

Welcome

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Quae, perferendis nisi nihil debitis aspernatur doloribus quam eveniet quibusdam tempora esse! Repudiandae, assumenda, debitis doloribus ea eius ab at quae totam. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Asperiores, inventore aliquid ipsam debitis nulla vitae omnis nemo quisquam?

345 Carl Street Apt 12, Carrol Rd. Fort Myers, FL 33913

Phone:: +1 239 555 1234 | Email:
marti@rdolcegroup.com

FOR THE LATEST NEWS & UPDATES

SUBSCRIBE

©2021- ERG, ALL RIGHTS RESERVED. MARTI@RDOLCEGROUP.COM | +1 239 555 1234

Building A Sinatra App With MySQL

Phase 2 Project Development for Flatiron School Software Engineering

DRAFT Version: 20210124:813AM

Getting Started

Now the SPEC's As Defined by the Project

DEVELOPMENT + GitHub

HTML + CSS

DEVELOPMENT + GitHub

SETTING UP THE FILE STRUCTURE

INSTALL MYSQL

| - CONFIG/DATABASE.YML PREP - LOCAL ENVIRONMENT

Adding the database.yml

AGENT CONTACT INFO TABLE

(OFFICE/FINANCIAL) PORTFOLIO TABLE

LISTINGS DATA TABLE

OBJECT.RELATIONAL.MAPPING

Resources:

Getting Started

As I am a new Ruby developer, I've learned the importance of understanding how the process works for getting a project started from scratch.

You have to start with a plan. Typically the work begins when the stakeholders and the business team have decided on a new technology 'business' requirement which then evolves into a list of specifications outlining the functionality and features of this new business initiative.

This application uses Sinatra with Active Record with a MySQL database. My 'client' is a project I need to complete to graduate from Flatiron School.

With this idea in mind, my Phase 2 Requirements from Flatiron School are represented in the chart below.



REQUIREMENTS & SPECIFICATIONS

1. Build an MVC (Links to an external site.) Sinatra application.
2. Use ActiveRecord (Links to an external site.) with Sinatra.
3. Use multiple models.
4. Use at least one has_many relationship on a User model and one belongs_to relationship on another model.
5. Must have user accounts - users must be able to sign up, sign in, and sign out.
6. Validate uniqueness of user login attribute (username or email).
7. Once logged in, a user must have the ability to create, read, update and destroy the resource that belongs to user.
8. Ensure that users can edit and delete only their own resources - not resources created by other users.
9. Validate user input so bad data cannot be persisted to the database.
10. BONUS: Display validation failures to user with error messages (Links to an external site.).
(This is an optional feature, challenge yourself and give it a shot!)

Now the SPEC's As Defined by the Project

Technical specifications define functional and feature goals for the project. These can be further defined as user stories to ensure you have met each requirement as a developer.

- [] Use Sinatra to build the app
- [] Use ActiveRecord for storing information in a database
- [] Include more than one model class (e.g. User, Post, Category)
- [] Include at least one has_many relationship on your User model (e.g. User has_many Posts)
- [] Include at least one belongs_to relationship on another model (e.g. Post belongs_to User)
- [] Include user accounts with unique login attribute (username or email)
- [] Ensure that the belongs_to resource has routes for Creating, Reading, Updating and Destroying
- [] Ensure that users can't modify content created by other users
- [] Include user input validations
- [] BONUS - not required - Display validation failures to user with error message (example form URL e.g. /posts/new)
- [] Your README.md includes a short description, install instructions, a contributors guide and a link to the license for your code

Confirm

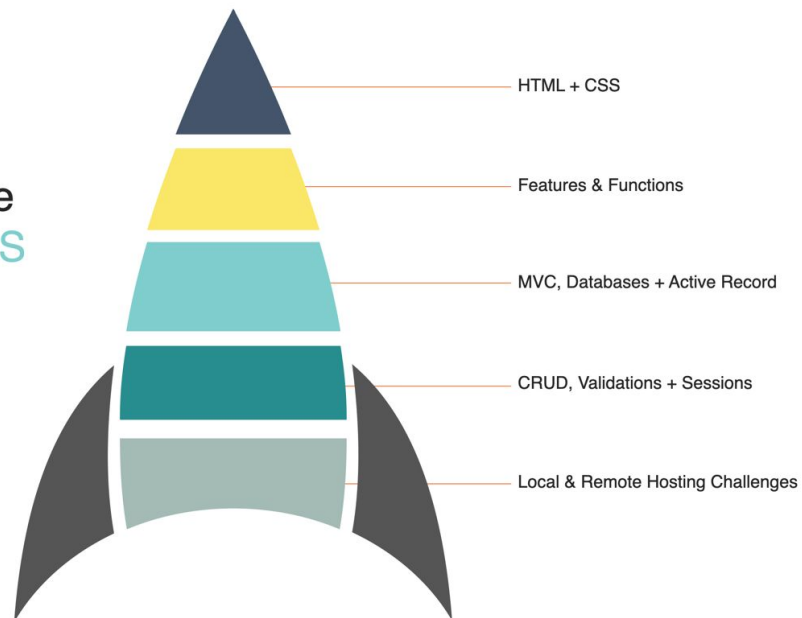
- [] You have a large number of small Git commits
 - [] Your commit messages are meaningful
 - [] You made the changes in a commit that relate to the commit message
 - [] You don't include changes in a commit that aren't related to the commit message
-



