Google On Fire

Project Overview

Holly Robertson
callow16@email.franklin.edu

Contents

Project Overview

Amazon Fire Tablets restricts users to only allowing applications to be downloaded from the Amazon Appstore versus the Google Play Store. The issue with this setup is that the Amazon Appstore is not as robust as the Google Play Store. A few examples of popular applications that are not downloadable via the Amazon Appstore:

- YouTube
- YouTube TV
- Gmail
- Chrome
- Hangouts
- Google Maps
- Ring
- WhatsApp Messenger

This website will be a tunnel for Amazon Fire Tablets to download and install the Google Account and Play Services APK files via the Silk Browser.

Technical Stack:

- Written with Ruby on Rails
- MySQL Database
- Developed within RubyMine IDE
- Hosted on AWS Elastic Beanstalk
- Using Okta for sign-up / login framework
- Routed through Route 53 via Amazon Web Services
- Site is accessible via GoogleonFire.com
- Code is available on GitHub
- Written by Holly Robertson for WEBD 435 course at Franklin University

Technical Requirements

- Application must save to an external database.
    - User credentials and information will be stored with MySQL
- Application must provide a user interface.
    - User will access application through a webpage (http://googleonfire.com)
- Application will offer basic CRUD services to external database.
    - User will have access to create, read, update, and delete records from the MySQL database.

Project Deliverables

- Implement Login Process through ~~Okta~~
- Implement Account Management
- Implement Session/User Application Management
- Testing:
    - Test for login encryption
    - Test for login authentication
    - Test for authentication-required before able to download files
    - Usability Testing by end-user

Testing Requirements

Application will be tested to ensure the following:

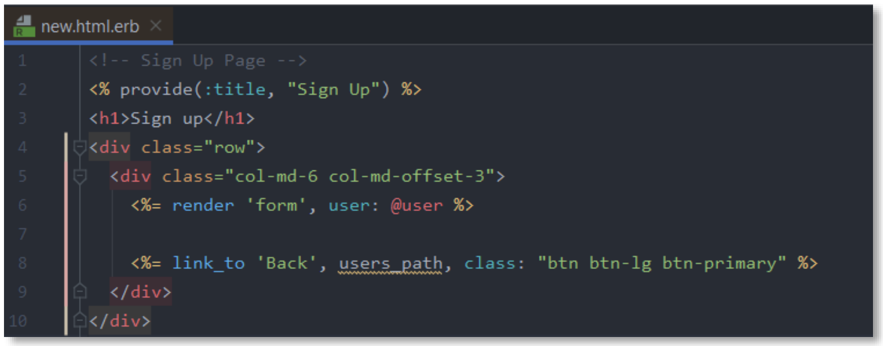
- Functional Requirements:
  - End-user can successfully navigate to the URL on Silk Browser
  - End-user can successfully create an account and login
  - End-user can successfully download files to Amazon tablet from URL via Silk Browser
- Usability Requirements:
  - End-user finds the login process “easy”
  - End-user finds the downloading of files “easy”
  - End-user can successfully complete application’s goal

Testing Methods

- Functionality Testing
    - Developers of the application are utilizing basic local testing for the application.
        - Start lifecycle with new Task:
            - Push to localhost (development environment)
                - Review/Edit/Approve current version
            - Push Approved current version to GitHub
            - Upload zip file to AWS ELB (production environment)
                - Review/Edit/Approve current version
            - Restart lifecycle
- Usability Testing
    - Usability of the application is tested by the developer in the development and production environment.
    - Also utilizing a third-party tester – male, 36, active Google user.

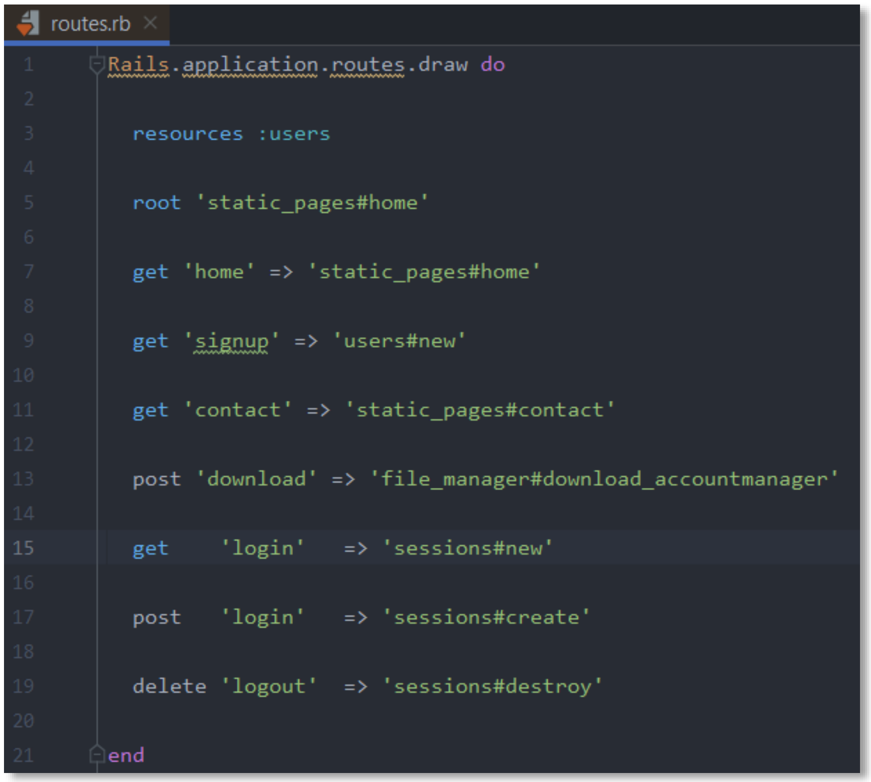
Application Design

- **Sign-Up Process**
  - User navigates to Sign-Up Page (*app\views\users\new.html.erb* via routes.rb)



