

MANUAL ELABORACIÓN CRUD LARAVEL

APRENDIZ

JUAN SEBASTIAN DURAN CASTELLANOS

INSTRUCTOR

CESAR ARTURO ESQUIVEL CORTES

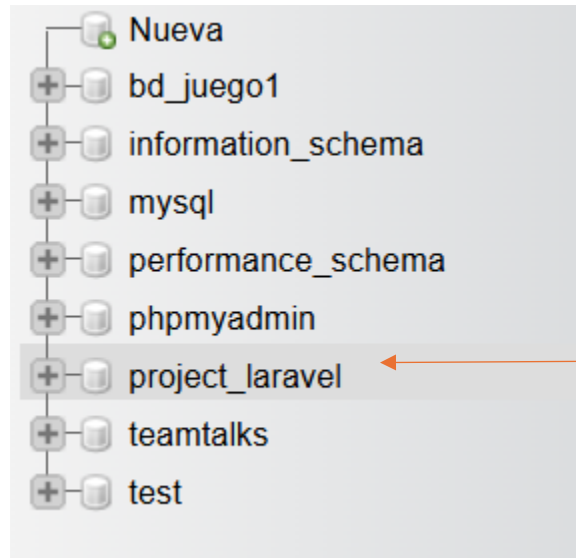


SERVICIO NACIONAL DE APRENDIZAJE SENA

IBAGUE-TOLIMA

2025

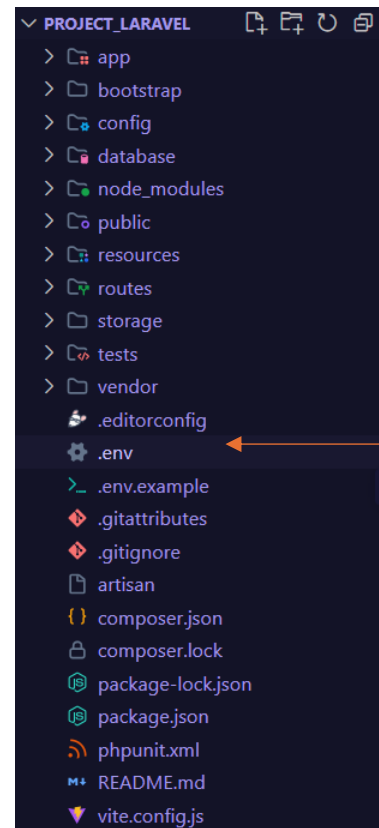
DEBEREMOS CREAR UNA BASE DE DATOS DE NUESTRO PROYECTO EN PHPMYADMIN.



Seguidamente nos dirigiremos a nuestro proyecto el cual llamamos project_laravel en donde va a contener toda la estructura del proyecto que acaba de crear laravel.

Luego de esto debemos ir a el archivo que se llama .env el cual aloja todos los parámetros de conexión a la base de datos.

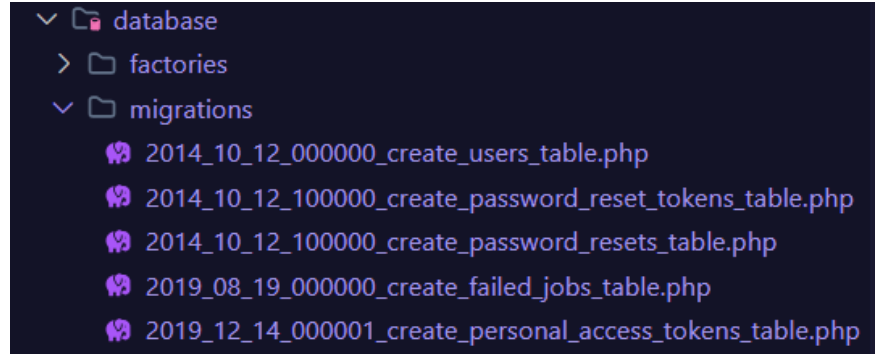
```
.env
1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=base64:6gtPoCtci7bGqWHP0Bm1bEnmudGX/aaDfGmHCpFAtTk=
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 LOG_CHANNEL=stack
8 LOG_DEPRECATIONS_CHANNEL=null
9 LOG_LEVEL=debug
10
11 DB_CONNECTION=mysql
12 DB_HOST=127.0.0.1
13 DB_PORT=3306
14 DB_DATABASE=project_laravel
15 DB_USERNAME=root
16 DB_PASSWORD=
```



