Manual de Uso

“Demo Conoce Nayarit”

Introducción

La aplicación web/pagina web ¨DemoConoceNayarit¨ se encuentra desarrollada sobre el lenguaje de programación Ruby 2.6.0 apoyado con el Framework Ruby on Rails en su versión 5.2.3, Desarrollado como una propuesta ante las peticiones de gobierno del estado. La aplicación se basa en un gestor de contenido de una página web para incentivar el turismo del estado.

Gemas utilizadas

PaperClip

PaperClip es una Gema de Ruby on Rails que permite la carga de Imagen en Rails.

Link a la documentación: https://github.com/thoughtbot/paperclip

NOTA cuando se trabaja sobre un entorno Windows la gema de Papaerclip necesita un gestor de imágenes por lo cual es necesario instalar ImageMagicDisplay en Windows, además de agregar un archivo adicional a la carpeta de “initializers” dentro del proyecto de rails.

Crear un archivo llamado **“paperclip_media_type_spoof_detector_override.rb”** con lo siguiente dentro.

```
require 'paperclip/media_type_spoof_detector'
module Paperclip
  class MediaTypeSpoofDetector
    def spoofed?
      false
    end
  end
end
```

Además de agregar una la definición de las medidas de las imágenes dentro del modelo donde utilizaran.

Ckeditor

Ckeditor es un editor de texto que permite agregar cierta personalización texto además de incrustar imágenes dentro del mismo contenido

https://github.com/galetahub/ckeditor

NOTA: Para que ckeditor guarde imágenes es necesario tener instalada y configurada la gema de paperclip

Administración de Contenido

Para agregar nuevo contenido a la página web es necesario una vez iniciada la aplicación entrara a la **URL:** http://localhost:3000/admin

Esto nos desplegara un pequeño menú de las cosas que a la fecha se pueden administrar desde la aplicación web.

Por defecto al iniciar la aplicación esta nos re direccionara a la vista principal de la página web. https://localhost:3000/

MVC

Para la construcción de una aplicación en Ruby on Rails es necesario de tres archivos, para la construcción de ello es posible hacerlo paso a paso o con una simple instrucción

Como crear una aplicación

El siguiente ejemplo toma la construcción del modelo de eventos

En ese caso el comando necesario para crear una aplicación con posgresql es el siguiente.

```
rails new DemoConoceNayarit –d=postgresql
```

NOTA: Si no se especifica el tipo de conexión de base de datos se utilizará SQLite por defecto.

Una vez hecha la configuración del archivo de la base de datos es necesario sobre la terminal poner los comandos

rake db:create db:migrate

Paso a paso:

Rails nos generara una carpeta con las rutas necesarias para empezar a trabajar. En primera instancia necesitaremos declara nuestro modelo a trabajar (un modelo es la representación de una tabla hablando en términos de base de datos)

```
rails g event name:string short_description:string long_description:string latitude:string longitude:string cost:string state:string begining_hour:string ending_hour:string begining_date:string ending_date:string
```

NOTA:

g es abreviación de “**generate**”

event es el nombre que recibe el modelo

los demás atributos son los campos de la tabla y es necesario especificar su tipo de dato. Si no rails tomara por defecto como string

