UNIVERSIDAD NACIONAL AUTÓNOMA DE NICARAGUA

(UNAN-LEÓN)
FACULTAD DE CIENCIAS Y TECNOLOGÍA

Carrera:
- Ingeniería en Telemática

Componente:
- Software como un Servicio.

Grupo:
- Grupo #1

Elaborado por:

- Daniela Merary Gómez Forbes.
- Josh David Pastrana Soriano.
- Roderick Alberto Pérez Quintana.
- Chelsea María Moraga Noguera
- Josué Gamaliel Pauth García

Profesor:
- Ervin.

Fecha:
- 06/10//2024

"A la Libertad por la Universidad"

```ruby
class LibrosController < ApplicationController
  before_action :set_libro, only: %i[ show edit update destroy ]

  # GET /libros or /libros.json
  def index
    if params[:query].present?
      @libros = Libro.where("autor LIKE ? OR nombre LIKE ? OR genero LIKE ?", "%#{params[:quer
    else
      @libros = Libro.all
    end
  end

  # GET /libros/1 or /libros/1.json
  def show
  end

  # GET /libros/new
  def new
    @libro = Libro.new
  end
```

El código proporcionado es un controlador en Ruby on Rails que gestiona las operaciones relacionadas con un modelo llamado `Libro`. En resumen, este controlador permite realizar las siguientes acciones:

1. **Listar Libros**: La acción `index` muestra todos los libros o filtra los resultados según una consulta de búsqueda proporcionada por el usuario.
2. **Mostrar un Libro Específico**: La acción `show` permite ver los detalles de un libro en particular.
3. **Crear un Nuevo Libro**: La acción `new` inicializa un nuevo objeto `Libro`, y la acción `create` guarda este objeto en la base de datos, manejando tanto el éxito como los errores en el proceso.
4. **Editar un Libro Existente**: La acción `edit` permite cargar un libro para su edición, y la acción `update` guarda los cambios realizados en el libro.
5. **Eliminar un Libro**: La acción `destroy` elimina un libro de la base de datos y redirige a la lista de libros.
6. **Buscar Libros**: La acción `search` permite buscar libros según un criterio específico y devuelve los resultados en formato JSON.

   Además, el controlador utiliza un método privado `set_libro` para encontrar un libro por su ID y otro método `libro_params` para asegurar que solo se permiten parámetros seguros al crear o actualizar libros. En conjunto, este controlador facilita la gestión de libros en una aplicación web, permitiendo a los usuarios interactuar con los datos de manera efectiva.

```ruby
Rails.application.routes.draw do
  #resources :libros
  root "libros#index"
  resources :libros do
    collection do
      get "search"
    end
  end
```

- **`Rails.application.routes.draw do`**: Este bloque define las rutas de la aplicación. Dentro de este bloque, se especifican las diferentes rutas que la aplicación puede manejar.

- **`root`**: Esta línea establece la ruta raíz de la aplicación, que es la URL que se carga cuando un usuario visita la raíz del sitio (por ejemplo, `http://tuaplicacion.com/`). En este caso, redirige a la acción `index` del controlador `LibrosController`, mostrando la lista de libros.

- **`resources :libros`**: Esta línea crea automáticamente un conjunto de rutas RESTful para el recurso `libros`. Esto incluye rutas para las acciones estándar: `index`, `show`, `new`, `create`, `edit`, `update`, y `destroy`. Cada una de estas acciones corresponde a métodos en el controlador `LibrosController`.
- **`collection do ... end`**: Dentro del bloque de recursos, se define una ruta adicional que no está asociada a un libro específico, sino a la colección de libros. Esto permite agregar rutas personalizadas que operan sobre el conjunto de recursos.
- **`get "search"`**: Esta línea define una ruta que responde a solicitudes GET en la URL `/libros/search`. Esta ruta está asociada a la acción `search` en el `LibrosController`, permitiendo a los usuarios buscar libros según un criterio específico.

Resumen

En resumen, este código configura las rutas de una aplicación Rails para gestionar libros. Establece la página de inicio que muestra la lista de libros y define rutas RESTful para las operaciones CRUD, además de una ruta personalizada para buscar libros. Esto permite que los usuarios interactúen con el recurso `libros` de manera estructurada y eficiente.

```ruby
class Libro < ApplicationRecord
    validates :autor, :nombre, :genero, presence: true
end
```

1. Definición de la Clase:

- **`class Libro < ApplicationRecord`**: Esta línea define la clase `Libro`, que representa un modelo en la base de datos. Al heredar de `ApplicationRecord`, `Libro` tiene acceso a todas las funcionalidades que Rails proporciona para interactuar con la base de datos.