RESEARCH & PLANNING

BUILDING the WORKFLOWS

Describes my methodology for approaching my development workflow



Getting started on a project of this size as a newby can be overwhelming for some. I've elected to break the overall specifications into modular topics. That way I can plan my approach.

DEVELOPMENT + GITHub

Discusses how I use RubyMine to speed up the development of Github projects and how I manage to merge different branches for each iteration of my project.

HTML + CSS

The User Experience or the Front - End defines the structure of the application and how an end-user interacts with the features and functionality of the application. I'll use a layout.erb to define my overall theme and build my pages using CSS.

Features & Functions

My project tracks the status of our guest rentals and payments received. First I'll build the application in Sinatra using basic concepts in Ruby.

MVC, Databases + Active Record

Choosing a database framework is the way to achieve MVC using Sinatra. Based on the requirements for this project we will use Active Record. However I'll use MySQL as the backend for my application.

CRUD, REST, USERS, Validations + Sessions

I'll need to determine how to use routes to get and post data from the back-end. In addition, I'll also need to add in features for making changes and/or deleting that data. Using a combination of HTML/ERB pages will achieve this. Thinking of it as a series of actions: list all, creating, saving, editing, searching, filtering and deleting data.

Users Roles & Logins

Understanding roles or user permission should validate the current users permissions and update the database should those permissions change.

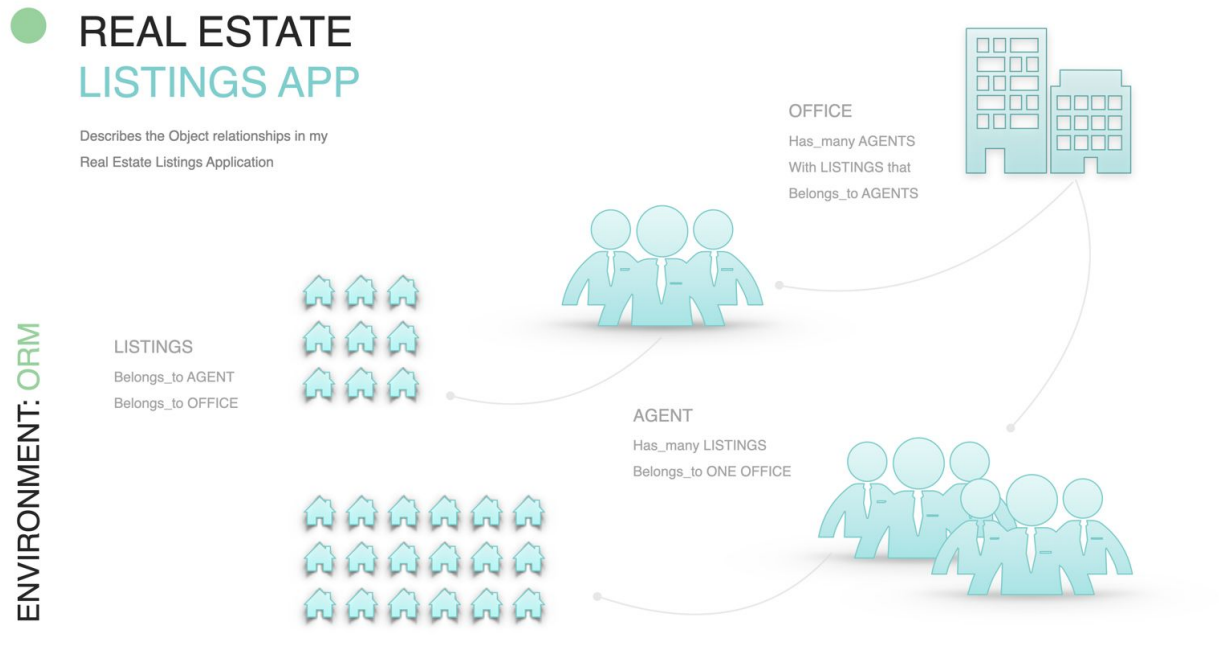
Sessions works with permissions to validate if the current user is logged in or not, establishes that current user's permissions and tracks the users activities while visiting the site.

