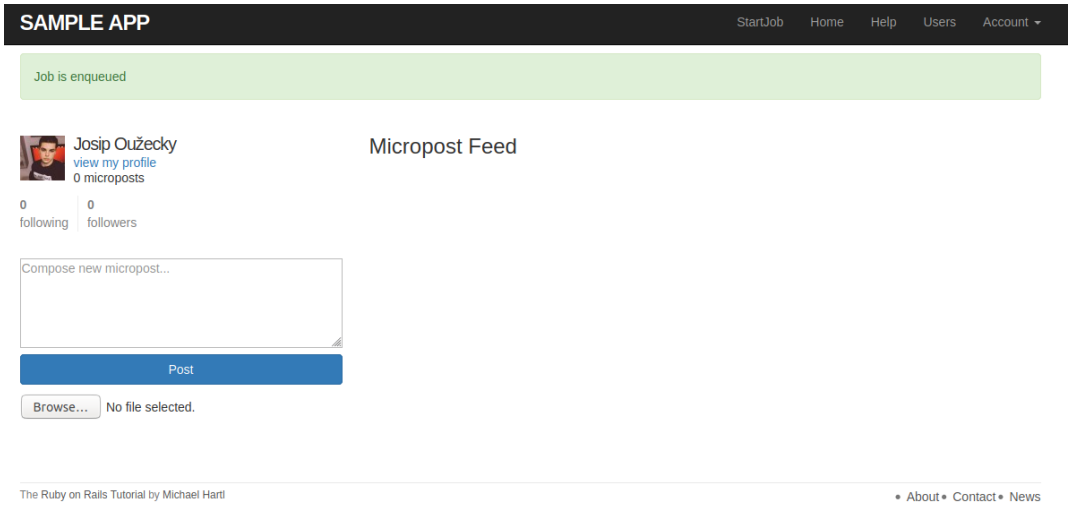
config > routes.rb

```ruby
Rails.application.routes.draw do
  get 'password_resets/new'
  get 'password_resets/edit'
  root "static_pages#home"
  get '/help', to:'static_pages#help'
  get '/about', to:'static_pages#about'
  get '/contact', to:'static_pages#contact'
  get '/start_job', to: 'user#start_job'
  get '/signup', to:'users#new'
  get '/login', to: 'sessions#new'
  post 'login', to: 'sessions#create'
  delete '/logout', to: 'sessions#destroy'
  resources :users do
    member do
    get :following, :followers
    end
  end
  resources :account_activations, only: [:edit]
  resources :password_resets, only: [:new, :create, :edit, :update]
  resources :microposts, only: [:create, :destroy]
  resources :relationships, only: [:create, :destroy]
end

```

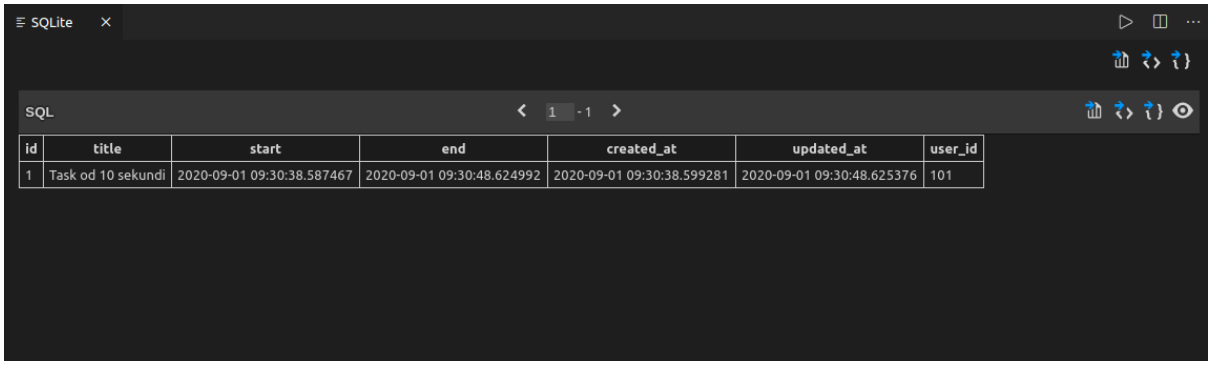
Sl. 3.55. Dodavanje rute kako bi se znalo u kojem kontroleru treba koji metou izvršiti

Kako bi se testiralo da li se posao zapravo izvršava kako treba i da li se u tablicu ***my_jobs*** upisuju podaci koji bi trebali, koristi se Visual Studio Code ekstenzija koja se zove SQLite da bi se dobio uvid u sve tablice u aplikaciji te podaci u njima. Posao će se izvršiti samo ako je korisnik ulogiran jer je sve preko njega implementirano i on mora stisniti link ***StartJob***. Na slici 3.56.prikazan je flash poruka nakon pritiska ***StartJob*** linka.



Sl. 3.56. Pritiskom na link StartJob dobiva se poruka da je uspješno posao postavljen u red izvršavanja

Na slici 3.57. nalazi se tablica ***my_jobs*** koja daje podatke o poslu kada je započet i izvršen te kojem korisniku pripada. Naravno posao se izvršava samo 10 sekundi jer je to zadana vrijednost i ne predaje se parametar za vrijeme koliko će posao trajati prilikom pritiska na link ***StartJob***.