después de crear un modelo es necesario ejecutar el comando **rake db:migrate**

event.rb

```ruby
class Event < ApplicationRecord
end

```

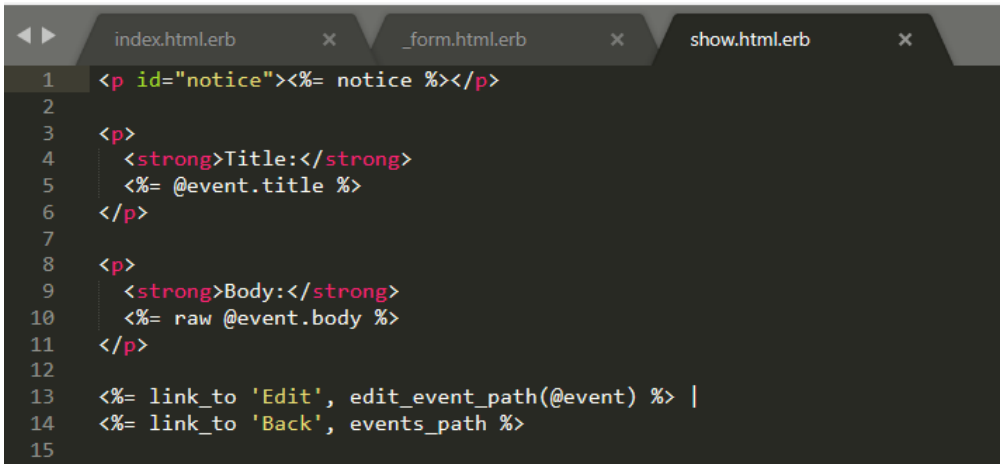
Ahora para la estructura básica de la vista de eventos es necesario crear una carpeta en la Views con el mismo nombre del modelo. La estructura se basa en un

- **Index.html.erb:** el cual es un concentrado de todos los registros

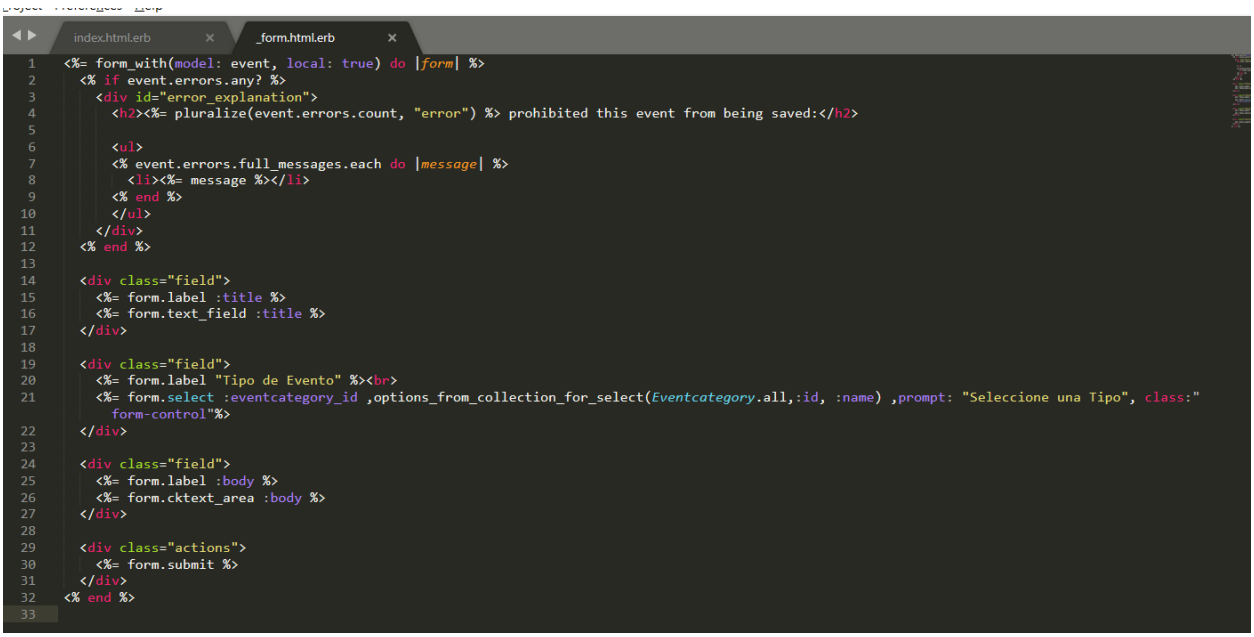
index.html.erb

```erb
<p id="notice"><%= notice %></p>

<h1>Events</h1>

<table>
  <thead>
    <tr>
      <th>Title</th>
      <th>Body</th>
      <th colspan="3"></th>
    </tr>
  </thead>

  <tbody>
    <% @events.each do |event| %>
      <tr>
        <td><%= event.title %></td>
        <td><%= link_to 'Show', event %></td>
        <td><%= link_to 'Edit', edit_event_path(event) %></td>
        <td><%= link_to 'Destroy', event, method: :delete, data: { confirm: 'Are you sure?' } %></td>
      </tr>
    <% end %>
  </tbody>
</table>

<br>

<%= link_to 'New Event', new_event_path %>