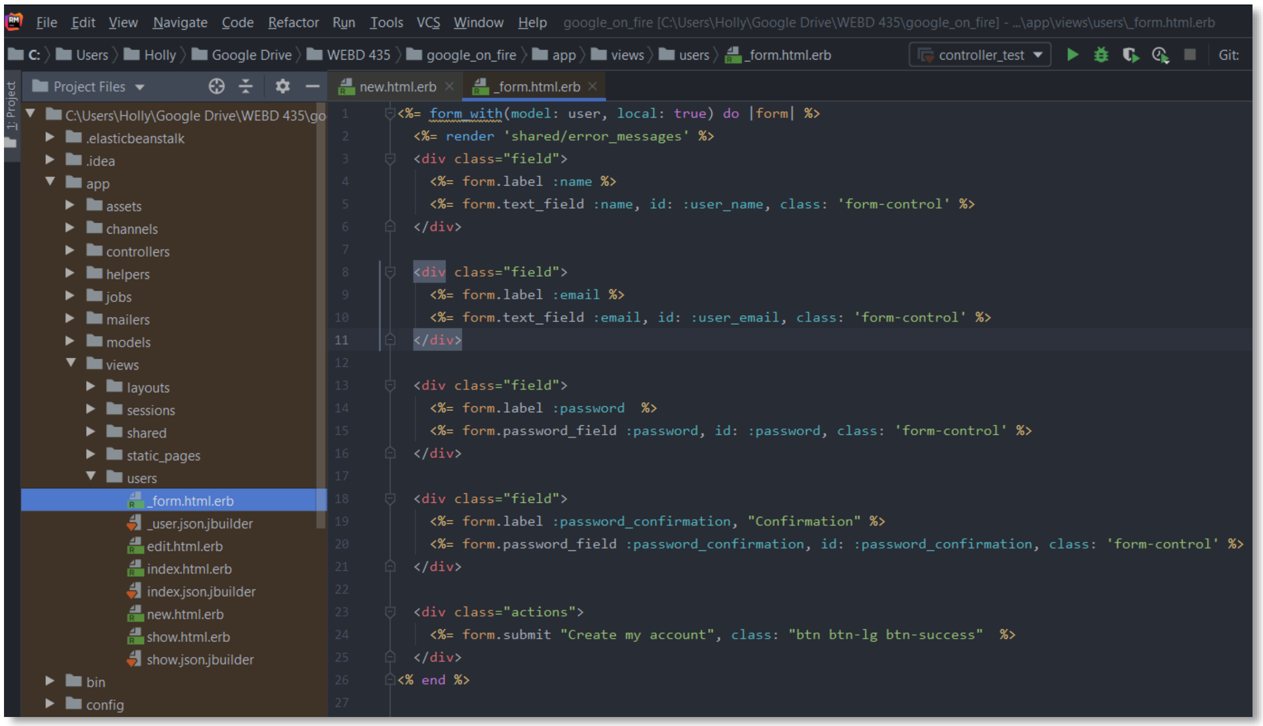
new.html.erb

```
<!-- Sign Up Page -->
<% provide(:title, "Sign Up") %>
<h1>Sign up</h1>
<div class="row">
  <div class="col-md-6 col-md-offset-3">
    <%= render 'form', user: @user %>

    <%= link_to 'Back', users_path, class: "btn btn-lg btn-primary" %>
  </div>
</div>
```



routes.rb

```
Rails.application.routes.draw do

  resources :users

  root 'static_pages#home'

  get 'home' => 'static_pages#home'

  get 'signup' => 'users#new'

  get 'contact' => 'static_pages#contact'

  post 'download' => 'file_manager#download_accountmanager'

  get    'login'   => 'sessions#new'

  post   'login'   => 'sessions#create'

  delete 'logout'  => 'sessions#destroy'

end
```

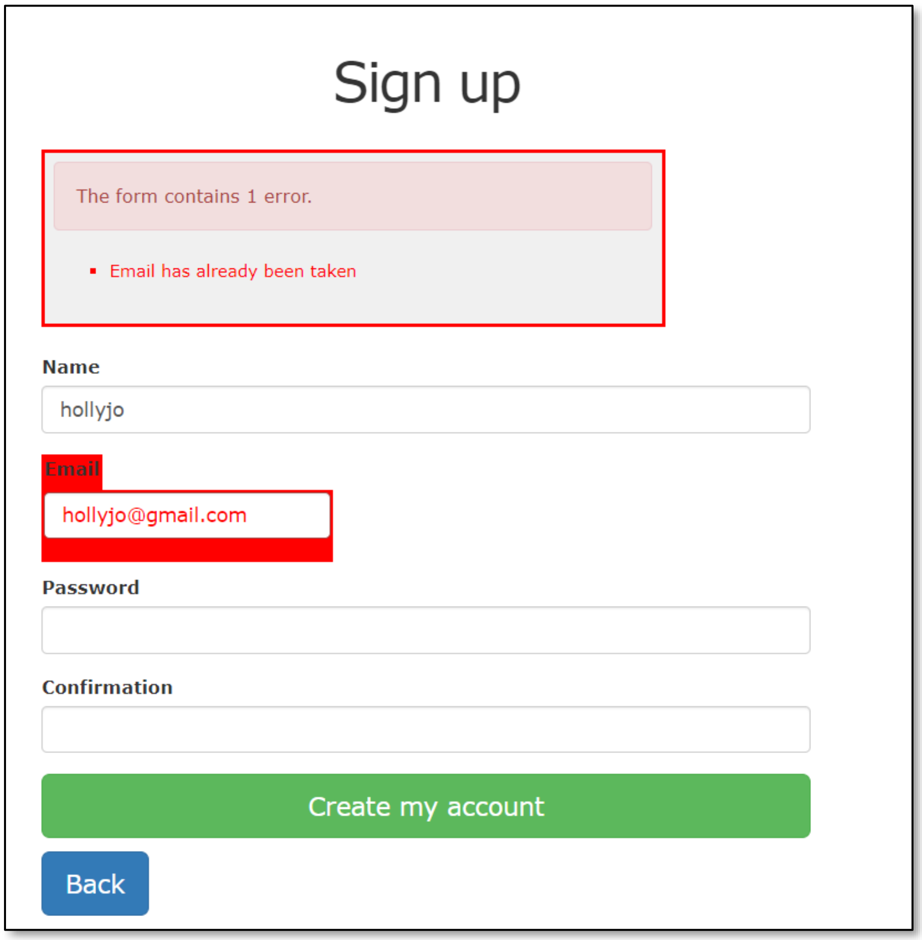

File Edit View Navigate Code Refactor Run Tools VCS Window Help google_on_fire [C:\Users\Holly\Google Drive\WEBD 435\google_on_fire] - ...\app\views\users_form.html.erb
C: 〉 Users 〉 Holly 〉 Google Drive 〉 WEBD 435 〉 google_on_fire 〉 app 〉 views 〉 users 〉 _form.html.erb
controller_test
Git:
Project Files
C:\Users\Holly\Google Drive\WEBD 435\go
.elasticbeanstalk
.idea
app
assets
channels
controllers
helpers
jobs
mailers
models
views
layouts
sessions
shared
static_pages
users
_form.html.erb
_user.json.jbuilder
edit.html.erb
index.html.erb
index.json.jbuilder
new.html.erb
show.html.erb
show.json.jbuilder
bin
config
new.html.erb
_form.html.erb

```erb
<%= form_with(model: user, local: true) do |form| %>
  <%= render 'shared/error_messages' %>
  <div class="field">
    <%= form.label :name %>
    <%= form.text_field :name, id: :user_name, class: 'form-control' %>
  </div>

  <div class="field">
    <%= form.label :email %>
    <%= form.text_field :email, id: :user_email, class: 'form-control' %>
  </div>

  <div class="field">
    <%= form.label :password  %>
    <%= form.password_field :password, id: :password, class: 'form-control' %>
  </div>

  <div class="field">
    <%= form.label :password_confirmation, "Confirmation" %>
    <%= form.password_field :password_confirmation, id: :password_confirmation, class: 'form-control' %>
  </div>

  <div class="actions">
    <%= form.submit "Create my account", class: "btn btn-lg btn-success"  %>
  </div>
<% end %>
```