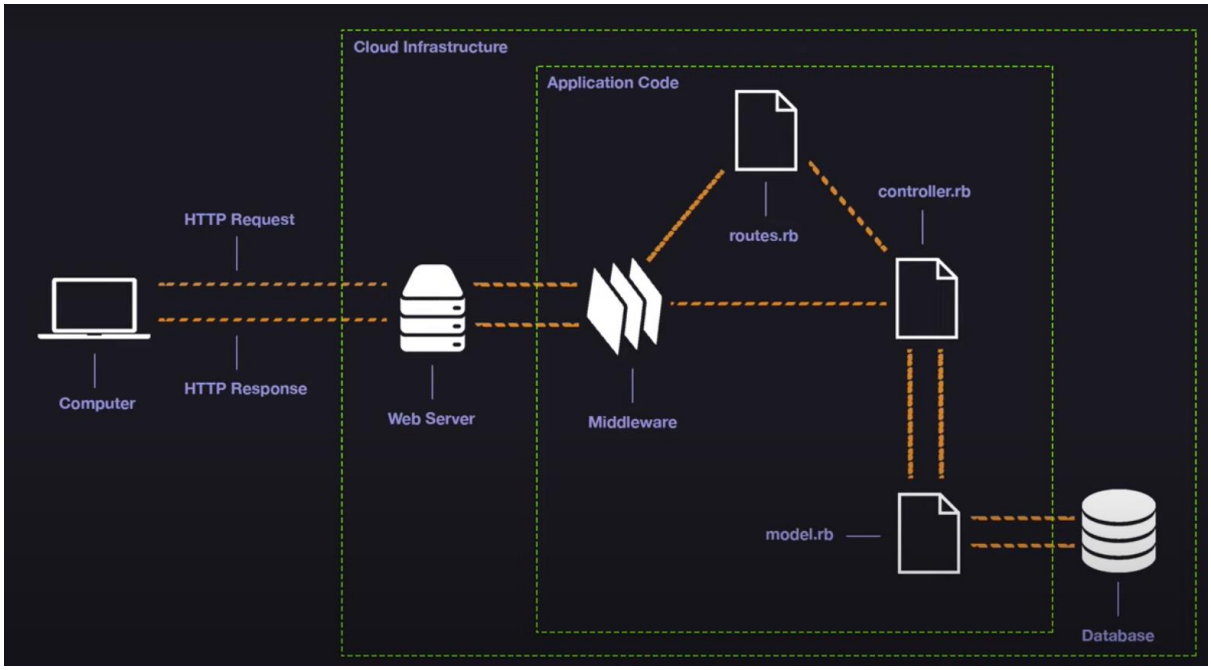


Sl. 3.57. Prikaz tablice my_jobs u kojom su prikazane informacije u poslu koji je pritisnut na prošloj slici

3.7. Sidekiq i Redis

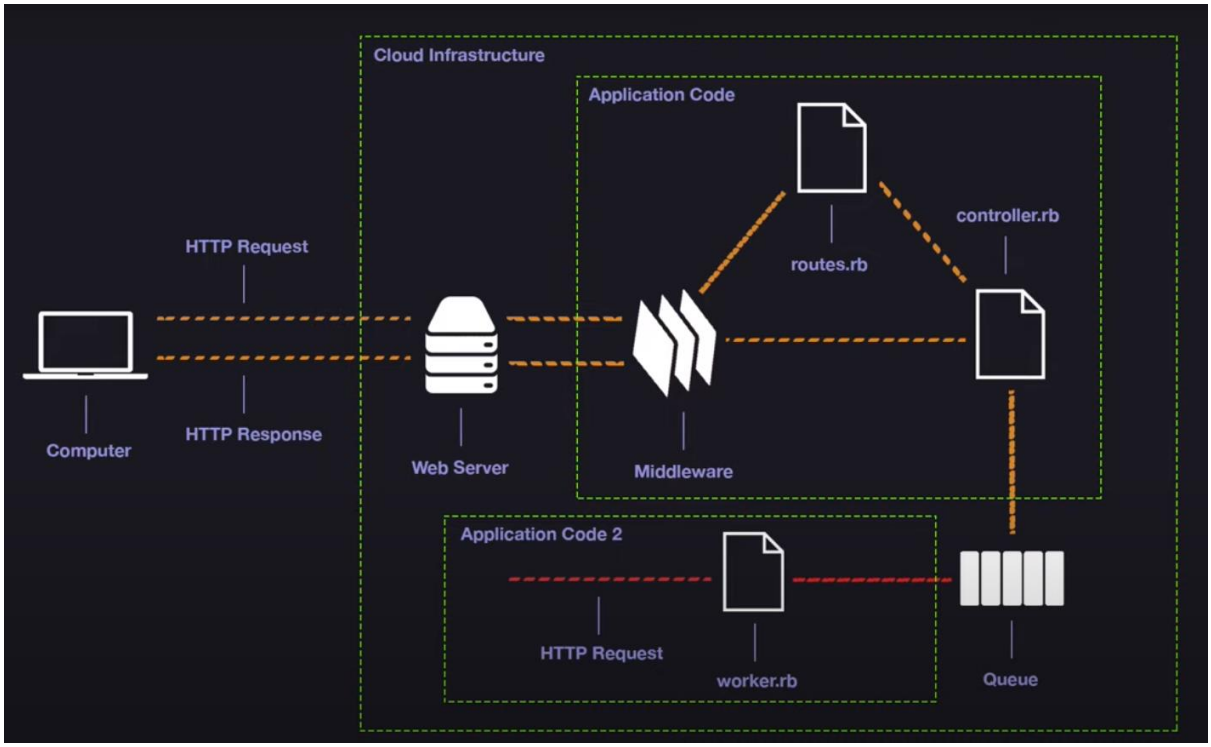
Dobar primjer bi bio kada bi imali internet aplikaciju u kojoj se treba dobiti izvješće o prodaji, takve stvari imaju jako puno informacija i trebaju dosta vremena da se učitaju. Prilikom izrade ovakve aplikacije kada bi poslali zahtjev serveru da se dobiju sva izvješća te da ih se prikaže i

ako naprimjer proces traje tridesetak sekundi pričekalo bi se to vrijeme i podaci bi se dobili. U produkciji je situacija nešto drukčija i ako traje zahtjev više od trideset sekundi (Heroku) dobije se "Application Error". Zbog toga je sinkroni proces vrlo loša ideja u produkciji. Stoga se koriste pozadinski poslovi kako bi se asinkrono napravila ova zadaća kao što je prikazano u [4,29]. Na slici 3.58. prikazan je sinkronosni proces u Rails aplikaciji.



