Primjena Active Recorda za rad s bazom podataka

Falak, Antonio

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij računarstva

PRIMJENA ACTIVE RECORDA ZA RAD S BAZOM PODATAKA

Završni rad

Antonio Falak

Osijek, 2016.

Sadržaj

1. UVOD

Tema završnog rada je "Primjena Active Recorda za rad s bazom podataka". U teorijskom dijelu je opisan programski jezik Ruby pomoću kojeg se koristi programsko okruženje temeljeno na Ruby jeziku po imenu Ruby on Rails. Koristit će se Ruby on Rails programsko okruženje kako bi se izradila *web* aplikacija, što je ujedno i praktični dio zadatka. Potrebno je teorijski obuhvatiti Object Relational Mapping (ORM) te ponajviše ORM biblioteku koja će se koristiti - Active Record. Praktični dio završnog rada se izvodi na operacijskom sustavu Ubuntu Linux, rad zahtijeva osnovno poznavanje shell naredbi, HTML, CSS i SQL. Alati koji će se koristiti za izradu *web* aplikacije su Bootstrap i Git (s kojim će se postaviti kod *web* aplikacije na sustav GitHub).

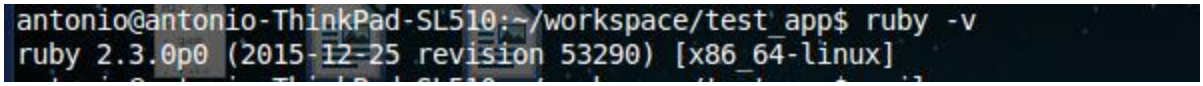
1.1 Zadatak završnog rada

U teorijskom dijelu potrebno je objasniti primjenu ORM sustava u radu s bazama podataka. Ukratko opisati ORM sustave za rad s najpoznatijim programskim jezicima. Detaljno opisati Active Record gem za rad s Ruby programskim jezikom. U praktičnom dijelu rada potrebno je izraditi jednostavnu web aplikaciju u Ruby on Rails okruženju koja će koristiti Active Record za rad s bazom te na primjeru razvijene aplikacije prikazati razlike u radu s bazom pomoću ORM sustava u odnosu na ručno pisane SQL upite. Tehnologije: Ruby on Rails Sumentor je Bruno Bušić, mag.ing.comp., Bamboo Lab, Osijek Sumentor je i Hrvoje Leventić, mag.ing.comp..

2. PROGRAMSKI JEZIK I OKRUŽENJE

2.1 Ruby

Dinamičan, *open source* programski jezik usredotočen na jednostavnost i produktivnost. Posjeduje elegantnu sintaksu koja je prirodno razumljiva i lako zapisiva. Tvorac Yukihiro "Matz" Matsumoto, izmiješao je dijelove svojih omiljenih programskih jezika (Perl, Smalltalk, Eiffel, Ada i Lisp) kako bi formirao novi jezik koji uravnotežuje funkcijsko programiranje i imperativno programiranje [1]. Ruby je izvorno objektno orijentirani programski jezik. Sve s čime se manipulira je objekt, rezultati tih manipulacija su također objekti. Ruby programi pisani su u 7-bitnom ASCII (Ruby raspolaže s opširnom podrškom za Kanji, koristeći EUC, SJIS, ili UTF-8 sustav kodiranja). Ruby je linijsko-orijentiran jezik. Ruby izrazi i izjave se završavaju na kraju linije osim ako je izjava očigledno nepotpuna, npr.ako je zadnji znak u liniji operator ili zarez. Točka-zarez se koristi za odvajanje višestrukih izraza u liniji. Također se može staviti obrnuta kosa crta na kraj linije kako bi se nastavilo u sljedećoj liniji. Komentari započinju sa '#' i protežu se do kraja fizičke linije, tijekom kompilacije se zanemaruju. Ruby se svrstava među 10 najboljih u većini popisa koji mjere rast i popularnost programskih jezika širom svijeta. Trenutna stabilna verzija (u vrijeme pisanja završnog rada) je 2.3.0.



Sl. 2.1. Ruby -v

RubyGems je paketni upravitelj za Ruby programski jezik koji pruža standardni format za distribuciju Ruby programa i biblioteka (u samostalnom formatu zvanom "gem"), alat dizajniran za laku instalaciju gem-ova te server za njihovu distribuciju. RubyGems kreiran je u studenom 2003. godine i sada je dio standardne biblioteke od Ruby verzije 1.9. Sadrže informacije o paketima s datotekama za instalaciju. Gem-ovi su obično izgrađeni od ".gemspec" datoteka, koje su YAML datoteke koje sadrže informacije o gem-ovima. Međutim, Ruby kod može izravno izraditi gem-ove. Takva praksa se uobičajeno koristi s Rake (*Ruby + make = Rake*). Svaki gem ima ime, verziju i platformu, npr. rake gem posjeduje verziju 11.2.2, platforma mu je Ruby, što znači da se pokreće na svakoj platformi gdje se Ruby pokreće [2].

Ruby je " interpretirani skriptni jezik za brzo i jednostavno objektno orijentirano programiranje" - što to znači? [3]

Interpretirani skriptni jezik:

- Mogućnost direktnog obavljanja sistemskih poziva
- Moćne string operacije i regularni izrazi
- Neposredni podaci o rezultatima tijekom razvoja

Brz i jednostavan:

- Deklaracija varijabli je nepotrebna
- Varijable nisu upisane
- Sintaksa je jednostavna i konzistentna
- Automatsko memorijsko upravljanje

Objektno orijentirano programiranje:

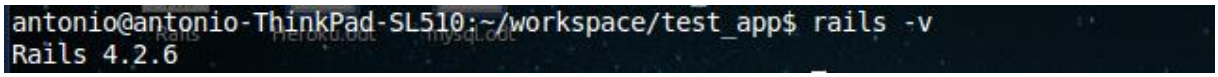
- Sve je objekt
- Klase, metode, nasljeđivanje, itd.
- *singleton* metode
- "mixin" funkcionalnost po modulu
- Iteratori i zatvarači

Također:

- *Integer*-i višestruke preciznosti
- Prikladno procesiranje iznimki
- Dinamično učitavanje
- Nitna podrška

2.2 Ruby on Rails

Rails je *web* aplikacijsko razvojno okruženje napisano u Ruby jeziku. Dizajnirano je da učini programiranje *web* aplikacija jednostavnijim pravljenjem pretpostavki o tome što svaki *developer* treba za početak izrade. Ruby on Rails je u potpunosti *open-source*, dostupan pod MIT licencom, kao rezultat ne košta ništa za preuzimanje i korištenje. Omogućava korisniku pisanje manje koda dok ostvaruje više od mnogih drugih jezika i okruženja. Rails je *Model-View-Controller* (MVC) okruženje, pružajući zadanu strukturu bazi podataka, *web* servisa i *web* stranica. Potiče i olakšava korištenje *web* standarda poput JSON ili XML za prijenos podataka te HTML, CSS i JavaScript za prikaz i korisničku razmjenu. Od svojeg prvog nastupa 2004., Ruby on Rails je brzo postao jedan od najmoćnijih i najpopularnijih alata za izgradnju dinamičkih *web* aplikacija. Rails se korisni od strane raznovrsnih kompanija poput Airbnb, Basecamp, Disney, GitHub, Hulu, Twitch, Kickstarter, Shopify, Twitter te Yellow Pages.[4]



Sl. 2.2. Rails -v

Rails filozofija uključuje 2 vodeća principa:

- ***Don't Repeat Yourself*:** DRY je princip izrade *software*-a koji kazuje da svaki dio znanja mora imati jedinstven, nedvosmislen, mjerodavan prikaz unutar sistema. Ne pišući istu informaciju iznova, kod je učinkovitije održiv, proširiv i manje podložan greškama.

- ***Convention Over Configuration*:** Rails obuhvaća mišljenja o najboljem načinu izvođenja mnogih zadataka u *web* aplikaciji te se zadaje na te *setove* skupova, radije nego da zahtijeva specificiranje svakog detalja kroz beskrajne konfiguracijske datoteke

Rails se također brzo prilagođava novim razvojima u web tehnologijama i dizajnu okruženja. Npr. rails je jedan od prvih okruženja koji je sistematizirao i implementirao REST stil izgradnje za strukturiranje *web* aplikacija.

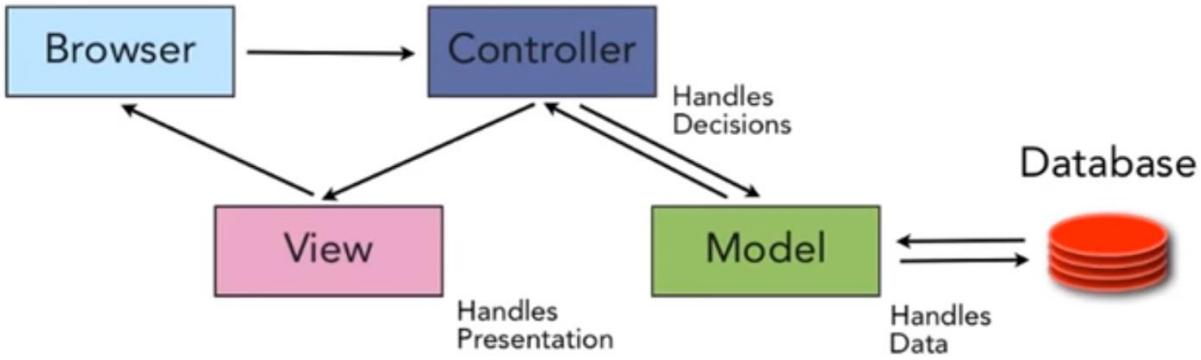
Model-View-Controller (MVC) je strukturalni obrazac koji razdvaja aplikaciju na tri glavne logičke komponente: *model*, *view* i *controller*. Svaka od ovih komponenata je izgrađena da rukuje specifičnim razvojnim aspektima aplikacije. MVC je jedan od najčešće korištenih industrijskih standarda *web* razvojnog okruženja za izradu proširivih projekata promjenjivih veličina.

***Model*:** komponenta model odgovara svoj podatkovnoj logici s kojom korisnici rade. Može predstavljati podatke koji se prenose između prikaz (engl. *View*) i kontroler (engl. *Controller*) komponenti ili neku drugu poslovnu logiku podataka. Npr. objekt naziva "Kupac" će vratiti informacije kupca iz baze podataka, manipulirat će i ažurirati podatke natrag u bazu podataka ili će izraziti (engl. *render*) podatke. Klase koje se koriste za spremanje i manipuliranje stanja, obično u nekoj vrsti baze podataka.

***View* (prikaz):** prikaz komponenta se koristi za svu UI logiku aplikacije. Npr. prikaz naziva "Kupac" sadržavao bi sve UI komponente poput tekstualnih okvira, padajućih izbornika i ostale komponente s kojima konačni korisnik ima interakciju.

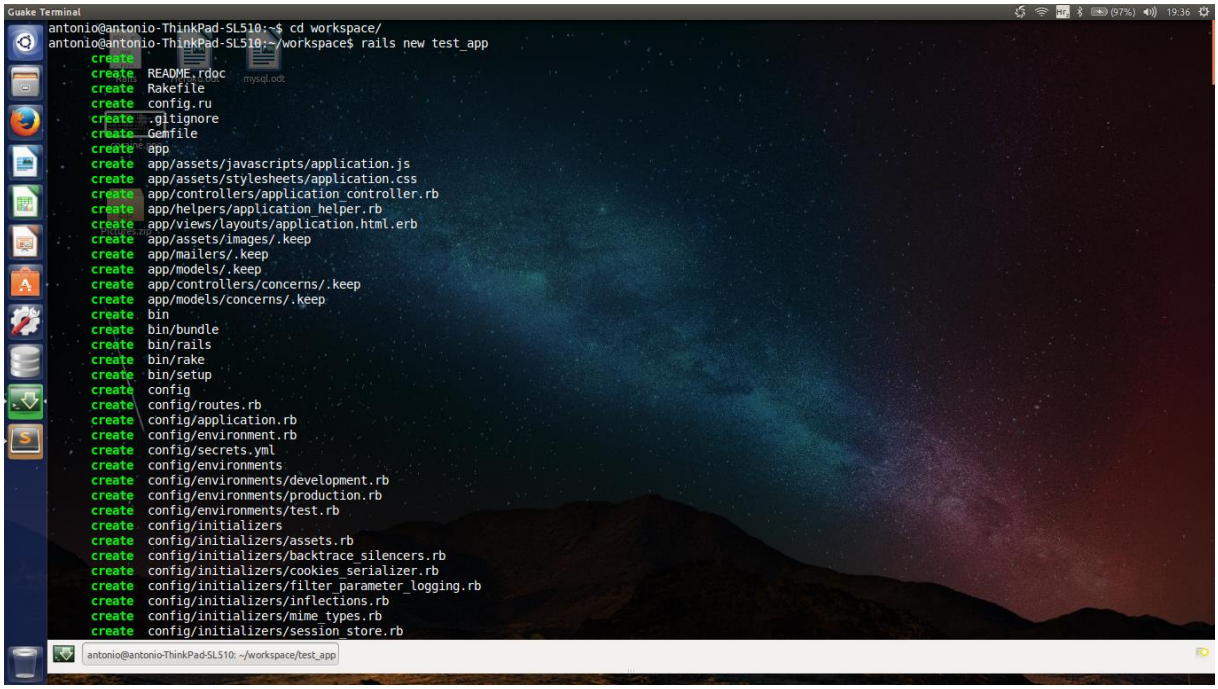
***Controller* (kontroler):** Mozak aplikacije. Kontroler komponenta odlučuje koji je korisnikov ulaz, kako se model treba ponašati u ovisnosti o tom ulazu te koji rezultirajući prikaz će se iskoristiti. Kontroler se ponaša poput sučelja između model i prikaz komponente kako bi

procesirao svu poslovnu logiku i nadolazeće zahtjeve, manipulirao podacima pomoću model komponente i komunicirao s prikaz komponentama za pregled konačnih izlaza. Npr. kontroler "Kupac" rukuje sa svim interakcijama i ulazima od strane prikaza "Kupac" i ažurira bazu podataka koristeći model "*Kupac*". Isti kontroler služi za pregled podataka kupca.



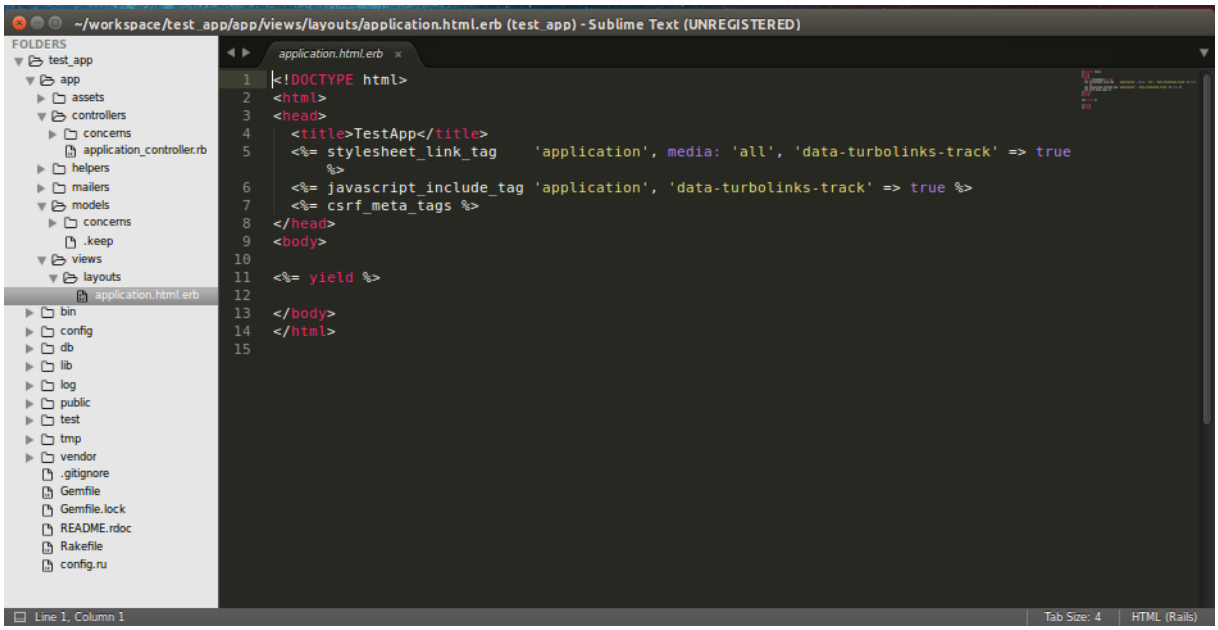
Sl. 2.3. MVC arhitektura [5]

Rails je zapravo Ruby gem, koji efikasno funkcionira s drugim gem-ovima kako bi omogućio platformu za programiranje. Čak i za iskusne Rails *developere*, instalacija Ruby-a, Rails-a i svih povezanih programa podrške, može biti frustrirajuća. Problem je u mnoštvu okruženja: drukčiji operacijski sustavi, verzije, prioritet tekstualnog uređivača (engl. *text editor*) i integriranog razvojnog okruženja (engl. *Integrated Development Environment* - IDE) , itd. Nakon instaliranja potrebnog *software*-a, poželjno je kreirati direktorij "workspace" (*$ mkdir workspace, $ cd workspace*) te u njemu generirati rails projekt (*$ rails new ime_projekta*).