Users should be able to register or sign-up. This will require: sign-up, login, and redirects to handle the user login/logout.

Local & Remote Hosting Challenges

After the project is completed and before I submit it for review, I'll need to plan on hosting it on a remote server. For the purpose of this session it was suggested we use Heroku.

Now that I have all that figured out, I can actually start planning my code.



DEVELOPMENT + GitHub

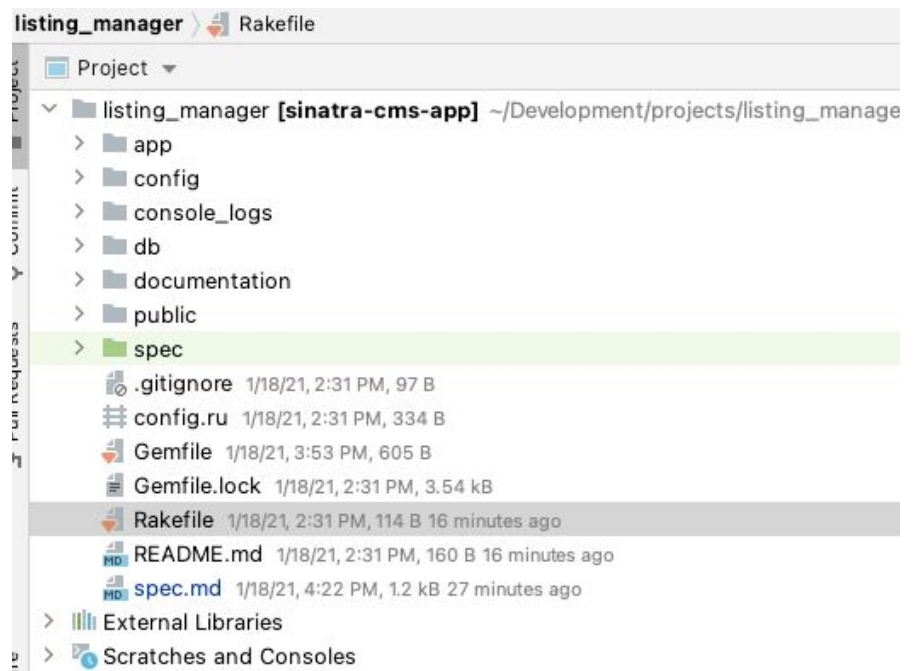
I like to work on both Mac and PC platforms because my work as a contractor gives me far more options if I'm familiar with both. My ORMS, in this case, is MySQL 5.7.32 with ActiveRecord and Sinatra.

	WINDOWS ¹	MAC BIG SUR
MySQL DB	Windows (x86, 32-bit), MSI Installer	brew install mysql@5.7.32
Ruby	GEMFILE	GEMFILE
IDE	Rubymine 2020.3	Rubymine 2020.3

¹ See Windows Eventmachine error: <https://github.com/eventmachine/eventmachine/issues/820>

SETTING UP THE FILE STRUCTURE

- app
 - controllers
 - application_controller.rb
- models
- views
- config
- db
- documentation
 - Sinatra With MySQL - Google Docs.pdf
- public
 - images
 - javascript
 - stylesheets
 - favicon.ico
- spec
- .gitignore
- config.ru
- Gemfile
- Gemfile.lock
- Rakefile
- README.md
- spec.md



INSTALL MYSQL

First, install MySQL for your OS and ensure the gem mysql2 is installed or added to your Gemfile prior to using bundle install.

| - CONFIG/DATABASE.YML PREP - LOCAL ENVIRONMENT

I install MySQL using Homebrew²

```
brew install mysql@5.7
```

After installing, I'll need to add the \$PATH to my zsh shell path using:

