Design and Implement Portfolio Projects

1. Requirements Analysis
2. Data Structure Design
    2.1. Project Model
    2.2. Database Migration for Projects
3. Model Relationships and Validations
    3.1. Student Model
    3.2. Project Model
4. Routes Setup
5. Controllers
    5.1. ProjectsController
6. Views
    6.1. Index View (index.html.erb)
    6.2. Show View (show.html.erb)
    6.3. New View (new.html.erb)
    6.4. Edit View (edit.html.erb)
7. Student Show View Modification
8. Testing
    8.1. Model Tests
    8.2. Controller Tests

1. Requirements Analysis

The following user stories and scenarios were analyzed for this feature:

- As a student, I want to add, view, edit, and delete projects associated with my profile.
- Projects should include attributes: title, description, and student_id.
- Navigation should support easy access between students and their associated projects.

2. Data Structure Design

2.1. Project Model

A Project model was created with the following attributes:

- title: String, representing the name of the project.
- description: Text, representing details about the project.
- student_id: Foreign key reference to the Student model, establishing a relationship between students and projects.

2.2. Database Migration for Projects

To implement this data structure, a migration was created and ran:

```
rails generate model Project title:string description:text student:references rails db:migrate
```

3. Model Relationships and Validations

In the Student and Project models, the relationships were established with proper validations.

3.1. Student Model

In app/models/student.rb, the following association was added:

```
ruby
Copy code
class Student < ApplicationRecord
  has_many :projects, dependent: :destroy
end
```

This has_many relationship specifies that a Student can have multiple projects and ensures that projects are deleted if the associated student is deleted.

3.2. Project Model

In app/models/project.rb, the belongs_to association and basic validations were added:

```
ruby
Copy code
class Project < ApplicationRecord
  belongs_to :student

  validates :title, presence: true, length: { maximum: 100 }
  validates :description, presence: true, length: { maximum: 1000 }
end
```

---

4. Routes Setup

Nested routes were added to represent the relationship between students and projects.

In config/routes.rb, the following routes were defined:

```
Rails.application.routes.draw do
  devise_for :students, controllers: {
    registrations: 'students/registrations',
    sessions: 'students/sessions',
    passwords: 'students/passwords'
  }

  resources :students do
    resources :projects
  end

  root "students#index"
  get "up" => "rails/health#show", as: :rails_health_check
end
```

---

5. Controllers

5.1. ProjectsController

The ProjectsController was created with CRUD operations for projects:

```
rails generate controller Projects
```

In app/controllers/projects_controller.rb, the controller actions were implemented:

```
class ProjectsController < ApplicationController
  before_action :set_student
  before_action :set_project, only: [:show, :edit, :update, :destroy]

  def index
    @projects = @student.projects
  end

  def show
  end

  def new
    @project = @student.projects.build
  end

  def create
    @project = @student.projects.build(project_params)
    if @project.save
      redirect_to student_project_path(@student, @project), notice: 'Project was successfully
created.'
    else
```

```
      render :new
    end
  end

  def edit
  end

  def update
    if @project.update(project_params)
      redirect_to student_project_path(@student, @project), notice: 'Project was successfully
updated.'
    else
      render :edit
    end
  end

  def destroy
    @project.destroy
    redirect_to student_projects_path(@student), notice: 'Project was successfully deleted.'
  end

  private

  def set_student
    @student = Student.find(params[:student_id])
  end

  def set_project
    @project = @student.projects.find(params[:id])
  end

  def project_params
    params.require(:project).permit(:title, :description)
  end
end
```

---

6. Views

To provide a user interface for managing projects, the following views were created in app/views/projects/.

6.1. Index View (index.html.erb)

Displays a list of all projects for a specific student and provides links to add, edit, or delete projects.

6.2. Show View (show.html.erb)

Displays detailed information about a single project.

6.3. New View (new.html.erb)

Contains a form for creating a new project.

6.4. Edit View (edit.html.erb)

Contains a form for editing an existing project.

---

7. Student Show View Modification

The student’s show.html.erb view was modified to display associated projects:

Updated app/views/students/show.html.erb:

```
<h2>Projects</h2>
<% if @student.projects.any? %>
  <ul>
    <% @student.projects.each do |project| %>
      <li>
        <h3><%= link_to project.title, student_project_path(@student, project) %></h3>
        <p><%= project.description %></p>
        <%= link_to 'Edit', edit_student_project_path(@student, project), class: 'btn btn-secondary'
%>
        <%= link_to 'Delete', student_project_path(@student, project), method: :delete, data: {
confirm: 'Are you sure?' }, class: 'btn btn-danger' %>
      </li>
    <% end %>
  </ul>
<% else %>
  <p>No projects available for this student.</p>
<% end %>

<%= link_to 'Add New Project', new_student_project_path(@student), class: 'btn btn-primary'
%>
```

---

8. Testing

RSpec tests were added to ensure that the new Project model, associations, and controllers function as expected.

8.1. Model Tests

Model tests for Project in spec/models/project_spec.rb:

```
require 'rails_helper'

RSpec.describe Project, type: :model do
  it { should belong_to(:student) }
  it { should validate_presence_of(:title) }
  it { should validate_presence_of(:description) }
end
```

8.2. Controller Tests

Controller tests for ProjectsController in spec/controllers/projects_controller_spec.rb included testing each CRUD action to verify project creation, viewing, editing, and deleting.