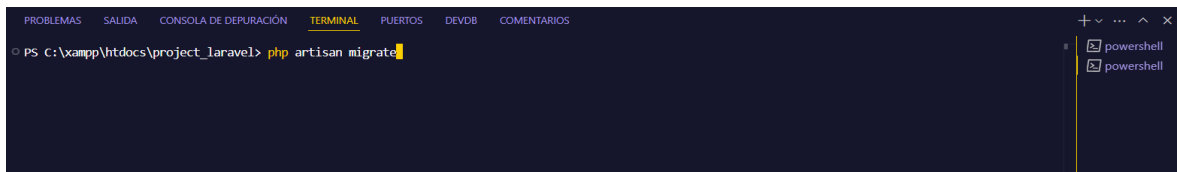
En este apartado deberemos realizar las respectivas configuraciones como agregar el nombre de nuestra base de datos o alguna configuración en especial.

Ahora deberemos dirigirnos hacia la carpeta **DATABASE** en donde encontraremos una subcarpeta llamada **MIGRATIONS**, el campo migrations es el que contiene todas las tablas que se van a migrar hacia la base de datos de phpmyadmin.

Generalmente laravel nos crea 4 migraciones en su estructura inicial.



Seguidamente deberemos migrar estas tablas a nuestra base de datos, Para esto deberemos dirigirnos hacia la terminal de nuestro visual studio code y digitar el comando “**php artisan migrate**”.

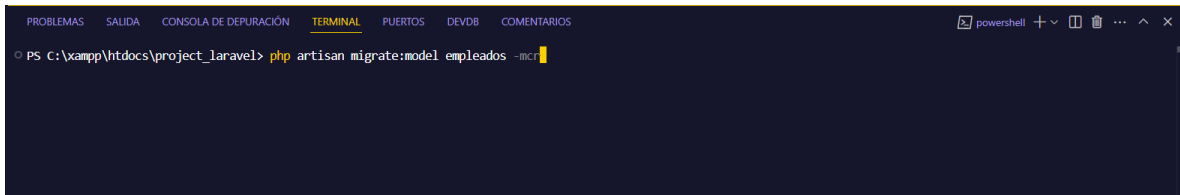


Con este comando lo que hacemos es migrar nuestras tablas hacia nuestra base de datos.

Tabla	Acción	Filas	Tipo	Cotejamiento	Tamaño	Residuo a depurar
<input type="checkbox"/> empleados	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_unicode_ci	16.0 KB	-
<input type="checkbox"/> failed_jobs	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_unicode_ci	32.0 KB	-
<input type="checkbox"/> migrations	Examinar Estructura Buscar Insertar Vaciar Eliminar	8	InnoDB	utf8mb4_unicode_ci	16.0 KB	-
<input type="checkbox"/> password_resets	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_unicode_ci	32.0 KB	-
<input type="checkbox"/> password_reset_tokens	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_unicode_ci	16.0 KB	-
<input type="checkbox"/> personal_access_tokens	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_unicode_ci	48.0 KB	-
<input type="checkbox"/> roles	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_unicode_ci	32.0 KB	-
<input type="checkbox"/> users	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_unicode_ci	48.0 KB	-
8 tablas	Número de filas	8	InnoDB	utf8mb4_general_ci	240.0 KB	0 B