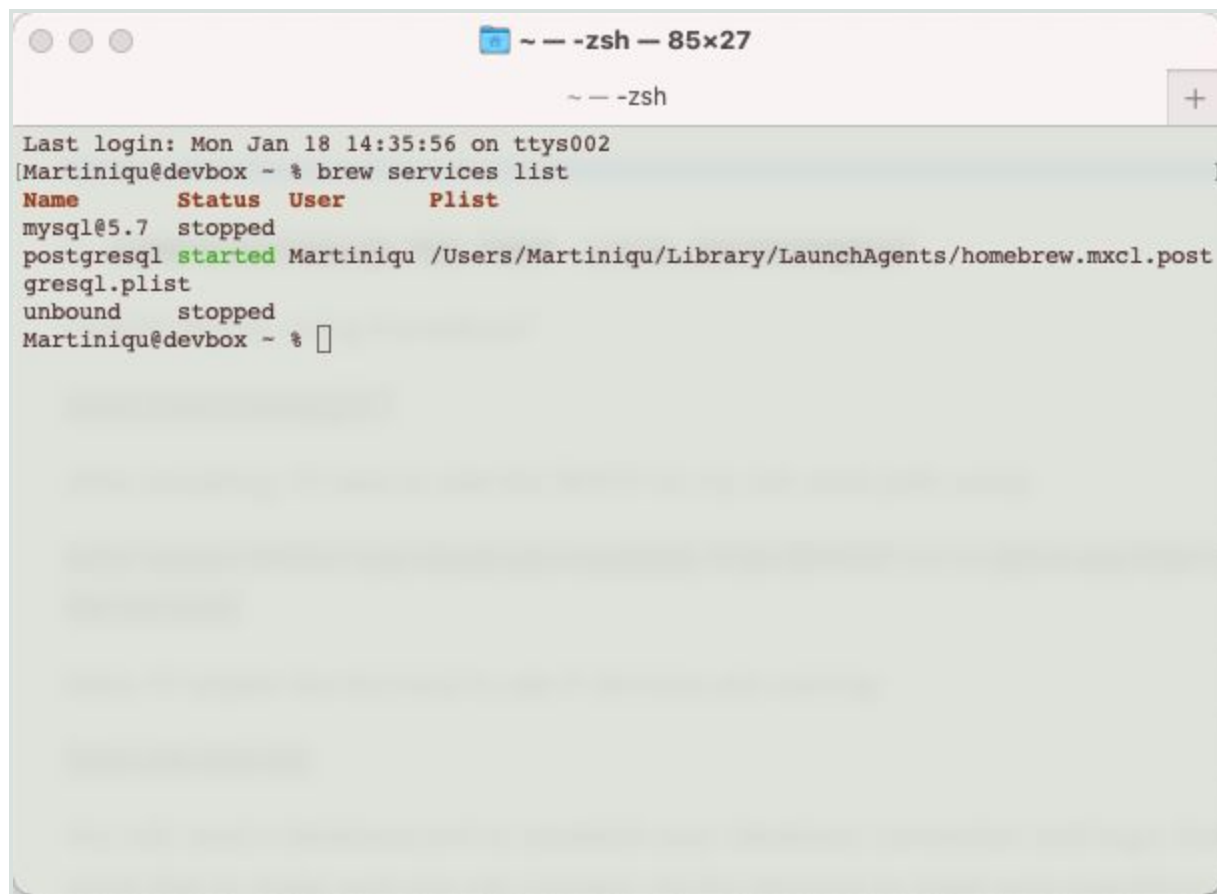
```
echo 'export PATH="/usr/local/opt/mysql@5.7/bin:$PATH"' >> ~/.zshrc and then restart the terminal.
```

Next, I'll reopen the terminal to see if services are running:

```
brew services list
```

² Install MySQL on Mac OS with homebrew:

<https://medium.com/macoclock/installing-mysql-5-7-using-homebrew-974cc2d42509>



```
~ --zsh -- 85x27
~ --zsh

Last login: Mon Jan 18 14:35:56 on ttys002
[Martiniqu@devbox ~ % brew services list
Name      Status User      Plist
mysql@5.7  stopped
postgresql started Martiniqu /Users/Martiniqu/Library/LaunchAgents/homebrew.mxcl.postgresql.plist
unbound    stopped
Martiniqu@devbox ~ %
```

Starting the MySQL Service:

```
brew services start mysql@5.7
```



```
~ --zsh -- 85x7
~ --zsh

[Martiniqu@devbox ~ % brew services start mysql@5.7
==> Successfully started `mysql@5.7` (label: homebrew.mxcl.mysql@5.7)
Martiniqu@devbox ~ %
```

Securing the MySQL Installation

```
mysql_secure_installation
```



```
~ -- mysql_secure_installation -- 85x14
~ -- mysql_secure_installation

[Martiniqu@devbox ~ % mysql_secure_installation

Securing the MySQL server deployment.

Connecting to MySQL using a blank password.

VALIDATE PASSWORD PLUGIN can be used to test passwords
and improve security. It checks the strength of password
and allows the users to set only those passwords which are
secure enough. Would you like to setup VALIDATE PASSWORD plugin?

Press y|Y for Yes, any other key for No: y
```

View clip online [here](#)

Finally, brew install mysql-connector-c

<https://medium.com/@nazrulworld/mysql-connector-c-api-libmysqlclient-install-and-configure-in-macos-ddd3ece8c3d3>

Adding the database.yml

You will need a database.yml to establish your database connection and login, but it's a good idea to make sure you can connect via the terminal to make sure everything is set up correctly.