2. Validaciones:

- **`validates`**: Esta línea establece validaciones para el modelo `Libro`. En este caso, se están validando tres atributos: `autor`, `nombre` y `genero`.
- **`presence: true`**: Esta opción asegura que los atributos mencionados no pueden estar vacíos. Si alguno de estos atributos no se proporciona al crear o actualizar un libro, la validación fallará y el objeto `Libro` no se guardará en la base de datos.

Este código define un modelo `Libro` en una aplicación Rails que requiere que los atributos `autor`, `nombre` y `genero` estén presentes para que un libro sea considerado válido. Esto ayuda a mantener la integridad de los datos en la base de datos, asegurando que cada libro tenga la información esencial antes de ser guardado.

```
<p style="color: green"><%= notice %></p>

<% content_for :title, "Libros" %>

<h1>Libros</h1>
<%= form_with(url: libros_path, method: :get, local: true) do %>
  <div>
    <%= label_tag :query, "Buscar libros:" %>
    <%= text_field_tag :query, params[:query] %>
    <%= submit_tag "Buscar" %>
  </div>
<% end %>
<div id="libros">
  <% @libros.each do |libro| %>
    <%= render libro %>
    <p>
      <%= link_to "Show this libro", libro %>
    </p>
  <% end %>
</div>

<%= link_to "New libro", new_libro_path %>
```

- **`<p style="color: green"><%= notice %></p>`**: Este párrafo muestra un mensaje de notificación (almacenado en la variable `notice`) en color verde. La sintaxis `<%= %>` se utiliza para evaluar y mostrar el contenido de la variable en la vista.

- **`<% content_for :title, "Libros" %>`**: Este código establece el contenido para el título de la página. `content_for` permite definir contenido que se puede utilizar en otras partes de la vista, como en el `<head>` de la página.

- **`<h1>Libros</h1>`**: Este encabezado de nivel 1 muestra el título "Libros" en la página, indicando que el contenido principal está relacionado con libros.
- **`<%= form_with(...) do %>`**: Este bloque crea un formulario que se enviará a la ruta `libros_path` utilizando el método GET. El formulario es local, lo que significa que no se enviará de forma asíncrona.

- **`label_tag :query, "Buscar libros:"`**: Crea una etiqueta para el campo de búsqueda.

- **`text_field_tag :query, params[:query]`**: Crea un campo de texto para que el usuario ingrese su consulta de búsqueda. Si hay un valor en `params[:query]`, se mostrará en el campo.
- **`submit_tag "Buscar"`**: Crea un botón de envío con el texto "Buscar".

- **`<div id="libros">`**: Este div contiene la lista de libros.
- **`<% @libros.each do |libro| %>`**: Itera sobre cada libro en la colección `@libros`.
- **`<%= render libro %>`**: Renderiza la vista parcial correspondiente a cada libro, mostrando su información.
- **`<p><%= link_to "Show this libro", libro %></p>`**: Crea un enlace que permite al usuario ver los detalles de cada libro. El enlace utiliza la ruta correspondiente al libro.

- **`<%= link_to "New libro", new_libro_path %>`**: Crea un enlace que lleva al usuario a la página para crear un nuevo libro. `new_libro_path` es la ruta que corresponde a la acción `new` del `LibrosController`.

En pocas palabras

Este código genera una vista para la gestión de libros en una aplicación Ruby on Rails. Muestra un mensaje de notificación, un formulario de búsqueda, una lista de libros con enlaces para ver detalles y un enlace para crear un nuevo libro. Utiliza ERB para integrar contenido dinámico y permite a los usuarios interactuar con la aplicación de manera efectiva.

