# Vistas de administración

## Introducción

En esta sección veremos cómo implementar las vistas, controladores y rutas para mantener nuestras **categorías, subcategorías y usuarios**. Al final del capítulo, también estudiaremos cómo crear un sistema de roles y permisos para nuestros **usuarios**.

## Categorías

A continuación, vamos a crear la vista, controlador y ruta necesarios para implementar nuestro mantenimiento de **categorías**.

### Creación del controlador

Todos los controladores para la administración los crearemos en una carpeta llamada **Admin** dentro de la carpeta **app\* para mantener cierto orden y control. De esta manera, separaremos los controladores del** Frontend\*\* de los controladores del panel de administración.

Para crear el controlador, teclee el siguiente comando:

php artisan make:controller Admin\CategoriaController

Una vez creado el controlador, diríjase a él e introduzca el siguiente código:

public function index()  
{  
 $categorias = Categoria::paginate(4);  
  
 return view('admin.categorias.index', compact('categorias'));  
}

El método **paginate** indica a Eloquent que devuelva el resultado en páginas de 4 categorías por página. Volveremos a este tema en la implementación de la vista.

### Creación de la vista

Todas las vistas asociadas con nuestro **panel de administración** estarán contenidas en una carpeta denominada **admin** dentro de la carpeta \*\*resources\*. En el caso de las categorías, incluiremos todas las vistas asociadas en \*\*resources\*.

Empecemos por la vista **index.blade.php**. Esta vista nos mostrará un listado paginado con todas las categorías disponibles.

@extends('adminlte::page')  
  
@section('title', 'Categorías')  
  
@section('content\_header')  
 <h1>Lista de Categorías</h1>  
@stop  
  
@section('content')  
 {{-- Comprobación de sesión --}}  
 @if (session('success'))  
 <div class="alert alert-success alert-dismissible fade show" role="alert">  
 {{ session('success') }}  
 <button type="button" class="close" data-dismiss="alert" aria-label="Cerrar">  
 <span aria-hidden="true">&times;</span>  
 </button>  
 </div>  
 @endif  
  
 <div class="card col-lg-12 col-md-4">  
 <div class="card-header mb-4">  
 <table>  
 <td>  
 <a href="{{ route('admin.categoria.create') }}" class="btn btn-primary btn-sm">  
 Agregar categoría  
 </a>  
 </td>  
 <!-- El botón "Agregar categoría" redirige a la ruta "admin.categoria.create" -->  
 </table>  
 </div>  
 <div class="card-body mt-2">  
 <table class="table">  
 <thead>  
 <tr>  
 <th scope="col">#</th>  
 <th scope="col">Nombre</th>  
 <th scope="col">Descripción</th>  
 <th scope="col">Imagen</th>  
 <  
  
th scope="col" colspan="2">Acción</th>  
 </tr>  
 </thead>  
 <tbody>  
 {{-- Iteración a través de las categorías --}}  
 @foreach ($categorias as $categoria)  
 <tr>  
 <th scope="row">{{ $categoria->id }}</th>  
 <td><b>{{ $categoria->nombre }}</b></td>  
 <td>{{ strip\_tags($categoria->descripcion) }}</td>  
 <td><img src="{{ $categoria->imagen }}" width="125px"></td>  
 {{-- Botón de edición --}}  
 <td width="10px">  
 <a href="{{ route('admin.categoria.edit', ['id' => $categoria->id]) }}" class="btn btn-primary btn-sm">Editar</a>  
 </td>  
 {{-- Botón de borrado --}}  
 <td width="10px">  
 <form action="{{ route('admin.categoria.delete', ['id' => $categoria->id]) }}" method="POST">  
 @csrf  
 @method('DELETE')  
 <button type="submit" class="btn btn-danger btn-sm" onclick="return confirm('Si borra esta categoría, borrará todas las subcategorías y anuncios asociados.\n¿Estás seguro de que deseas eliminar esta categoría?')">Eliminar</button>  
 </form>  
 </td>  
 </tr>  
 @endforeach  
 </tbody>  
 </table>  
 </div>  
 </div>  
 {{ $categorias->links('pagination::bootstrap-4') }}  
@stop

:::info ¿Qué hace?

* @extends('adminlte::page'): Esta línea indica que la vista extiende el layout proporcionado por AdminLTE, lo cual significa que heredará estilos y funcionalidades predefinidas del tema.
* @section('title', 'Categorías'): Aquí se define el título de la página, que en este caso será “Categorías”.
* @section('content\_header'): Esta sección define el contenido que se mostrará en el encabezado de la página. En este caso, se muestra el título “Lista de Categorías”.
* @section('content'): Aquí comienza la sección principal de contenido de la página. A partir de este punto, se empieza a definir el contenido que se mostrará en el cuerpo de la página.
* @if (session('success')): En esta línea se comprueba si hay un mensaje de éxito almacenado en la sesión. Si existe, se muestra un cuadro de alerta con el mensaje.
* Luego se crea una tarjeta (<div class="card">) que contiene una tabla con las categorías. En esta tabla se muestra información como el id, nombre, descripción e imagen de cada categoría.
* Dentro de la iteración @foreach ($categorias as $categoria) se muestran los datos de cada categoría en filas de la tabla. También se incluyen botones de edición y eliminación para cada categoría.
* Al final de la vista, se muestra la paginación utilizando el componente de paginación de Bootstrap 4 ({{ $categorias->links('pagination::bootstrap-4') }}).

En resumen, esta vista muestra una lista de categorías con la posibilidad de agregar, editar y eliminar categorías. Utiliza el diseño y los estilos proporcionados por AdminLTE para lograr una apariencia agradable y coherente. :::

Si después de implementar el código y especificar la ruta desea probar la vista, sustituya las rutas especificadas por el símbolo “#” para evitar errores. Una vez probado, le recomendamos volver a las rutas originales.