The screenshot shows an IDE window titled "listing_manager - database.yml". The editor displays the contents of the `database.yml` file, which is configured for a development database named `listings_development` with a pool size of 5, and fields for `username` and `password`. The file is highlighted with a yellow background.

Below the editor, a terminal window is open, showing the output of the `mysql> SHOW DATABASES;` command. The output lists four databases: `information_schema`, `mysql`, `performance_schema`, and `sys`. The terminal also shows the MySQL version (5.7.32) and copyright information.

```

database: listings_development
pool: 5
username:
password:

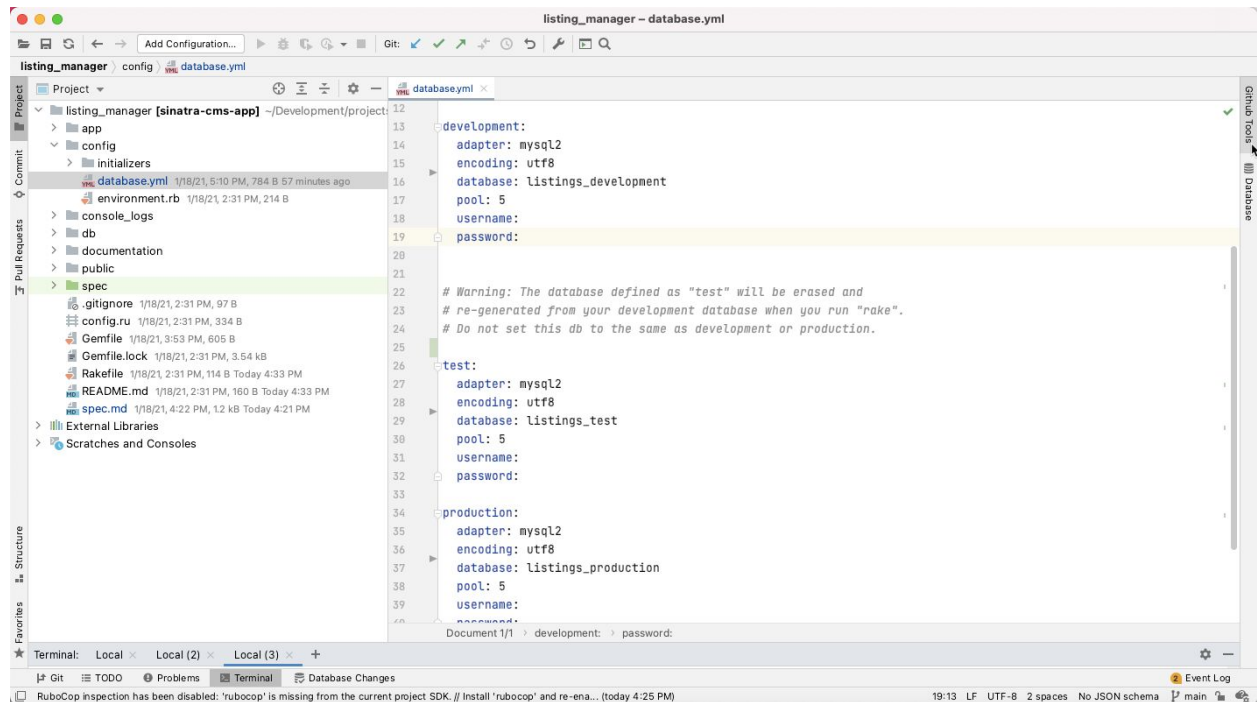
# Warning: The database defined as 'test' will be created and
# dropped and recreated every time you run the rails server.
Document 1/1 -> development: -> password:

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
4 rows in set (0.00 sec)

mysql>

```

The database.yml connects to the database environments for development, test, and production environments³ because I'll need to sync this to a GitHub account for review.



³ I'll need to research how to securely add a secret password for the logon.