- This is a basic sign-up form that uses some of Ruby on Rails magic. When this form is submitted it makes an @user object and an entry into the database based on the attributes (:email, :name, :password).
- There are several verifications that take place on the **Front-End** and **Back-End**
  - Valid Email
    - Ruby uses the :email attribute on the **Front-End** in the text_field to tell the browser to make sure that the text put into this field is an email address. Ruby also uses several verifications on the **Back-End** by inspecting the email attribute of the **User** model
      - Email is present (***presence: true***)
      - Email is less than 255 characters (***length: { maximum: 50 }***)

- Email (no matter lower-case) has to be unique in the table

  (***uniqueness: { case_sensitive: false })***



Sign up

The form contains 1 error.

- Email has already been taken

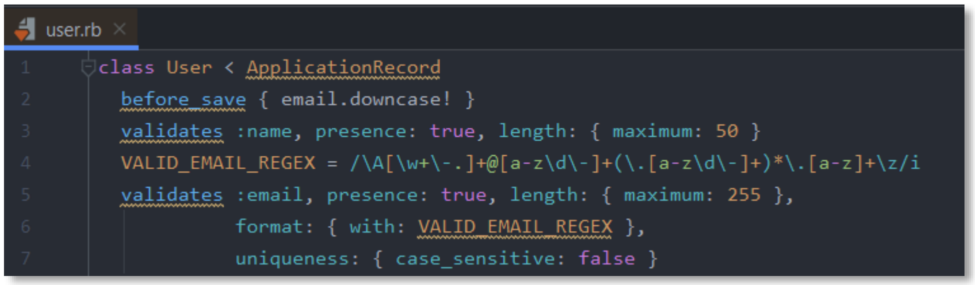
Name: hollyjo

Email: hollyjo@gmail.com

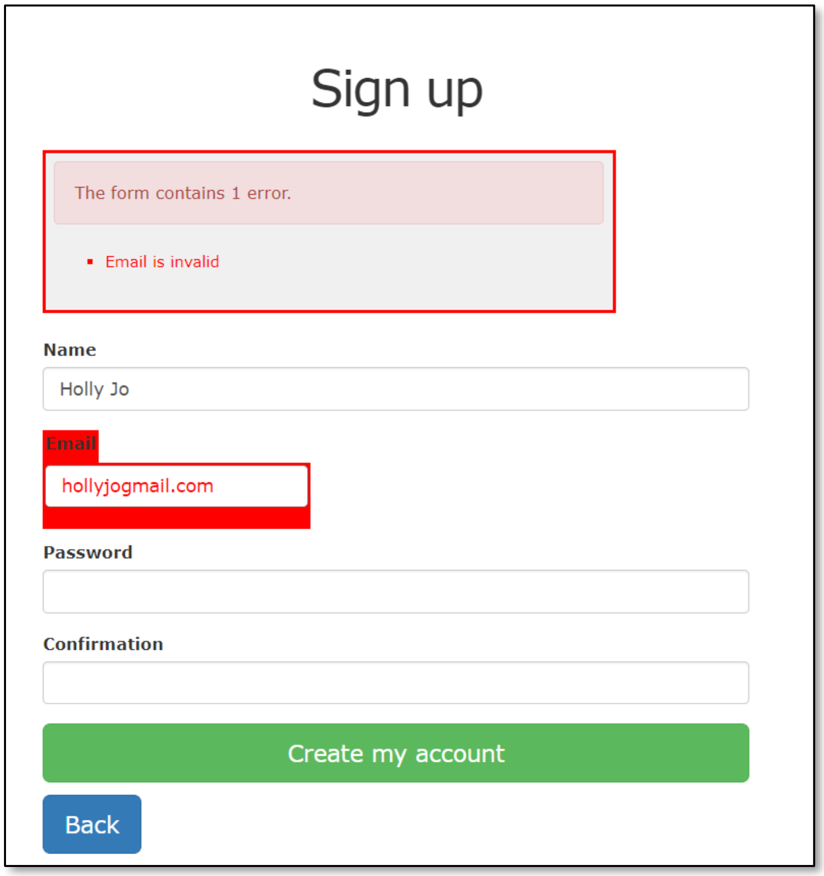
Password:

Confirmation:

- Email matches an industry acceptable format



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



Sign up

The form contains 1 error.

- Email is invalid

Name

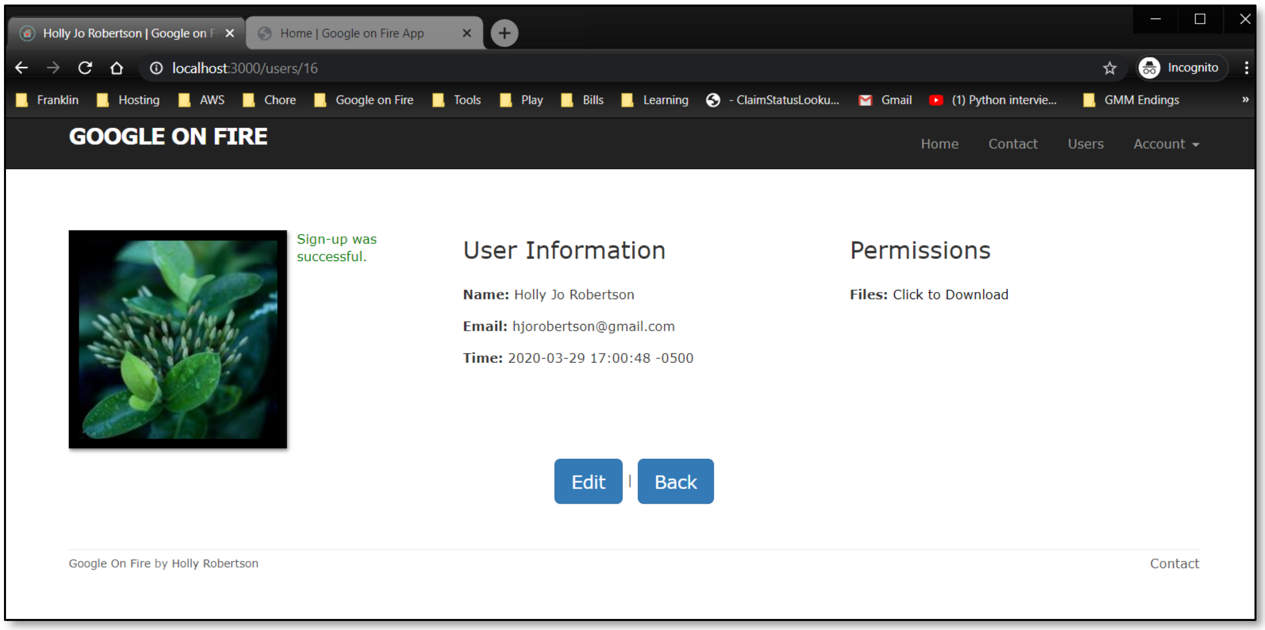
Holly Jo

Email

hollyjogmail.com

Password

Confirmation



Sign-up was successful.