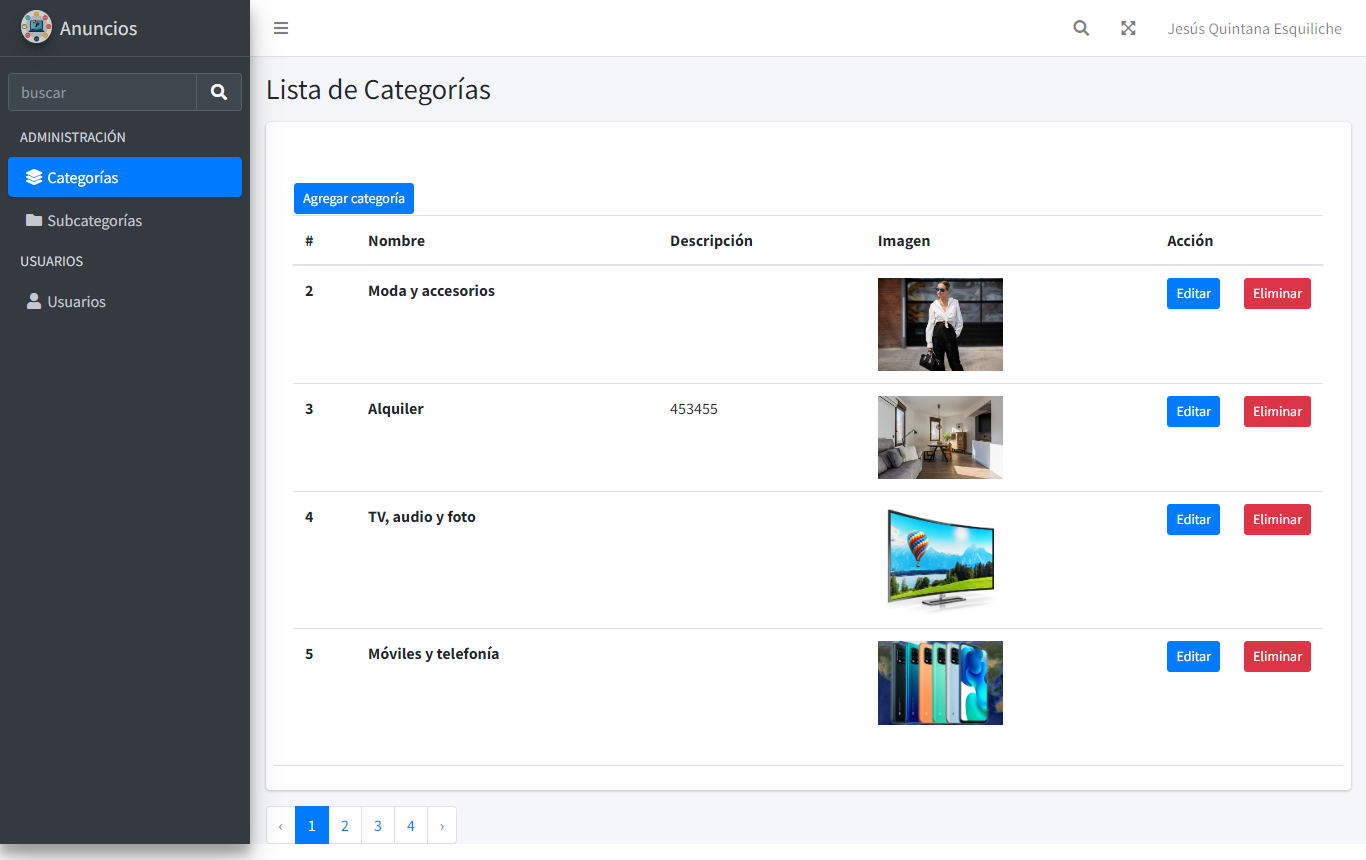
A continuación, vamos a definir la ruta asociada a la vista. Diríjase al archivo de rutas **routes.php** y escriba el siguiente código:

use App\Http\Controllers\Admin\CategoriaController;  
  
Route::get('/categorias', [CategoriaController::class, 'index'])->name('admin.categoria.index');

No olvide incluir la referencia al controlador **CategoriaController** al principio del archivo:

<?php  
  
use Illuminate\Http\Request;  
use Illuminate\Support\Facades\Route;  
use App\Http\Controllers\Admin\HomeController;  
use App\Http\Controllers\Admin\CategoriaController;  
use App\Http\Controllers\Admin\SubCategoriaController;  
use App\Http\Controllers\Admin\UserController;

Ahora si desde la barra de direcciones tecla la siguiente dirección 'http://localhost:8000/admin/categorias', obtendrá la siguiente vista.



Vista categorias

Como esta forma de acceder a la vista no es de lo más cómodo para el usuario, vamos a modificar el **sidebar** de nuestro **panel de administración**. Para ello editaremos el archivo **config.php**, busque la siguiente sección y modifiquela.

['header' => 'ADMINISTRACIÓN'],  
[  
 'text' => 'Categorías',  
 'route' => 'admin.categoria.index',  
 'icon' => 'fas fa-fw fa-layer-group',  
],

### Agregar categorías

En este apartado vamos a ver como habilitar el botón de **Agregar** de nuestra vista index. Para ello deberemos crear un nuevo método en nuestro controlador **CategoriaController**.

public function create()  
 {  
 return view('admin.categorias.create');  
 }

Como puede observar el código del método **create** es bastante simplemente, llamamos a la vista admin.categorias.create. Diríjase la carpeta resources\views\admin\categorias y cree el archivo create.blade.php.

@extends('adminlte::page')  
  
@section('title', 'Agregar Categoría')  
  
@section('content\_header')  
 <h1>Agregar Categoría</h1>  
@stop  
  
@section('content')  
 <div class="container">  
 <div class="card col-lg-12 p-4">  
 {!! Form::open(['route' => 'admin.categoria.store', 'files' => true, 'method' => 'POST']) !!}  
  
 <div class="form-group">  
 {!! Form::label('nombre', 'Nombre:') !!}  
 {!! Form::text('nombre', null, ['class' => 'form-control'.($errors->has('nombre') ? ' is-invalid' : ''), 'required']) !!}  
 @if ($errors->has('nombre'))  
 <div class="invalid-feedback">  
 {{ $errors->first('nombre') }}  
 </div>  
 @endif  
 </div>  
  
 <div class="form-group">  
 {!! Form::label('descripcion', 'Descripción:') !!}  
 {!! Form::textarea('descripcion', null, ['class' => 'form-control'.($errors->has('descripcion') ? ' is-invalid' : ''), 'required']) !!}  
 @if ($errors->has('descripcion'))  
 <div class="invalid-feedback">  
 {{ $errors->first('descripcion') }}  
 </div>  
 @endif  
 </div>  
  
 <div class="form-group">  
 {!! Form::label('imagen', 'Imagen:') !!}  
 {!! Form::file('imagen', ['class' => 'form-control-file'.($errors->has('imagen') ? ' is-invalid' : ''), 'accept' => 'image/\*', 'onchange' => 'previewImage(event)']) !!}  
 @if ($errors->has('imagen'))  
 <div class="invalid-feedback">  
 {{ $errors->first('imagen') }}  
 </div>  
 @endif  
 <img id="preview" src="#" alt="Preview de la imagen" style="display: none; max-width: 200px; margin-top: 10px;">  
 </div>  
  
 {!! Form::submit('Agregar', ['class' => 'btn btn-primary']) !!}  
  
 {!! Form::close() !!}  
 </div>  
 </div>  
  
 <script>  
 function previewImage(event) {  
 var input = event.target;  
 var preview = document.getElementById('preview');  
  
 if (input.files && input.files[0]) {  
 var reader = new FileReader();  
  
 reader.onload = function (e) {  
 preview.src = e.target.result;  
 preview.style.display = 'block';  
 }  
  
 reader.readAsDataURL(input.files[0]);  
 }  
 }  
 </script>  
@stop  
@extends('adminlte::page')  
  
@section('title', 'Agregar Categoría')  
  
@section('content\_header')  
 <h1>Agregar Categoría</h1>  
@stop

:::info ¿Qué hace?

* @extends('adminlte::page'): Esta línea indica que la vista extiende el layout proporcionado por AdminLTE, lo cual significa que heredará estilos y funcionalidades predefinidas del tema.
* @section('title', 'Agregar Categoría'): Aquí se define el título de la página, que en este caso será “Agregar Categoría”.
* @section('content\_header'): Esta sección define el contenido que se mostrará en el encabezado de la página. En este caso, se muestra el título “Agregar Categoría”.
* @section('content'): Aquí comienza la sección principal de contenido de la página. A partir de este punto, se empieza a definir el contenido que se mostrará en el cuerpo de la página.
* Dentro del div con clase “container” se crea una tarjeta (<div class="card">) que contendrá un formulario para agregar una categoría.
* El formulario se crea utilizando el método Form::open, que especifica la ruta de destino del formulario, los atributos files y method, que permiten el envío de archivos y el método HTTP POST respectivamente.
* Dentro del formulario, hay tres campos de entrada: nombre, descripción e imagen. Cada campo se crea utilizando el método Form::text o Form::textarea, y se le asignan atributos como clase y validación de errores. Si hay errores de validación, se muestra un mensaje de error debajo del campo correspondiente.
* El campo de imagen utiliza el método Form::file y tiene un evento onchange que llama a la función previewImage para mostrar una vista previa de la imagen seleccionada. La vista previa se muestra en un elemento img con id “preview”.
* El botón “Agregar” se crea utilizando el método Form::submit y tiene la clase “btn btn-primary”.
* Al final de la vista, se encuentra un script JavaScript que define la función previewImage. Esta función lee el archivo de imagen seleccionado por el usuario y muestra una vista previa de la imagen en el elemento img con id “preview”.