Sl. 3.58. Sinkronosni proces u Rails aplikaciji Izvor[4]

Kada se klikne na dugme "Report" u aplikaciji pravimo HTTP zahtjev internet serveru. On će zatim usmjeriti zahtjev našem aplikacijskom kodu koji je u ovom slučaju Rails aplikacija. Zatim prolazi kroz međuopremu te se usmjerava na odgovarajući kontroler koji se poziva na model koji će dohvatiti podatke iz baze podataka, a zatim će ih vratiti kontroleru. Kontroler će organizirati sve u odgovarajućem formatu i vratiti kroz međuopremu zatim kroz internet server te će se konačno vratiti uređaju kao HTTP odgovor. Ovo se zove ciklički sinkroni proces. Iz UX perspektive blokira se aplikacija da radi bilo što drugo sve dok se ne dobije rezultat. U produkciji se događa greška „Application Error“ nakon trideset sekundi od zahtjeva a kako bi se moguća greška spriječila mora se procesirati u pozadini tako što šaljemo zahtjev i odma dobijemo odgovor, a ne dok se zadatak izvrši, ovo u praksi radi pomoću reda za izvršavanja (eng. queue). Red služi za pohranu podataka i sličan je bazi podataka, ali je mnogo jednostavniji jer je to zapravo par ključa i vrijednosti bez potrebe modeliranja modela za podatke. Na slici 3.59. prikazan je asinkroni proces pomoću reda izvršavanja (eng. queue).



Sl. 3.59. Asinkroni proces u Rails aplikaciji Izvor[4]

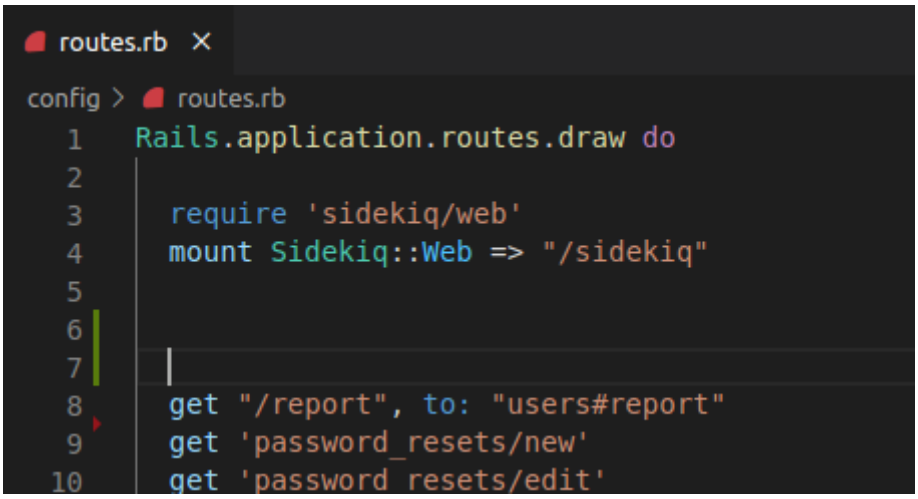
U asinkronom procesu umjesto da iz kontrolera se poziva model koji će generirati izvještaj staviti će se poruka u red za izvršavanja i odmah vratiti korisniku poruku da se izvršava njegov zahtjev i da će dobiti obavijest kasnije ili kada se posao završi. Trebamo postojati radnik (eng. worker). Radnik je zapravo Ruby objekt koji je razvijen na drugom procesu koji je radvojen od procesa koji zapravo upravlja internet zahtjevom. Radnik će pročitati posao iz reda izvršavanja i generirati izvještaj. Ovaj proces nije ograničen vremenski i možemo ga imati koliko god je potrebno. Kada proces bude izvršen može se izvršiti i nekakva akciju s podacima kao što je na primjer slanje emaila korisniku. Prvi korak je uključivanje Sidekiqa u gem file te je na slici 3.60. prikazano kako implementirati sidekiq.

Gemfile

```
1  source 'https://rubygems.org'
2  git_source(:github) { |repo| "https://github.com/#{repo}.git" }
3
4
5  #Sidekiq
6  gem 'sidekiq'
7  gem 'sinatra', github: 'sinatra/sinatra'
8
9
10 #Image Magick for resizing images
11 gem 'image_processing', '1.9.3'
12 gem 'mini_magick', '4.9.5'
13 #Active storage for image
```