A continuación, tenemos las imágenes de la interfaz:

Libros

Buscar libros: Buscar

Autor: Ruben Dario

Nombre: Azul

Genero: poema

Show this libro

Autor: J.K.Rowling

Nombre: Harry Potter y la piedra filosofal

Genero: fantasia

Show this libro

Autor: DISNEY

Nombre: GRAVITY FALLS. DIARIO 3

Genero: fantasia

Show this libro

Libros

Buscar libros: La fugitiva Buscar

Autor: Sergio Ramírez Mercado

Nombre: La fugitiva

Genero: novela

Show this libro

New libro

A continuación, tenemos las imágenes de la interfaz del form:

New libro

3 errors prohibited this libro from being saved:

- Autor can't be blank
- Nombre can't be blank
- Genero can't be blank

Autor

Nombre

Genero

Create Libro

Back to libros

Acá tenemos lo que es el código del form.html.erb:

```erb
<%= form_with(model: libro) do |form| %>
  <% if libro.errors.any? %>
    <div style="color: red">
      <h2><%= pluralize(libro.errors.count, "error") %> prohibited this libro from being saved:</h2>

      <ul>
        <% libro.errors.each do |error| %>
          <li><%= error.full_message %></li>
        <% end %>
      </ul>
    </div>
  <% end %>

  <div>
    <%= form.label :autor, style: "display: block" %>
    <%= form.text_field :autor %>
  </div>

  <div>
    <%= form.label :nombre, style: "display: block" %>
    <%= form.text_field :nombre %>
  </div>

  <div>
    <%= form.label :genero, style: "display: block" %>
    <%= form.text_field :genero %>
  </div>

  <div>
    <%= form.submit %>
  </div>
<% end %>
```

```erb
<%= form_with(model: libro) do |form| %>
```

- **`<%= form_with(model: libro) do |form| %>`**: Este código inicia un formulario utilizando el helper `form_with`, que está vinculado al objeto `libro`. Esto significa que el formulario se utilizará para crear o editar un libro, y Rails manejará automáticamente la URL y el método HTTP apropiados según si `libro` es un nuevo objeto o uno existente.

```erb
<% if libro.errors.any? %>
  <div style="color: red">
    <h2><%= pluralize(libro.errors.count, "error") %> prohibited this
libro from being saved:</h2>

    <ul>
      <% libro.errors.each do |error| %>
        <li><%= error.full_message %></li>
      <% end %>
    </ul>
  </div>
<% end %>
```

- **`if libro.errors.any?`**: Este bloque verifica si hay errores de validación en el objeto `libro`. Si hay errores, se muestra un mensaje.
- **`<div style="color: red">`**: Este div se utiliza para mostrar los errores en color rojo.
- **`<h2><%= pluralize(libro.errors.count, "error") %> prohibited this libro from being saved:</h2>`**: Muestra un encabezado que indica cuántos errores han ocurrido. `pluralize` se utiliza para mostrar "error" en singular o plural según la cantidad de errores.
- **`<ul>` y `<li>`**: Se crea una lista desordenada que muestra cada mensaje de error utilizando `error.full_message`, que proporciona un mensaje descriptivo para cada error.

```erb
<div>
  <%= form.label :autor, style: "display: block" %>
  <%= form.text_field :autor %>
</div>

<div>
  <%= form.label :nombre, style: "display: block" %>
  <%= form.text_field :nombre %>
</div>

<div>
  <%= form.label :genero, style: "display: block" %>
  <%= form.text_field :genero %>
</div>
```

- **Campos del Formulario**: Cada bloque `<div>` contiene un campo de entrada para los atributos del libro (`autor`, `nombre`, `genero`).
  - **`form.label`**: Crea una etiqueta para cada campo, que se muestra en bloque.
  - **`form.text_field`**: Crea un campo de texto para que el usuario ingrese el valor correspondiente a cada atributo del libro.

```erb
<div>
  <%= form.submit %>
</div>
```

- **`<div><%= form.submit %></div>`**: Este div contiene un botón de envío que permite al usuario enviar el formulario. El texto del botón será "Create Libro" o "Update Libro" dependiendo de si se está creando un nuevo libro o editando uno existente.

```erb
<% end %>
```

- **`<% end %>`**: Finaliza el bloque del formulario.

Resumen General
En resumen, este código genera un formulario para crear o editar un libro en una aplicación Ruby on Rails. Muestra mensajes de error si hay problemas de validación, permite al usuario ingresar los atributos del libro (autor, nombre y género) y proporciona un botón para enviar el formulario. Utiliza ERB para integrar contenido dinámico y manejar la lógica de validación de manera efectiva.

Enlace del video en Google drive:

https://drive.google.com/file/d/1RjCzls47QyV1DBRXIVNkpCXO4vtXutgm/view?usp=sharing

enlace del video en youtube:

https://youtu.be/08yjQ2wez3Q