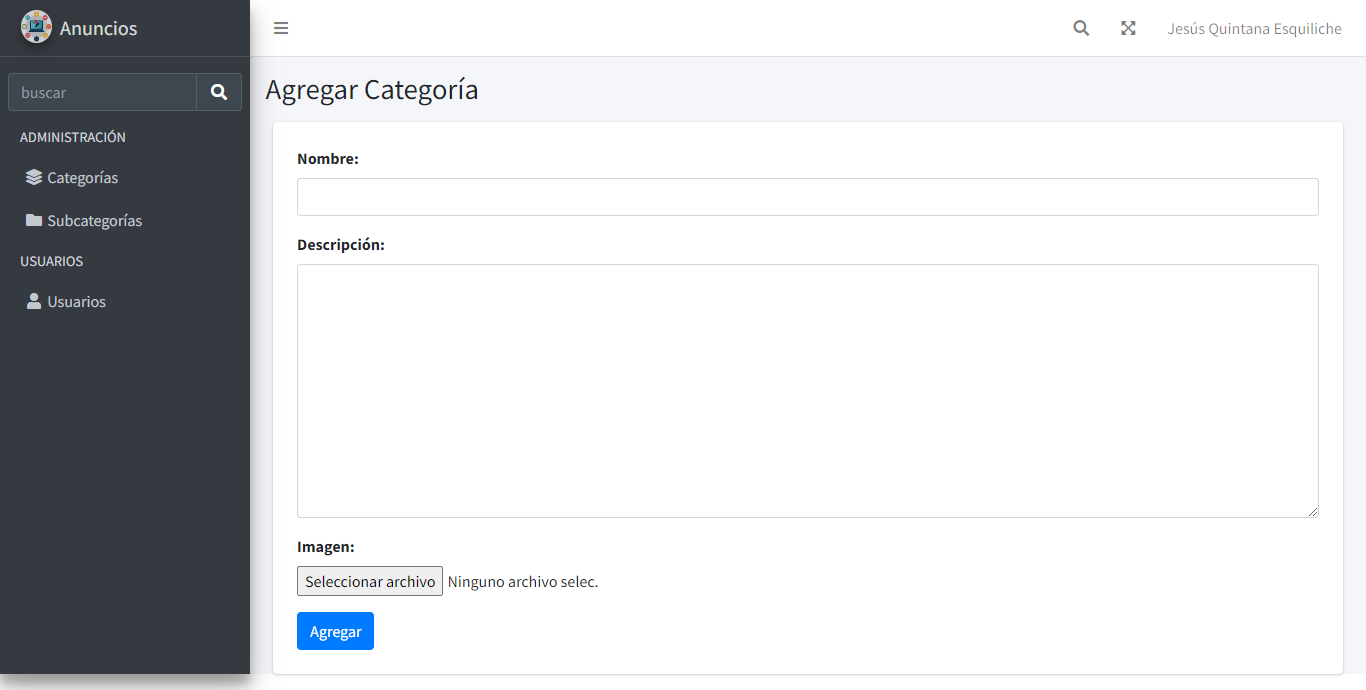
En definitiva, esta vista muestra un formulario para agregar una categoría, que incluye campos para el nombre, descripción e imagen de la categoría. También permite la visualización de una vista previa de la imagen seleccionada antes de enviar el formulario. :::

A continuación, dentro de la vista **index.blade.php** se encuentra el siguiente código:

<td>  
 <a href="{{ route('admin.categoria.create') }}" class="btn btn-primary btn-sm">  
 Agregar categoría  
 </a>  
</td>

Este enlace redirige a la ruta asociada a la vista **admin.categoria.create**. El código crea un enlace dentro de una celda de una tabla. Al hacer clic en el enlace, el usuario será redirigido a la ruta definida en route('admin.categoria.create'), que se ha configurado para estar asociada a la vista admin.categoria.create.

Ahora si se dirige al listado de categorías y hace click sobre el botón agregar se le debería mostrar la siguiente vista.



Agregar categoria

### Editar categoría

A continuación, vamos a estudiar cómo implementar la vista. Empecemos por el controlador. Diríjase al archivo **app.php** e introduzca el siguiente código:

public function edit($id)  
{  
 // Buscar la categoría que se va a editar  
 $categoria = Categoria::find($id);  
 return view('admin.categorias.edit', compact('categoria'));  
}

No es necesario comentar este código, ya que su funcionalidad es evidente. Aprovecharemos para crear otro método en el controlador llamado **update**. Este método se encargará de validar los datos introducidos en la vista y actualizar nuestra base de datos.

public function update(Request $request, $id)  
{  
 // Validar los campos de la petición  
 $request->validate([  
 'nombre' => 'required',  
 'descripcion' => 'required',  
 'imagen' => 'image|mimes:jpeg,png,jpg,gif|max:2048',  
 ]);  
  
 // Buscar la categoría que se va a actualizar  
 $categoria = Categoria::find($id);  
 $categoria->nombre = $request->nombre;  
 $categoria->descripcion = $request->descripcion;  
  
 // Verificar si se ha seleccionado un archivo de imagen para subir  
 if ($request->hasFile('imagen')) {  
 // Recuperar el archivo de imagen  
 $imagen = $request->file('imagen');  
 // Generar un nombre único para el archivo  
 $imagenNombre = time() . '\_' . $imagen->getClientOriginalName();  
 // Mover el archivo a la carpeta public/images  
 $imagen->move(public\_path('images'), $imagenNombre);  
 // Guardar la ruta de la imagen en el campo 'imagen' de la categoría  
 $categoria->imagen = '/images/' . $imagenNombre;  
 }  
  
 // Guardar los cambios en la base de datos  
 $categoria->save();  
  
 // Redirigir a la lista de categorías con un mensaje de éxito  
 return redirect()->route('admin.categoria.index')->with('success', 'Categoría actualizada correctamente');  
}

:::info ¿Qué hace?

1. El método update recibe dos parámetros: $request (que parece ser una instancia de la clase Request) y $id (el identificador de la categoría que se va a actualizar).
2. Se realiza una validación de los campos enviados en la petición utilizando el método validate() del objeto $request. En este caso, se verifica que los campos nombre y descripcion sean requeridos, y que el campo imagen sea una imagen válida con un tamaño máximo de 2048 kilobytes.
3. A continuación, se busca la categoría que se va a actualizar utilizando el modelo Categoria y el identificador $id. Esto permite obtener la categoría correspondiente en la base de datos.
4. Se asignan los valores recibidos en los campos nombre y descripcion al objeto $categoria, actualizando así los datos de la categoría.
5. Si se ha seleccionado un archivo de imagen en la petición utilizando el método hasFile() del objeto $request, se procede a procesarlo.
6. Se recupera el archivo de imagen utilizando el método file() del objeto $request y se genera

un nombre único para el archivo concatenando la marca de tiempo actual con el nombre original del archivo.

1. Luego, se mueve el archivo de imagen a una carpeta específica dentro de la carpeta pública (public/images) utilizando el método move() del objeto $imagen. La función public\_path() se utiliza para obtener la ruta completa de la carpeta pública.
2. La ruta del archivo de imagen actualizado se guarda en el campo imagen del objeto $categoria.
3. Se guardan los cambios en la base de datos utilizando el método save() del objeto $categoria.
4. Finalmente, se redirige a la lista de categorías utilizando la ruta con nombre 'admin.categoria.index' y se incluye un mensaje de éxito en la sesión ('success') que se mostrará en la vista de destino.