ACTIVERECORD PATTERNS & CONVENTIONS

- * Table name: lowercase plural name of model - agents
- * Model filename: singular lowercase (underscored) - agent.rb
- * Class name: singular camelcase - Agent

I've added this as a note from the Rails class we were given this week as it filled in some gaps for me.

USE RAKE -T

To create my database using the database.yml, I'll need to run rake db:create, afterwards I'll plan my ORM and begin my migrations.

CREATING THE DATABASE

% `rake db:create`

Created database 'listings_development'

Created database 'listings_test'

DESIGNING ROLES/PERMISSION TABLE

I'd like to start with the Roles table. In essence it will look something like this:

FIELDS	DATATYPE	NULL	KEY	EXTRA
Role_ID	INTEGER	No	PRI	Auto_Inc
Role_name	INTEGER	No		

ROLES TABLE: CREATING THE MIGRATION, MODEL & SEEDS

```
% rake db:create_migration create_roles_table role_name:string
```

Role.rb

```
class Role < ActiveRecord::Base

  # Validate if table exists == true, if not...

end
```

seeds.rb

```
# db/seeds.rb

require_relative '../app/models/role'

Role.create({role_name: "admin"}, {role_name: "agent"}, {role_name: "guest"})
```

```
% rake db:seed
```

The screenshot displays a web application interface with a database table named 'roles'. The table has four columns: 'id', 'role_name', 'created_at', and 'updated_at'. It contains three rows of data:

id	role_name	created_at	updated_at
1	admin	2021-01-20 14:0...	2021-01-20 14:0...
2	agent	2021-01-20 14:0...	2021-01-20 14:0...
3	guest	2021-01-20 14:0...	2021-01-20 14:0...

The interface also shows a file explorer on the left with a project structure including 'db', 'migrate', 'schema.rb', 'seeds.rb', 'documentation', 'public', and 'spec'. The bottom status bar indicates that the roles were synchronized successfully.

ROLES TABLE: ADDING A CONTROLLER AND VIEWS

Now that my role table is set up, I'll need to add a controller and views to manage the data in this table.

The controller and views introduce concepts related to CRUD and REST. These include patterns for ROUTES and their associated actions.


VERB	ROUTE	ACTION	DESCRIPTION
GET	<code>'/roles'</code>	<code>@role = Role.All</code>	Display All
GET	<code>'/roles/new'</code>	<code>@role = Role.new</code>	Add new with Form
POST	<code>'/roles'</code>	<code>@role = Role.save(role_params) + redirect</code>	Add new Form Action
GET / ID	<code>'/roles/:id'</code>	<code>@role = Role.find(params[:id])</code>	Show/Find by ID
GET /EDIT ID	<code>'/roles/:id/edit'</code>	<code>@role = Role.update(params[:id]) + flash</code>	Edit by ID
PATCH	<code>'/roles/:id'</code>	<code>@role.update(role_params) + flash</code>	Update by ID
PUT	<code>'/roles/:id'</code>	<code>@role.update(role_params) + flash</code>	Replace by ID
DELETE	<code>'/roles/:id'</code>	<code>@role.destroy(params[:id])</code>	Delete by ID

ROLES PAGES: INDEX, SHOW, NEW, EDIT

I've added RoleController routes for index, show and new edit erb's. Next, I'll start working on CRUD allowing for new, editing, patching, and putting entries. Because the RoleController will serve as my permissions admin, I will not include a delete option.

The R. Dolce Group

HOMELOGINLISTINGSAGENTS COMPANIES



About Roles

Roles determine whether or not a user has permission to view or edit data on the site.

[Roles Home](#) [View](#) [New](#) [Edit](#)

345 Carl Street Apt 12, Carrol Rd. Fort Myers, FL 33913

Phone:: +1 239 555 1234 | Email:
marti@rdolcegroup.com


REGISTER YOUR PROPERTIES TODAY!

SUBSCRIBE

©2021- ERG, ALL RIGHTS RESERVED. MARTI@RDOLCEGROUP.COM | +1 239 555 1234

The R. Dolce Group

HOMELOGINLISTINGSAGENTSCOMPANIES



Show All Roles

ID.	Role Name	Created At	Updated
1	admin	2021-01-20 16:03:48 UTC	2021-01-20 16:03:48 UTC
2	agent	2021-01-20 16:03:48 UTC	2021-01-20 16:03:48 UTC
3	new_user	2021-01-20 16:03:48 UTC	2021-01-20 16:03:48 UTC

Roles HomeViewNewEdit

345 Carl Street Apt 12, Carrol Rd. Fort Myers, FL 33913

Phone:: +1 239 555 1234 | Email: marti@rdolcegroup.com

REGISTER YOUR PROPERTIES TODAY!