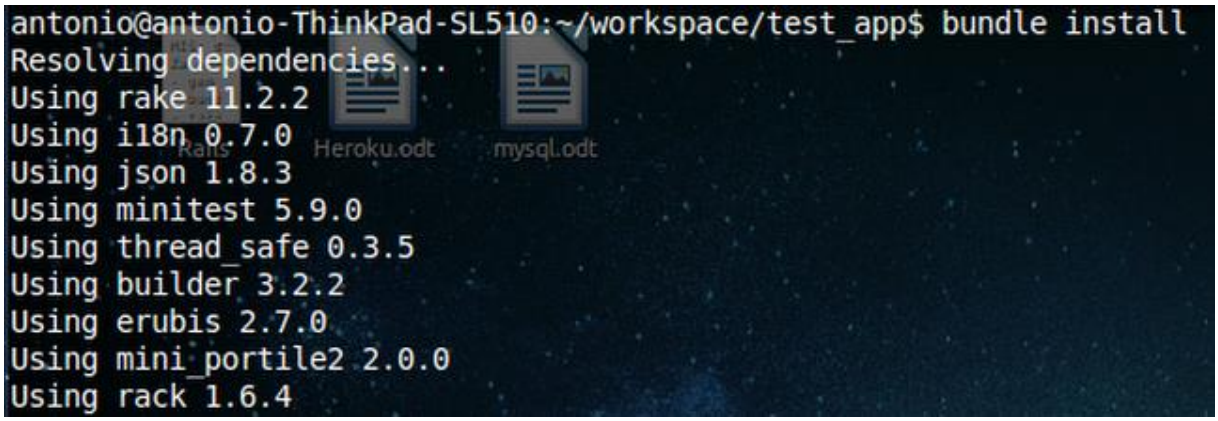
Sl. 2.4. Rails new naredba

Uočljivo je koliko mnogo datoteka i direktorija stvori rails naredba. Standardni direktorij i struktura podataka jedna je od mnogih prednosti Rails-a, odmah gradi od nule do funkcionalne (ali minimalne) aplikacije. Štoviše, zato što je struktura zajednička svim Rails aplikacijama, moguće je smjesta shvatiti funkcionalnost gledajući u tuđi kod.

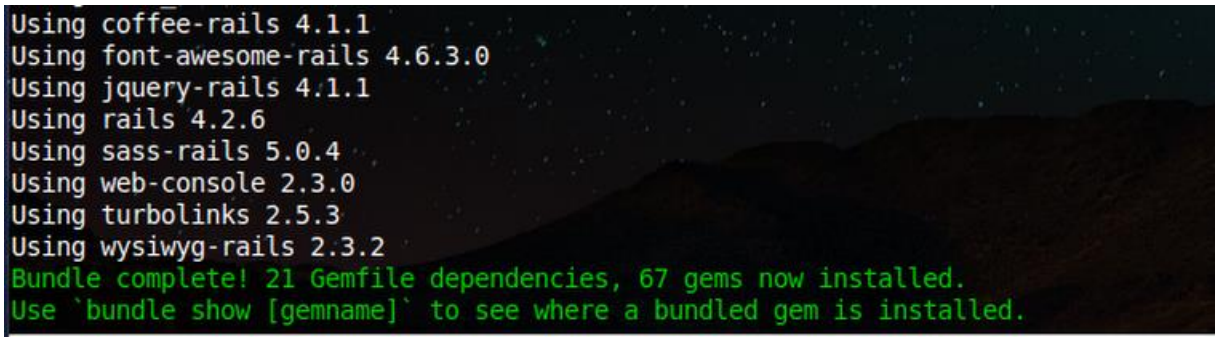


Sl 2.5. Generirani rails direktoriji i datoteke

U datoteci **Gemfile** se dodaju gem-ovi ključni za projekt (gem 'sqlite3', gem 'uglifier', gem 'coffee-rails' i svi drugi potrebni gem-ovi, što ovisi o vrsti željenog projekta), zatim se instaliraju naredbom *$ bundle install*.

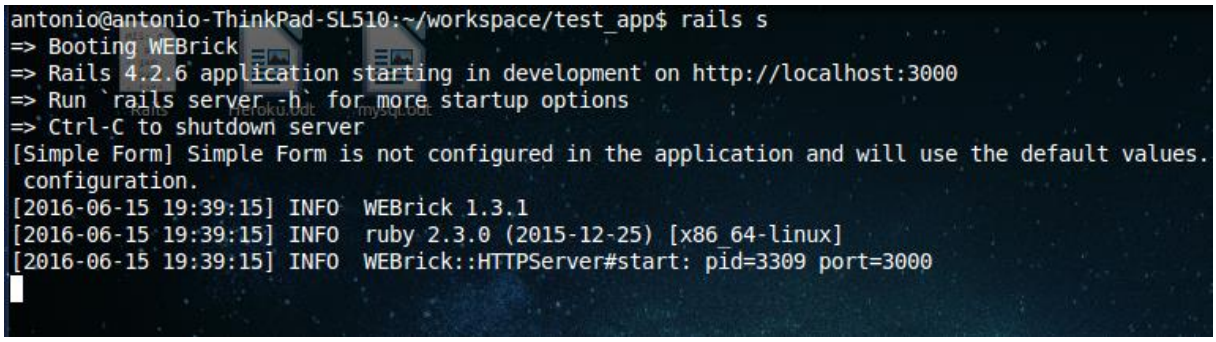


Sl. 2.6. Bundle install naredba



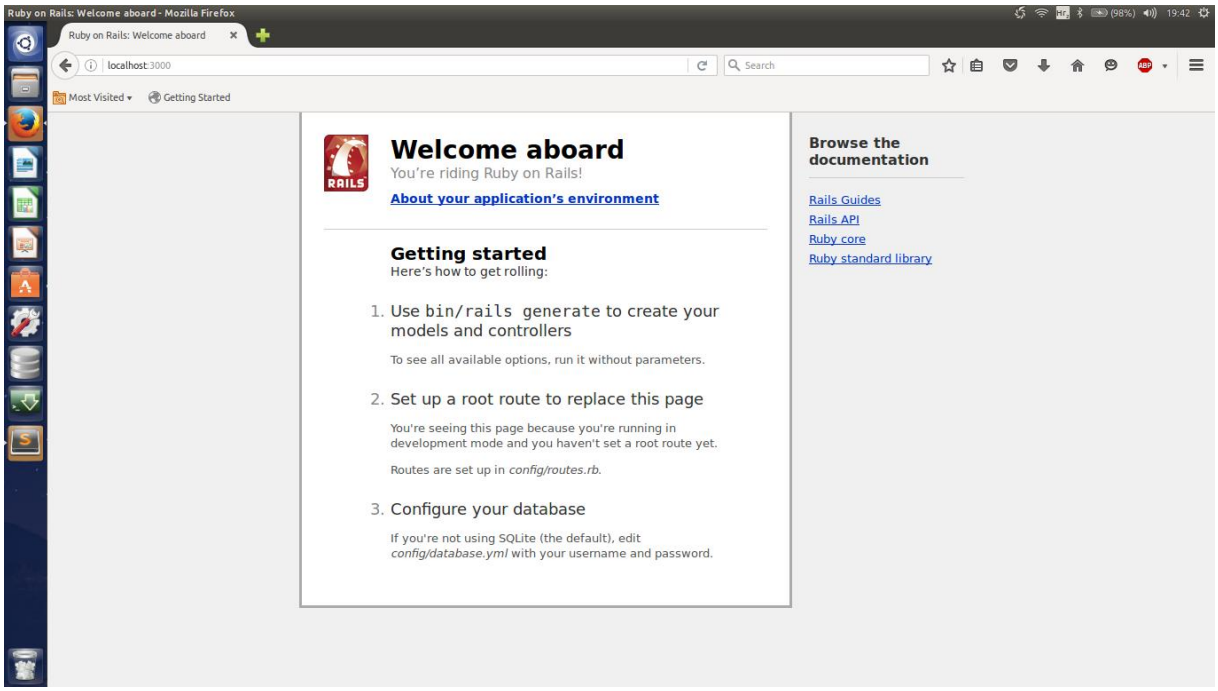
Sl. 2.7. Bundle install naredba

Sljedeće, vrši se ulaz u direktorij novostvorenog Rails projekta (*$ cd workspace/ime_projekta*) i pokreće se server na *port*-u 3000 (*$ rails server*).



Sl. 2.8. Rails server naredba

Ako se *web* pretraživačem posjeti adresa localhost:3000, trebala bi se pojaviti sljedeća stranica.



Sl. 2.9. Localhost:3000

3. ORM (Object Relational Mapping)

ORM akronim predstavlja *Object-Relational-Mapping* (objektno-relacijska preslikavanja). U osnovi znači da Active Record preuzima podatke spremljene u tablici baze podataka koristeći stupce i redove, koje treba modificirati ili dohvatiti pisanjem SQL izraza (ako se koristi SQL baza podataka) te omogućava interakciju s podacima na način kao da se upravlja normalnim Ruby objektima. *Object-relational mapping* (*ORM, O/RM i O/R mapping*) u računalnom *software*-u je programska tehnika za pretvaranje podataka između inkompatibilnih sistema u relacijskim bazama i objektno orijentiranog programskog jezika. Ovo stvara, ustvari, "virtualnu bazu objekata" koja se koristi unutar programskog jezika. Postoje besplatni i komercijalni paketi koji obavljaju objektno-relacijska preslikavanja, iako se neki programeri odlučuju za stvaranje vlastitih ORM alata. Npr. (unutar Rails-a) umjesto konstruiranja SQL upita za pronalaženje prvog korisnika unutar *users* tablice, moguće je učiniti sljedeće:

```
$ User.first
```

Što vraća instancu *User* modela s atributima prvog korisnika unutar tablice.

Objekti baze podataka izgledaju poput objekta programskog jezika. Često, sučelje za proizvode objektno-relacijskog preslikavanja je identično sučelju od baze objekata. *Object-relational mapping* proizvodi se trebaju koristiti kad se nastoji iskoristiti transparentna dosljednost i korištenje relacijske baze. Relacijska baza može biti potrebna zato što:

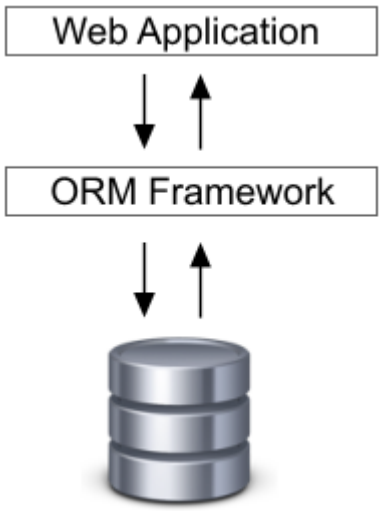
- Su podaci već unutar jedne ili više relacijske baze podataka
- Podaci su novi, ali postoje tehnički ili politički razlozi za korištenje relacijske baze podataka

U bilo kojem slučaju, objektno-orijentirano preslikavanje reducirat će programski kod i kroz *caching* će poboljšati performanse naspram korištenja ugrađenog SQL-a ili sučelja na razini poziva s upraviteljem relacijske baze podataka [6].

U usporedbi s tradicionalnim tehnikama izmjene između objektno orijentiranog jezika i relacijske baze, ORM često smanjuje količinu koda kojeg se treba upisati. Nedostatci ORM alata općenito proizlaze iz visoke razine apstrakcije koja čini nerazumljivim što se točno događa u implementacijskom kodu. Također, veliko oslanjanje na ORM *software* je navedeno kao glavni faktor u proizvodnji loše dizajniranih bazi podataka.

Postoje brojni nedostatci ORM-a, uključujući:

- Neslaganje impedancije
- Potencijalno smanjene performanse
- Prebacivanje kompleksnosti s baze podataka u kod aplikacije



Sl. 3.1. Princip ORM-a [6]

ORM-ovi također teoretski omogućavaju prebacivanje aplikacije između različitih relacijskih baza podataka. Npr. *developer* može koristiti SQLite za lokalno razvijanje, ali MySQL u produkciji. Produkcijska aplikacija se može prebaciti s MySQL na PostgreSQL bazu uz minimalne modifikacije u kodu.

Neki od poznatijih ORM *software*-a za popularne programske jezike su:

- C++ - LiteSQL, ODB, Wt::Dbo, QxOrm
- Java - Hibernate, ActiveJDBC, ActiveJPA, JOOQ, Toplink, Torque...
- .NET - Castle ActiveRecord, Nhibernate, Entity Framework, DataObjects.NET...
- PHP - CakePHP, CodeIgniter, Doctrine, FuelPHP, Redbean, Yii...
- Python - Django, PeeWee, SQLAlchemy, SQLObject, PonyORM...
- Ruby - Active Record

3.1 Active Record

Najvažnije mjesto gdje je logično razmišljanje potrebno kod izgradnje *web* stranice leži u pravilnom konstruiranju modela podataka. Podaci su temelj skoro svih značajnih *web* aplikacija, od jednostavnog bloga do Facebook-ove masivne mreže podataka. Active Record je M u MVC tj. *Model*, sloj koji je zadužen za prikaz poslovnih podataka i logike. Active Record olakšava stvaranje i korištenje poslovnih objekata čiji podaci zahtijevaju dosljedno skladištenje u bazi podataka. Implementacija Active Record obrasca je opis objektno-relacijskog preslikavanja. U Active Record-u objekti nose dosljedne podatke i ponašanja koja upravljaju tim podacima. Active record nosi mišljenje da omogućavanje logike podatkovnog pristupa kao dio objekta će educirati korisnike objekta kako pisati i čitati iz baze podataka. Objektno-relacijsko preslikavanje (ORM) je način povezivanja objekata aplikacije s tablicama u relacijskom DBMS. Koristeći ORM, svojstva i veze objekta u aplikaciji mogu lako biti spremljene i dohvaćene iz baze bez direktnog pisanja SQL upita i s manje cjelokupnog pristupnog koda baze podataka. Active Record daje nekoliko mehanizama, a najvažniji imaju mogućnosti:

- Reprezentacije modela i njihovih podataka
- Reprezentacije asocijacija između modela
- Reprezentacije hijerarhije nasljeđivanja kroz srodne modele
- Validacije modela prije njihovog spremanja u bazu podataka
- Izvršavanje operacija baze na objektno orijentiran način

Kod pisanja aplikacija koristeći druge programske jezike ili okruženja, može biti potrebno pisati puno konfiguracijskog koda. To je naročito istina za ORM okruženja općenito. Međutim, ako se prate konvencije usvojene u Rails-u, zahtijeva se pisanje vrlo male količine koda (u nekim slučajevima nimalo) u stvaranju Active Record modela. Ideja je ta da ako se aplikacije konfiguriraju na isti način u većini slučajeva, tada je to zadani način. Prema tome, eksplicitna konfiguracija bi bila potrebna samo u onim slučajevima gdje nije moguće pratiti standardne konvencije [7].

Zadano, Active Record koristi konvencije imenovanja kako bi našao preslikavanje između modela i tablica baze podataka. Rails će pretvoriti naziv klase u množinu kako bi našao dotičnu tablicu. Stoga, za klasu "Book" treba postojati tablica u bazi podataka imena "books". Rails pluralizacijski mehanizam je veoma moćan te je u mogućnosti je načiniti

množinu ili jedninu regularnih ili iregularnih riječi (engleski jezik). Postoji internacionalizacijski gem imena "I18n" (počinjući od Rails -v 2.2) koji omogućava prevođenje aplikacije na alternativni jezik. Pri korištenju imena klasa koja sadrže više od jedne riječi, ime modela mora pratiti Ruby konvencije, koristeći *CamelCase* oblik, dok imena tablica trebaju sadržavati riječi odvojene podvlakama.

Model / Class	Table / Schema
Article	articles
LineItem	line_items
Deer	deers
Mouse	mice
Person	people

Sl 3.2. Preslikavanje model-tablica

Active Record koristi konvencije imenovanja za stupce u tablicama baze, u ovisnosti o namjeni tih stupaca.

- **Strani ključevi** - ova polja se imenuju prateći obrazac jednina_imena_tablice_id (npr. item_id, order_id). Ova polja će Active Record tražiti kada se stvore asocijacije između modela.
- **Primarni ključevi** - zadano, Active Record će koristiti integer stupac imena id kao primarni ključ tablice. Koristeći Active Record migracije za stvaranje tablica, ovaj stupac se stvara automatski.

Postoje i neobavezna imena stupaca koja daju dodatne značajke Active Record instancama:

- *created_at* - automatski postavlja trenutni datum i vrijeme kada se zapis prvi put stvara
- *updated_at* - automatski postavlja trenutni datum i vrijeme kada se zapis ažurira
- *lock_version* - dodaje optimistično zaključavanje modela
- *type* - specificira da model koristi jednotablično nasljeđivanje
- *(association_name)_type* - sprema tip za polimorfne asocijacije
- *(table_name)_count* - koristi se za *caching* broja pripadajućih objekata na asocijaciji

Kreiranje Active Record modela je lako, sve što je potrebno napraviti je podklasa ActiveRecord::Base klase:

```
class Product < ActiveRecord::Base
end
```

Ovo će kreirati model imena *Product*, preslikanog u tablicu *products* u bazi podataka. Radeći ovo, također je moguće preslikati stupce svakog reda tablice s atributima instanci modela. Pretpostavit će se da je products tablica kreirana putem SQL-a poput:

```
CREATE TABLE products (
        id int(11) NOT NULL auto_increment,
        name varchar(255),
        PRIMARY KEY  (id)
);
```

Prateći shemu tablice, Active Record omogućava pisanje koda poput:

```
p = Product.new
p.name = "Some Book"
puts p.name # "Some Book"
```

3.2 CRUD

CRUD je akronim za četiri glagola koja se koriste za rukovanje podacima*: Create, Read, Update* i *Delete* (stvori, čitaj, ažuriraj i obriši). Active Record automatski stvara metode omogućavajući aplikaciji čitanje i manipuliranje podacima spremljenih unutar tablica.

Active Record objekti se mogu kreirati iz *hash-a*, bloka ili im se atributi ručno postavljaju poslije stvaranja. Metoda *new* vraća novi objekt dok metoda *create* vraća objekt i sprema ga u bazu podataka [7].

Npr. za zadani *User* model s atributima *name* i *occupation, create* metoda će stvoriti i spremiti zapis u bazu podataka:

```
user = User.create(name: "Ivan", occupation: "Doktor medicine")
```

Koristeći metodu *new*, objekt se može instancirati bez da se spremi u bazu:

```
user = User.new
user.name = "Ivan"
user.occupation = "Doktor medicine"
```

Pozivanjem *user.save* zapis se sprema u bazu podataka.
Konačno, ako postoji blok, obje metode *create* i *new* će prepustiti novi objekt bloku za inicijalizaciju:

```
user = User.new do |u|
        u.name = "Ivan"
        u.occupation = "Doktor medicine"
end
```

Active Record pruža bogat API za pristup podacima unutar baze podataka. Navedeno je nekoliko primjera metoda za pristup podacima danih Active Record-om.

```
#vrati kolekciju svih instanci klase User i spremi ih u users
users = User.all
#vraća prvog user-a
user = User.first

#vrati prvog user-a pod imenom Mislav
mislav = User.find_by(name: 'Mislav')

#pronađi sve user-e imena Mislav koji koji su po zanimanju Novinar te ih sortiraj po
#created_at suprotnim kronološkim slijedom
users = User.where(name: 'Mislav', occupation: 'Novinar').order(created_at: :desc)
```

Jednom kad se Active Record objekt vrati, atribute je moguće modificirati i spremiti nazad u bazu podataka.

```
user = User.find_by(name: 'Mislav')
user.name = 'Goran'
user.save
```

Još kraće, moguće je koristiti *hash* preslikavanje atributa imena na željenu vrijednost, poput:

```
user = User.find_by(name: 'Mislav')
user.update(name: 'Goran')
```

Najkorisnije je kada se ažurira više atributa istovremeno. Ako je potrebno ažurirati nekoliko zapisa naveliko, primjerena je *update_all* klasna metoda:

```
User.update_all "max_login_attempts = 3, must_change_password =
'true'"
```

Na isti način, jednom dohvaćen, Active Record objekt se može uništiti što će ga obrisati iz baze podataka.

```
user = User.find_by(name: 'Mislav')
user.destroy
```

Active Record dopušta potvrđivanje stanja modela prije nego se spremi u bazu podataka. Postoji nekoliko metoda za provjeru modela i provjeru da neki atribut nije prazan, da je jedinstven i da već ne postoji u bazi, prati određeni format,itd.