En resumen, este código actualiza una categoría en la base de datos, permitiendo cambiar su nombre, descripción y también subir una imagen asociada a ella. Después de realizar la actualización, se redirige a la lista de categorías con un mensaje de éxito. :::

#### Creación de la vista

@extends('adminlte::page')  
  
@section('title', 'Agregar Categoría')  
  
@section('content\_header')  
 <h1>Agregar Categoría</h1>  
@stop  
  
@section('content')  
 <div class="container">  
 <div class="card col-lg-12 p-4">  
 {!! Form::model($categoria, [  
 'route' => ['admin.categoria.update', $categoria->id],  
 'files' => true,  
 'method' => 'PUT',  
 ]) !!}  
  
 <div class="form-group">  
 <!-- Etiqueta del campo "nombre" -->  
 {!! Form::label('nombre', 'Nombre:') !!}  
 <!-- Campo de entrada de texto para el nombre -->  
 {!! Form::text('nombre', $categoria->nombre, [  
 'class' => 'form-control' . ($errors->has('nombre') ? ' is-invalid' : ''),  
 'required',  
 ]) !!}  
 @if ($errors->has('nombre'))  
 <div class="invalid-feedback">  
 <!-- Mensaje de error si el campo "nombre" tiene errores de validación -->  
 {{ $errors->first('nombre') }}  
 </div>  
 @endif  
 </div>  
  
 <div class="form-group">  
 <!-- Etiqueta del campo "descripcion" -->  
 {!! Form::label('descripcion', 'Descripción:') !!}  
 <!-- Campo de entrada de texto (área de texto) para la descripción -->  
 {!! Form::textarea('descripcion', $categoria->descripcion, [  
 'class' => 'form-control' . ($errors->has('descripcion') ? ' is-invalid' : ''),  
 'required',  
 ]) !!}  
 @if ($errors->has('descripcion'))  
 <div class="invalid-feedback">  
 <!-- Mensaje de error si el campo "descripcion" tiene errores de validación -->  
 {{ $errors->first('descripcion') }}  
 </div>  
 @endif  
 </div>  
  
 <div class="form-group">  
 <table>  
 <td>  
 <!-- Etiqueta del campo "imagen" -->  
 {!! Form::label('imagen', 'Imagen:') !!}  
 <!-- Campo de selección de archivo de imagen -->  
 {!! Form::file('imagen', [  
 'class' => 'form-control-file' . ($errors->has('imagen') ? ' is-invalid' : ''),  
 'accept' => 'image/\*',  
 'onchange' => 'previewImage(event)',  
 ]) !!}  
 @if ($errors->has('imagen'))  
 <div class="invalid-feedback">  
 <!-- Mensaje de error si el campo "imagen" tiene errores de validación -->  
 {{ $errors->first('imagen') }}  
 </div>  
 @endif  
 </td>  
 <td>  
 <!-- Elemento de imagen para previsualizar la imagen seleccionada -->  
 <img id="preview" src="{{ $categoria->imagen }}" alt="Preview de la imagen"  
 style="max-width: 200px; margin-top: 10px; padding-left: 20px;">  
 </td>  
 </table>  
 </div>  
  
 <!-- Botón de envío del formulario -->  
 {!! Form::submit('Actualizar', ['class' => 'btn btn-primary']) !!}  
  
 {!! Form::close() !!}  
 </div>  
 </div>  
  
 <script>  
 // Función para previsualizar la imagen seleccionada  
 function previewImage(event) {  
 var input = event.target;  
 var preview = document.getElementById('preview');  
  
 if (input.files && input.files[0]) {  
 var reader = new FileReader();  
  
 reader.onload = function(e) {  
 // Actualizar la fuente de la imagen para mostrar la previsualización  
 preview.src = e.target.result;  
 preview.style.display = 'block';  
 }  
  
 // Leer el archivo de imagen como una URL de datos  
 reader.readAsDataURL(input.files[0]);  
 }  
 }  
 </script>  
@stop

Este código pertenece a la vista **resources.blade.php**. A continuación, comentaré cada sección del código para una mejor comprensión:

@extends('adminlte::page')

Esta línea indica que esta vista extiende la plantilla base proporcionada por AdminLTE.

@section('title', 'Agregar Categoría')

Esta sección define el título de la página.

@section('content\_header')  
 <h1>Agregar Categoría</h1>  
@stop

Esta sección define el encabezado de contenido de la página.

@section('content')  
 <!-- Contenido de la página -->  
@stop

Esta sección define el contenido principal de la página.

<div class="container">  
 <div class="card col-lg-12 p-4">

Aquí se crea un contenedor y una tarjeta para agrupar el formulario.

{!! Form::model($categoria, [  
 'route' => ['admin.categoria.update', $categoria->id],  
 'files' => true,  
 'method' => 'PUT',  
]) !!}

Se inicia un formulario utilizando la función Form::model de Laravel Collective. El modelo $categoria se asocia con el formulario, y se especifica la ruta de actualización, permitiendo enviar archivos ('files' => true) y utilizando el método PUT.

<div class="form-group">  
 {!! Form::label('nombre', 'Nombre:') !!}  
 {!! Form::text('nombre', $categoria->nombre, [  
 'class' => 'form-control' . ($errors->has('nombre') ? ' is-invalid' : ''),  
 'required',  
 ]) !!}  
 @if ($errors->has('nombre'))  
 <div class="invalid-feedback">  
 {{ $errors->first('nombre') }}  
 </div>  
 @endif  
</div>

Este código representa un campo de formulario para el nombre de la categoría. Se utiliza la función Form::label para generar una etiqueta, Form::text para generar un campo de entrada de texto, y se utiliza la variable $errors para mostrar mensajes de error si existen.

Se sigue una estructura similar para el campo de descripción y el campo de imagen.

{!! Form::submit('Actualizar', ['class' => 'btn btn-primary']) !!}

Este código representa el botón de envío del formulario con el texto “Actualizar” y la clase de estilo CSS “btn btn-primary”.

{!! Form::close() !!}

Aquí se cierra el formulario.

<script>  
 function previewImage(event) {  
 var input = event.target;  
 var preview = document.getElementById('preview');  
  
 if (input.files && input.files[0]) {  
 var reader = new FileReader();  
  
 reader.onload = function(e) {  
 preview.src = e.target.result;  
 preview.style.display = 'block';  
 }  
  