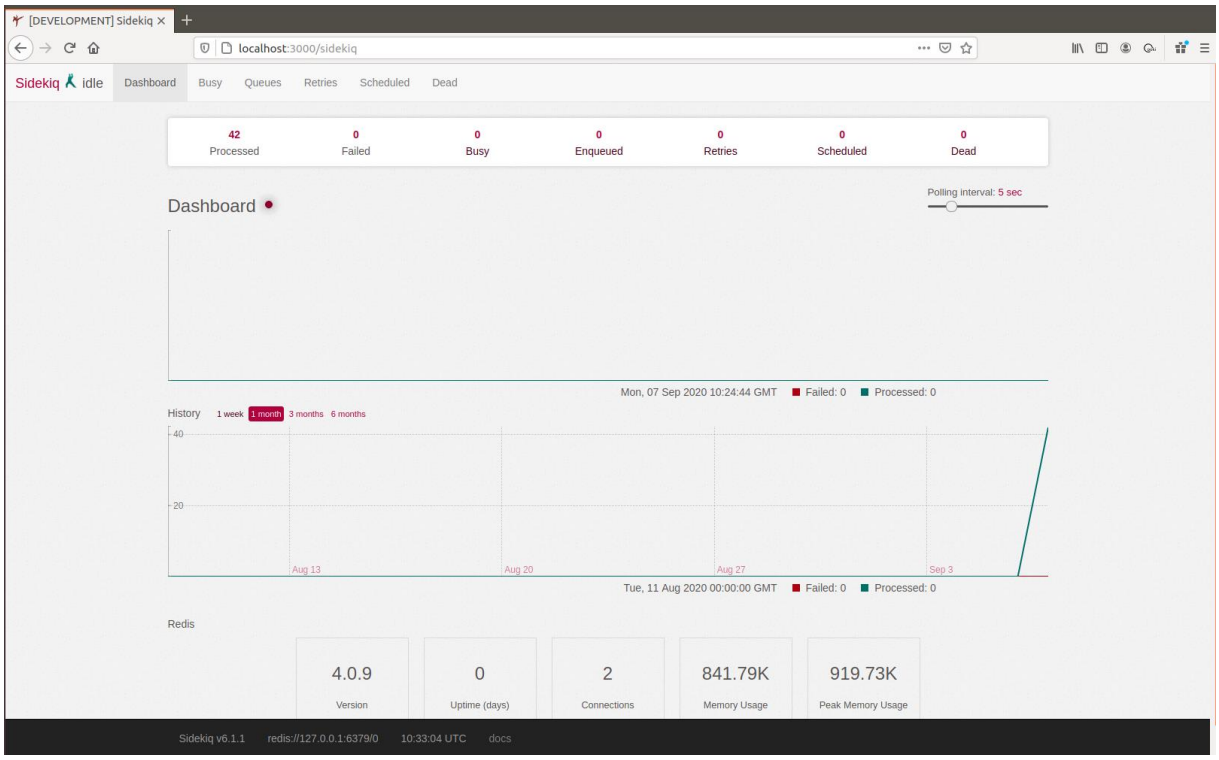
Sl. 3.60. Kod u gemfile datoteci koji prikazuje kako implementirati Sidekiq

Nakon ovoga se pokreće komanda "bundle install" u konzoli kako bi sve bilo spremno za uporabu. Redis se ne mora upisati u ***gemfile*** datoteci jer automatski sidekiq koristi redis te ga mora instalirati kako bi ga mogao koristiti. Sinatra se uključuje u gemfile radi korisničkog sučelja od Sidekiqa. Kako bi vidjeli sučelje Sidekiqa u aplikaciji moramo u ***routes.rb*** datoteci napisati kod koji prikazan na slici 3.61.



Sl. 3.61. Prikaz koda kako bi omogućili sučelje Sidekiqa u internet aplikaciji

Kada je to napravljeno nakon pokretanja servera u ruti ***/sidekiq*** se vidi sučelje Sidekiqa te je ono prikazano na slici 3.62.



Sl. 3.62. Prikaz Sidekiq sučelja

U ovom sučelju se vide podatci o poslovima kao što je na primjer koliko je poslova procesirano i koji su poslovi u redu za izvršavanja i čekaju na svoje izvršavanje. Treba se napraviti mapa ***workers*** u ***app*** direktoriju te napravi Ruby klasu koja je u ovom slučaju ***ReportWorker***, a ime datoteke je ***report_worker.rb***. Na slici 3.63. prikazan je kod datoteke ***report_worker.rb***. Prvom linijom se daje do znanje da ovaj worker pripada Sidekiqu. Ostale opcije koje se mogu dodavati su naprimjer ***retry: false*** i uobičajeno je postavljena ova opcija da se automatski opet proba izvršiti posao ako dođe do pogreške, ali u ovom slučaju smo ugasili tu opciju da ako se posao ne izvrši ili bude nekakva greška da se posao ne pokušava opet izvršiti. Svaki radnik mora imati ***perform*** metodu da se zna šta će taj posao raditi. U ovom slučaju se ispisuje u konzolu obična poruka.

report_worker.rb ×

app > workers > report_worker.rb

```
1  class ReportWorker
2      include Sidekiq::Worker
3      sidekiq_options retry: false
4
5      def perform(start_date, end_date)
6          puts "SIDEKIQ WORKER GENERATING A REPORT FROM #{start_date} to #{end_date}"
7      end
8  end
```

Sl. 3.63. Prikaz koda od workera

Kako bi se postavila poruka da je posao u redu za izvršavanja ta se funkcionalnost napravi u kontroleru. Na slici 3.64. prikazan je kod kojim dobivamo flash poruku nakon stavljanja posla u red u ***User*** kontroleru.

