Portfolio App Project

The Portfolio App project is a comprehensive web application designed to showcase student portfolios. To facilitate development, the project required the setup of a Docker container for an isolated environment, along with the integration of an IDE (VS Code) for efficient coding. Additionally, a GitHub repository was established for version control and collaboration. Ruby on Rails was incorporated as the core framework, and various terminal commands were used to configure the application and start the server on a localhost for local testing. This setup enables seamless development and deployment while ensuring a stable and reproducible environment. Each setup step is detailed in its respective document for easy reference.

Step 1:

To begin working on your project, ensure you're inside the Linux container via the terminal. Open your project folder in your preferred IDE. Next, create a new branch named ge03models from the main branch to isolate your changes. Use the checkout command to switch to the new branch, allowing you to commit and push updates directly to it. Once the branch is set, run the Rails scaffold command to automatically generate the necessary files for the Student model, including controllers, views, and routes.

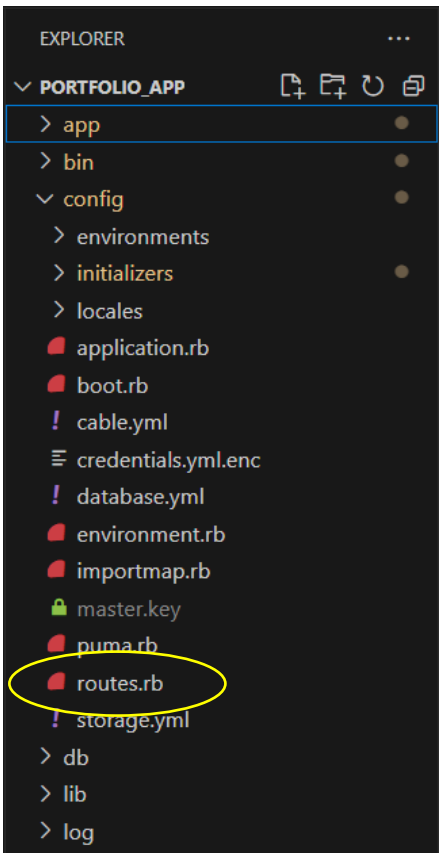
#rails generate scaffold Student name: string school_email:string major:string minor:string graduation_date:date

#rails db:migrate

This scaffold command defines attributes like name, school_email, major, minor, and graduation_date. After scaffolding, execute rails db:migrate to apply these changes to the database, creating the corresponding table for the Student model. Further details about the database migration process will be covered later.

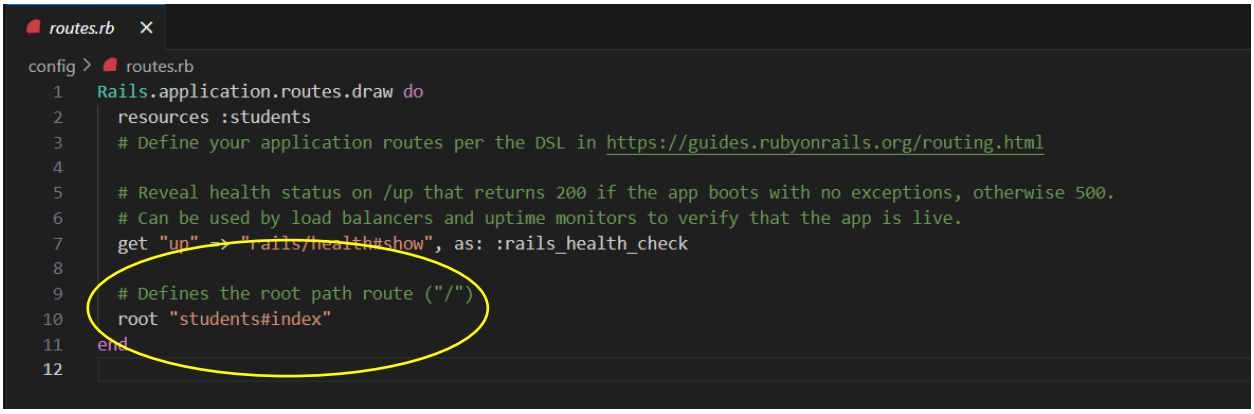
Step 2:

Once the Ruby project files have been downloaded, the docker container has been created, and an IDE is set up we can configure the root route.



To configure the root route of the Portfolio App to direct users to the students#index action, you need to modify the config/routes.rb file in your IDE. This file manages how URLs are mapped to specific actions in the application.

By default, the line resources :students generates all the necessary routes for performing CRUD operations on the Student model. To set the homepage to display the list of students, you add root "students#index". This ensures that when users visit the root URL (/), they are directed to the index action of the StudentsController, which displays all students in the system. With this configuration, the home page of the app will automatically show the list of students as the entry point for users.