 reader.readAsDataURL(input.files[0]);  
 }  
 }  
</script>

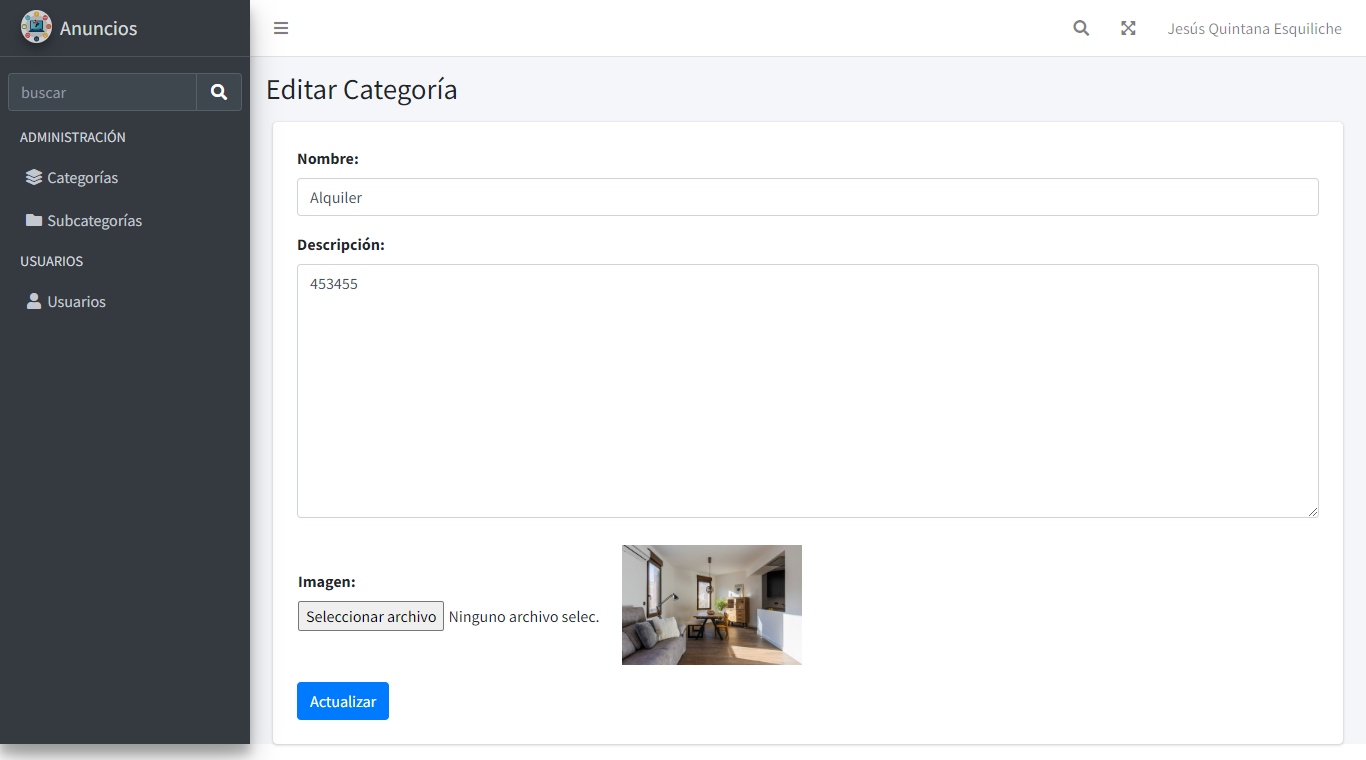
Este es un script JavaScript que maneja la vista previa de la imagen seleccionada. Cuando se selecciona un archivo de imagen, se lee el archivo utilizando la clase FileReader y se asigna la URL de la imagen al elemento <img> con el id “preview”.

En resumen, este código representa una vista de formulario para editar una categoría. El formulario muestra los campos de nombre, descripción e imagen de la categoría. También incluye la funcionalidad para mostrar una vista previa de la imagen

Por último deberíamos dirigirnos al archivo de rutas **routes.php** y comprobar si la siguiente ruta existe, de no existir ahora es el momento justo de crearla.

Route::put("/categorias/update/{id}",[CategoriaController::class,'update'])->name('admin.categoria.update');

Nuestra vista debería quedar tal como se muestra en la imagen:



Editar imagen

### Borrado de categoría

Esta es la parte más sencilla de implementar en nuestra vista **resources.blade.php**. Empezaremos creando el método adecuado del controlador:

public function destroy($id)  
{  
 $categoria = Categoria::find($id);  
  
 if (!$categoria) {  
 return redirect()->route('admin.categoria.index')->with('error', 'Categoría no encontrada');  
 }  
  
 // Eliminar la imagen asociada si existe  
 /\*  
 if ($categoria->imagen) {  
 // Obtener la ruta completa de la imagen  
 $imagenPath = public\_path($categoria->imagen);  
  
 // Verificar si la imagen existe en el servidor  
 if (file\_exists($imagenPath)) {  
 // Eliminar la imagen  
 unlink($imagenPath);  
 }  
 }\*/  
  
 $categoria->delete();  
  
 return redirect()->route('admin.categoria.index')->with('success', 'Categoría eliminada correctamente');  
}

El código proporcionado realiza las siguientes acciones:

1. Se busca una categoría en la base de datos utilizando el identificador $id proporcionado.
2. Si no se encuentra la categoría, se redirige a la lista de categorías (admin.categoria.index) y se muestra un mensaje de error en la sesión indicando que la categoría no fue encontrada.
3. (Comentario): Se comprueba si la categoría tiene una imagen asociada y, de ser así, se obtiene la ruta completa de la imagen y se verifica si existe en el servidor. Si la imagen existe, se elimina utilizando la función unlink().
4. Se elimina la categoría de la base de datos utilizando el método delete().
5. Se redirige a la lista de categorías (admin.categoria.index) y se muestra un mensaje de éxito en la sesión indicando que la categoría fue eliminada correctamente.

Es importante destacar que el bloque de código comentado (desde /\* hasta \*/) indica una sección que ha sido deshabilitada y no se ejecutará. En este caso, se trata de la eliminación de la imagen asociada a la categoría. Si deseas utilizar esta funcionalidad, deberás eliminar los comentarios (/\* y \*/) para activar ese bloque de código.

A continuación vamos a definir la ruta en nuestro fichero **routes.php**.

Route::delete("/categorias/delete/{id}",[CategoriaController::class,'destroy'])->name('admin.categoria.delete');

Por último editaremos la vista **resources.blade.php** y donde se define el código de borrado deberemos introducir el siguiente código si este no esta definido, este el último tag <td> de la tabla.

{{-- Botón de borrado --}}  
<td width="10px">  
<form action="{{ route('admin.categoria.delete', ['id' => $categoria->id]) }}"" method="POST">  
 @csrf  
 @method('DELETE')  
 <button type="submit" class="btn btn-danger btn-sm"  
 onclick="return confirm('Si borrar esta cagegoria borra todas las subcegorias y anuncios asociados,\n¿Estás seguro de que deseas eliminar esta categoría?')">  
 Eliminar  
 </button>  
</form>  
  
</td>

El código proporcionado muestra un botón de borrado dentro de una celda de una tabla. A continuación, se explica su funcionamiento:

* Se crea un formulario con el atributo action que apunta a la ruta admin.categoria.delete y se le pasa el parámetro id con el valor de $categoria->id. Esto indica que al enviar el formulario, se realizará una solicitud HTTP DELETE a la ruta especificada, enviando el ID de la categoría a ser eliminada.
* Se utiliza la directiva @csrf para agregar un campo de token CSRF al formulario. Este token ayuda a proteger contra ataques de falsificación de solicitudes entre sitios (CSRF).
* Se utiliza la directiva @method('DELETE') para indicar que la solicitud HTTP debe ser de tipo DELETE.
* Se muestra un botón con la clase CSS btn btn-danger btn-sm, que representa un botón de color rojo y tamaño pequeño.
* Se agrega un evento onclick al botón, que muestra un cuadro de confirmación con un mensaje antes de enviar la solicitud de eliminación. El mensaje de confirmación incluye una advertencia sobre la eliminación de la categoría y cómo esto también eliminará todas las subcategorías y anuncios asociados.
* Si el usuario confirma la acción, el formulario se envía y se realiza la solicitud de eliminación.

En resumen, este código muestra un botón de eliminación en forma de formulario dentro de una celda de una tabla. Al hacer clic en el botón y confirmar la acción, se envía una solicitud de eliminación de la categoría correspondiente a través de una solicitud HTTP DELETE a la ruta especificada.

## Subcategorías

En esta sección estudiaremos como crear la vista, el controlador para el mantenimiento de **subcategorías**.