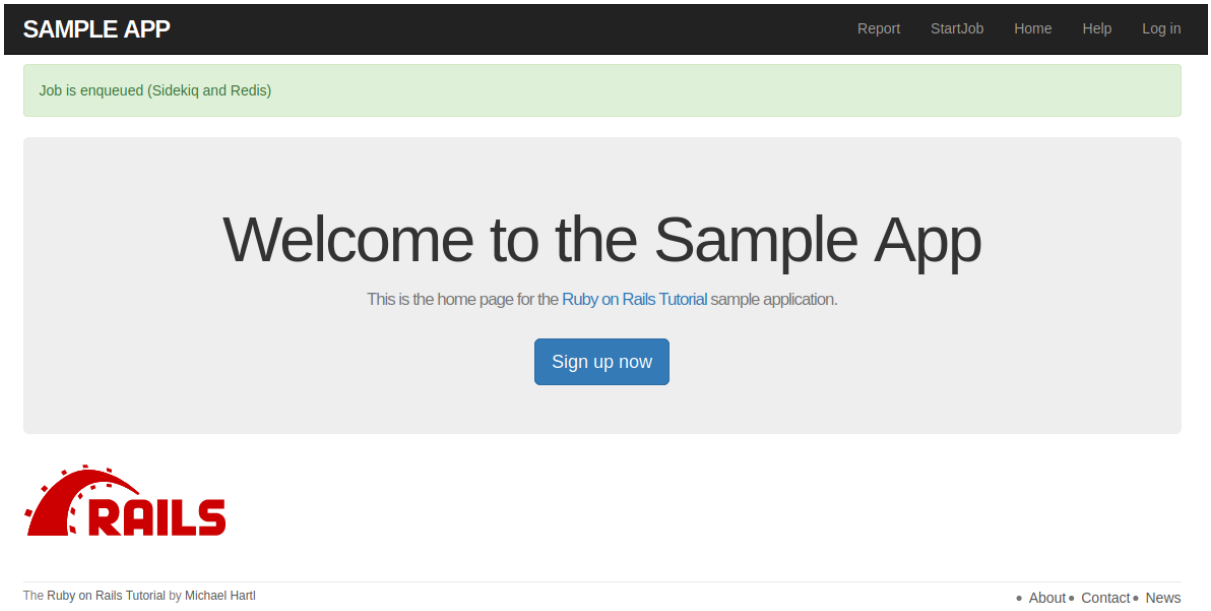
users_controller.rb ×

app > controllers > users_controller.rb

```
65
66   def start_job
67     duration = params[:duration].present? ? params[:duration] : 10
68     TestWorkloadJob.perform_later user: current_user, duration: duration
69     flash[:success] = "Job is enqueued"
70     redirect_to root_path
71   end
72
73   def report
74     #generate_report()
75     ReportWorker.perform_async("07-01-2020", "08-01-2020")
76     flash[:success] = "Job is enqueued (Sidekiq and Redis)"
77     redirect_to root_path
78   end
79
80   private
81
82   def generate_report
83     sleep 30
84   end
85
86     def user_params
87       params.require(:user).permit(:name,:email,:password,:password_confirmation)
88     end
89
90   # Before filters
91
92   # Confirms a logged-in user.
93   def logged_in_user
```

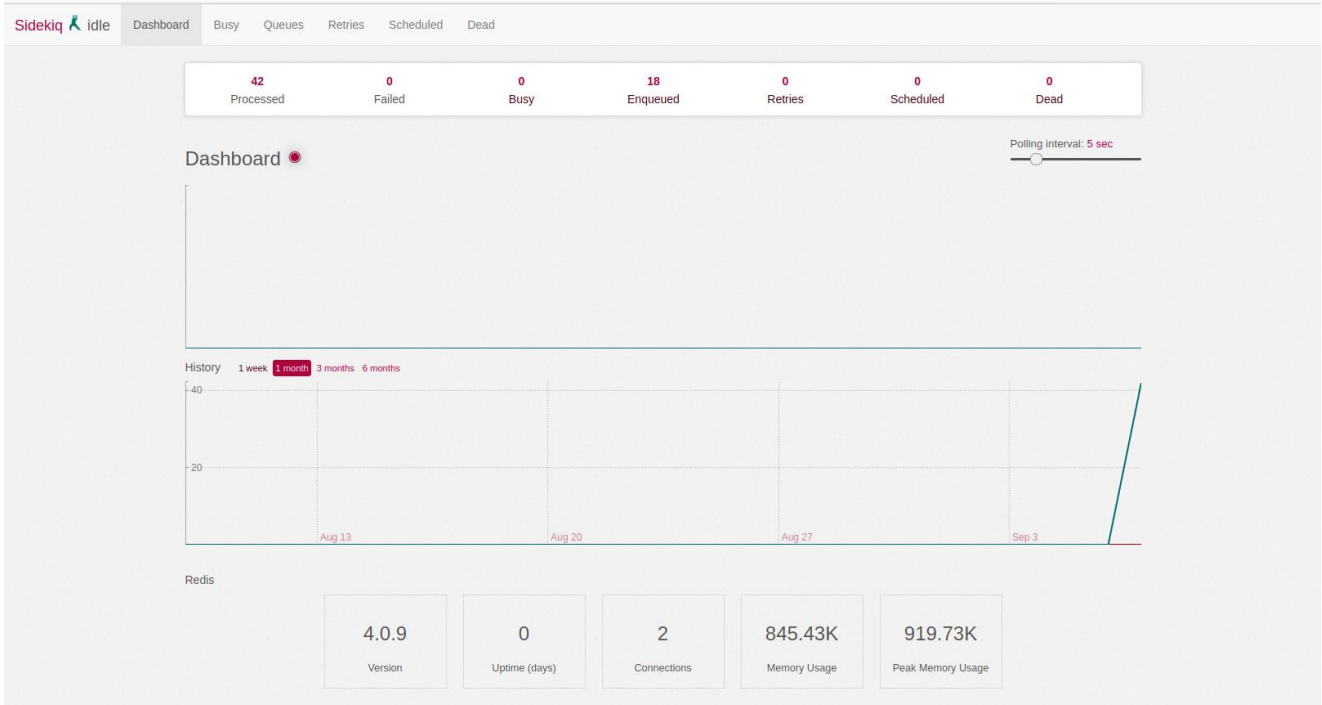
Sl. 3.64. Prikaz koda u kontroleru kako bi dobili poruku kada se posao stavi u red čekanja

Na slici 3.65. prikazan je izgled stranice te novi link u navigaciji ***`Report`*** s kojim se zapravo pokreće posao preko Sidekiq a i Redisa.



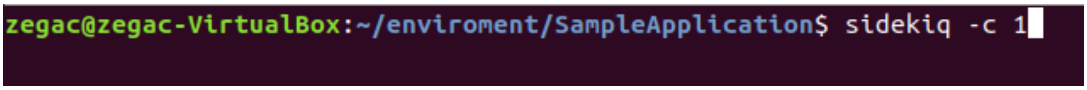
Sl. 3.65. Prikaz izgleda stranice nakon pritiska na link Report u navigaciji

Nakon pritiska više puta dugmeta "Report" na slici 3.66. se može vidjeti sučelje Sidekiqa koje pokazuje koliko je poslova smješteno u red čekanja.