User Information

Name: Holly Jo Robertson

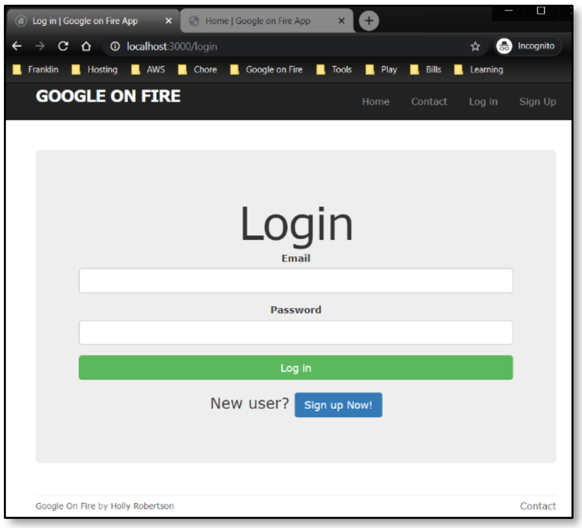
Email: hjorobertson@gmail.com

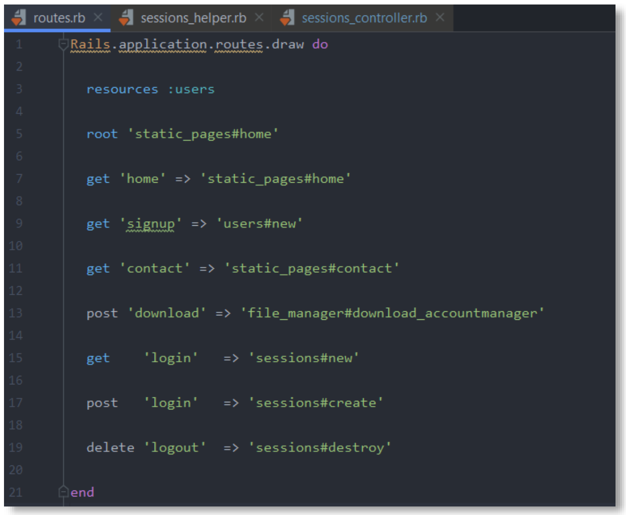
Time: 2020-03-29 17:00:48 -0500

Permissions

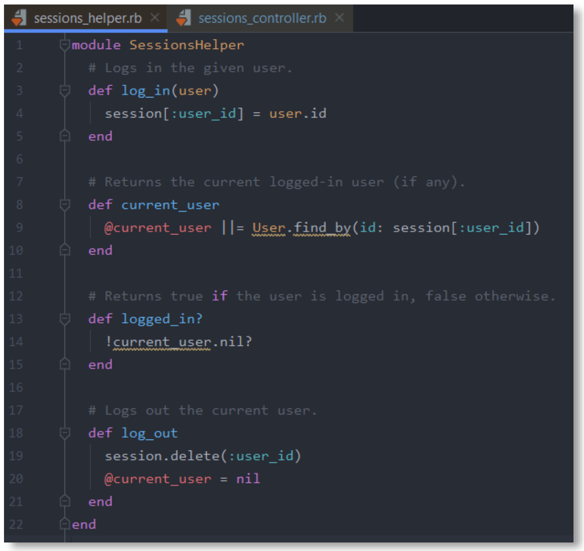
Files: Click to Download

- We wanted to Log the user in when the Sign Up, so we added a **Sessions Controller** (there is a lot of ***Ruby on Rails Magic*** happening here). The **Sessions Controller** sets the :user_id for the **Session Object** (this is the ***Ruby on Rails Magic***) and lets you :authorize the session[:user_id] before allowing certain / specific pages / data to be accessed asking "is session[:user_id] equal to nil?"
  - If so, not logged in.
  - If not, user is logged in.

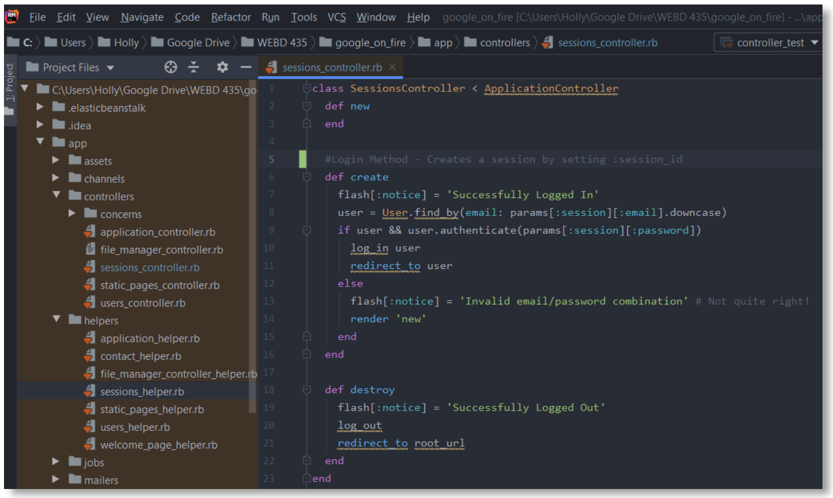




```ruby
Rails.application.routes.draw do

  resources :users

  root 'static_pages#home'

  get 'home' => 'static_pages#home'

  get 'signup' => 'users#new'

  get 'contact' => 'static_pages#contact'

  post 'download' => 'file_manager#download_accountmanager'

  get    'login'   => 'sessions#new'

  post   'login'   => 'sessions#create'

  delete 'logout'  => 'sessions#destroy'

end
```