SUBSCRIBE

©2021- ERG, ALL RIGHTS RESERVED. MARTI@RDOLCEGROUP.COM | +1 239 555 1234

Show's all the roles currently in the system. This feature would only be accessible to admin users.

The R. Dolce Group

HOMELOGINLISTINGSAGENTS COMPANIES



Add New Role

Role Name

CREATE ROLE

[Roles Home](#) [View](#) [New](#) [Edit](#)

345 Carl Street Apt 12, Carrol Rd. Fort Myers, FL 33913

Phone:: +1 239 555 1234 | Email:
marti@rdolcegroup.com

REGISTER YOUR PROPERTIES TODAY!

SUBSCRIBE

©2021- ERG, ALL RIGHTS RESERVED. MARTI@RDOLCEGROUP.COM | +1 239 555 1234

ROLES PAGES: ADDING THE CRUD

The forms & Pages:

```
<h1>Add New Role Form</h1>

<section id="page">

  <form class="form" action="/roles" method="post">

    <label for="role_name" class="label-name">Role Name</label>

    <input type="text" id="role_name" name="name" maxlength="40" class="field
field-name" />

    <input type="submit" value="Create Role" class="button" />

  </form>

  <nav>

    <ol>

      <li><a href="/roles/">Roles Home</a></li>

      <li><a href="/roles/show">View</a></li>

      <li><a href="/roles/new">New</a></li>

      <li><a href="/roles/edit">Edit</a></li>

    </ol>

  </nav>

</section>

<h1>Show Roles Page</h1>

<section id="page">

  <table class="table-striped">

    <thead>

      <tr>

        <th>ID</th>
```

```
<th>Role Name</th>

<th>Created</th>

<th>Updated</th>

<th></th>

</tr>

</thead>

<tbody>

<tr>

<td><a href='/roles/<%= @role.id %>/edit'><%= @role.id %></a></td>

<td><%= @role.role_name %></td>

<td><%= @role.created_at.to_s %></td>

<td><%= @role.updated_at.to_s %></td>

<td> <form method="post" action="/roles/<%= @role.id %>">

    <input type="hidden" name="_method" value="delete" />

    <input type="submit" value="Delete" class="button" />

</form></td>

</tr>

</tbody>

</table>

<nav>

<ol>

<li><a href="/roles">Roles Home</a></li>

<li><a href="/roles/new">New</a></li>

</ol>

</nav>

</section>
```

```

<h1>Edit Role</h1>

<section id="page">

  <form class="form" action="/roles/<%= @role.id %>" method="post">

    <input type="hidden" name="_method" value="patch">

    <label for="role_name" class="label-name">Role Name</label>

    <input type="text" id="role_name" name="role[role_name]" value="<%=
@role.role_name %>" />

    <input type="submit" value="Update Role" class="button" />

  </form>

  <nav>

    <ol>

      <li><a href="/roles/new">Roles Home</a></li>

      <li><a href="/roles/new">New</a></li>

    </ol>

  </nav>

</section>

# ROLES CONTROLLER

# frozen_string_literal: true

# Roles

class RoleController < ApplicationController

```

```
get "/roles" do

  @role = Role.all

  erb :"/roles/index.html"

end

# new

get '/roles/new' do

  @role = Role.new

  erb :"/roles/new.html"

end

# create

post '/roles' do

  @role = Role.create(params)

  redirect to "/roles/#{@role.id}"

end

# show

get '/roles/:id' do

  @role = Role.find(params[:id])

  erb :"/roles/show.html"

end

# edit

get '/roles/:id/edit' do

  @role = Role.find(params[:id])
```

```
    erb :"/roles/edit.html"
  end

  # update
  patch '/roles/:id' do

    @role = Role.find(params[:id])

    @role.update(params[:role])

    redirect to "/roles/#{@role.id}"
  end

  # destroy

  delete '/roles/:id' do

    Role.destroy(params[:id])

    redirect to '/roles'
  end

  def destroy

    Role.destroy(params[:id])

    redirect to '/roles'
  end
end
```

Rack::MethodOverride - config.ru

Working on my EDIT/PATCH in the project has been challenging. Sinatra conforms for GET/POST but to EDIT/UPDATE/PATCH, there needs to be a middle tier using the Rack::MethodOverride in addition to making sure routes are set up properly.

```
# config.ru

#

require './config/environment'

begin

  fi_check_migration

  use Rack::MethodOverride

  run ApplicationController

  use RoleController

  use ArticleController

rescue ActiveRecord::PendingMigrationError => e

  warn e

  exit 1

end
```

AGENT CONTACT INFO TABLE

Agents can update/edit personal information and their listings. Mapping out the models helps me understand how they will work with Routes.