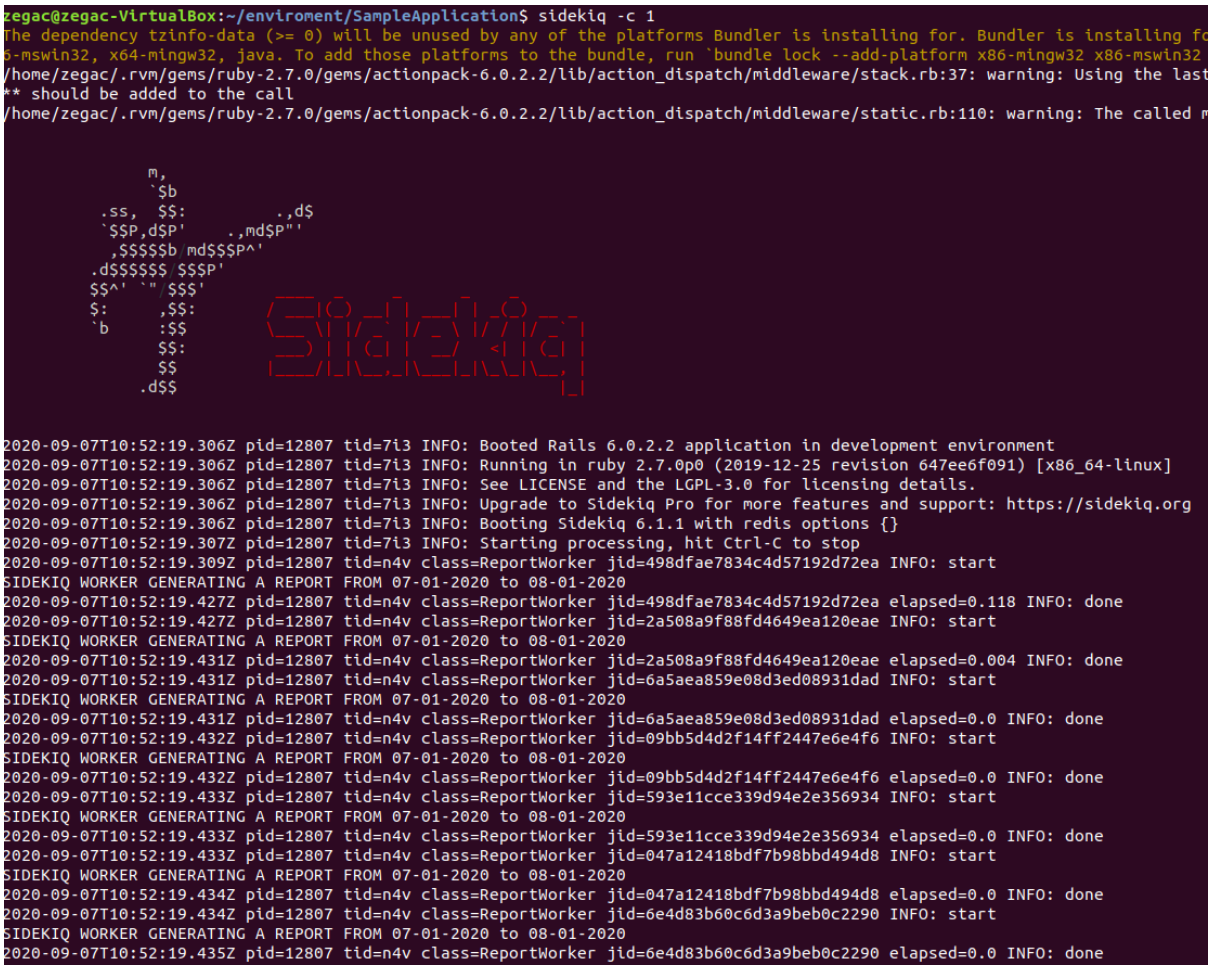
Sl. 3.66. Prikaz da je 18 poslova stavljeno u red izvršavanja

Kako bi se sidekiq pokrenuo u terminalu se upisuje komanda koja je prikazana na slici 3.67. Time će se izvršiti svi poslovi koji su na čekanju te bi se u sučelju Sidekiqa trebalo prikazati da je 60 poslova izvršeno i nijedan više ne čeka na red.