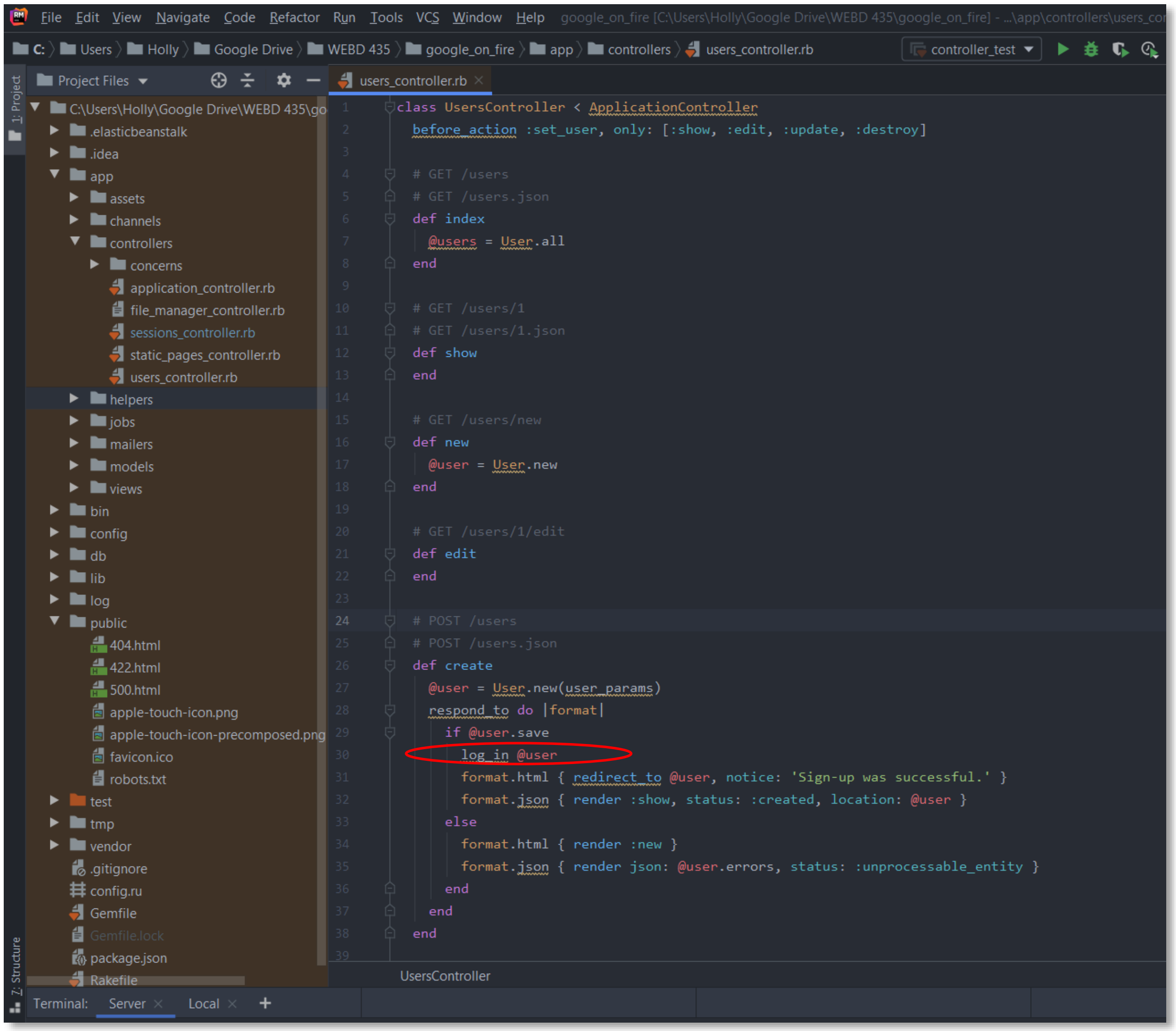
```ruby
module SessionsHelper
  # Logs in the given user.
  def log_in(user)
    session[:user_id] = user.id
  end

  # Returns the current logged-in user (if any).
  def current_user
    @current_user ||= User.find_by(id: session[:user_id])
  end

  # Returns true if the user is logged in, false otherwise.
  def logged_in?
    !current_user.nil?
  end

  # Logs out the current user.
  def log_out
    session.delete(:user_id)
    @current_user = nil
  end
end
```



File
Edit
View
Navigate
Code
Refactor
Run
Tools
VCS
Window
Help
google_on_fire [C:\Users\Holly\Google Drive\WEBD 435\google_on_fire] - ...\app
C: 〉Users 〉Holly 〉Google Drive 〉WEBD 435 〉google_on_fire 〉app 〉controllers 〉sessions_controller.rb
controller_test
Project Files
C:\Users\Holly\Google Drive\WEBD 435\go
.elasticbeanstalk
.idea
app
assets
channels
controllers
concerns
application_controller.rb
file_manager_controller.rb
sessions_controller.rb
static_pages_controller.rb
users_controller.rb
helpers
application_helper.rb
contact_helper.rb
file_manager_controller_helper.rb
sessions_helper.rb
static_pages_helper.rb
users_helper.rb
welcome_page_helper.rb
jobs
mailers

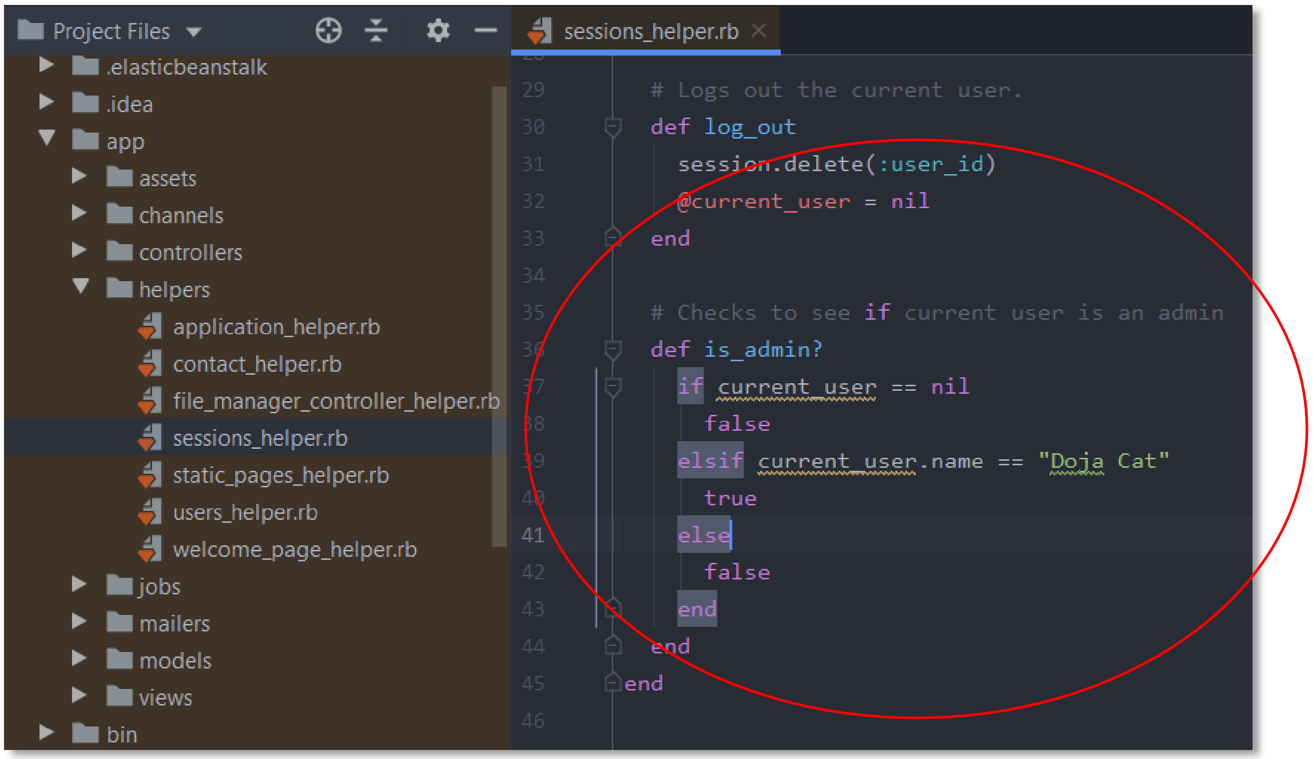
```ruby
class SessionsController < ApplicationController
  def new
  end

  #Login Method - Creates a session by setting :session_id
  def create
    flash[:notice] = 'Successfully Logged In'
    user = User.find_by(email: params[:session][:email].downcase)
    if user && user.authenticate(params[:session][:password])
      log_in user
      redirect_to user
    else
      flash[:notice] = 'Invalid email/password combination' # Not quite right!
      render 'new'
    end
  end

  def destroy
    flash[:notice] = 'Successfully Logged Out'
    log_out
    redirect_to root_url
  end
end
```