### Creación del controlador

Teclee desde la terminal dentro de la carpeta del proyecto:

artisan make:controller Admin\SubCategoriaController

Una vez creado diríjase al fichero **app.php**. Edite el contenido del fichero he introduzca el siguiente código:

<?php  
  
namespace App\Http\Controllers\Admin;  
  
use App\Http\Controllers\Controller;  
use App\Models\SubCategoria;  
use App\Models\Categoria;  
use Illuminate\Http\Request;  
  
class SubCategoriaController extends Controller  
{  
 /\*\*  
 \* Display a listing of the resource.  
 \*  
 \* @param \Illuminate\Http\Request $request  
 \* @return \Illuminate\Http\Response  
 \*/  
 public function index(Request $request)  
 {  
 $categorias = Categoria::all();  
  
 $subcategorias = SubCategoria::query();  
  
 // Filtrar subcategorías por categoría si se proporciona el parámetro 'categoria\_id'  
 if ($request->has('categoria\_id')) {  
 $subcategorias->where('categoria\_id', (int)$request->categoria\_id);  
 }  
  
 $subcategorias = $subcategorias->paginate(4);  
  
 return view('admin.subcategorias.index', compact('subcategorias', 'categorias'));  
 }  
  
 /\*\*  
 \* Store a newly created resource in storage.  
 \*  
 \* @param \Illuminate\Http\Request $request  
 \* @return \Illuminate\Http\Response  
 \*/  
 public function store(Request $request)  
 {  
 $rules = [  
 'nombre' => 'required|unique:subcategorias|string|max:150',  
 'descripcion' => 'required|string',  
 'categoria\_id' => 'required|exists:categorias,id'  
 ];  
  
 $request->validate($rules);  
  
 $data = $request->only([  
 'nombre',  
 'descripcion',  
 'categoria\_id'  
 ]);  
  
 Subcategoria::create($data);  
  
 return redirect()->route('admin.subcategorias.index')->with('success', 'Subcategoría agregada correctamente');  
 }  
  
 /\*\*  
 \* Show the form for editing the specified resource.  
 \*  
 \* @param int $id  
 \* @return \Illuminate\Http\Response  
 \*/  
 public function edit($id)  
 {  
 $subcategoria = SubCategoria::find($id);  
  
 return view('admin.subcategorias.edit', compact('subcategoria'));  
 }  
  
 /\*\*  
 \* Update the specified resource in storage.  
 \*  
 \* @param \Illuminate\Http\Request $request  
 \* @param int $id  
 \* @return \Illuminate\Http\Response  
 \*/  
 public function update(Request $request, $id)  
 {  
 $request->validate([  
 'nombre' => 'required',  
 'descripcion' => 'required'  
 ]);  
  
 $subcategoria = Subcategoria::find($id);  
 $subcategoria->nombre = $request->nombre;  
 $subcategoria->descripcion = $request->descripcion;  
  
 $subcategoria->save();  
  
 return redirect()->route('admin.subcategorias.index')->with('success', 'Subcategoría actualizada correctamente');  
 }  
  
 /\*\*  
 \* Show the form for creating a new resource.  
 \*  
 \* @return \Illuminate\Http\Response  
 \*/  
 public function create()  
 {  
 $categorias = Categoria::all();  
  
 return view('admin.subcategorias.create', compact('categorias'));  
 }  
  
 /\*\*  
 \* Remove the specified resource from storage.  
 \*  
 \* @param int $id  
 \* @return \Illuminate\Http\Response  
 \*/  
 public function destroy($id)  
 {  
 $subcategoria = SubCategoria::find($id);  
  
 $subcategoria->delete();  
  
 return redirect()->route('admin.subcategorias.index')->with('success', 'Subcategoría eliminada correctamente');  
 }  
}

#### ¿Qué hace?

El controlador SubCategoriaController es responsable de manejar las acciones relacionadas con las subcategorías en la parte administrativa de la aplicación. Aquí hay una descripción de cada método en el controlador:

* index: Este método muestra la lista de subcategorías. Obtiene todas las categorías y subcategorías disponibles y las pasa a la vista ‘admin.subcategorias.index’. También permite filtrar las subcategorías por categoría si se proporciona el parámetro ‘categoria\_id’.
* store: Este método almacena una nueva subcategoría en la base de datos. Primero, valida los datos proporcionados por el formulario utilizando reglas de validación. Luego, crea un nuevo objeto SubCategoria con los datos validados y lo guarda en la base de datos. Finalmente, redirige a la lista de subcategorías con un mensaje de éxito.
* edit: Este método muestra el formulario de edición para una subcategoría específica. Recibe el ID de la subcategoría como parámetro y busca el objeto SubCategoria correspondiente. Luego, pasa el objeto SubCategoria a la vista ‘admin.subcategorias.edit’ para que pueda ser editado.
* update: Este método actualiza una subcategoría existente en la base de datos. Valida los datos proporcionados por el formulario y busca el objeto SubCategoria correspondiente según el ID proporcionado. Luego, actualiza los campos nombre y descripcion con los datos proporcionados y guarda los cambios en la base de datos. Finalmente, redirige a la lista de subcategorías con un mensaje de éxito.
* create: Este método muestra el formulario para crear una nueva subcategoría. Obtiene todas las categorías disponibles y las pasa a la vista ‘admin.subcategorias.create’ para su selección en el formulario de creación.
* destroy: Este método elimina una subcategoría de la base de datos. Busca el objeto SubCategoria según el ID proporcionado y lo elimina de la base de datos. Luego, redirige a la lista de subcategorías con un mensaje de éxito.

Cada método realiza acciones específicas relacionadas con la manipulación de subcategorías, como mostrar, crear, editar y eliminar. Estas acciones se corresponden con las operaciones básicas de CRUD (Crear, Leer, Actualizar, Eliminar) para las subcategorías en la aplicación administrativa.

### Creación de la vista index

Para la creación de nuestras vistas **resources\* y una vez creada la carpeta dentro de esta cree el fichero** index.blade.php\*\* e introduzca el siguiente código:

@extends('adminlte::page')  
  
@section('title', 'Lista de subcategorías')  
  
@section('content\_header')  
 <h1>Lista de subcategorias</h1>  
@stop  
  
@section('content')  
 <!-- Mensaje de éxito si existe -->  
 @if(session('success'))  
 <div class="alert alert-success alert-dismissible fade show" role="alert">  
 {{ session('success') }}  
 <button type="button" class="close" data-dismiss="alert" aria-label="Cerrar">  
 <span aria-hidden="true">&times;</span>  
 </button>  
 </div>  
 @endif  
 <div class="card">  
 <div class="card-header">  
 <!-- Formulario de filtrado -->  
 {!! Form::open(['route' => 'admin.subcategorias.index', 'method' => 'GET']) !!}  
 {!! Form::select('categoria\_id', $categorias->pluck('nombre', 'id'), request('categoria\_id'), [  
 'class' => 'form-control col-md-4',  
 'placeholder' => 'Escoge categoría',  
 ]) !!}  
 <br />  
 <table>  
 <tr>  
 <td>  
 {!! Form::submit('Filtrar', ['class' => 'btn btn-primary']) !!}  
 {!! Form::close() !!}  
 </td>  
 <td>  
 <!-- Enlace para agregar una nueva subcategoría -->  
 <a href="{{ route('admin.subcategorias.create') }}" class="btn btn-primary ml-2">Agregar</a>  
 </td>  
   
 </tr>  
 </table>  
 </div>  
  
 <div class="card-body">  
 <table id="subcategorias-table" class="table table-bordered table-hover">  
 <thead>  
 <tr>  
 <th>#</th>  
 <th>Nombre</th>  
 <th>Descripción</th>  
 <th colspan="2">Acciones</th>  
 </tr>  
 </thead>  
 <tbody>  
 <!-- Iteración sobre las subcategorías -->  
 @foreach ($subcategorias as $subcategoria)  
 <tr>  
 <td>{{ $subcategoria->id }}</td>  
 <td>{{ $subcategoria->nombre }}</td>  
 <td>{{ $subcategoria->descripcion }}</td>  
   
 <td widt="10px">  
 <!-- Enlace para editar la subcategoría -->  
 <a href="{{ route('admin.subcategorias.edit', ['id' => $subcategoria->id]) }}" class="btn btn-primary btn-sm">Editar</a>  
 </td>  
 <td widt="10px">  
 <!-- Formulario para eliminar la subcategoría -->  
 <form action="{{ route('admin.subcategorias.delete', ['id' => $subcategoria->id]) }}" method="POST">  
 @csrf  
 @method('DELETE')  
 <!-- Botón de eliminación con confirmación -->  
 <button type="submit" class="btn btn-danger btn-sm"  
 onclick="return confirm('Si borrar esta subcategoria borra todas las subcegorias y anuncios asociados,\n¿Estás seguro de que deseas eliminar esta categoría?')">  
 Eliminar  
 </button>  
 </form>  
  
 </td>  
   
 </tr>  
 @endforeach  
 </tbody>  
 </table>  
 </div>  
 {{ $subcategorias->links('pagination::bootstrap-4') }}  
 </div>  
@endsection

¿Qué hace?