Validacija je veoma važno pitanje za proučavanje u vezi ustrajnosti baze, tako da metode *save* i *update* uzimaju u obzir pri izvođenju metodu vraćanja *false* pri neuspješnoj validaciji i ne izvršavanje operacija nad bazom podataka. Sve navedene metode imaju svoj *bang* duplikat (*save!* i *update!*), koje su strože tako da podižu iznimku ActiveRecord::RecordInvalid pri neuspješnoj validaciji. Primjer:

```
class User < ActiveRecord::Base
validates :name, presence: true
end

user = User.new
user.save  # => false
user.save! # => ActiveRecord::RecordInvalid: Validation failed:
Name can't be blank
#klasa User zahtijeva da atribut name ne bude prazan, novostvoreni objekt nije instanciran s
#nikakvim imenom, stoga se ne može spremiti u bazu podataka
```

Active Record povratni pozivi omogućavaju pridavanje koda određenim događajima u životnom ciklusu modela. Ovo omogućava dodavanje ponašanja modelima kroz transparentno izvršavanje koda kada se ti događaji dogode, poput stvaranja novog zapisa, njegovog ažuriranja, uništavanja, itd.

3.3 Active Record migracije

Migracije su odlika Active Record-a koja omogućava evoluiranje baze podataka kroz vrijeme. Umjesto pisanja modifikacijskih shema u SQL-u, migracije dopuštaju korištenje jednostavnog Ruby DSL-a za opisivanje tabličnih promjena. Migracije su prikladan način za promijeniti shemu baze kroz vrijeme na dosljedan i jednostavan način. Koriste Ruby DSL tako da nije potrebno ručno pisati SQL upite, omogućavajući shemi i promjenama da budu neovisni o bazi podataka. Migracija je u osnovi skripta koja govori Rails-u kako postaviti ili promijeniti bazu podataka. Sasvim drugi dio Active Record magije omogućava izbjegavanje ručnog pisanja SQL koda za stvaranje tablica. Migracija je samo skripta, kako reći Rails-u da pokrene tu skriptu te zapravo izvrši kod za stvaranje tablica i ažurira shemu baze podataka? Koristeći naredbu *$ rake db:migrate*, koja pokreće sve migracije koje se nisu izvršile. Rails zna koje migracije su izvršene jer prati pokrenute migracije (koristeći vremenske oznake) u pozadini. Kada se naredba pokrene, Rails će izvršiti odgovarajući SQL kod za postavljanje tablica baze te se korisnik može vratiti izgradnji *web* stranice [8].

Svaka migracija se može smatrati kao nova "verzija" baze podataka. Shema započinje bez ikakvog sadržaja te ju svaka migracija modificira pri dodavanju ili uklanjanju tablica, stupaca ili unosa. Active Record zna kako ažurirati shemu tijekom vremenskog perioda, dovodeći ju iz bilo koje točke u prošlosti u konačnu verziju. Active Record također ažurira db/schema.rb datoteku da odgovara konačnoj strukturi baze podataka.

Primjer migracije:

```
class CreateProducts < ActiveRecord::Migration[5.0]
  def change
    create_table :products do |t|
      t.string :name
      t.text :description
     t.timestamps
    end
  end
end
```

Ova migracija dodaje tablicu zvanu products sa *string* stupcem zvanim *name* i s *text* stupcem zvanim *description*. Implicitno će se stvoriti stupac koji je primarni ključ tablice zvan *id*, koji je zadani primarni ključ za sve Active Record modele. Macro vremenske oznake dodaje dva stupca, *created_at* i *updated_at*. Ovi posebni stupci su upravljani automatski od strane Active Record-a ako postoje.

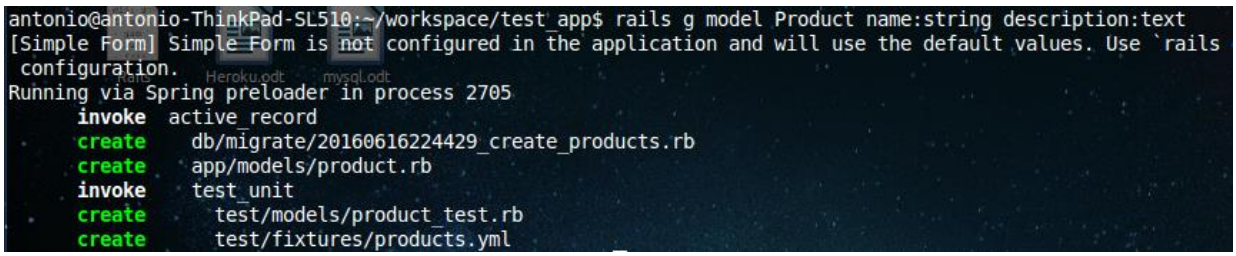
Važno je primjetiti da se definira željena promijena koja se nastoji izvršiti krećući se naprijed u vremenu. Prije pokretanja migracije, ne postoji tablica. Nakon izvršenja, tablica postoji. Active Record zna kako obrnuti migraciju također, ako se obrne migracija unazad, obrisat će tablicu.

U bazama koje podržavaju transakcije s izjavama koje mijenjaju shemu, migracije su umotane u transakciju. Ako baza ne podržava ovo, kada migracija podbaci, dijelovi koji su uspješno izvršeni neće biti vraćeni u prethodno stanje. Bit će potrebno ručno vratiti nastale promjene.

Migracije su spremljene kao datoteke unutar db/migrate direktorija, jedna za svaku klasu migracije. Ime datoteke je u formatu YYYYMMDDHHMMSS_create_products.rb, što znači UTC vremenska oznaka identificirajući migraciju, nakon koje slijedi podcrta, nakon koje slijedi ime migracije. Ime klase migracije (CamelCase verzija) se treba podudarati s drugim dijelom imena datoteke. Npr. 20080906120000_create_products.rb treba definirati klasu *CreateProducts* te 20080906120001_add_details_to_products.rb treba definirati AddDetailsToProducts. Rails koristi vremensku oznaku za određivanje koje migracije se trebaju izvršiti i kojim slijedom, stoga ako se kopira migracija iz druge aplikacije ili se generira vlastita datoteka, *developer* treba biti svjestan njezine pozicije u redu.

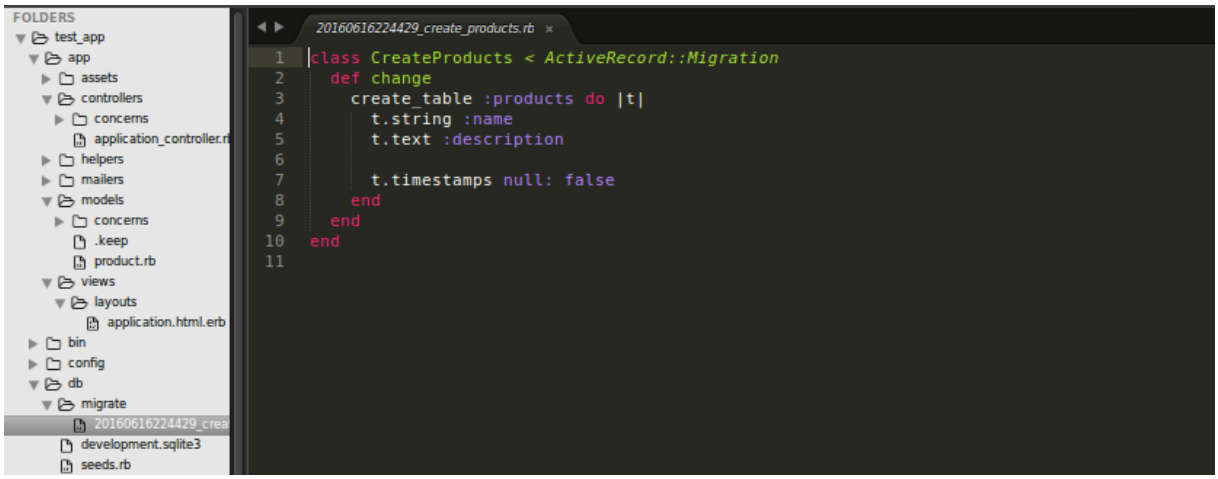
Model i *scaffold* generatori će stvoriti migracije prikladne za dodavanje novog modela. Ako se Rails-u narede željeni stupci, tada će se izjava za dodavanje tih stupaca također stvoriti. Npr. pokrenuvši:

```
$ rails generate model Product name:string description:text
```



Sl. 3.3. Rails generate (g) naredba

Stvara migraciju koja izgleda ovako:



Sl. 3.4. Nastala migracijska datoteka

Prva migracija koja će se koristit će najvjerojatnije biti *$ rake db:migrate*. U svojoj osnovnoj formi, samo izvrši promjenu ili *up* metodu za sve neizvršene migracije. Ako ne postoje takve migracije, jednostavno izlazi. Izvršit će migracije redoslijedom datuma tih migracija.

```
antonio@antonio-ThinkPad-SL510:~/workspace/test_app$ rake db:migrate
[Simple Form] Simple Form is not configured in the application and will use the default values.
 configuration.
== 20160616224429 CreateProducts: migrating ===================================
-- create_table(:products)
   -> 0.0018s
== 20160616224429 CreateProducts: migrated (0.0019s) ==========================
```

Sl. 3.5. Rake db:migrate naredba

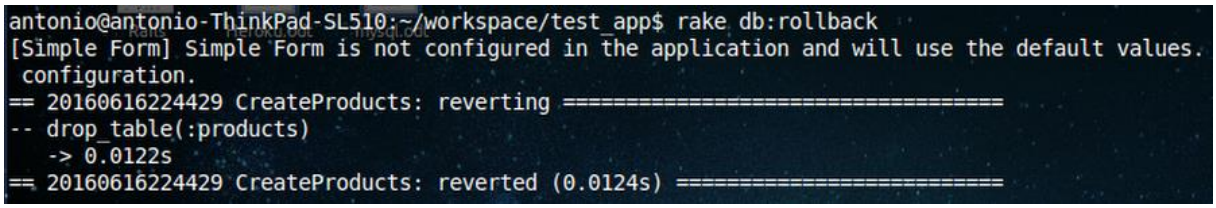
Važno je uočiti da pokretanje *$ rake db:migrate* zadatka također poziva *db:schema:dump* zadatak, koji će ažurirati db/schema.rb datoteku da odgovara strukturi baze podataka. Ako se odredi ciljana verzija, Active Record će izvesti tražene migracije (*change, up ,down*) dok ne dođe do tražene verzije. Verzija je brojčani prefiks imena migracije. Npr. da bi se migriralo na verziju 20080906120000, treba se pokrenuti:

```
$ rake db:migrate VERSION=20080906120000
```

Ako je verzija 20080906120000 veća od trenutne verzije (to bi bila migracija unaprijed), ovo će pokrenuti *change* (ili *up*) metodu na svim migracijama do i uključujući 20080906120000 te neće izvršiti niti jednu naknadnu migraciju. Ako se migrira unazad, ovo će pokrenuti *down* metodu na sve migracije do, ali ne uključujući, 20080906120000.

Uobičajen slučaj je vraćanje migracije na prethodnu verziju. Primjerice, ako se dogodila greška na migraciji te ju je potrebno ispraviti. Umjesto pronalaženja broja povezanog s prošlom migracijom, može se pokrenuti:

$ rake db:rollback



Sl. 3.6. Rake db:rollback naredba

Ovo će vratiti unazad zadnju migraciju, ili vraćanjem *change* metode unazad ili pokretanjem *down* metode. U slučaju da je potrebno vratiti nekoliko migracija, moguće je pružiti STEP parametar:

$ rake db:rollback STEP=3

će vratiti unazad zadnje 3 migracije.

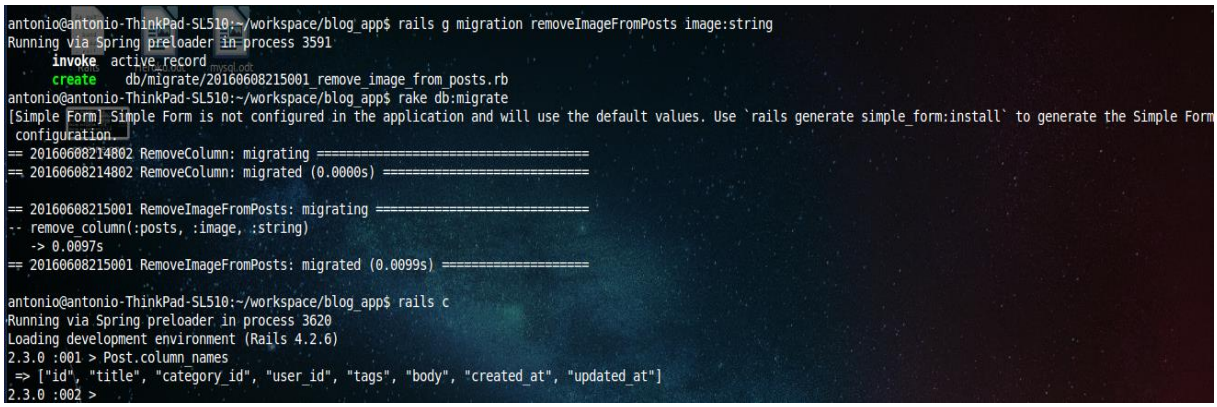
Naredba *db:migrate:redo* je kratica za vraćanje unazad te ponavljanje migracije iznova. Kao i sa *db:rollback* naredbom, moguće je koristiti STEP parametar ako je potrebno ići više od jedne verzije unazad, npr:

$ rake db:migrate:redo STEP=3

Niti jedan od *bin/rails* zadataka ne radi ništa što *db:migrate* ne može. Jednostavno su više pogodni, jer nije potrebno eksplicitno specificirati verziju na koju se treba migrirati.

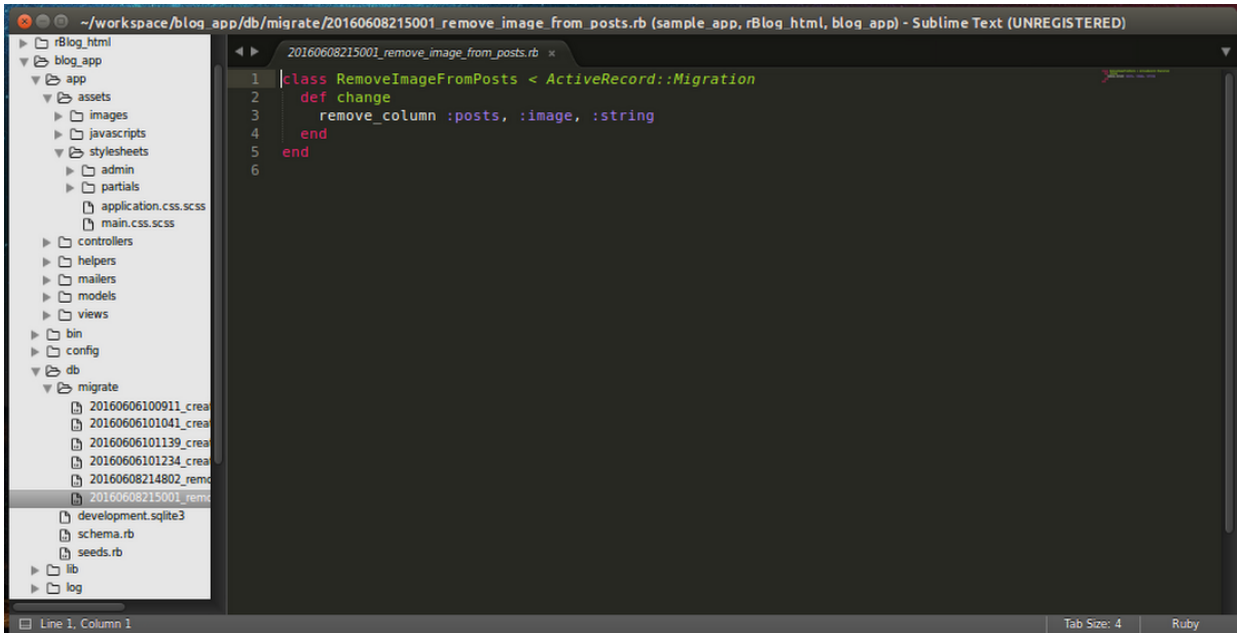
Ponekad će se desiti greška pri pisanju migracije. Ako je migracija već izvršena, nije moguće samo popraviti migraciju i iznova ju pokrenuti, Rails smatra da je migracija već izvršena i neće izvršiti ništa pri pokretanju *$ rails db:migrate*. Potrebno je vratiti migraciju unazad (*rollback*), s naredbom *bin/rails db:rollback*, urediti migraciju i zatim pokrenuti *$ rails db:migrate* za izvršenje točne verzije.

Općenito, uređivanje postojeće migracije nije dobra ideja. Stvori se dodatni posao sebi i svojim kolegama, a to može prouzrokovati probleme ako je postojeća verzija migracije već izvršena na proizvodnim uređajima. Umjesto toga, treba se napisati nova migracija koja izvršava tražene promjene. Uređivanje novonastale migracije koja još nije predana kontroli izvora (ili općenito, koja nije propagirana izvan razvojnog uređaja) je relativno bezopasno.