- By adding ***log_in @user*** to our signup_path in the **create** method of the **Users Controller** we are automatically logging them in if the user is created
  - by setting the session[:user_id] to the user[:user_id] that is auto-magically created by MySQL when a User is saved to the database)

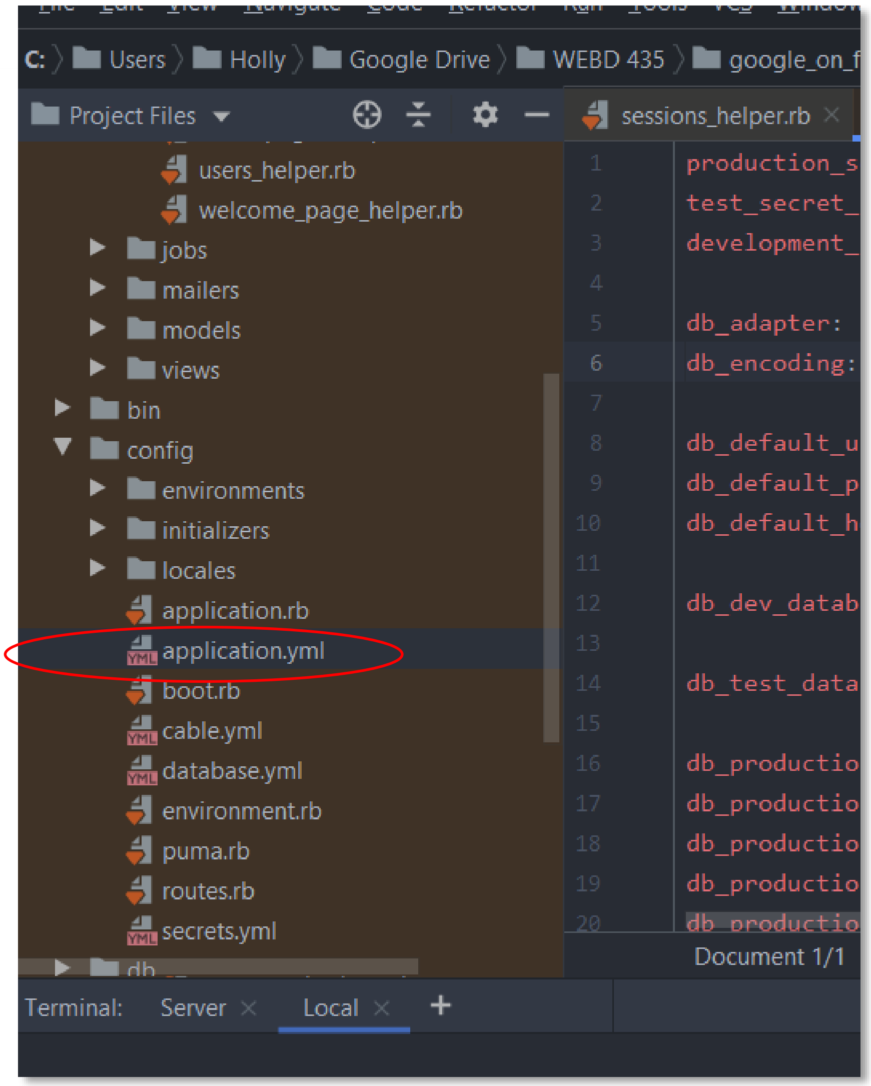


```
class UsersController < ApplicationController
  before_action :set_user, only: [:show, :edit, :update, :destroy]

  # GET /users
  # GET /users.json
  def index
    @users = User.all
  end

  # GET /users/1
  # GET /users/1.json
  def show
  end

  # GET /users/new
  def new
    @user = User.new
  end

  # GET /users/1/edit
  def edit
  end

  # POST /users
  # POST /users.json
  def create
    @user = User.new(user_params)
    respond_to do |format|
      if @user.save
        log_in @user
        format.html { redirect_to @user, notice: 'Sign-up was successful.' }
        format.json { render :show, status: :created, location: @user }
      else
        format.html { render :new }
        format.json { render json: @user.errors, status: :unprocessable_entity }
      end
    end
  end
```