```

- **Show.html.erb:** el cuál es la vista individual de un evento registrado



- **Form.html.erb:** el cual es un formulario para el registro de eventos



- **New.html.erb:** render del formulario

```
<h1>New Event</h1>
<%= render 'form', event: @event %>
<%= link_to 'Back', events_path %>
```

- **Edit.html.erb:** render del formulario

```
<h1>Editing Event</h1>
<%= render 'form', event: @event %>
<%= link_to 'Show', @event %> |
<%= link_to 'Back', events_path %>
```

Y por último es necesario de que cada vista tenga una función declarada sobre el archive de controlador.

Para la vista del **index**

```
def index
    @events = Event.all
 end
```

NOTA: @events es una variable que contiene todos los eventos registrados.

New

```
def new
    @event = Event.new
 end
```

NOTA: @events es una variable que junto la palabra reservada permite crear un nuevo evento

```
def create
    @event = Event.new(event_params)

    respond_to do |format|
      if @event.save
        format.html { redirect_to @event, notice: 'Event was successfully created.' }
        format.json { render :show, status: :created, location: @event }
      else
        format.html { render :new }
        format.json { render json: @event.errors, status: :unprocessable_entity }
      end
    end
  end
```

Función que ayuda a la creación de eventos tomando como parámetros el “event_params” para evitar inyecciones SQL

Update

```
def update
    respond_to do |format|
      if @event.update(event_params)
        format.html { redirect_to @event, notice: 'Event was successfully updated.' }
        format.json { render :show, status: :ok, location: @event }
      else
        format.html { render :edit }
        format.json { render json: @event.errors, status: :unprocessable_entity }
      end
    end
  end
```

NOTA: @events es una variable que junto la palabra reservada permite actulizar un evento

Delete

```
def destroy
    @event.destroy
    respond_to do |format|
      format.html { redirect_to events_url, notice: 'Event was successfully destroyed.' }
      format.json { head :no_content }
    end
  end
```

NOTA: @events es una variable que junto la palabra reservada permite eliminar un evento

Declaración de **event_params**

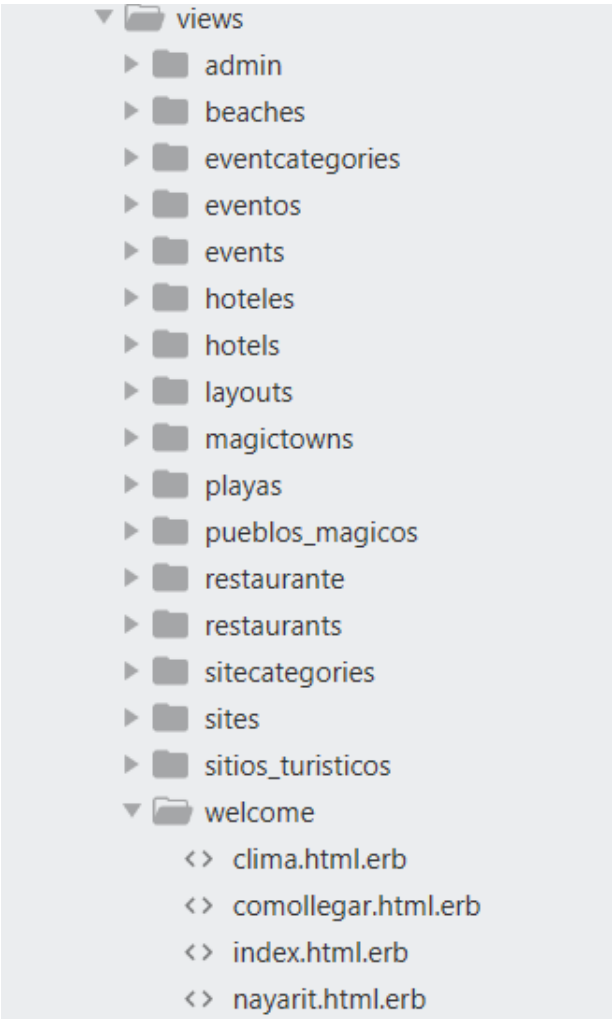
```
private
    def set_event
      @event = Event.find(params[:id])
    end

    def event_params
      params.require(:event).permit(:title, :body)
    end
```