Sl. 3.7. Rails g migration

U danom primjeru generira se migracija pomoću narede *$ rails g migration* koja će ukloniti stupac *image* u tablici *posts*. Migracija stvara sljedeću datoteku na lokaciji *db/migrate*, u njoj se definira metoda *change*, koja uklanja stupac *image*.



Sl. 3.8. Kod u datoteci migracije

Nakon generiranja migracije koristi se naredba *$ rake db:migrate* koja mijenja model. Otvara se *$ rails console* i naredbom *$ Post.column_names* se potvrđuje da u modelu *Post* više ne postoji stupac image.

Zašto je ovo korisno? Očigledno dopušta postavljanje baze podataka koristeći *user-friendly* Ruby kod umjesto SQL-a, ali ima više pogodnosti od toga. Kroz vrijeme, izradit će se mnogo migracijskih datoteka. Ako se odluči odustati od baze podataka i započeti iznova, lako se može izvesti i zatim nanovo izvršiti migracije. Pri postavljanju na *web*, izvršit će se iste migracije, a

produkcijska baza podataka će čekati, čak i ako je riječ o drugom tipu baze podataka. Active Record obavlja teži dio posla, dok *developer* može izrađivati *web* stranicu.

3.4 Asocijacije između modela

Zašto su potrebne asocijacije između modela? Zato što čine uobičajene operacije jednostavnijima i lakšima u kodu. Primjerice, razmotrit će se jednostavna Rail aplikacija koja uključuje model za *customers* i model za *orders*. Svaki *customer* može imati više *orders.* Bez asocijacija, deklaracije modela bi izgledale ovako:

```
class Customer < ActiveRecord::Base
end

class Order < ActiveRecord::Base
end
```

pretpostavit će se da je trenutno potrebno dodati novi *order* za postojećeg *customer*. Moralo bi se učiniti nešto poput sljedećeg:

```
@order = Order.create(order_date: Time.now, customer_id:
@customer.id)
```

Ili, u slučaju da je potrebno razmotriti brisanje instance klase *Customer,* da se pri tom osigura da se sve njegove *orders* također obrišu:

```
@orders = Order.where(customer_id: @customer.id)
@orders.each do |order|
        order.destroy
end
@customer.destroy
```

S Active Record asocijacijama se pojednostavljuje mnogo operacija poput ovih, tako da se deklarativno kaže Rails-u poveznica između dva modela. Slijedi prerađen kod za postavljanje customers i orders:

```
class Customer < ActiveRecord::Base
        has_many :orders, dependent: :destroy
end

class Order < ActiveRecord::Base
        belongs_to :customer
end
```

s ovom promjenom, stvaranje nove *order* za pojedinog *customer* je puno jednostavnije:

```
@order = @customer.orders.create(order_date: Time.now)
```

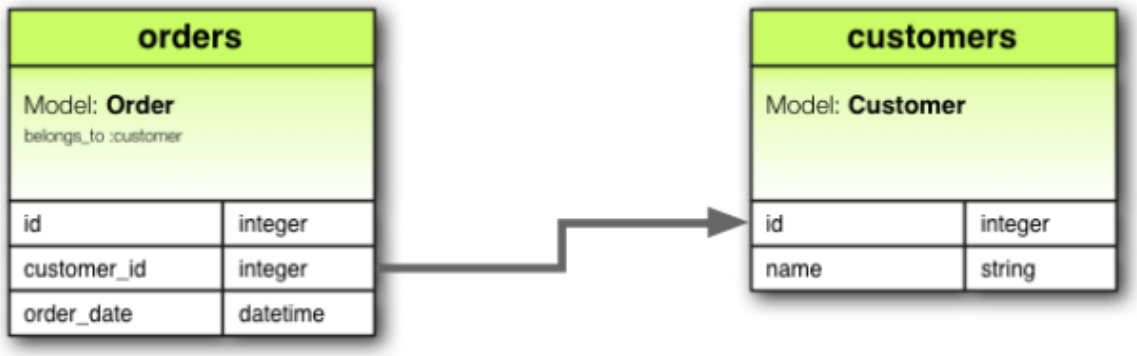
Brisanje instance klase *Customer* i svih njegovih *orders* je jednostavnije:

```
@customer.destroy
```

Unutar Rails-a, asocijacija je poveznica između dva Active Record modela. Asocijacije se implementiraju koristeći pozive macro stila, stoga je moguće deklarativno dodavati značajke modelima. Primjerice, deklarirajući da jedan model *belongs_to* (pripada) drugom, poučava se Rails da održava informacije o primarnom ključu-stranom ključu između instanci dva modela, također se dodaju brojne uslužne metode zadanom modelu. Rails podržava šest tipova asocijacija:

- belongs_to
- has_one
- has_many
- has_many :through
- has_one :through
- has_and_belongs_to_many

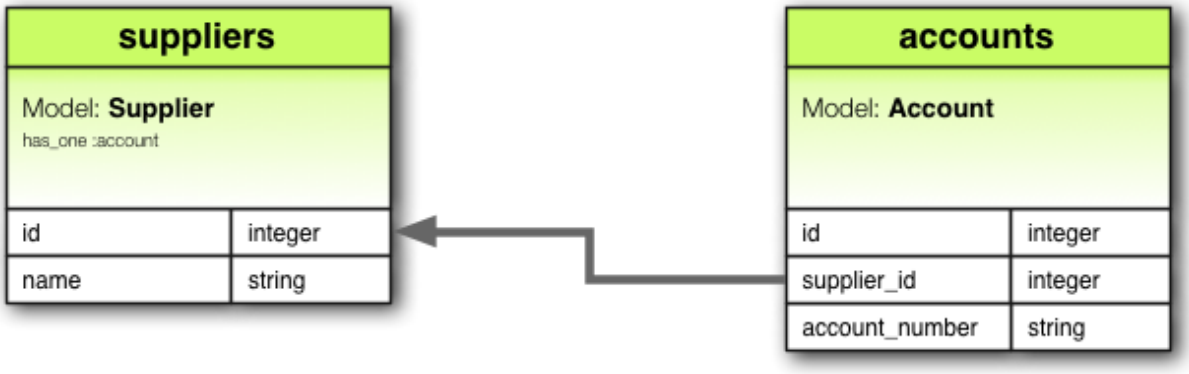
Asocijacija belongs_to uspostavlja jedan-naprama-jedan vezu s drugim modelom, takvu da svaka instanca deklariranog modela pripada (*belongs_to*) jednoj instanci drugog modela. Primjerice, ako aplikacija sadrži *customers* (korisnici) i *orders* (narudžbe) tablice te se svaka *order* (narudžba) može pripisati točno jednom *customer*-u (korisniku), *Order* model bi se deklarirao na ovaj način:




```
class Order < ActiveRecord::Base
  belongs_to :customer
end
```

Sl. 3.9. Belongs_to asocijacija [9]

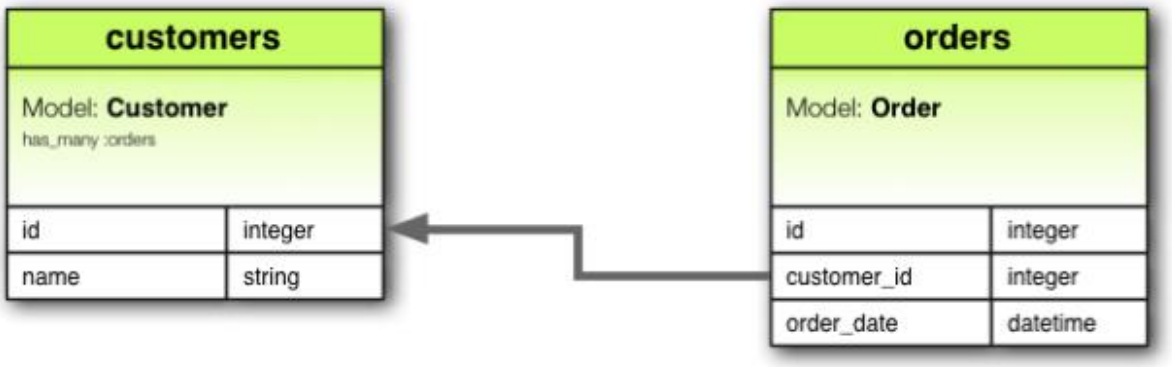
Asocijacija *has_one* također postavlja jedna-naspram-jedan vezu s drugim modelom, ali s nešto drugačijom semantikom (i značajem). Ova asocijacija ukazuje da svaka instanca modela sadrži ili posjeduje jednu instancu drugog modela. Primjerice, ako svaki *supplier* (dobavljač) u aplikaciji ima samo jedan *account* (račun), *Supplier* model bi se deklarirao:




```
class Supplier < ActiveRecord::Base
  has_one :account
end
```

Sl. 3.10. Has_one asocijacija [9]

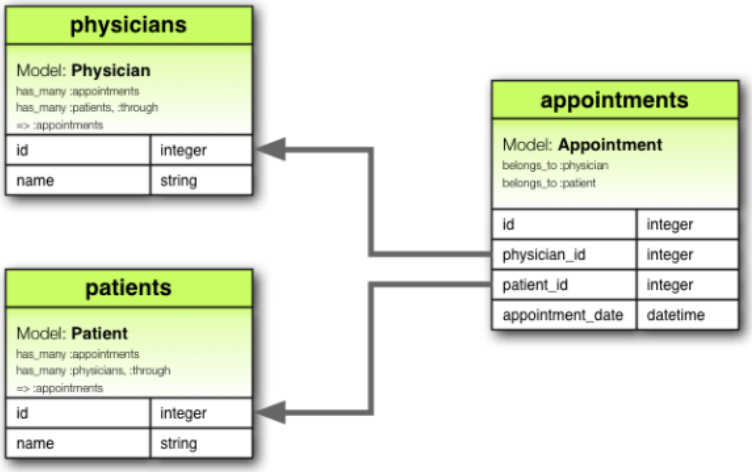
Asocijacija *has_many* ukazuje na jedan-naspram-više vezu s drugim modelom. Često će se naći na "drugoj strani" *belongs_to* asocijacije. Ova asocijacija ukazuje da svaka instanca modela ima nula ili više instanci drugog modela. Primjerice, u aplikaciji koja sadrži *customers* (korisnici) i *orders* (narudžbe) tablice, *Customer* model bi se mogao deklarirati ovako:




```
class Customer < ActiveRecord::Base
  has_many :orders
end
```

Sl. 3.11. Has_many asocijacija [9]

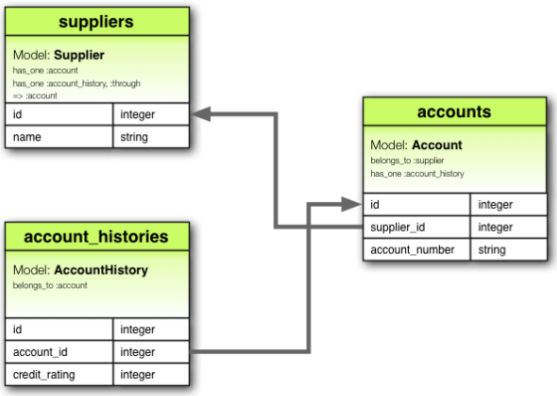
Asocijacija *has_many* :*through* se često koristi za postavljanje više-naspram-više vezu s drugim modelom. Ova asocijacija ukazuje da se deklarirani model može spojiti s nula ili više instanci drugog modela tako da se upućuje kroz treći (posredni) model. Primjerice, razmatra se medicinska praksa gdje *patients* (pacijenti) stvaraju zakazani pregled da posjete *physicians* (liječnike). Relevantna asocijacijska deklaracija može izgledati ovako:




```
class Physician < ActiveRecord::Base
  has_many :appointments
  has_many :patients, :through => :appointments
end

class Appointment < ActiveRecord::Base
  belongs_to :physician
  belongs_to :patient
end

class Patient < ActiveRecord::Base
  has_many :appointments
  has_many :physicians, :through => :appointments
end
```

Sl. 3.12. Has_many :through asocijacija [9]

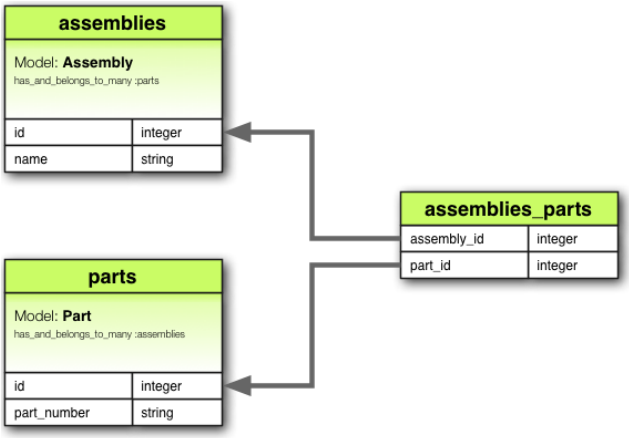
Asocijacija *has_one* :*through* postavlja jedan-naspram-jedan vezu s drugim modelom. Ova asocijacija ukazuje da se deklarirani model povezuje s jednom instancom drugog modela tako da se upućuje kroz treći (posredni model). Primjerice, ako svaki *supplier* (dobavljač) posjeduje jedan *account* (račun) te se svaki *account* udružuje s jednom *account history* (povijest računa), tada *Supplier* model izgleda ovako:




```
class Supplier < ActiveRecord::Base
  has_one :account
  has_one :account_history, :through => :account
end

class Account < ActiveRecord::Base
  belongs_to :supplier
  has_one :account_history
end

class AccountHistory < ActiveRecord::Base
  belongs_to :account
end
```

Sl. 3.13. Has_one :through asocijacija [9]

Asocijacija *has_and_belongs_to_many* stvara direktnu više-naspram-više vezu s drugim modelom. Primjerice, ako aplikacija sadrži *assemblies* (sklopovi) i *parts* (dijelovi) tablice, tako da svaki *assembly* ima više *parts* i da se svaki *part* pojavljuje u više *assemblies*, model se može deklarirati kao:




```
class Assembly < ActiveRecord::Base
  has_and_belongs_to_many :parts
end

class Part < ActiveRecord::Base
  has_and_belongs_to_many :assemblies
end
```

Sl. 3.14. Has_and_belongs_to_many asocijacija [9]

4. WEB APLIKACIJA

Praktični dio završnog rada obuhvaća izradu *web* aplikacije u Rails okruženju. Izrađena je *web* aplikacija zvana "Oglasi" koja omogućava pregledavanje korisnički postavljenih oglasa, po uzoru na popularnu stranicu Njuškalo. Postoji mogućnost registracije korisnika kao i otvaranja i zatvaranja sesije. Kad je korisnik prijavljen u sustav, ima mogućnost kreiranja svog oglasa (odabire kategoriju, dodaje sliku te upisuje naslov oglasa i njegov opis), zatim svoj oglas može naknadno uređivati i obrisati. Oglase drugih korisnika nije u stanju uređivati ili brisati, ali može ocijeniti oglas i/ili ostaviti komentar.

Aplikacija slijedi MVC arhitekturu, u sljedećim potpoglavljima su navedeni i opisani dijelovi MVC arhitekture, tj. modeli, prikazi i kontroleri. Poslije MVC arhitekture je prikazana gotova *web* aplikacija.

Nakon kreiranja novog Rails projekta, dodani su *gem*-ovi koji pomažu u zasebnim aspektima aplikacije. Također je definirana izvorna putanja (engl. *root route*), ostale putanje su definirane pri kreiranju modela.

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


# Bundle edge Rails instead: gem 'rails', github: 'rails/rails'
gem 'rails', '4.2.6'
# Use SCSS for stylesheets
gem 'sass-rails', '~> 5.0'
# Use Uglifier as compressor for JavaScript assets
gem 'uglifier', '>= 1.3.0'
# Use CoffeeScript for .coffee assets and views
gem 'coffee-rails', '~> 4.1.0'
# See https://github.com/rails/execjs#readme for more supported runtimes
# gem 'therubyracer', platforms: :ruby
gem 'simple_form', '~> 3.2', '>= 3.2.1'
gem 'devise', '~> 4.2'
gem 'paperclip', '~> 5.0'
gem 'bootstrap-sass', '~> 3.3', '>= 3.3.6'
# Use jquery as the JavaScript library
gem 'jquery-rails'
# Turbolinks makes following links in your web application faster. Read more: https://github.com/rails/turbolinks
gem 'turbolinks'
# Build JSON APIs with ease. Read more: https://github.com/rails/jbuilder
gem 'jbuilder', '~> 2.0'
# bundle exec rake doc:rails generates the API under doc/api.
gem 'sdoc', '~> 0.4.0', group: :doc

group :development, :test do
  # Call 'byebug' anywhere in the code to stop execution and get a debugger console
  gem 'byebug'
end

group :development do
  # Access an IRB console on exception pages or by using <%= console %> in views
  gem 'web-console', '~> 2.0'

  # Spring speeds up development by keeping your application running in the background. Read more: https://github.com/rails/spring
  gem 'spring'
  gem 'sqlite3'
end

group :production do
    gem 'pg'
    gem 'rails_12factor'
end
```

Sl. 4.1. Gemfile

Većina *gem*-ova je automatski uključena u projekt, no *gem*-ovi poput simple_form, devise, paperclip i bootstrap-sass su naknadno dodani projektu. Također je *gem* sqlite3 premješten u *:development* te je dodano okruženje *:production*.[10]

Uloga naknadno dodanih Gem-ova je:

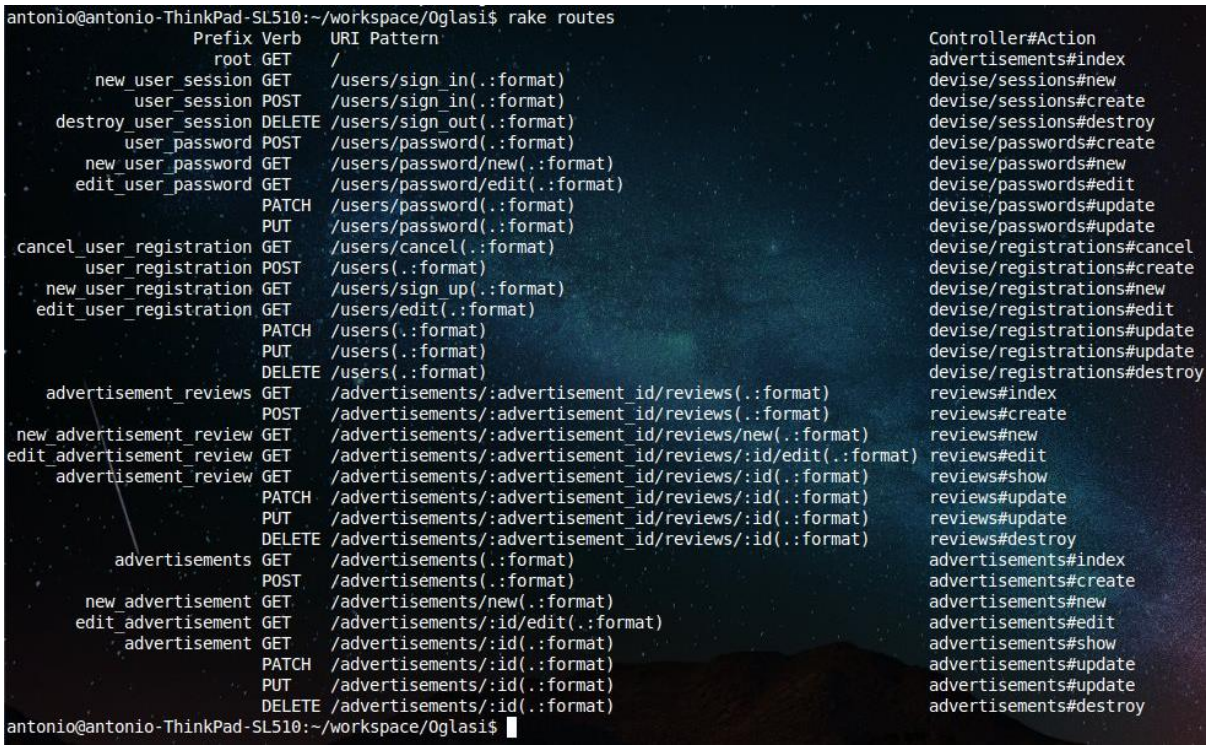
- **simple_form** - lako i jednostavno kreiranje formi, iskorišten za kreiranje forme oglasa i recenzija
- **devise** - služi za korisničku autentifikaciju (generira sustav za imena, lozinke, sigurnost, forme...), iskorišten za kreiranje modela *User* (korisnik)
- **paperclip** - jednostavna biblioteka za dodavanje slika namijenjena Active Record-u, iskorišten za dodavanje slika oglasima
- **bootstrap-sass** - okruženje za izradu responsive *web* projekata temeljeno na Sass jeziku, iskorišten za uređivanje izgleda aplikacije

Za temeljniji pregled funkcionalnosti pojedinog *gem*-a, moguće je pristupiti službenoj dokumentaciji na: https://rubygems.org/ .

Potrebne putanje su definirane u datoteci *routes.rb*, također naredbom *$ rake routes* je moguće pregledati sve putanje projekta.