Step 3:

Rails scaffolding automatically generates basic CRUD functionality for models, but to customize specific fields like making the school_email unique for each student, you need to modify the model and database. First, create a standalone migration by running the command

 # rails generate migration AddIndexToStudentsSchoolEmail

This will generate a migration file in the db/migrate folder. Open this file and add the necessary code to ensure that the school_email field is unique.

```
20240930194140_add_index_to_students_school_email.rb U
db > migrate > 20240930194140_add_index_to_students_school_email.rb
1  class AddIndexToStudentsSchoolEmail < ActiveRecord::Migration[7.1]
2    def change
3      add_index :students, :school_email, unique: true
4    end
5  end
6
```

Once the migration file is updated, run **rails db:migrate** to apply the changes, and verify the updates by checking the db/schema.rb file to confirm that the unique index was applied. Additionally, in the Student model, add a validation rule to enforce uniqueness at the application level by including **validates :school_email, presence: true, uniqueness: true**. This ensures that duplicate school emails cannot be saved, both in the database and within the Rails application logic.

```
student.rb M ×   20240930194140_add_index_to_students_school_email.rb U
app > models > student.rb
1  class Student < ApplicationRecord
2    has_one_attached :image
3    validates :school_email, presence: true, uniqueness: true
4  end
5
```

Step 4:

To add the profile picture feature using Active Storage, first, install Active Storage in your Rails application by running:

 # rails active_storage:install

This creates the necessary database tables to handle file attachments. After running the migration with **rails db:migrate**, Active Storage will be set up with two tables: active_storage_blobs to store file metadata and active_storage_attachments to associate files with models. To add the profile picture feature to the Student model, include **has_one_attached :image** in the model, allowing each user to upload a single profile image.

student.rb M × 20240930194140_add_index_to_students_school_email.rb U

app > models > student.rb

```ruby
class Student < ApplicationRecord
  has_one_attached :image
  validates :school_email, presence: true, uniqueness: true
end
```

In the students_controller, make sure to permit the **:image** parameter within student_params to ensure the image is accepted during registration.

students_controller.rb × development.rb student.rb M 20240930194140_add_index_to_students_school_email.rb U

app > controllers > students_controller.rb

```ruby
class StudentsController < ApplicationController
  private
    # Use callbacks to share common setup or constraints between actions.
    def set_student
      @student = Student.find(params[:id])
    end

    # Only allow a list of trusted parameters through.
    def student_params
      params.require(:student).permit(:name, :school_email, :major, :minor, :graduation_date, :bio, :image)
    end
end
```

Next, configure Active Storage to store files locally by modifying config/storage.yml to specify the local disk storage service and updating config/environments/development.rb to use **config.active_storage.service = :local**.

development.rb × | student.rb M | 20240930194140_add_index_to_students_school_email.rb U

config > environments > development.rb

```ruby
Rails.application.configure do
  # Enable/disable caching. By default caching is disabled.
  # Run rails dev:cache to toggle caching.
  if Rails.root.join("tmp/caching-dev.txt").exist?
    config.action_controller.perform_caching = true
    config.action_controller.enable_fragment_cache_logging = true

    config.cache_store = :memory_store
    config.public_file_server.headers = {
      "Cache-Control" => "public, max-age=#{2.days.to_i}"
    }
  else
    config.action_controller.perform_caching = false

    config.cache_store = :null_store
  end

  # Store uploaded files on the local file system (see config/storage.yml for options).
  config.active_storage.service = :local

  # Don't care if the mailer can't send.
  config.action_mailer.raise_delivery_errors = false
```

On the front-end, add a file upload field in the user registration form (app/views/students/_form.html.erb) using **form.file_field :image**, which allows users to upload their profile pictures.

students_controller.rb | development.rb | student.rb M | _form.html.erb ×

app > views > students > _form.html.erb

```erb
<%= form_with(model: student) do |form| %>
  <div>
    <%= form.label :graduation_date, style: "display: block" %>
    <%= form.date_field :graduation_date %>
  </div>


  <div>
    <%= form.label :image, "Upload Profile Picture:" %>
    <%= form.file_field :image %>
  </div>
```

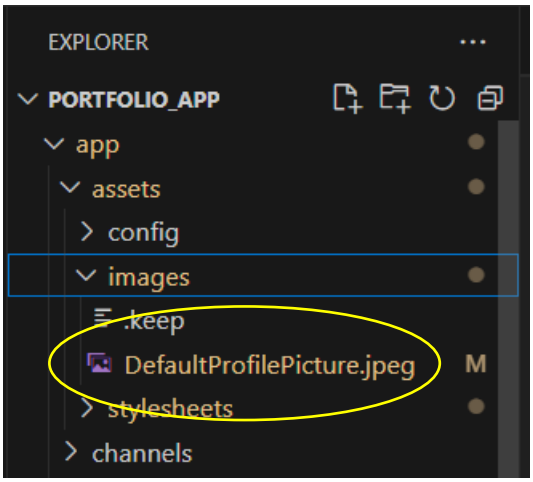
After a successful registration, you can display the uploaded image in the student's profile by calling .image on the student instance. Creating an if-else system for displaying the image allows for the instance of a default image. The code first checks if a student exists and has an attached image using **student.image.attached?**. If the student has uploaded an image, it uses the image_tag helper to render the profile picture, setting the size to "150x150". If no image is attached, a default profile picture (**DefaultProfilePicture.jpeg**) is displayed in its place.

students_controller.rb | development.rb | student.rb M | _form.html.erb | _student.html.erb

app > views > students > _student.html.erb

```
<div id="<%= dom_id student %>">

  <p>
    <% if student && student.image.attached? %>
      <%= image_tag url_for(student.image), size: "150x150", style: "float: top" %>
    <% else %>
      <%= image_tag 'DefaultProfilePicture.jpeg', size: "150x150", style: "float: top" %>
    <% end %>
  </p>
```



The stored image that is called is placed in the images folder (app/assets/images).

Student Profile:

The completion of the student profile feature in the Portfolio App ensures that users can create and manage their personal profiles, with unique names and emails to maintain data integrity. The inclusion of the profile picture upload feature adds personalization to each profile, allowing students to display their images alongside their information. Together, these features provide a comprehensive and user-friendly experience, offering core functionality for the management of student profiles within the app.