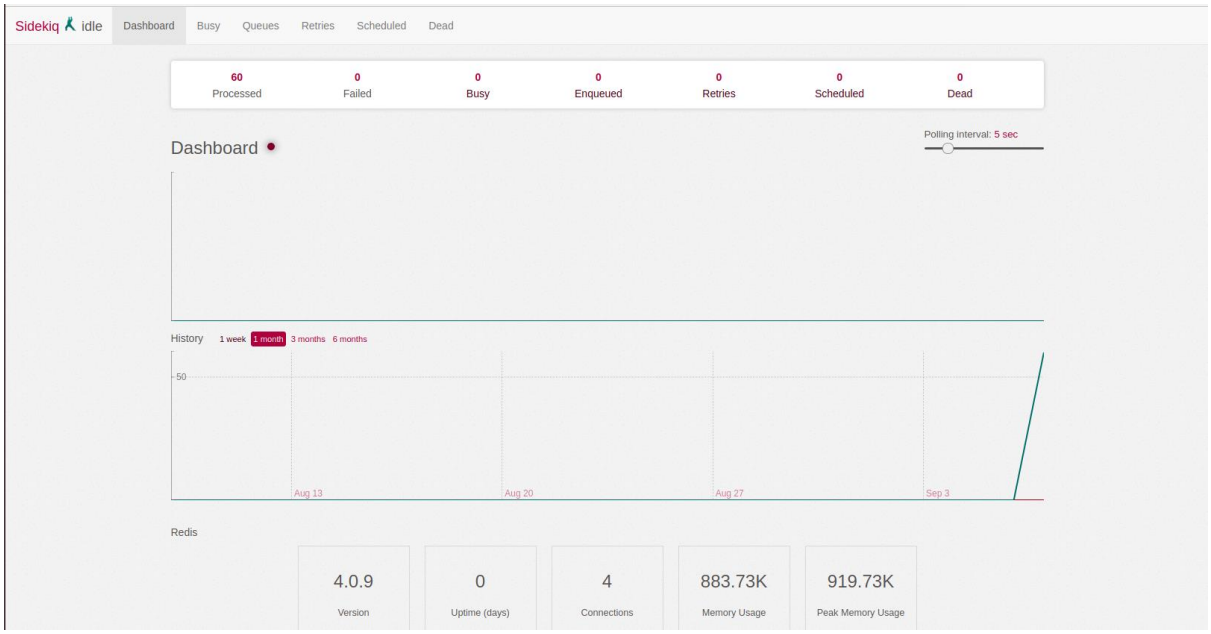


Sl. 3.67. Prikaz komande s kojom pokrećemo Sidekiq s 1 niti što je u ovom slučaju je dosta 1 nit

Nakon pokretanja komande Sidekiq izvršava posao te ispisuje poruku koja je napisana u ***report_worker.rb*** datoteci te je na slici 3.68. prikazan proces izvršavanja svih poslova u redu dok je na slici 3.69. prikazano sučelje Sidekiq-a u kojem se vidi da je 60 poslova izvršeno i više nijedan posao nije u redu čekanja te ostale informacije u poslovima. Prilikom ovog postupka Redis mora biti upaljen, a pokreće se komandom ***redis-server***.



Sl. 3.68. Prikaz izvršavanja poslova koji su bili postavljeni u redu



Sl. 3.69. Prikaz Sidekiq sučelja sa dodatnim informacijama u poslovima

4. ZAKLJUČAK

Svrha ovog završnog rada je predstaviti internetski mrežni okvir Ruby On Rails te pozadinske poslove i različite sustave s kojim se mogu isti poslovi implementirati. U sklopu rada predstavljene su tehnologije koje su korištene kako bi se mogla izraditi aplikacija u Ruby On Rails okruženju, a to su HTML, CSS, SASS, Bootstrap, Ruby i Ruby on Rails koji su služili kako bi se internet aplikacija uspješno napravila i razvila. Tehnologije kao što su Git, Github i Heroku su služile kako bi se programski kod mogao postaviti na internet te pokrenuti u produkciji.

Ruby on Rails je mrežni okvir koji je svoj vrhunac doživio 2014. godine, ali i danas je jako popularan u razvijanju internet aplikacija. Popularan je upravo zbog toga što se aplikacije mogu jako brzo razviti u odnosu na druge tehnologije. Ima svojih nedostataka kao što su skalabilnost i spore perfomanse, ali koje se tek uočavaju na velikim projektima.

Što se tiče pozadinskih poslova u Ruby on Rails okruženju, glavna svrha je predstaviti sustave za raspoređivanje poslova kao što je Sidekiq. Naime, bitna stvar je da su pozadinski poslovi vrlo zastupljeni jer neke operacije i dohvaćanje informacija znaju oduzeti mnogo vremena i sadrže veliku količinu podataka. Korisniku želimo omogućiti da kada neki posao dugo traje da mu se ne blokira cijela aplikacija prilikom izvršavanja tog posla i tako dobijemo bolje korisničko iskustvo i sretnog korisnika.

Sidekiq i Redis su najzastupljeniji kada je u pitanju Ruby on Rails okruženje, a to je zato što je i cijeli taj sustav za raspoređivanje upravo napisan u programskom jeziku Ruby i jako se dobro slaže s Rails aplikacijom. Redis je zapravo baza podataka koja sprema par ključ i vrijednost te služi da se tamo spremaju poslovi.

Autorov zaključak je da su pozadinski poslovi jako bitni u razvijanju internet aplikacija, a sustavi za raspoređivanje tih poslova su veoma korisni. Sidekiq sustav najlakši je što se tiče implementacije u Rails aplikaciju zato se on i najviše koristi. Osim jednostavnosti implementacije, Sidekiq ima prednost nad ostalim sustavima zbog njegovih perfomansi.

LITERATURA

[1] M. Slesar, Advantages of Web Development with Ruby on Rails (https://onix-systems.com/blog/advantages-of-web-development-with-Ruby-on-Rails), pristupljeno 15.06.2020

[2] M. Heartl, The Ruby on Rails Tutorial, Sixth Edition, pristupljeno 06.05.2020

[3] RailsGuides, Active Job Basics (https://guides.rubyonrails.org/active_job_basics.html), pristupljeno 20.7.2020

[4] J. Leigh, Background Processing with Rails, Redis and Sidekiq (https://www.youtube.com/watch?v=GBEDvF1_8B8), pristupljeno 15.7.2016.

[5] Wikipedia, Ruby (https://en.wikipedia.org/wiki/Ruby_(programming_language)), pristupljeno 25.8.2020

[6] Wikipeda , Ruby on Rails (https://en.wikipedia.org/wiki/Ruby_on_Rails), pristupljeno 17.9.2020

[7] Wikipeda, Model-View-Controller (https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller), pristupljeno 1.8.2020

[8] TK, Ruby on Rails: HTTP, MVC and Routes (https://medium.com/the-renaissance-developer/ruby-on-rails-http-mvc-and-routes-f02215a46a84), pristupljeno 12.7.2017

[9] RailsGuides, Active Record Basics (https://guides.rubyonrails.org/active_record_basics.html), pristupljeno 20.7.2020

[10] RailsGuides, Action View Overview (https://guides.rubyonrails.org/action_view_overview.html), pristupljeno 20.7.2020