```
1  Rails.application.routes.draw do
2    root 'advertisements#index'
3    devise_for :users
4    resources :advertisements do
5      resources :reviews
6    end
7  end
8
```

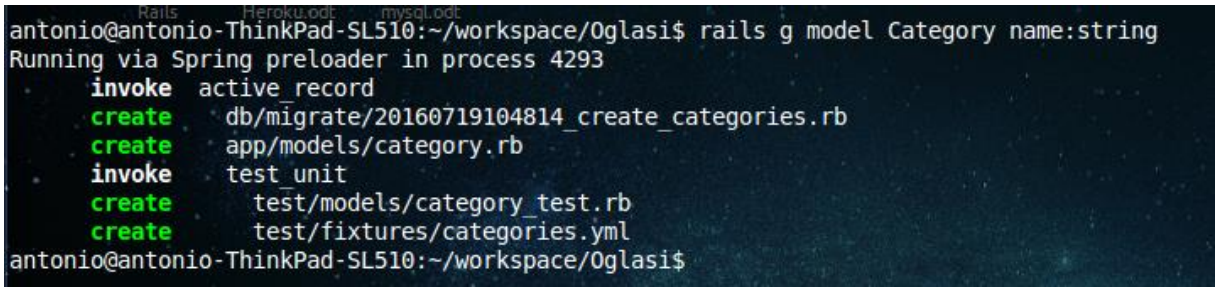
Sl. 4.2. Routes.rb



Sl. 4.3. Rake routes

4.1 Models

Modeli predstavljaju instance baze podataka, tj. svaki model (poput *Advertisement*) ima tablicu u bazi podataka imena u množini (*advertisements*) sa svim atributima koji se definiraju modelu pri kreiranju ili naknadno. Nakon stvaranja modela, definirane su asocijacije između modela.



Sl. 4.4. Stvaranje Category modela

```
antonio@antonio-ThinkPad-SL510:~/workspace/Oglasi$ rails g model Review rating:integer comment:text
Running via Spring preloader in process 4526
      invoke  active_record
      create    db/migrate/20160719105131_create_reviews.rb
      create    app/models/review.rb
      invoke    test_unit
      create      test/models/review_test.rb
      create      test/fixtures/reviews.yml
antonio@antonio-ThinkPad-SL510:~/workspace/Oglasi$ rake db:migrate
== 20160719104814 CreateCategories: migrating ================================
-- create_table(:categories)
   -> 0.0386s
== 20160719104814 CreateCategories: migrated (0.0388s) =======================

== 20160719105131 CreateReviews: migrating ====================================
-- create_table(:reviews)
   -> 0.0014s
== 20160719105131 CreateReviews: migrated (0.0015s) ===========================

antonio@antonio-ThinkPad-SL510:~/workspace/Oglasi$
```

Sl. 4.5. Stvaranje Review modela

U primjerima se vidi način stvaranja dva modela, *Category* i *Review*. Nakon stvaranja modela potrebno je izvršiti naredbu *$ rake db:migrate* kako bi se nadogradila baza podataka novim tablicama.

U prethodnom poglavlju su navedene i ukratko objašnjene asocijacije između modela. Aplikacija koristi četiri modela s pripadajućim atributima:

- *User* - model korisnika, stvoren pomoću Devise gem-a
- *Advertisement* - model oglasa (:title, :body, :image, :user_id, :category_id)
- *Category* - model kategorije (:name)
- *Review* - model recenzije (:rating, :comment, :user_id, :advertisement_id)

Nakon stvaranja modela putem terminala, potrebno je u datotekama modela definirati njihove međusobne asocijacije.

```ruby
class Advertisement < ActiveRecord::Base
    belongs_to :user
    belongs_to :category
    has_many :reviews

    has_attached_file :image, styles: { large: "600x600>", medium: "400x200>", thumb: "150x150#" }
    validates_attachment_content_type :image, content_type: /\Aimage\/.*\Z/
end
```

Sl. 4.6. Advertisement.rb

Iz slike 4.6. je uočljivo da model oglasa pripada modelu korisnika i modelu kategorije te da sadrži model recenzija. Također sadrži kod za paperclip *gem*, koji definira tri različite rezolucije slike (*large, medium i thumb*).

```ruby
class Category < ActiveRecord::Base
    has_many :advertisements
end

```

Sl. 4.7. Category.rb

Model kategorije sadrži samo jednu asocijaciju, da sadrži model oglasa tj. jedna kategorija može imati puno oglasa. Iz modela oglasa se vidi obrnuta situacija, tada svaki oglas pripada određenoj kategoriji.

```ruby
class Review < ActiveRecord::Base
    belongs_to :advertisement
    belongs_to :user
end

```

Sl. 4.8. Review.rb

Model recenzije pripada modelu oglasa i modelu korisnika, zbog takve asocijacije *Review* model sadrži atribute :*user_id* i :*advertisement_id*.

```ruby
class User < ActiveRecord::Base
  # Include default devise modules. Others available are:
  # :confirmable, :lockable, :timeoutable and :omniauthable
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :trackable, :validatable

  has_many :advertisements
  has_many :reviews, dependent: :destroy
end

```

Sl. 4.9. User.rb

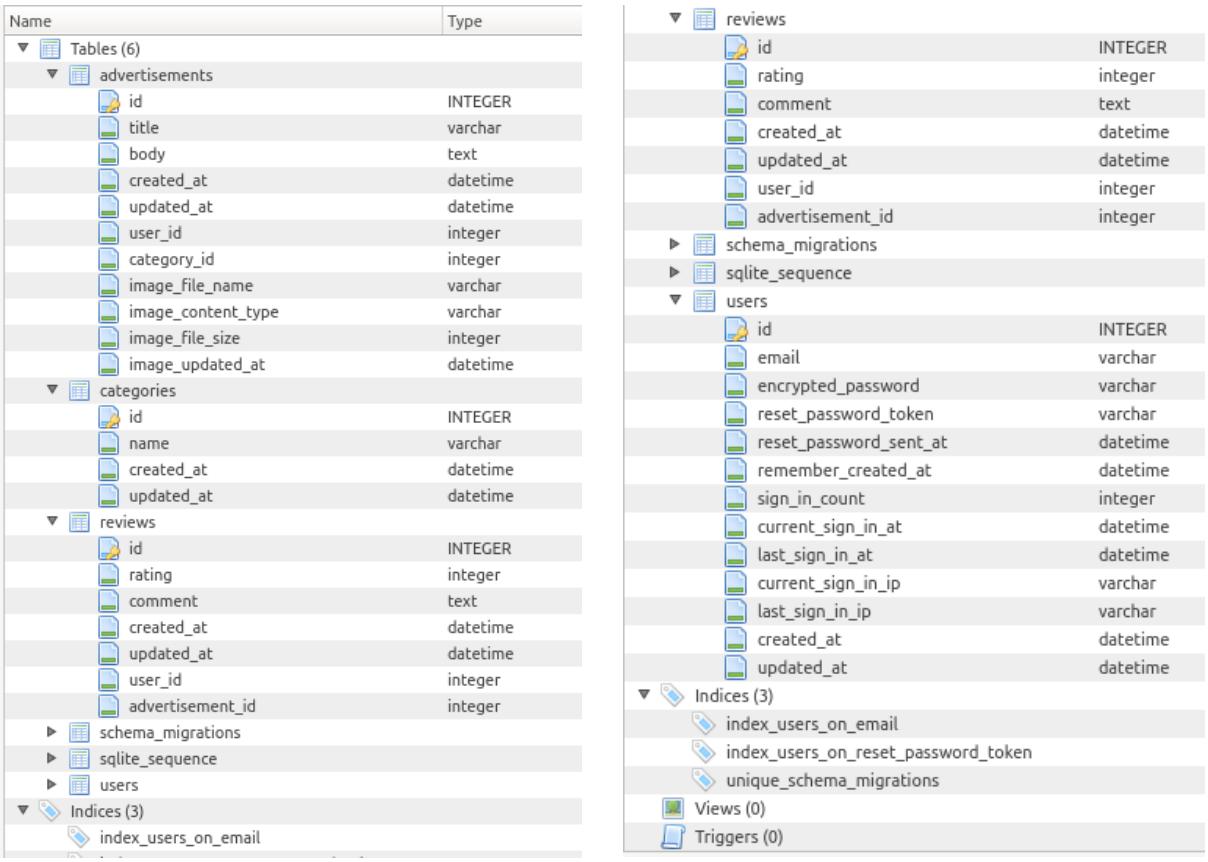
Model korisnika je generiran pomoću Devise *gem*-a. Model korisnika sadrži model oglasa i model recenzija tj. svaki korisnik može imati puno oglasa i recenzija.

```ruby
class AddAttachmentImageToAdvertisements < ActiveRecord::Migration
  def self.up
    change_table :advertisements do |t|
      t.attachment :image
    end
  end

  def self.down
    remove_attachment :advertisements, :image
  end
end

```

Sl. 4.10. Migracija

U prethodnom poglavlju su ukratko objašnjene migracije i njihova primjena. Pomoću terminala je generiran kod migracije sa slike 4.10, a migracija ima metode *self.up* i *self.down*. Pri naredbi *$ rake db:migrate*, tablica *advertisements* dobiva još jedan stupac - *image*. Kod suprotne naredbe $ rake db:rollback, stupac *image* se uklanja iz tablice *advertisements*.

Pomoću programa *DB Browser for SQLite* se može grafički predočiti baza podataka s novonastalim modelima tj. tablicama u bazi.



Sl. 4.11. Tablice baze podataka

	id	title	body	created_at	updated_at	user_id	category_id	image_file_name	mage_content_type	image_file_size
		Filter	Filter	Filter	Filter	F...	Filter	Filter	Filter	Filter
1	2	Prodajem kutnu garnituru	Siva kožna garnitura iz dva ...	2016-07-24 16:51:41.582019	2016-07-24 16:51:41.582019	1	3	kutna.jpg	image/jpeg	87165
2	3	Usluge prijevoza po cijeloj Hrvats...	Naš tim Vam je na usluzi 24 sata d...	2016-07-24 17:56:36.563281	2016-07-24 17:56:36.563281	1	2	prijevoz.jpg	image/jpeg	73692
3	4	Peugeot 308	Opširnije: (Peugeot 308 1,...	2016-08-10 13:01:00.770708	2016-08-10 13:01:00.770708	3	8	peugeot-308-1.6-hdi-sportium2-sl...	image/jpeg	123625
4	5	Škoda Yeti	AUTOMATSKA KLIMAALU NAPL...	2016-08-10 13:03:41.960217	2016-08-10 13:03:41.960217	3	8	skoda-yeti-2.0-tdi-4x4-limited-s...	image/jpeg	86922
5	6	BMW F800GT	MOTOCIKL IMA FULL OPREMU:E...	2016-08-10 13:07:13.620081	2016-08-10 13:07:13.620081	3	8	bmw-f800gt-2013-full...	image/jpeg	84037
6	7	Flauta Yamaha 311	Prodajem vrlo očuvanu flautu Y...	2016-08-10 13:09:45.749243	2016-08-10 13:09:45.749243	1	5	flauta-yamaha-311-slik...	image/jpeg	56143
7	8	Razglas aktivni 600W	Komplet se sastoji od:□	2016-08-10 13:12:48.344879	2016-08-10 13:12:48.344879	1	5	razglas-aktivni-600w-st...	image/jpeg	42573
8	9	Philips 273E3LH SmartImage, S...	-Full HD LED display, with a 1...	2016-08-10 13:17:48.292511	2016-08-10 13:17:48.292511	1	4	philips-273e3lh-smartimage-sm...	image/jpeg	115952
9	10	Hyundai T7 Quad Core 1,4 GHz Sa...	Cijena: 799kn Model: Hyundai ...	2016-08-10 13:22:03.477241	2016-08-10 13:22:03.477241	2	4	hyundai-t7-quad-core-1.4-ghz-sa...	image/jpeg	33765
10	11	Ribolovni štap ZEBCO BW 808-...	Cijena: 290kn Ribolovni štap Z...	2016-08-10 13:24:35.963676	2016-08-10 13:24:35.963676	2	7	ribolovni-stap-zebco-bw-808-7...	image/jpeg	85478
11	12	Hrvatski opći leksikon	Leksikografski zavod Miroslav ...	2016-08-10 13:26:26.919285	2016-08-10 13:26:26.919285	2	6	hrvatski-opci-leksikon-a-z-slik...	image/jpeg	41570
12	13	Osnove financijskog me...	Osnove financijskog me...	2016-08-10 13:27:39.335038	2016-08-10 13:27:39.335038	2	6	osnove-financijskog-me...	image/jpeg	70991
13	14	Otvoreni regal	Cijena: 4.200kn Otvoreni regal, ...	2016-08-10 13:29:52.353029	2016-08-10 13:29:52.353029	2	3	otvoreni-regal-slika-52129706.j...	image/jpeg	56682

Sl. 4.12. Tablica oglasa

Slika 4.11. prikazuje tablice i njihove atribute, dok se na slici 4.12. može vidjeti tablica oglasa sa svim do tada unošenim vrijednostima.

4.2 Views

Views (prikazi) određuju izgled pojedine URL adrese *web* aplikacije. *Views* direktorij sadrži direktorije *advertisements*, *devise*, *layouts* i *reviews*. *Advertisements* i *reviews* direktoriji sadrže datoteke čija imena odgovaraju akcijama kontrolera *advertisements_controller* i *reviews_controller*. Devise mapa sadrži sve datoteke kreirane od istoimenog *gem*-a, svi prikazi su vezani za registraciju korisnika i manipulaciju sesijama. *Layouts* direktorij sadrži datoteku *application.html.erb* u kojoj se definira generalni izgled (kostur) cijele aplikacije. Sve datoteke u *Views* direktoriju imaju ekstenziju .html.erb, stoga se u njima nalazi *html* i *embedded ruby* kod.