Esta vista es la plantilla que se utiliza para mostrar la lista de subcategorías en la interfaz de administración. Aquí hay una descripción de lo que hace cada sección de la vista:

* @extends('adminlte::page'): Esta línea indica que la vista extiende la plantilla proporcionada por el paquete AdminLTE. Proporciona una estructura básica para la página.
* @section('title', 'Lista de subcategorías'): En esta sección se establece el título de la página como “Lista de subcategorías”. Esto se mostrará en la pestaña del navegador.
* @section('content\_header'): Aquí se define el encabezado de contenido de la página. Muestra el título “Lista de subcategorías” como encabezado principal de la página.
* @section('content'): Esta sección contiene el contenido principal de la página.
* @if(session('success')): Este bloque de código verifica si existe una sesión flash llamada ‘success’. Si existe, muestra un mensaje de éxito en forma de una alerta verde. El contenido del mensaje se obtiene de la sesión flash.
* <div class="card">: Este div envuelve el contenido de la tarjeta que contiene el formulario de filtrado y la tabla de subcategorías.
* <div class="card-header">: Esta sección contiene el encabezado de la tarjeta. Aquí se encuentra el formulario de filtrado de subcategorías, que permite filtrar las subcategorías por categoría. El formulario utiliza el método GET y se envía a la ruta ‘admin.subcategorias.index’. También muestra un botón para agregar una nueva subcategoría que enlaza a la ruta ‘admin.subcategorias.create’.
* <table id="subcategorias-table" class="table table-bordered table-hover">: Esta tabla muestra la lista de subcategorías. Utiliza la clase de estilo de Bootstrap ‘table’ para dar formato a la tabla. Cada fila muestra información de una subcategoría, incluyendo su ID, nombre, descripción y acciones disponibles como editar y eliminar.
* @foreach ($subcategorias as $subcategoria): Este bucle itera sobre cada subcategoría en la variable $subcategorias y muestra la información correspondiente en cada fila de la tabla.
* <td widt="10px">: Estas celdas de la tabla contienen los botones de “Editar” y “Eliminar” para cada subcategoría. El botón de “Editar” enlaza a la ruta ‘admin.subcategorias.edit’ con el ID de la subcategoría correspondiente. El botón de “Eliminar” muestra un formulario de eliminación que se envía a la ruta ‘admin.subcategorias.delete’ con el ID de la subcategoría correspondiente. Antes de eliminar, muestra una confirmación utilizando el método JavaScript confirm().
* {{ $subcategorias->links('pagination::bootstrap-4') }}: Esta línea muestra la paginación de la lista de subcategorías utilizando el método links() del objeto $subcategorias. Utiliza la plantilla de paginación proporcionada por Bootstrap 4.

En resumen, esta vista muestra una lista paginada de subcategorías en una tabla, permite filtrar las subcategorías por categoría, proporciona botones para editar y eliminar cada subcategoría, y muestra un mensaje de éxito si existe.

### Listado de subcategorias

Dentro de la ruta del fichero **routes.php** cree la siguiente ruta.

Route::get("/subcategorias",[SubCategoriaController::class,'index'])->name('admin.subcategorias.index');

Recuerde que la vista todavía no es totalmente operativa pues nos quedan vistas por implementar.

### Agregar categoría

A continuación cree el siguiente fichero **resources.blade.php** edite el fichero y copie y pegue el siguiente código:

@extends('adminlte::page')  
  
@section('title', 'Tablero')  
  
@section('content\_header')  
 <h1>Agregar categoria</h1>  
@stop  
  
@section('content')  
<div class="card">  
 <div class="card-body">  
 {!! Form::open(['route' => 'admin.subcategorias.store']) !!}  
   
 <div class="form-gropup">  
 {!! Form::label('categoria', 'Categoría') !!}  
 {!! Form::select('categoria\_id', $categorias->pluck('nombre', 'id'), null, ['class' => 'form-control']) !!}  
 </div>  
 <div class="form-group">  
 {!! Form::label('nombre', 'Nombre') !!}  
 {!! Form::text('nombre', null, [  
 'class' => 'form-control',  
 'placeholder' => 'Ingrese el nombre de la categoria'  
 ]) !!}  
   
 @error('nombre')  
 <span class="text-danger">{{ $message }}</span>  
 @enderror  
   
 </div>  
   
 <div class="form-group">  
 {!! Form::label('descripcion', 'Descripcion') !!}  
 {!! Form::textarea('descripcion', null, [  
 'class' => 'form-control',  
 'placeholder' => 'Ingrese la descripción de la categoria'  
 ]) !!}  
   
 @error('descripcion')  
 <span class="text-danger">{{ $message }}</span>  
 @enderror  
   
 </div>  
   
 {!! Form::submit('Crear categoria', [  
 'class'=>'btn btn-primary'  
 ]) !!}  
   
 {!! Form::close() !!}  
 </div>  
</div>  
@stop  
   
  
@section('css')  
 <link rel="stylesheet" href="/js/ckeditor/ckeditor.css">  
@stop  
  
@section('js')  
 <script src="https://cdn.ckeditor.com/ckeditor5/30.0.0/classic/ckeditor.js"></script>  
 <script>  
 ClassicEditor  
 .create(document.querySelector('#descripcion'))  
 .catch(error => {  
 console.error(error);  
 });  
 </script>  
 <script>  
 $(document).ready(function() {  
 CKEDITOR.replace('descripcion');  
 });  
 </script>  
@stop

### ¿Qué hace?

A continuación, se comentan las secciones principales del código:

@extends('adminlte::page')  
  
@section('title', 'Tablero')

Esta línea establece que la vista extiende la plantilla “adminlte::page” y establece el título de la página como “Tablero”.

@section('content\_header')  
 <h1>Agregar categoria</h1>  
@stop

Esta sección define el encabezado del contenido de la página, que muestra el título “Agregar categoría”.

@section('content')  
<div class="card">  
 <div class="card-body">  
 {!! Form::open(['route' => 'admin.subcategorias.store']) !!}  
 <!-- Contenido del formulario -->  
 {!! Form::close() !!}  
 </div>  
</div>  
@stop

Esta sección define el contenido principal de la página, que incluye un formulario para agregar una categoría. El formulario utiliza la función Form::open() para abrir el formulario y especifica la ruta admin.subcategorias.store como destino para enviar los datos del formulario. El contenido específico del formulario se encuentra dentro de las etiquetas {!! Form::open() !!} y {!! Form::close() !!}.

@section('css')  
 <link rel="stylesheet" href="/js/ckeditor/ckeditor.css">  
@stop

Esta sección define el CSS adicional que se debe cargar en la página. En este caso, se carga un archivo CSS de CKEditor para estilos específicos.