NOTA: por buenas practicas se recomienda crear la función de **params**, esto evitara que ataques y solo permitirá los campos especificados dentro del formulario.

La estructura de construcción de los modelos es similar y es el mismo principio para los demás modelos como lo son :

- **Playa**
- **Hotel**
- **Restaurante**
- **Sitios turísticos**

A diferencia de Conoce Tepic, la esta aplicación alimenta a la una página web por lo cual se crean **“Vistas” y “Controladores”** para su administración que a su vez utiliza el mismo modelo previamente creado.



Las **vistas** que se encuentran en Ingles hacen referencia a la parte de la administración de la página web, mientras que todas las vistas que se encuentran con nombre en español hacen referencia a vistas de la página web.

NOTA: Todas las vistas agregan información desde la parte de administración a excepción de los archivos contenidos en “**welcome”**

- **Clima**
- **Como llegar**
- **Nayarit**

Dicha información se encuentra estática.

Rutas

routes.rb

```ruby
Rails.application.routes.draw do
  resources :sitecategories

  get 'playas', to: "playas#index"
  get 'playas/:id', to: "playas#show", as: "playa"
  resources :beaches

  get 'hoteles', to: "hoteles#index"
  get 'hoteles/:id', to: "hoteles#show",as: "hotel"
  resources :hotels

  get 'restaurante', to: "restaurante#index"
  get 'restaurante/:id', to: "restaurante#show", as: "restaurant"
  resources :restaurants

  get 'sitios_turisticos', to: "sitios_turisticos#index"
  get 'sitios_turisticos/:id', to: "sitios_turisticos#show", as: "sitio_turistico"
  resources :sites

  resources :eventcategories
  get 'pueblos_magicos', to: "pueblos_magicos#index"
  get 'pueblos_magicos/:id', to: "pueblos_magicos#show", as: "pueblo_magico"

  resources :magictowns
  get 'eventos/index'
  mount Ckeditor::Engine => '/ckeditor'

  resources :events
  get 'comollegar', to: "welcome#comollegar"
  get 'clima', to: "welcome#clima"
  get 'nayarit', to: "welcome#nayarit"

  get 'admin', to: "admin#admin"
  root 'welcome#index'


  # For details on the DSL available within this file, see http://guides.rubyonrails.org/routing.html
end

```

resources es una forma de especificar todas las rutas de comprendidas por las vistas de un modelo.

cuando tenemos una vista sin un modelo directamente, es necesario referenciar que acción en este caso **GET** para poder acceder a esta vista, y por buenas practicas referenciamos las rutas con un solo nombre

Esto hace que redirecciones al consentrado de todos lo hoteles
get 'hoteles', to: "hoteles#index"

Esto hace que obtenga como parametron el ID del hotel y lo muestre individualmente
get 'hoteles/:id', to: "hoteles#show",as: "hotel"

Esto hace todas las rutas de la parte admin, es mas fácil que especificar vista por vista
resources :hotels