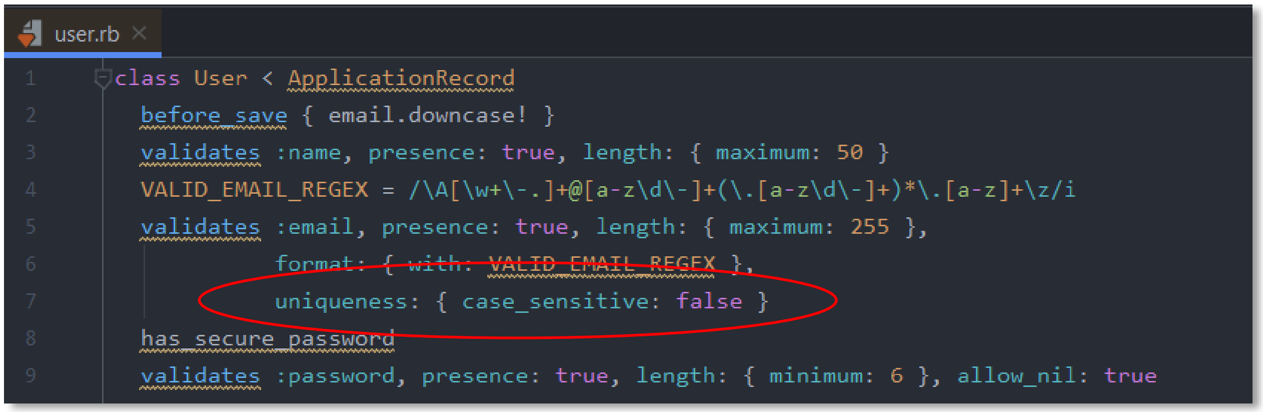
- **Admin Account**
  - The admin account is hard-coded into the project code in the file app/helpers/sessions_helper.rb
  - Right now this is hard-coded by verifying the ***current_user.name*** is "Doja Cat"
  - Eventually this will change to a Ruby Environment Variable that will call the actual value held in the private file app/config/application.yml.
  - The only protection the admin account has currently is that the database is checked for an existing email before a new one can be saved. This code is held in the app/models/user.rb file. As long as the admin account is not deleted nor the email address edited, only one account will have admin access. No one else can sign up with the admin's current email address (not case-sensitive).



```
  # Logs out the current user.
  def log_out
    session.delete(:user_id)
    @current_user = nil
  end

  # Checks to see if current user is an admin
  def is_admin?
    if current_user == nil
      false
    elsif current_user.name == "Doja Cat"
      true
    else
      false
    end
  end
end
```

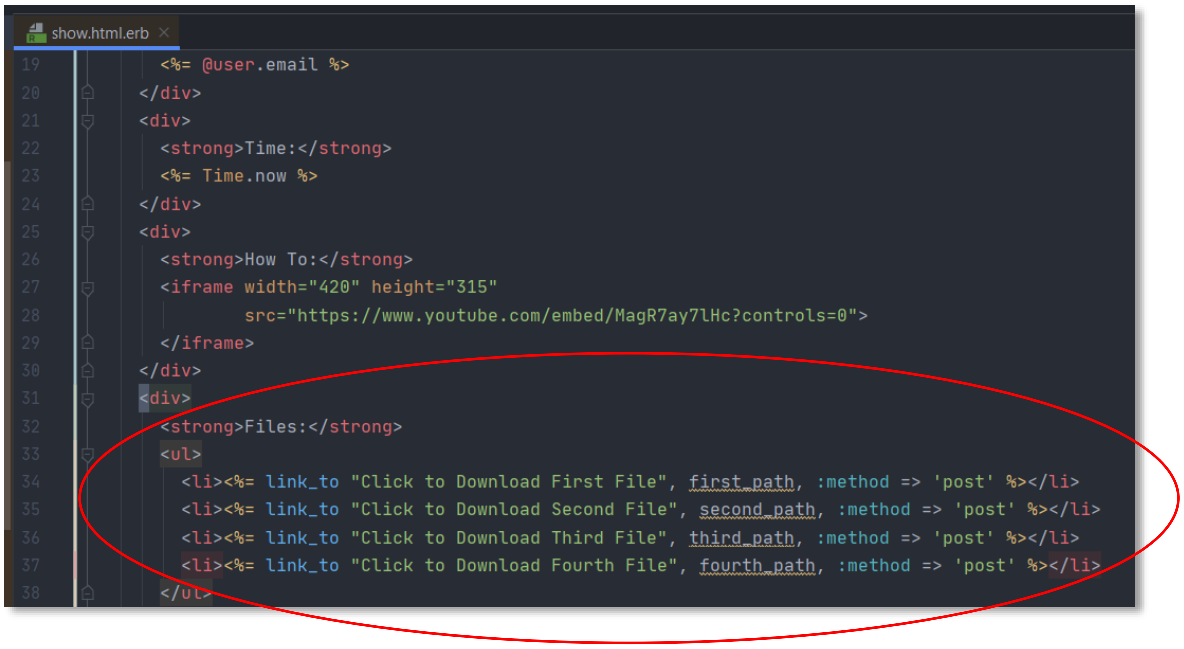

C: 〉 Users 〉 Holly 〉 Google Drive 〉 WEBD 435 〉 google_on_f

```
production_s
test_secret_
development_

db_adapter:
db_encoding:

db_default_u
db_default_p
db_default_h

db_dev_datab

db_test_data

db_productio
db_productio
db_productio
db_productio
db_productio
```



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

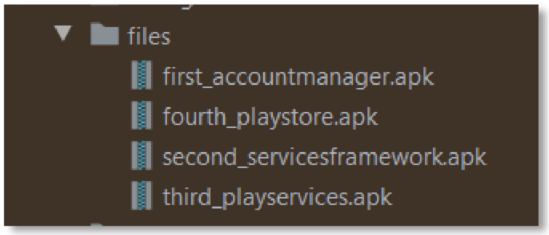
- **File Management**
  - In order for a user to be able to download the APK Files needed to download the Google Play Store on their Amazon Tablet, they need to be signed up and logged in to website. The Download File Links only appear on the Logged-In Page (app/views/users/show.html.erb)