```
<!DOCTYPE html>
<html>
<head>
  <title>Oglasi</title>
  <%= stylesheet_link_tag    'application', media: 'all', 'data-turbolinks-track' => true %>
  <%= javascript_include_tag 'application', 'data-turbolinks-track' => true %>
  <%= csrf_meta_tags %>
</head>

<body>

    <!-- NAVBAR -->
    <nav class="navbar navbar-default navbar-fixed-top">
        <div class="container">

            <div class="navbar-header">
                <%= link_to "Oglasi", root_path, class: "navbar-brand" %>
            </div>

            <ul class="nav navbar-nav">
                <% if user_signed_in? %>
                    <li><%= link_to "Sign out", destroy_user_session_path, method: :delete %></li>
                <% else %>
                    <li><%= link_to "Sign in", new_user_session_path %></li>
                    <li><%= link_to "Sign up", new_user_registration_path %></li>
                <% end %>
            </ul>

            <ul class="nav navbar-nav navbar-right">
                <% if user_signed_in? %>
                <li><%= link_to "Predaj Oglas!", new_advertisement_path %></li>
                <% end %>
            </ul>
        </div>
    </nav>

    <div class="row">
        <!-- SIDEBAR -->
        <div id="sidebar">
            <div class="col-md-1">
                <div class="container">
                    <div id="logo">
                        <%= link_to root_path do %>
                            <%= image_tag "oglasi.jpg" %>
                        <% end %>
                    </div>
                </div>
```

Sl. 4.13. Application.html.erb

```
37      <div class="row">
38          <!-- SIDEBAR -->
39          <div id="sidebar">
40              <div class="col-md-1">
41                  <div class="container">
42                      <div id="logo">
43                          <%= link_to root_path do %>
44                              <%= image_tag "oglasi.jpg" %>
45                          <% end %>
46                      </div>
47                  </div>
48
49                  <ul>
50                      <li id="category">Kategorije</li>
51                  </ul>
52
53                  <div class="btn-group-vertical btn-group-lg" role="group">
54                      <% Category.all.each do |category| %>
55                          <button type="button" class="btn btn-default">
56                              <div class="<%= 'active' if params[:category] == category.name %>">
57                                  <%= link_to category.name, advertisements_path(category: category.name) %>
58                              </div>
59                          </button>
60                      <% end %>
61                  </div>
62
63              </div>
64          </div>
65
66          <div class="well">
67              <div class="main-content"></div>
68                  <div class="col-md-10 col-md-offset-1">
69                      <div class="container-fluid">
70                          <p class="notice"><%= notice %></p>
71                          <p class="alert"><%= alert %></p>
72                          <%= yield %>
73                      </div>
74                  </div>
75              </div>
76          </div>
77
78      </div>
79  </body>
80  </html>
```

Sl. 4.14. Application.html.erb

Nakon *stylesheet* i *JavaScript* tagova, slijedi kod koji definira *navbar*. Kod *navbar*-a sadrži ime aplikacije "Oglasi" te *linkove* poput "*Sign in*" i "Predaj Oglas!". Nakon njega slijedi kod za *sidebar* i glavni sadržaj stranice (gdje se prikazuju oglasi). *Sidebar* ima svrhu prikazivanja svih kategorija koje se definiraju unutar Rails konzole (*$ rails c*). *Bootstrap* omogućava dijeljenje sadržaja stranice po stupcima (kojih ima sveukupno 12), stoga je za *sidebar* rezerviran jedan stupac, a za glavni sadržaj deset stupaca s jednim stupcem razmaka između. Naredba *erb* koda *yield* služi za dohvaćanje svih prikaza (engl. *View*) na koje upućuje zadana putanja (engl. *route*).

Kod *application.html.erb* se izvršava na svakoj stranici *web* aplikacije, dok *yield* naredba poziva određene prikaze na zahtjev *web browser*-a. *Root route* poziva prvi prikaz koji se generira za prikaz u dijelu glavnog sadržaja stranice, taj prvi prikaz je u datoteci *routes.rb* podešen na *index* stranicu oglasa tj. *index.html.erb* u direktoriju *Advertisements*.

```
<h1 class="current-category"><%= params[:category] %></h1>
<% if params[:category].blank? %>
    <h2>Najnoviji oglasi</h2>
<% end %>
<% if @advertisements.count == 0 %>
    <h1>Trenutno nema oglasa u kategoriji <%= params[:category] %></h1>

<% else %>

    <div class="row">
        <% @advertisements.each do |ad| %>
            <div class="col-md-4">
                <div class="box">
                    <a href="/advertisements/<%= ad.id %>">
                        <%= image_tag ad.image.url(:medium), class: "img-responsive" %>
                    </a>
                </div>
            </div>
        <% end %>
    </div>

<% end %>
```

Sl. 4.15. Index.html.erb

Kod u *index.html.erb* datoteci služi za prikaz svih najnovije dodanih oglasa iz svih kategorija ili prikaz svih oglasa u određenoj kategoriji u slučaju da je odabrana pojedina kategorija. Za svaki oglas je rezervirano 4 stupca, što znači da ih stane 3 u svakom redu glavnog sadržaja stranice.

Advertisement direktorij također sadrži prikaze *new*, *edit*, *_form* te najvažniji - *show* prikaz. Prva dva prikaza služe za stvaranje i uređivanje oglasa, dok *_form* parcijalna datoteka služi za svrstavanje istog koda na jedno mjesto (*DRY - Don't Repeat Yourself*). *Show* prikaz služi za prikaz jednog odabranog oglasa sa svim podacima i recenzijama.

```
    <div class="row-fluid">

        <div class="col-md-6">
            <div class="info">
                <%= image_tag @advertisement.image.url(:medium), class: "ad-show" %>
                <div class="star-rating" data-score= <%= @avg_review %> ></div>
                <div class="table-responsive"></div>
                    <table class="table">
                        <tbody>
                            <tr>
                                <td><strong>Kategorija: </strong></td>
                                <td><%= @advertisement.category.name %></td>
                            </tr>

                            <tr>
                                <td><strong>Naslov: </strong></td>
                                <td><%= @advertisement.title %></td>
                            </tr>

                            <tr>
                                <td><strong>Opis: </strong></td>
                                <td><%= @advertisement.body %></td>
                            </tr>

                            <tr>
                                <td><strong>Kontakt: </strong></td>
                                <td><%= @advertisement.user.email %></td>
                            </tr>
                        </tbody>
                    </table>
            </div>
        </div>
    </div><!-- oglas -->
```

Sl. 4.16. Show.html.erb

```
34
35            <div class="col-md-6">
36                <div class="recenzije">
37                    <h3>Recenzije</h3>
38                    <% if @reviews.blank? %>
39                      <h3>Trenutno nema recenzija, ostavite prvu!</h3>
40                      <%= link_to "Napiši recenziju", new_advertisement_review_path(@advertisement), class: 'btn btn-danger' %>
41                    <% else %>
42                      <% @reviews.each do |review| %>
43                        <div class="reviews">
44                          <div class="star-rating" data-score= <%= review.rating %> ></div>
45                          <p><%= review.comment %></p>
46                        </div>
47                      <% end %>
48                      <br>
49                      <%= link_to "Napiši recenziju", new_advertisement_review_path(@advertisement), class: 'btn btn-danger' %>
50                    <% end %>
51                </div>
52            </div><!-- recenzije -->
53
54        </div><!-- row -->
55
56  <br>
57  <%= link_to 'Nazad', advertisements_path, class: 'btn btn-default show-btn' %>
58
59  <% if user_signed_in? %>
60      <% if @advertisement.user_id == current_user.id %>
61          <%= link_to 'Uredi', edit_advertisement_path(@advertisement), class: 'btn btn-default' %>
62          <%= link_to 'Ukloni', advertisement_path(@advertisement), method: :delete, data: { confirm: "Jeste li sigurni da želite ukloniti
                oglas?"}, class: 'btn btn-default' %>
63      <% end %>
64  <% end %>
65
66
67  <script>
68    $('.star-rating').raty({
69      path: '/assets/',
70      readOnly: true,
71      score: function() {
72        return $(this).attr('data-score');
73      }
74    });
75  </script>
```

Sl. 4.17. Show.html.erb

Sadržaj *show* prikaza je vertikalno podijeljen u dva dijela, u lijevom dijelu se nalazi slika oglasa sa svim podacima poput ukupne ocjene i opisa. Na desnoj polovici su smještene recenzije tj. ocjene (po metodi zvjezdica) s komentarima. Ispod oglasa se nalazi tipka "Nazad" te dvije uvjetne tipke "Uredi" i "Ukloni" koje su vidljive samo ako se prikazuje oglas čiji je vlasnik trenutni korisnik. Na samom dnu se nalazi *script* koji omogućava ocjenjivanje pomoću zvjezdica. Uz prikaze oglasa, postoje i samostalno generirani devise prikazi te prikazi recenzija. Recenzije imaju samo tri prikaza: *new*, *edit* i *_form* (koji se poziva u prethodna dva prikaza).

```
1   <%= simple_form_for([@advertisement, @advertisement.reviews.build]) do |f| %>
2       <div id="rating-form">
3           <label>Ocjena</label>
4       </div>
5       <div class="field">
6           <%= f.label :comment %><br>
7           <%= f.text_area :comment %>
8       </div>
9       <div class="actions">
10          <%= f.button :submit, "Ocijeni" %>
11      </div>
12  <% end %>
13
14  <script>
15      $('#rating-form').raty({
16          path: '/assets/',
17          scoreName: 'review[rating]'
18      });
19  </script>
```

Sl. 4.18. _form.html.erb

```
<h2>Nova ocjena</h2>

<%= render 'form' %>
```

```
<h2>Uredi ocjenu</h2>

<%= render 'form' %>

<%= link_to 'Nazad', advertisement_path(@advertisement) %>
```

Sl. 4.19. New.html.erb i edit.html.erb

Unutar direktorija *assets/stylesheets* se nalazi datoteka *application.css.scss*, u njoj je definiran CSS kod koji služi za vizualni izgled cijele aplikacije.

```
@import "bootstrap-sprockets";
@import "bootstrap";

/* Styles */
body {
    background: #FC354C;
    background: -webkit-linear-gradient(right, #FC354C, #0ABFBC);
    background: -o-linear-gradient(right, #FC354C, #0ABFBC);
    background: linear-gradient(to left, #FC354C, #0ABFBC);
}

.box {
    background-color: #fce697;
}

.main-content {
    padding-top: 10px;
    padding-left: 300px;

}

.well {
    padding-left: 3em;

    .ad-show {
        max-width: 100%;
        max-height: 100%;
    }
}

.col-md-4 {
    padding-top: 0;
}

.navbar {
    background: #FC354C;
    background: -webkit-linear-gradient(right, #FC354C, #0ABFBC);
    background: -o-linear-gradient(right, #FC354C, #0ABFBC);
    background: linear-gradient(to left, #FC354C, #0ABFBC);
    border-bottom: 3px solid black;

    .navbar-brand {
        color: #000066;
    }
}
```

Sl. 4.20. Application.css.scss

```scss
#sidebar {
    width: 150px;
    position: fixed;
    left: 0;
    top: 0;
    height: 100%;
    background-color: #fce697;
    padding: 7em 0 0 0;
    border-right: 3px solid black;

    .btn-default {
        .active {
            background-color: #fce697;
        }
    }

    #category {
        list-style: none;
        position: absolute;
        padding-top: 2em;
        padding-right: 3em;
        font-weight: 700;
        font-size: 1.3em;
        color: #33acb7;
    }

    #logo {
        transform: scale(0.85, 0.85);
        position: absolute;
        left: -1em;
        top: -4em;
    }
    .btn-group-lg {
        padding-top: 4em;
        text-align: center;

        .btn {
            font-weight: 300;
            font-size: 1.5em;
            padding: 0.2em 0.4em;
            color: #9eafba;
            text-decoration: none;
            margin-right: -5px;
            text-align: center;

            &:hover {
                color: #36acb7;
            }
        }
            .active {
                color: #f3acb7;
            }
        }
}

.info {
    border-right: 5px solid #ffff80;
    border-top: 5px solid #ffff80;
    padding-top: 5px;
    border-radius: 5px;
}

.recenzije {
    border-left: 5px solid #ffff80;
    padding-left: 5px;
    border-bottom: 5px solid #ffff80;
    padding-bottom: 5px;
}

.reviews {
    padding: 10px 0;
    border-bottom: 1px solid #ffff80;
    .star-rating {
        padding-bottom: 8px;
    }
}

.img-responsive {
    height: 200px;
    width: 400px;
    padding: 5px;
    border: 3px solid black;
    border-radius: 5px;
}

.show-btn {
    margin-left: 10px;
}
```

Sl. 4.21. Application.css.scss

Datoteci *application.css* je dodana ekstenzija .scss što omogućava djelovanje SASS (*Syntactically Awesome Style Sheets*) pretprocesora. Bez Sass-a gniježđenje (engl. *nesting*) koje se vidi na slikama 4.20. i 4.21. ne bi bilo moguće.

4.3 Controllers

Svrha kontrolera je da se ponaša kao sučelje između modela i prikaza te da pritom kontrolira zahtjeve *web browser*-a. Rails projekt automatski sadrži *application_controller* kojeg svi ostali kontroleri nasljeđuju. U projektu Oglasi su stvorena dva kontrolera: *advertisements_controller* i *reviews_controller*.

```ruby
class AdvertisementsController < ApplicationController
    before_action :find_ad, only: [:show, :edit, :update, :destroy]
    before_action :authenticate_user!, only: [:new, :edit]

    def index
        if params[:category].blank?
            @advertisements = Advertisement.all.order("created_at DESC")
        else
            @category_id = Category.find_by(name: params[:category]).id
            @advertisements = Advertisement.where(:category_id => @category_id).order("created_at DESC")
        end
    end

    def show
        @reviews = Review.where(advertisement_id: @advertisement.id).order("created_at DESC")

        if @advertisement.reviews.blank?
            @avg_review = 0
        else
            @avg_review = @advertisement.reviews.average(:rating).round(2)
        end
    end

    def new
        @advertisement = current_user.advertisements.build
        @categories = Category.all.map { |c| [c.name, c.id] }
    end

    def create
        @advertisement = current_user.advertisements.build(advertisement_params)
        @advertisement.category_id = params[:category_id]

        if @advertisement.save
            redirect_to root_path
        else
            render 'new'
        end
    end

    def edit
        @categories = Category.all.map { |c| [c.name, c.id] }
    end

    def update
        @advertisement.category_id = params[:category_id]

        if @advertisement.update(advertisement_params)
            redirect_to advertisement_path(@advertisement)
        else
            render 'new'
        end
    end

    def destroy
        @advertisement.destroy
        redirect_to root_path
    end

    private
    def advertisement_params
        params.require(:advertisement).permit(:title, :body, :image, :user_id, :category_id)
    end

    def find_ad
        @advertisement = Advertisement.find(params[:id])
    end
end
```

Sl. 4.22. Advertisements_controller.rb

U kontrolerima se definiraju metode poput *index*, *new*, *create*, *edit*, *destroy* i ostale. Imena metoda odgovaraju imenima prikaza. Kad se definiraju funkcije metoda (ili akcija kontrolera), uspostavljaju se varijable instance koje započinju sa znakom @ (poput *@reviews*). Sve varijable instance definirane u pojedinoj akciji kontrolera su raspoložive u istoimenom prikazu, npr.

metoda *index* sadrži varijablu instance imena *@advertisements* koja je raspoloživa u prikazu *index.html.erb*.

U *private* djelu su definirane dvije metode za pronalaženje oglasa i odobravanje parametara oglasa, od kojih se *find_ad* metoda koristi u više akcija, stoga je napisana kao *before_action*. *Index* metoda služi za pronalaženje svih oglasa po kategorijama te ih slaže po kronološkom slijedu. *Show* metoda služi za prikaz recenzija i računanje ukupne ocjene kod prikaza pojedinog oglasa. Metode *new* i *create* služe za stvaranje novog oglasa trenutnog korisnika (kod u trećem redu onemogućava stvaranje oglasa ako ne postoji otvorena sesija korisnika). Metode *edit* i *update* su slične metodama *new* i *create*, samo što obrađuju postojeće oglase. Definirana je još i *destroy* akcija, koja briše oglas iz baze podataka.

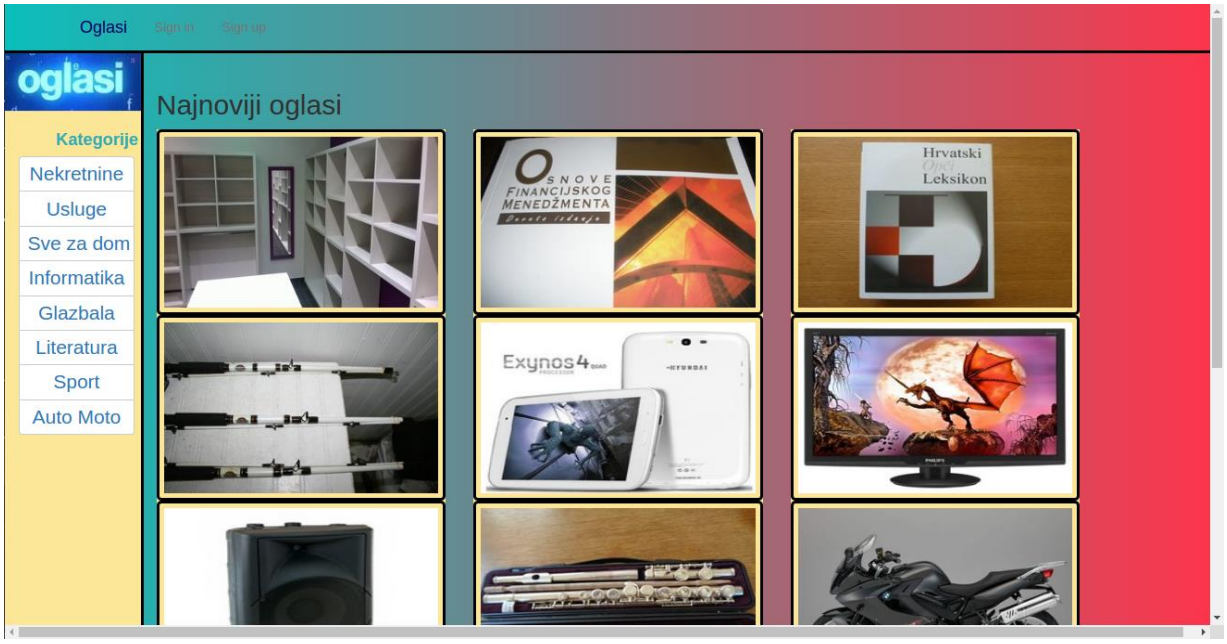
```ruby
class ReviewsController < ApplicationController
    before_action :find_ad
    before_action :find_review, only: [:edit, :update, :destroy]
    before_action :authenticate_user!, only: [:new, :edit]

    def new
        @review = Review.new
    end

    def create
        @review = Review.new(review_params)
        @review.user_id = current_user.id
        @review.advertisement_id = @advertisement.id

        if @review.save
            redirect_to advertisement_path(@advertisement)
        else
            render 'new'
        end
    end

    def edit

    end

    def update
        if @review.update(review_params)
            redirect_to advertisement_path(@advertisement)
        else
            render 'edit'
        end
    end

    def destroy
        @review.destroy
        redirect_to advertisement_path(@advertisement)
    end

    private
    def review_params
        params.require(:review).permit(:rating, :comment, :user_id, :advertisement_id)
    end

    def find_ad
        @advertisement = Advertisement.find(params[:advertisement_id])
    end