This table has information on each agent and is linked to the Listings table and the Portfolio table by way of a foreign key. For privacy reasons, I've not linked this table to any sessions or permissions.

In hindsight, I should have created a separate table to track logins. Maybe that will be a project for Phase 3 development at Flatiron.

In addition, I'll need to assign an association to the Roles table. The idea is to associate table foreign keys during the rake db:create_migrations.

Google Search: [ruby create migration association](#)

Agents Table

FIELDS	DATATYPE	NULL	KEY	EXTRA
Agent_ID	INTEGER	No	PRI	Auto_Inc
Listings_ID	INTEGER	No	FK	Listings Data
Role_ID	INTEGER	No	FK	Listings Data
Username	VARCHAR	Yes		
Password	VARCHAR	Yes		
Office	VARCHAR	Yes		
Email	VARCHAR	Yes		
Company	VARCHAR	Yes		
Contact	VARCHAR	Yes		
Photo	BLOB	Yes		
Created_At	TIMESTAMP	Yes		
Updated_At	TIMESTAMP	Yes		

MODELS/URLS & ROUTES

(OFFICE/FINANCIAL) PORTFOLIO TABLE

Functioning as a financial and visual portfolio for potential investors, prospective clients and other agents or offices, agents manage their portfolios and have control over who has access to the portfolio. The data also functions as a newsletter.

FIELDS	DATATYPE	NULL	KEY	EXTRA
Portfolio_ID	INTEGER	No	PRI	Auto_Inc
Agent_ID	INTEGER	No	FK	Agent
Listing_ID	VARCHAR	No	FK	Listings
Address	INTEGER	Yes		Address, City, State, Zip
Keywords	VARCHAR	Yes		Marketing SEO
Status	VARCHAR	Yes		Sold, New, Rent, Purchase, Lease
Shared_with	ENUM	Yes		(private, shared, public)
Date_Listed	DATE	Yes		4/11/2018
Bedrooms	VARCHAR	Yes		4BR
#Bathrooms	VARCHAR	Yes		5BA
Summary	BLOB	Yes		Description
Square_Footage	VARCHAR	Yes		2500 SQFT
Asking_Price	INTEGER	Yes		\$325,000.00
Photo 1	BLOB	Yes		Featured Image
Photo 2	BLOB	Yes		
Photo 3	BLOB	Yes		
Photo 4	BLOB	Yes		
Created_At	TIMESTAMP	Yes		
Updated_At	TIMESTAMP	Yes		

LISTINGS DATA TABLE

All new listings are added to this table, which functions as a tracker for managing portfolio data and views for each listing.

Listings - Permission - Logs Table

FIELDS	DATATYPE	NULL	KEY	EXTRA
Listings_ID	INTEGER	No	PRI	Auto_Inc
Portfolio_ID	INTEGER	Yes	FK	
Agent_ID	INTEGER	Yes	FK	
Listing_ID	INTEGER	Yes	FK	
Sessions	STRING	Yes	Hash	
Roles	ENUM	Yes		read_only, read_write
Created_At	TIMESTAMP	Yes		
Updated_At	TIMESTAMP	Yes		

OBJECT.RELATIONAL.MAPPING

Because Ruby objects don't map to the way databases store data, we use ORM to do this mapping. If you think of an application as having a front-end, middle-tier and back-end, then this concept falls within the logic of separating out the UI, logic and data in the application.

ACTIVE RECORD - DRY

MODEL.VIEW.CONTROLLER



Resources:

Development + Github

Windows Ruby Eventmachine error

<https://github.com/eventmachine/eventmachine/issues/820>. Run from terminal >
gem install eventmachine --platform ruby -- --use-system-libraries
--with-ssl-dir=c:/msys64/mingw64

MySQL Installation Mac OS

<https://coderwall.com/p/os6woq/uninstall-all-those-broken-versions-of-mysql-and-re-install-it-with-brew-on-mac-mavericks>

<https://www.positronx.io/how-to-install-mysql-on-mac-configure-mysql-in-terminal/>

<https://medium.com/macoclock/installing-mysql-5-7-using-homebrew-974cc2d42509>

Heroku

MySQL

<https://devcenter.heroku.com/articles/cleardb>

<https://devcenter.heroku.com/articles/cleardb#the-cleardb-shared-mysql-complete-tutorial>

Ruby

<https://devcenter.heroku.com/articles/ruby-support#ruby-versions>