show.html.erb

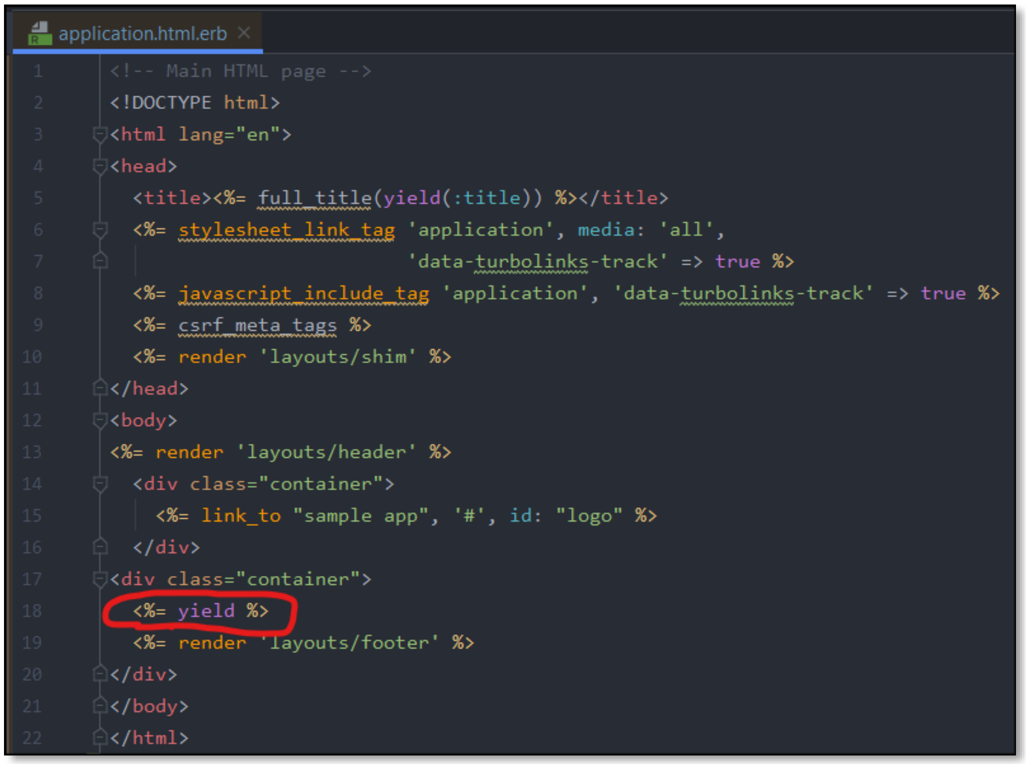
```
        <%= @user.email %>
      </div>
      <div>
        <strong>Time:</strong>
        <%= Time.now %>
      </div>
      <div>
        <strong>How To:</strong>
        <iframe width="420" height="315"
                src="https://www.youtube.com/embed/MagR7ay7lHc?controls=0">
        </iframe>
      </div>
      <div>
        <strong>Files:</strong>
        <ul>
          <li><%= link_to "Click to Download First File", first_path, :method => 'post' %></li>
          <li><%= link_to "Click to Download Second File", second_path, :method => 'post' %></li>
          <li><%= link_to "Click to Download Third File", third_path, :method => 'post' %></li>
          <li><%= link_to "Click to Download Fourth File", fourth_path, :method => 'post' %></li>
        </ul>
```

  - Ruby does allow the downloading of multiple files at one time, but unfortunately this application has no yet integrated that technology in. For right now, the end-users will have to download each file separately. These files are hosted in the Ruby on Rails Asset Pipeline (app/assets/files)



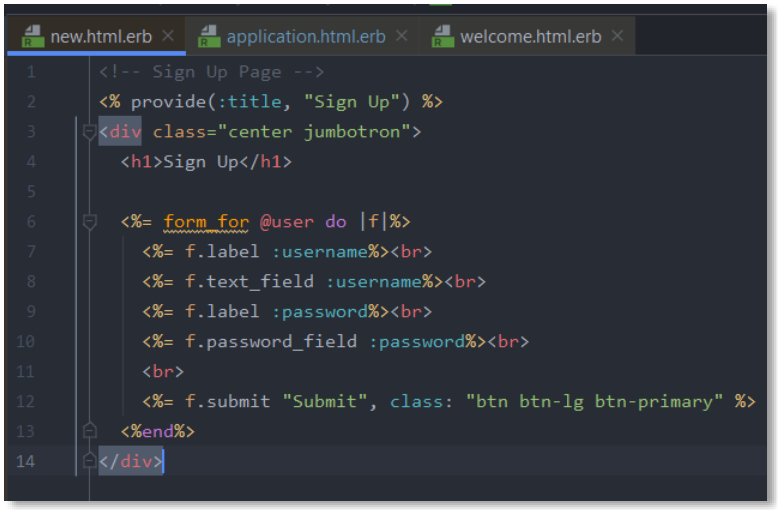
```
▼ files
    first_accountmanager.apk
    fourth_playstore.apk
    second_servicesframework.apk
    third_playservices.apk
```

- Basic HTML structure
    - Every page inherits the following pages:
        - _footer.html.erb
        - _header.html.erb
        - _shim.html.erb
        - application.html.erb
    - Each page that is called is rendered through the <%= yield %> tag in the application.html.erb page

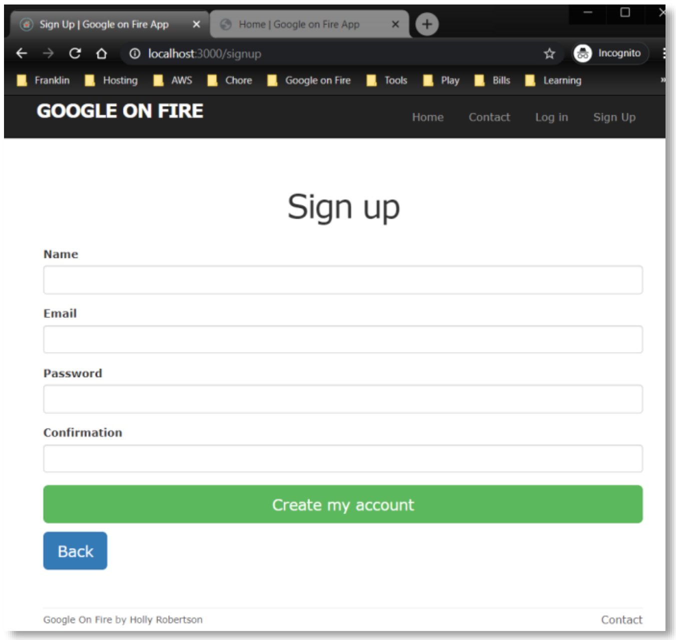

application.html.erb

```
<!-- Main HTML page -->
<!DOCTYPE html>
<html lang="en">
<head>
  <title><%= full_title(yield(:title)) %></title>
  <%= stylesheet_link_tag 'application', media: 'all',
                          'data-turbolinks-track' => true %>
  <%= javascript_include_tag 'application', 'data-turbolinks-track' => true %>
  <%= csrf_meta_tags %>
  <%= render 'layouts/shim' %>
</head>
<body>
<%= render 'layouts/header' %>
  <div class="container">
    <%= link_to "sample app", '#', id: "logo" %>
  </div>
<div class="container">
  <%= yield %>
  <%= render 'layouts/footer' %>
</div>
</body>
</html>
```

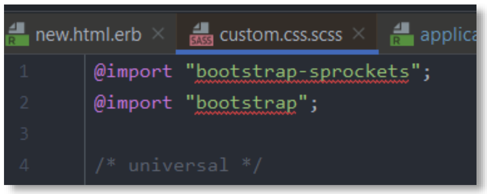
- Example of a page that will be rendered within the <%= yield %> tag



```
<!-- Sign Up Page -->
<% provide(:title, "Sign Up") %>
<div class="center jumbotron">
  <h1>Sign Up</h1>

  <%= form_for @user do |f|%>
    <%= f.label :username%><br>
    <%= f.text_field :username%><br>
    <%= f.label :password%><br>
    <%= f.password_field :password%><br>
    <br>
    <%= f.submit "Submit", class: "btn btn-lg btn-primary" %>
  <%end%>
</div>
```



- Using Font Awesome 4.7, Bootstrap, Boostrap-Sprockets



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

/* universal */
```