    def find_review
        @review = Review.find(params[:id])
    end
end
```

Sl. 4.23. Reviews_controller.rb

Kontroler recenzija je sličan kontroleru za oglase, definirane su tri privatne metode koje se pozivaju s *before_action*. Također se provjerava postoji li trenutna sesija korisnika pri *new* i *edit* akciji.

4.4 Aplikacija

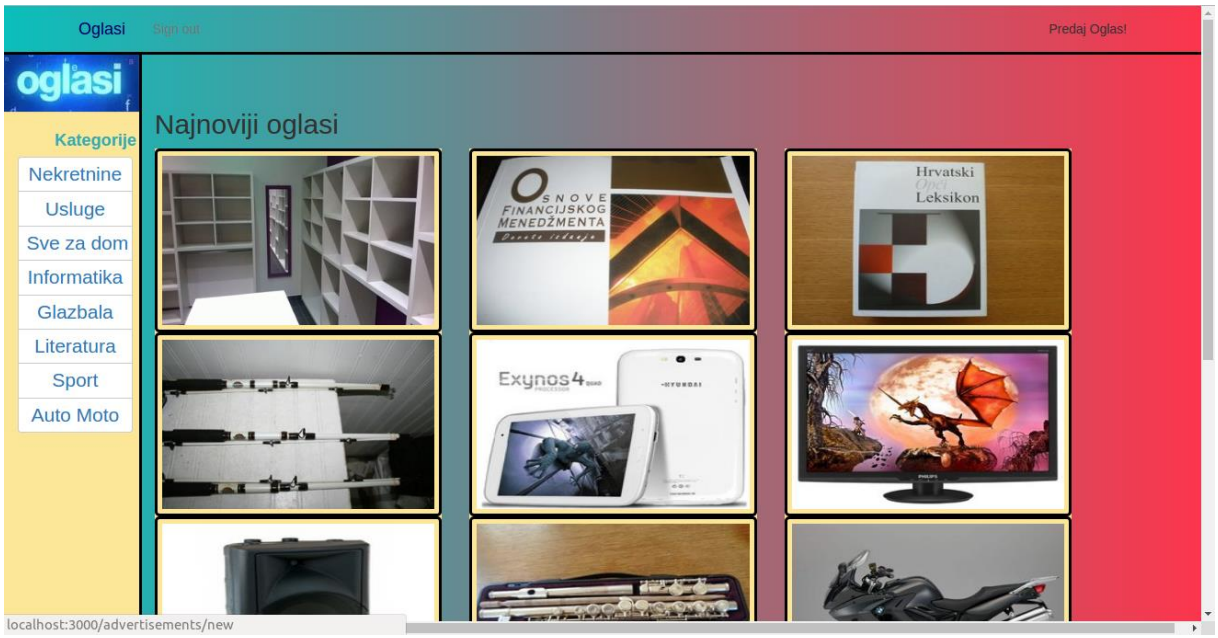
Konačna aplikacija pri pristupanju stranici otvara listu najnovijih oglasa iz svih kategorija. Postoje mogućnosti *Sign in* i *Sign up* te dok se ne stvori sesija korisnika, nije moguće stvoriti oglas. Međutim, moguće je pregledavati oglase bez kreiranja korisničkog računa.



Sl. 4.24. Početni zaslon

Sl. 4.25. Sign up zaslon

Nakon kreiranja novog korisničkog računa ili otvaranja nove sesije s postojećim računom, moguće je stvoriti novi oglas pritiskom na "Predaj Oglas!". Pri otvorenoj korisničkoj sesiji, umjesto tipki *Sign in* i *Sign up*, postoji tipka *Sign out* koja služi za zatvaranje sesije trenutnog korisnika.

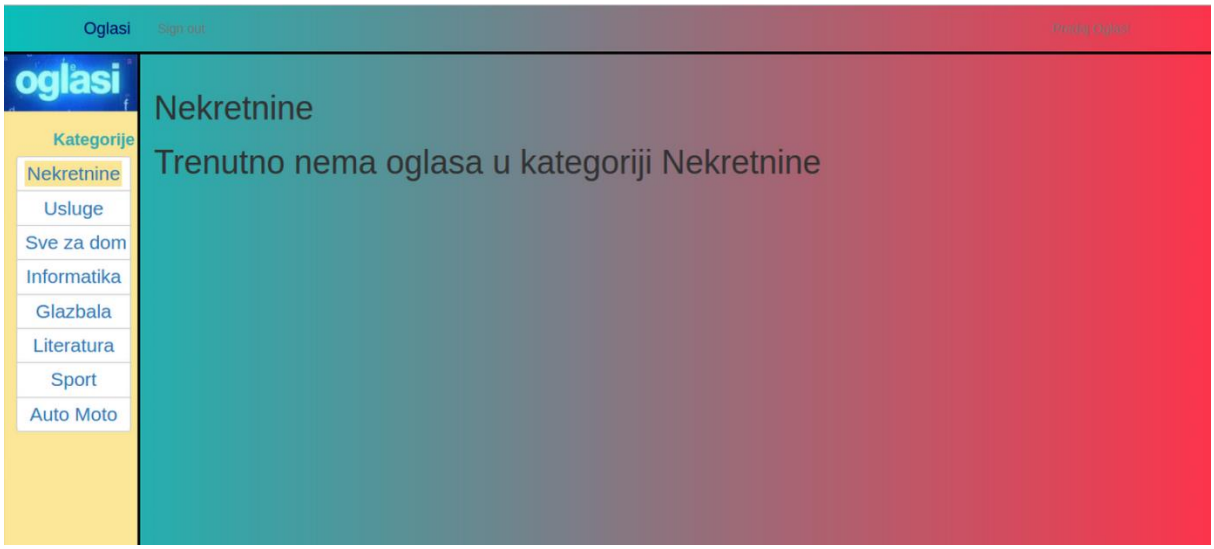


Sl. 4.26. Početna stranica nakon otvaranja sesije

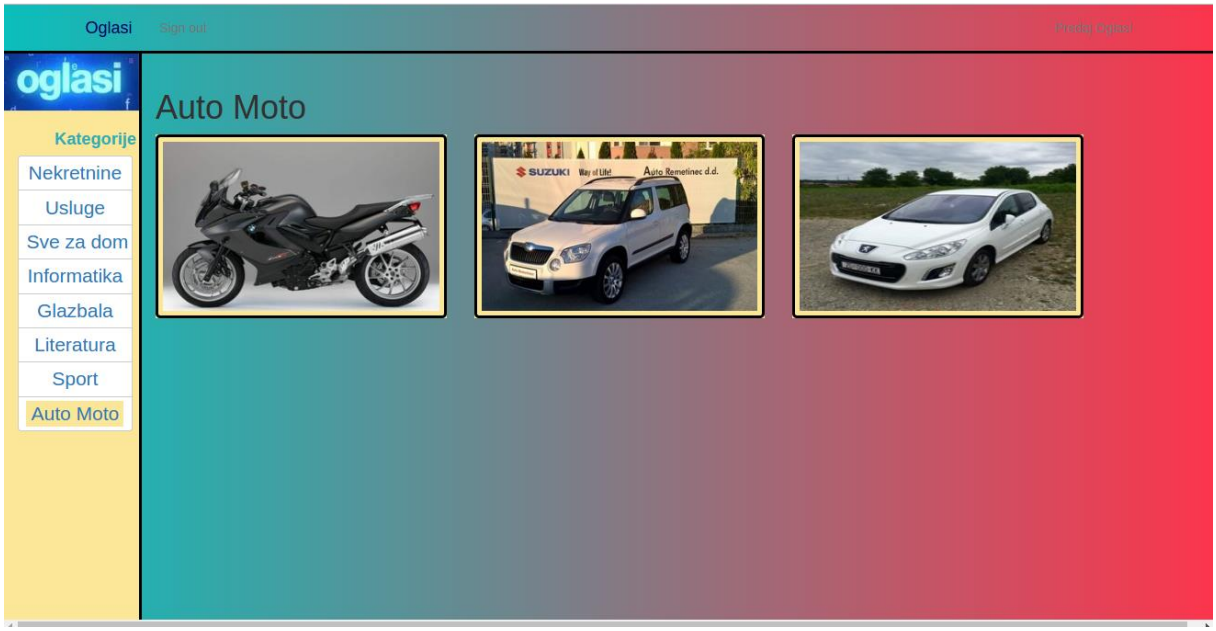
Sl. 4.27. Stvaranje novog oglasa

Pri kreiranju novog oglasa odabire se kategorija kojoj će oglas pripadati (moguće kategorije su izlistane u *sidebar*-u). Odabire se slika oglasa te se upisuju naslov i opis oglasa, nakon čega se pritiskom na "Stvori oglas" novonastali oglas prikazuje na početnoj stranici.

U *sidebar*-u se nalaze sve stvorene kategorije, pritiskom na određenu kategoriju, lista oglasa će ukloniti sve oglase koji ne pripadaju u odabranu kategoriju. U slučaju da ne postoje oglasi u odabranoj kategoriji, ispiše se poruka da trenutno nema oglasa u odabranoj kategoriji.

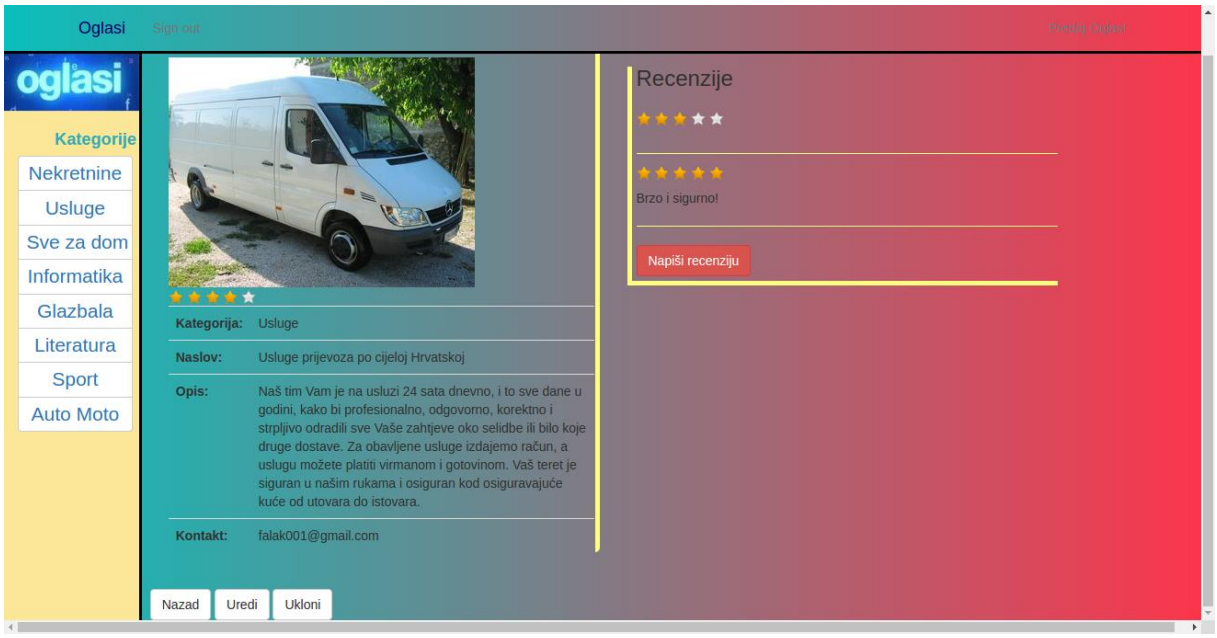


Sl. 4.28. Kategorija bez oglasa



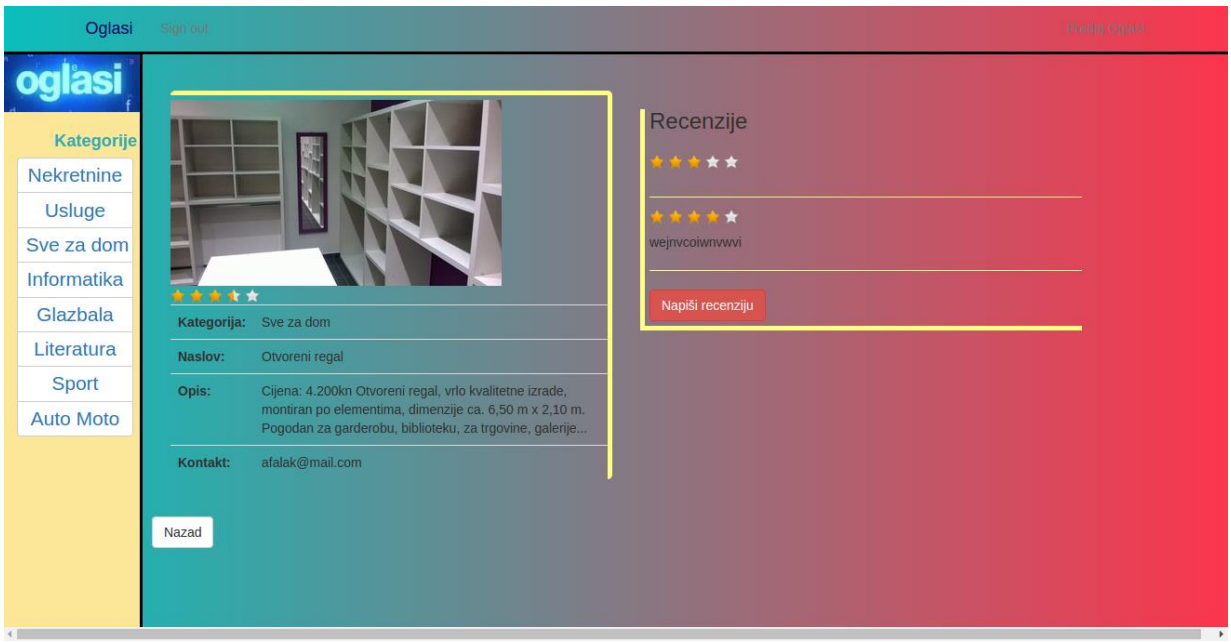
Sl. 4.29. Kategorija sa oglasima

Nakon što je oglas kreiran, može mu se pristupiti pritiskom na sliku oglasa, tada se otvara *show* prikaz koji je zadužen za izgled stranice oglasa.



Sl. 4.30. Oglas 1

S lijeve strane se nalazi slika i podaci o oglasu, s desne strane su smješteni komentari i ocjene te je moguće ostaviti individualnu recenziju. Na slici 4.30. se nalazi oglas koji pripada trenutnom korisniku, stoga ga korisnik može uređivati i obrisati pritiskom na tipke "Uredi" i "Obriši". Pod dijelom za kontakt se nalazi *email* adresa korisnika koji je stvorio oglas.



Sl. 4.31. Oglas 2

Na slici 4.31. se tipke "Uredi" i "Ukloni" ne vide zbog razloga što korisnik koji održava sesiju, ne posjeduje prikazani oglas.

4.5 Primjena Active Record-a

Active Record omogućava automatsko mapiranje između klasa i tablica, atributa i stupaca. Asocijacije između objekata su definirane jednostavnim klasnim metodama. U prethodnim poglavljima se vidi primjena Active Record-a na kreiranju modela, definiranju asocijacija među modelima te dodavanje novih stupaca pomoću migracija. U trenutnom potpoglavlju će se usporediti upiti bazi podataka putem rails konzole (primjenjujući Active Record) i SQL (*Structured Query Language*) naredbi.

U terminalu se pristupa rails konzoli naredbom *$ rails c*, može se također iskoristiti naredba *$ rails c --sandbox*, kako bi se sve promjene izvršene pri izlasku iz konzole vratile u prvobitno stanje (*rollback*). Naredbom *$ rails db* se pristupa bazi podataka direktno pišući upite SQL naredbama.

```
antonio@antonio-ThinkPad-SL510:~/workspace/Oglasi$ rails c
Running via Spring preloader in process 3592
Loading development environment (Rails 4.2.6)
2.3.0 :001 >
```

Sl. 4.32. $ rails c

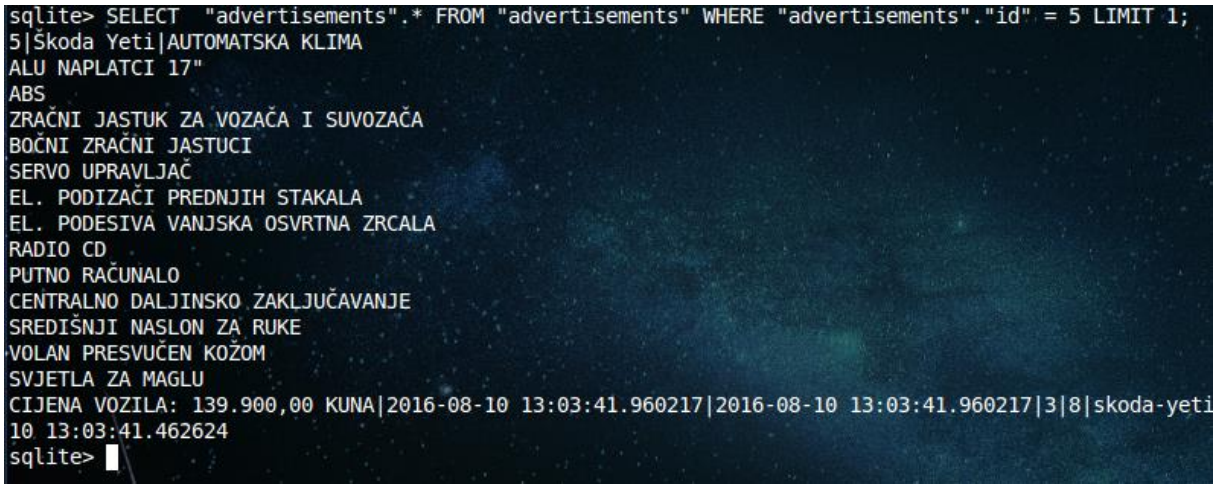
```
antonio@antonio-ThinkPad-SL510:~/workspace/Oglasi$ rails db
SQLite version 3.8.2 2013-12-06 14:53:30
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
```

Sl. 4.33. $ rails db

Baza podataka kojoj se pristupa je SQLite3. U sljedećim primjerima će se pokazati i usporediti korištenje ORM biblioteke i ručno pisanih SQL upita.