[11] Wikipedia, Representational state transfer (https://en.wikipedia.org/wiki/Representational_state_transfer), pristupljeno 8.9.2020

[12] Upcase by thoughbot, REST (https://thoughtbot.com/upcase/videos/rest), pristupljeno 25.7.2020

[13] Restfulapi.net, REST API Tutorial (https://restfulapi.net/rest-architectural-constraints/), pristupljeno 07.07.2017

[14] J. Duckett, HTML & CSS design and build websites, pristupljeno 29.9.2014

[15] Wikipedia, Bootstrap (https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)), pristupljeno 28.9.2020

[16] V. Kanaprthi, Git and Github in a NutShell (https://codeburst.io/git-and-github-in-a-nutshell-b0a3cc06458f), pristupljeno 14.02.2018

[17] S. Rostad, What is Heroku? A Simple Explanation for Non-Techies (https://trifinlabs.com/what-is-heroku/), pristupljeno 28.02.2018

[18] Heroku, What is Heroku? (https://www.heroku.com/what), pristupljeno 20.6.2020

[19] Wikipedia, Sass (stylesheet language) (https://en.wikipedia.org/wiki/Sass_(stylesheet_language)), pristupljeno 17.9.2020

[20] Sass-lang.com, Sass Basics (https://sass-lang.com/guide), pristupljeno 18.9.2020

[21] RubyOnRails.org, Active Job adapters (https://api.rubyonrails.org/v6.0.3.2/classes/ActiveJob/QueueAdapters.html), pristupljeno 20.7.2020

[22] N. Esquenazi (github), Backburner (https://github.com/nesquena/backburner). pristupljeno 16.5.2020

[23] Collective Idea (github), Delayed Job (https://github.com/collectiveidea/delayed_job), pristupljeno 16.08.2019

[24] Que-rb (github), que (https://github.com/que-rb/que)

[25] QueueClasic (github), Queue Classic (https://github.com/QueueClassic/queue_classic), pristupljeno 4.9.2020

[26] Resque (github), Resque (https://github.com/resque/resque), pristupljeno 8.4.2020

[27] D. J. Nahum (github), Sneakers (https://github.com/jondot/sneakers), pristupljeno 30.6.2020

[28] B. Hilkert (github), Sucker Punch (https://github.com/brandonhilkert/sucker_punch), pristupljeno 8.2.2020

[29] K. Juell, How to Add Sidekiq and Redis to a Ruby on Rails Application (https://www.digitalocean.com/community/tutorials/how-to-add-sidekiq-and-redis-to-a-ruby-on-rails-application), pristupljeno 22.11.2019

[30] Wikipedia, Sidekiq (https://en.wikipedia.org/wiki/Sidekiq)

[31] M. Drake, How To Install and Secure Redis on Ubuntu 18.04 (https://www.digitalocean.com/community/tutorials/how-to-install-and-secure-redis-on-ubuntu-18-04), pristupljeno 15.9.2020

[32] M. Perham (github), sidekiq (https://github.com/mperham/sidekiq) pristupljeno 20.9.2020

[33] Wikipedia, Redis (https://en.wikipedia.org/wiki/Redis) pristupljeno 1.10.2020

[34] Redis (github), Redis (https://github.com/redis/redis) pristupljeno 2.10.2020

[35] D. Eisinger, Adding a NOT NULL Column to an Existing Table (https://www.viget.com/articles/adding-a-not-null-column-to-an-existing-table/),

pristupljeno 30.9.2014

SAŽETAK

Cilj ovog završnog rada je razmotriti metode i načine izvršavanja pozadinskih poslova u Ruby on Rails okruženju. Opisati Activejob modul i najčešće korištene sustave za izvršavanje ActiveJob poslova (Sidekiq, Backburner, Que, Sneakers). Ovaj rad pruža pregled osnovnih tehnologija za izradu internet aplikacije te razvijanje aplikacije koja je slična Twitteru. U radu su korištene tehnologije kao što su HTML, CSS, SASS, Git, Github, Heroku, Ruby i Ruby on Rails. Za pozadinske poslove je korištena integracija ActiveJob modula pomoću zadanog Ruby on Rails sustava za raspoređivanja poslova te imamo i primjer implementacije poslova pomoću sustava za raspoređivanje poslova Sidekiqa koji se koristi zajedno s Redisom.

Ključne riječi: pozadinski poslovi, Ruby on Rails, ActiveJob, Sidekiq, Redis

ABSTRACT

BACKGROUND JOBS IN RUBY ON RAILS ENVIRONMENT

The aim of this thesis is to consider methods and ways for performing background jobs in Ruby on Rails environment. Describe the ActiveJob module and most commonly used backends for execution tasks and jobs in Rails application (Sidekiq, Backburner, Que, Sneakers). This paper work provides overview of basic technologies for making web applications such as HTML, CSS, SASS, Git, Github, Heroku, Ruby and Ruby on Rails. For background jobs was used ActiveJob module with default Ruby on Rails backend and also have an example of implementing jobs using Sidekiq which is widely used with Redis.

Keywords: background jobs, Ruby on Rails, ActiveJob, Sidekiq, Redis

ŽIVOTOPIS

Josip Oužecky rođen je 19.ožujka 1998. godine, u Virovitici. Osnovno obrazovanje započinje 2005. godine upisom u Osnovnu školu Josipa Kozarca Slatina. Po završetku osnovne škole, 2013. godine nastavlja obrazovanje u Srednjoj školi Marka Marulića Slatina, smjer elektrotehnika. Stručni studij informatike upisuje 2017. godine na Elektrotehničkom fakultetu Osijek. Istraživanje provedeno tijekom izrade završnog rada izvedeno je u 2020. godini.

_____

(potpis studenta/ice)

PRILOG 1 – REPOZITORIJ NA GITHUB-U

Kompletan programski kod prezentiran u ovom radu dostupan je na ovom LINKU.