@section('js')  
 <script src="https://cdn.ckeditor.com/ckeditor5/30.0.0/classic/ckeditor.js"></script>  
 <script>  
 ClassicEditor  
 .create(document.querySelector('#descripcion'))  
 .catch(error => {  
 console.error(error);  
 });  
 </script>  
 <script>  
 $(document).ready(function() {  
 CKEDITOR.replace('descripcion');  
 });  
 </script>  
@stop

Esta sección define el JavaScript adicional que se debe cargar en la página. Primero, se carga el script de CKEditor desde un CDN. Luego, se inicializa un editor CKEditor en el campo de texto con el ID “descripcion” utilizando la función ClassicEditor.create(). También se utiliza jQuery para reemplazar el campo de texto con el ID “descripcion” con un editor CKEditor adicional.

En general, este código define una vista de Blade que muestra un formulario para agregar una categoría y utiliza CKEditor para mejorar la experiencia de edición del campo de descripción.

### Editar subcategoría

A continuación vamos a explicar como editar subcategorías. Cree el siguiente fichero **resources.blade.php**. Edite y copie el siguiente código:

@extends('adminlte::page')  
  
@section('title', 'Tablero')  
  
@section('content\_header')  
 <h1>Editar Subcategoria</h1>  
@stop  
  
@section('content')  
<div class="card">  
 <div class="card-body">  
 {{-- Genera un formulario de actualización de bloque utilizando el modelo $subcategoria --}}  
 {!! Form::model($subcategoria,  
 ['route' => ['admin.subcategorias.update',$subcategoria->id],  
 'method'=>'PUT']) !!}  
 {{-- Define un campo de texto para el nombre de la subcategoría --}}  
 <div class="form-group">  
 {!! Form::label('nombre', 'Nombre') !!}  
 {!! Form::text('nombre', null, [  
 'class' => 'form-control',  
 'placeholder' => 'Ingrese el nombre de la Subcategoría'  
 ]) !!}  
   
 {{-- Muestra un mensaje de error si hay un error de validación para el campo nombre --}}  
 @error('nombre')  
 <span class="text-danger">{{ $message }}</span>  
 @enderror  
   
 </div>  
   
 {{-- Define un campo de texto para la descripción del bloque --}}  
 <div class="form-group">  
 {!! Form::label('descripcion', 'Descripcion') !!}  
 {!! Form::textarea('descripcion', null, [  
 'class' => 'form-control',  
 'placeholder' => 'Ingrese la descripción de la Subcategoría'  
 ]) !!}  
   
 {{-- Muestra un mensaje de error si hay un error de validación para el campo descripcion --}}  
 @error('descripcion')  
 <span class="text-danger">{{ $message }}</span>  
 @enderror  
   
 </div>  
   
 {{-- Define el botón de envío del formulario --}}  
 {!! Form::submit('Actualizar categoria', [  
 'class'=>'btn btn-primary'  
 ]) !!}  
   
 {!! Form::close() !!} {{-- Cierra el formulario --}}  
 </div>  
</div>  
@stop  
  
@section('css')  
 <link rel="stylesheet" href="/js/ckeditor/ckeditor.css">  
@stop  
  
@section('js')  
 <script src="https://cdn.ckeditor.com/ckeditor5/30.0.0/classic/ckeditor.js"></script>  
 <script>  
 ClassicEditor  
 .create(document.querySelector('#descripcion'))  
 .catch(error => {  
 console.error(error);  
 });  
 </script>  
 <script>  
 $(document).ready(function() {  
 CKEDITOR.replace('descripcion');  
 });  
 </script>  
@stop

### ¿Qué hace?

A continuación, se comentan las secciones principales del código:

@extends('adminlte::page')  
  
@section('title', 'Tablero')

Esta línea establece que la vista extiende la plantilla “adminlte::page” y establece el título de la página como “Tablero”.

@section('content\_header')  
 <h1>Editar Subcategoría</h1>  
@stop

Esta sección define el encabezado del contenido de la página, que muestra el título “Editar Subcategoría”.

@section('content')  
<div class="card">  
 <div class="card-body">  
 {{-- Genera un formulario de actualización de subcategoría utilizando el modelo $subcategoria --}}  
 {!! Form::model($subcategoria,  
 ['route' => ['admin.subcategorias.update',$subcategoria->id],  
 'method'=>'PUT']) !!}  
 <!-- Contenido del formulario -->  
 {!! Form::close() !!} {{-- Cierra el formulario --}}  
 </div>  
</div>  
@stop

Esta sección define el contenido principal de la página, que incluye un formulario para editar una subcategoría existente. El formulario utiliza la función Form::model() para vincular el modelo $subcategoria al formulario y especifica la ruta admin.subcategorias.update con el ID de la subcategoría como destino para enviar los datos del formulario. El contenido específico del formulario se encuentra dentro de las etiquetas {!! Form::model() !!} y {!! Form::close() !!}.

Dentro del formulario, se definen los campos de texto para editar el nombre y la descripción de la subcategoría. Estos campos utilizan las funciones Form::label(), Form::text() y Form::textarea() para generar los campos y tienen valores predeterminados establecidos mediante el uso del modelo $subcategoria. Además, se muestra un mensaje de error si hay algún error de validación para los campos nombre y descripcion mediante el uso de la directiva @error.

@section('css')  
 <link rel="stylesheet" href="/js/ckeditor/ckeditor.css">  
@stop

Esta sección define el CSS adicional que se debe cargar en la página. En este caso, se carga un archivo CSS de CKEditor para estilos específicos.

@section('js')  
 <script src="https://cdn.ckeditor.com/ckeditor5/30.0.0/classic/ckeditor.js"></script>  
 <script>  
 ClassicEditor  
 .create(document.querySelector('#descripcion'))  
 .catch(error => {  
 console.error(error);  
 });  
 </script>  
 <script>  
 $(document).ready(function() {  
 CKEDITOR.replace('descripcion');  
 });  
 </script>  
@stop

Esta sección define el JavaScript adicional que se debe cargar en la página. Primero, se carga el script de CKEditor desde un CDN. Luego, se inicializa un editor CKEditor en el campo de texto con el ID “descripcion” utilizando la función ClassicEditor.create(). También se utiliza jQuery para reemplazar el campo de texto con el ID “descripcion” con un editor CKEditor adicional.

En resumen, este código define una vista de Blade que muestra un formulario para editar una subcategoría existente y utiliza CKEditor para mejorar la experiencia de edición del campo de descripción.

### Borrado de categoría

El borrado de categoría no implementa ninguna vista. El método se implementa en el controlador **SubCategoriaController**.

public function destroy($id)  
 {  
 $subcategoria = SubCategoria::find($id);  
  
 $subcategoria->delete();  
  
 return redirect()->route('admin.subcategorias.index')->with('success', 'Subcategoría eliminada correctamente');  
 }

### Definición de rutas

Vamos a ver las rutas definidas en el el fichero **admin.php** relacionadas con **subcategorias**. Comprueba que están definidas.

Route::get("/subcategorias",[SubCategoriaController::class,'index'])->name('admin.subcategorias.index');  
Route::post("/subcategorias/store",[SubCategoriaController::class,'store'])->name('admin.subcategorias.store');  
Route::get("/subcategorias/create",[SubCategoriaController::class,'create'])->name('admin.subcategorias.create');  
Route::get("/subcategorias/edit/{id}",[SubCategoriaController::class,'edit'])->name('admin.subcategorias.edit');  
Route::put("/subcategorias/update/{id}",[SubCategoriaController::class,'update'])->name('admin.subcategorias.update');  
Route::delete("/subcategorias/delete/{id}",[SubCategoriaController::class,'destroy'])->name('admin.subcategorias.delete');