```
2.3.0 :003 > Advertisement.find(5)
  Advertisement Load (0.4ms)  SELECT  "advertisements".* FROM "advertisements"
 => #<Advertisement id: 5, title: "Škoda Yeti", body: "AUTOMATSKA KLIMA\r\nALU
 "2016-08-10 13:03:41", user_id: 3, category_id: 8, image_file_name: "skoda-ye
le_size: 86922, image_updated_at: "2016-08-10 13:03:41">
2.3.0 :004 >
```

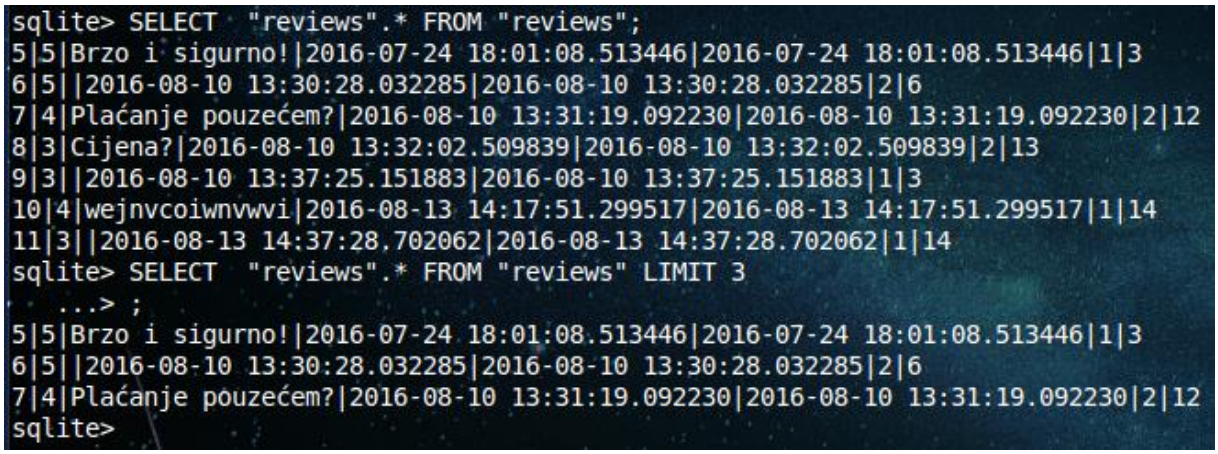
Sl. 4.34. Active Record primjer 1



Sl. 4.35. SQL primjer 1

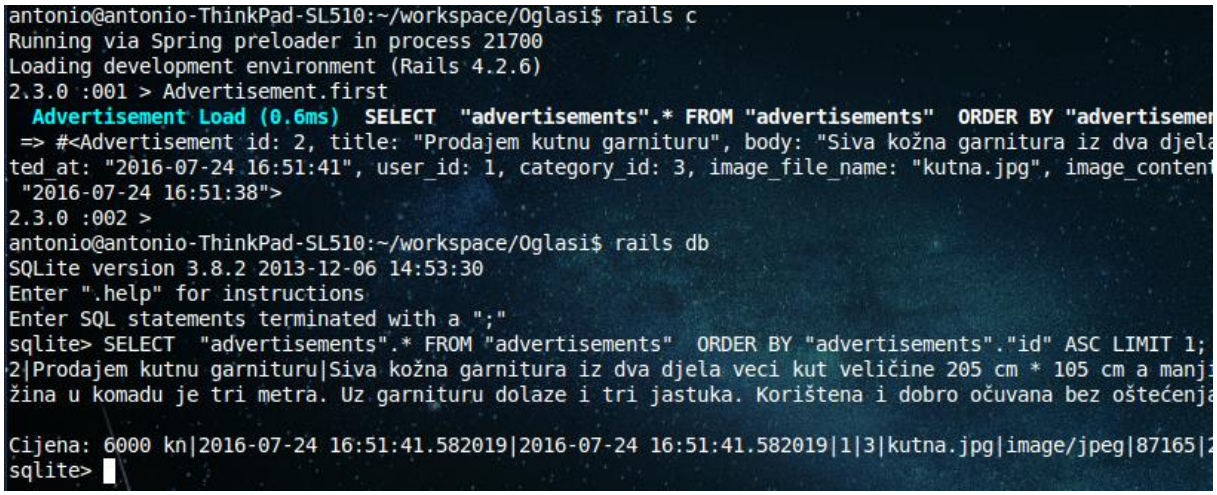
U prvom primjeru je pomoću ORM biblioteke pronađen oglas čiji je atribut id = 5, upisani izraz generira SQL kod koji izvršava upit na bazi podataka. Na slici 4.35. se vidi identični upit u SQLite3 konzoli.

Sl. 4.36. Active Record primjer 2



Sl. 4.37. SQL primjer 2

U primjeru 2 se vidi upotreba jednostavnih upita unutar rails konzole te unutar db konzole.

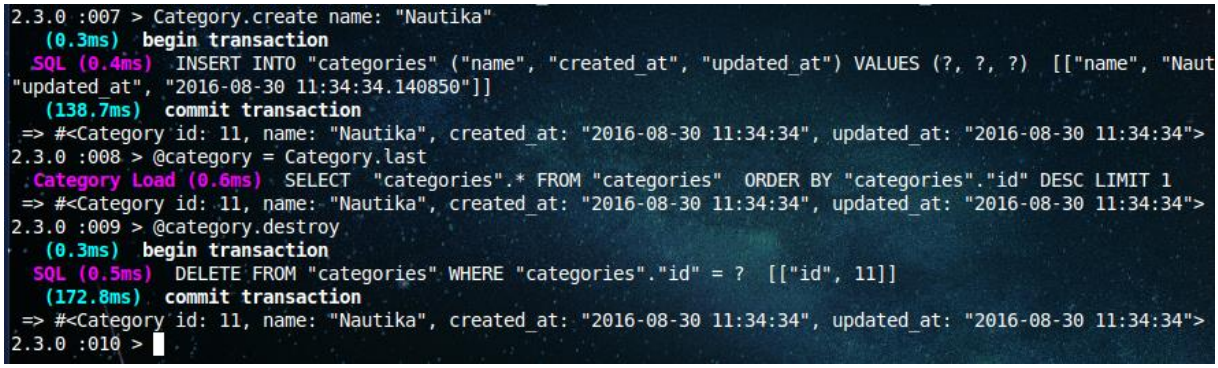


Sl. 4.38. Active Record i SQL primjer 3

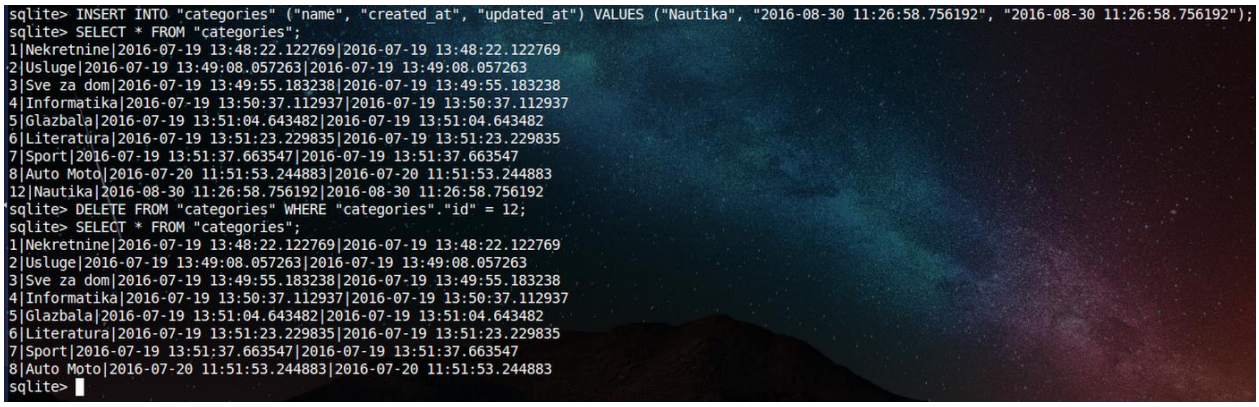
U trećem primjeru se dohvaća prvi oglas iz tablice oglasi, po primarnom ključu (*id*). Očigledna je jednostavnost ORM biblioteke koja naredbom *first* automatski dohvaća prvi objekt po primarnom ključu, ako se eksplicitno ne definira neki drugi atribut za dohvaćanje.

Sl. 4.39. Active Record i SQL primjer 4

U četvrtom primjeru se pretražuje tablica kategorije po zadanom imenu kategorije.

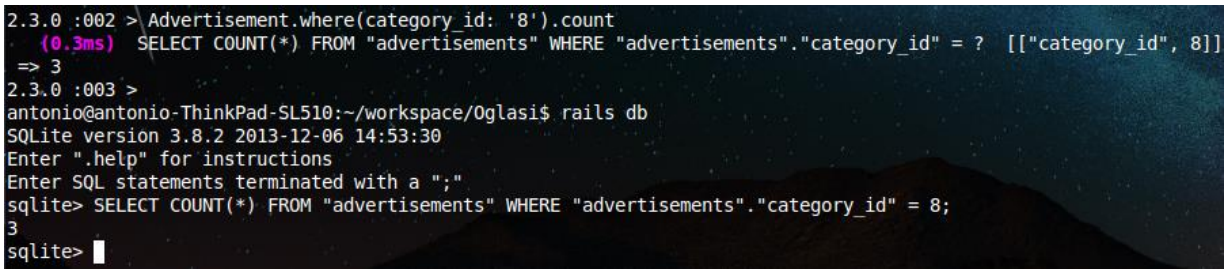


Sl. 4.40. Active Record primjer 5



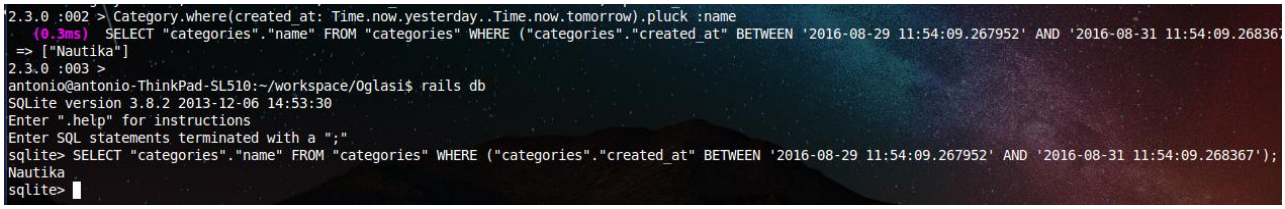
Sl. 4.41. SQL primjer 5

U petom primjeru stvorena je kategorija imena "Nautika", zatim se kategorija briše iz tablice kategorije. Unutar rails konzole kategorija je obrisana tako da je u varijablu *@category* spremljena zadnje stvorena kategorija, tj. "Nautika". Potom je pozvana metoda *destroy* kako bi se kategorija obrisala. Unutar SQLite3 konzole je kategorija obrisana tako da je pronađena pomoću njezinog jedinstvenog ključa (*id* = 12).



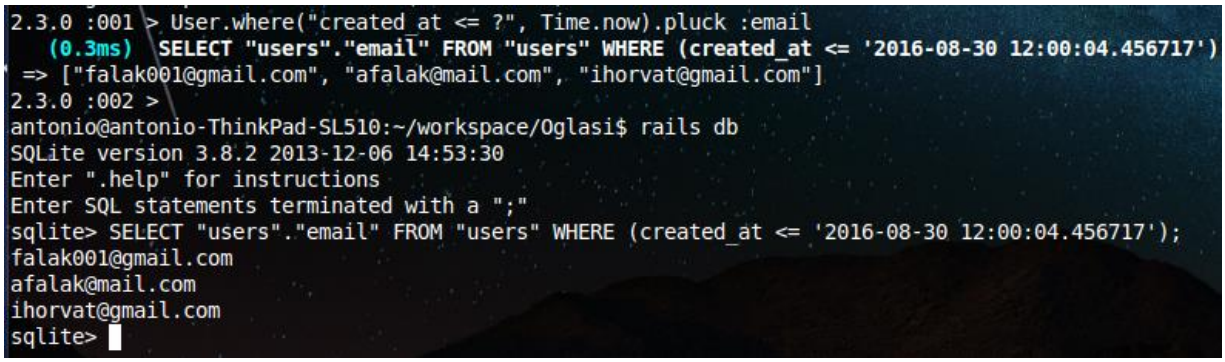
Sl. 4.42. Active Record i SQL primjer 6

Šesti primjer prikazuje rezultat zbrajanja svih oglasa čiji je atribut *category_id* = 8.



Sl. 4.43. Active Record i SQL primjer 7

U sedmom primjeru se dohvaćaju sve kategorije koje su stvorene trenutnog dana, tj. samo njihova imena. U ovom slučaju je trenutnog dana stvorena samo kategorija "Nautika".



Sl. 4.44. Active Record i SQL primjer 8

Osmi primjer prikazuje sve korisnike koji su stvoreni do današnjeg dana, tj. vraća korisničke *email* adrese.

Active Record pruža objektno orijentirano sučelje za pristup i upravljanje podacima unutar bazi. Active Record klasa je povezana s tablicom baze, Active Record instanca odgovara redu tablice te atribut Active Record instance predstavlja vrijednost određenog stupca unutar zadanog reda. Umjesto pisanja sirovih SQL izraza, pristupa se Active Record atributima i služi se Active Record metodama kako bi se pristupilo i upravljalo spremljenim podacima unutar tablica baze podataka.

5. ZAKLJUČAK

Active Record je standardni gem koji dolazi s instalacijom Rails okruženja. Active Record je ORM (*Object Relational Mapping*) biblioteka koja preslikava relacije iz (u ovom slučaju SQL) baze podataka u *Ruby* objekte. Iz prošlog poglavlja može se vidjeti utjecaj Active Record-a od sinteze modela, pa sve do kreiranja kategorija. Funkcionalnost Active Record-a se najviše očituje u migracijama, one omogućavaju kreiranje sheme baze podataka iterativno uz Ruby jezik. CRUD je akronim za *Create-Read-Update-Destroy* te služi kao osnovni set funkcionalnosti kojeg Active Record omogućava.

Cilj završnog rada je teoretski obraditi Rails okruženje i Active Record kao ORM biblioteku, što je obrađeno u prva tri poglavlja. Opisane su značajke *Ruby* programskog jezika te kako i u koju svrhu se koristi Ruby on Rails okruženje. Dan je uvid u značajke Active Record sistema, CRUD strukturu te značajne Active Record migracije i asocijacije između modela.

U četvrtom poglavlju je opisana izrađena *web* aplikacija, što je ujedno bio i praktični dio završnog rada. *Web* aplikacija se naziva "Oglasi" te služi za korisnički pregled i pretraživanje oglasa po kategorijama. Koristi SQLite3 bazu podataka, preko db konzole se unose SQL upiti koji se uspoređuju s *Ruby* metodama Active Record-a. Iz zadnjeg poglavlja se vidi jednostavnost i jasnoća upotrebe ORM biblioteke naspram dugačkih SQL naredbi. Objektno-orijentirano preslikavanje reducirat će programski kod i kroz *caching* će poboljšati performanse naspram korištenja ugrađenog SQL-a ili sučelja na razini poziva s upraviteljem relacijske baze podataka.

U usporedbi s tradicionalnim tehnikama izmjene između objektno orijentiranog jezika i relacijske baze, ORM često smanjuje količinu koda kojeg se treba upisati. Nedostatci ORM alata općenito proizlaze iz visoke razine apstrakcije koja čini nerazumljivim što se točno događa u implementacijskom kodu. Također, veliko oslanjanje na ORM *software* je navedeno kao glavni faktor u proizvodnji loše dizajniranih bazi podataka.

Literatura

[1] Ruby programski jezik - https://www.ruby-lang.org/en/

[2] D. Thomas, A. Hunt - Programming Ruby - The Pragmatic Programmer's Guide - http://ruby-doc.org/docs/ruby-doc-bundle/ProgrammingRuby/index.html?utm_source=twitterfeed&utm_medium=twitter

[3] Ruby vodič za korisnike - http://www.rubyist.net/~slagell/ruby/index.html

[4] M. Hartl - Ruby on Rails Tutorial, Third Edition

[5] MVC arhitektura - https://www.youtube.com/watch?v=3mQjtk2YDkM

[6] ORM biblioteke - http://www.killerphp.com/articles/what-are-orm-frameworks/

[7] Osnove Active Record-a - http://guides.rubyonrails.org/active_record_basics.html

[8] Active Record migracije - http://edgeguides.rubyonrails.org/active_record_migrations.html

[9] Active Record asocijacije - http://guides.rubyonrails.org/association_basics.html

[10] Blog s Rails projektima - https://mackenziechild.me/rails-courses/

Sažetak

Primjena Active Recorda za rad s bazom podataka

Završni rad opisuje razloge i praktičnu namjenu korištenja ORM (*Object Relational Mapping*) biblioteke za Ruby on Rails okruženje - Active Record. Njegova uloga je preslikavanje relacija baze podataka u *Ruby* objekte, u svrhu lakšeg kreiranja i održavanja baze podataka. Izrađena je i opisana *web* aplikacija na kojoj se primjenjuje Active Record za stvaranje modela, provođenje migracija, uspostavljanja asocijacija između modela te vršenja upita bazi podataka s *Ruby* metodama. Projekt koristi MVC arhitekturu, gdje je svaka komponenta (*Model, View i Controller*) posebno izrađena kako bi se uspostavila funkcionalnost *web* aplikacije. Nakon završetka praktičnog dijela završnog rada, uspoređuje se korištenje ORM biblioteke, tj. Active Record-a, sa SQL upitima. Objektno-orijentirano preslikavanje reducira programski kod i kroz *caching* poboljšava performanse naspram korištenja ugrađenog SQL-a.

Ključne riječi: Ruby, Rails, Active Record, ORM, SQL, migracije, asocijacije, modeli, prikazi, kontroleri

Abstract

Active Record application with databases

Bachelor's thesis describes reasons and practical usage of ORM (Object Relational Mapping) library for Ruby on Rails framework - Active Record. Its usage consists of mapping database relations into Ruby objects, for easier development and maintenance of databases. Web application was developed and described, which uses Active Record for creating models, conducting migrations, establishing associations between models and performing queries on database using Ruby methods. Project uses MVC arhitecture, where every component (Model, View and Controller) is specifically designed for establishing web application funcionality. With completion of the practical segment of bachelor's thesis, begins the comparison of ORM library (Active Record) with SQL queries. Object-oriented mapping reduces programming code and through caching improves performance in opposition to using embedded SQL.

Key words: Ruby, Rails, Active Record, ORM, SQL, migrations, associations, models, views, controllers

Životopis

Antonio Falak rođen je 1. travnja 1994. Godine u Osijeku, gdje trenutno živi i obrazuje se. Pohađao je OS "Tin Ujević" u Osijeku u razdoblju od 2001. do 2009. godine. Svoje daljnje obrazovanje nastavlja u srednjoj školi "Elektrotehnička i prometna škola Osijek" u kojoj 2013. godine dobiva zvanje elektrotehničar. Završetkom srednje škole, iste godine uspješno upisuje preddiplomski sveučilišni studij računarstva na Elektrotehničkom fakultetu u Osijeku.

Potpis:

_____

Prilozi

GitHub repozitorij završnog rada: https://github.com/afalak94/Oglasi