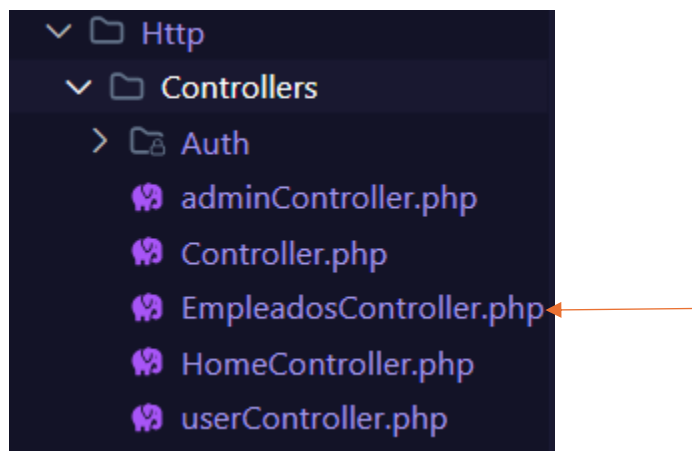
Nos debe de migrar 5 tablas a nuestra base de datos las cuales son users, personal_access_tokens ,password_reset_tokens, migrations y failed_jobs.

Luego de realizar las migraciones a nuestra base de datos debemos crear nuestro MVC (Modelo, Vista, Controlador), para esto debemos abrir nuevamente nuestra terminal y digitar el siguiente comando “**php artisan migrate:model empleados -mcr**”.

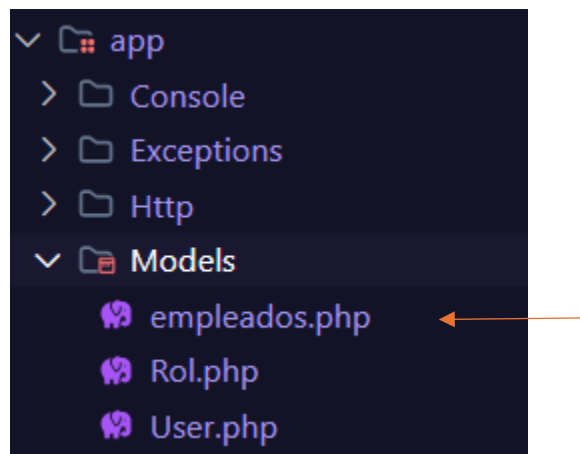


```
PS C:\xampp\htdocs\project_laravel> php artisan migrate:model empleados -mcr
```

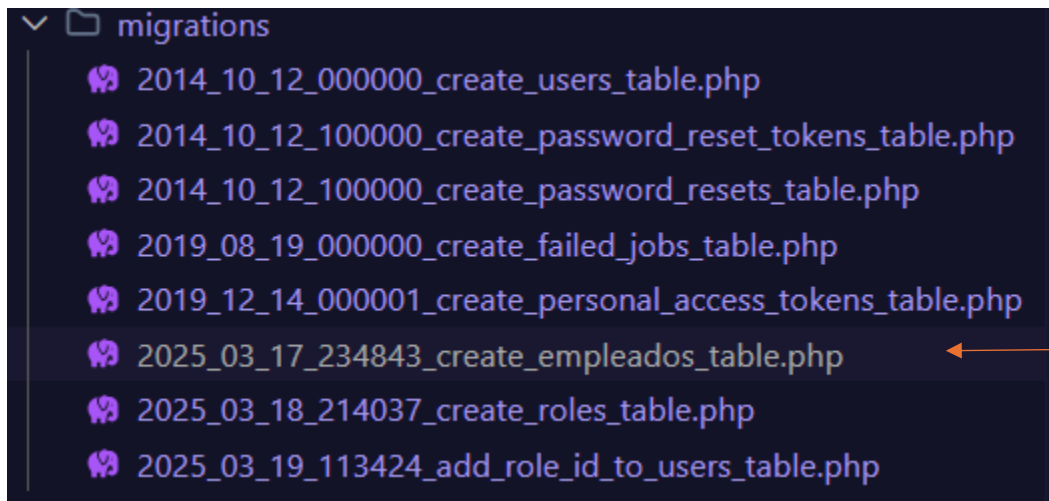
Si nos dirigimos a la carpeta “**app**” en la subcarpeta “**Http**” carpeta “**Controllers**” podemos evidenciar que nos creó un archivo controlador llamado “**EmpleadosController.php**”



Si verificamos también en “**app**” en la subcarpeta llamada “**Models**” podemos observar que se creó un archivo modelo llamado “**empleados.php**”.



Debemos dirigirnos a nuestras migraciones en donde se creó la de “**create_empleados_table.php**” y crear la estructura de nuestra tabla para almacenar nuestros empleados.

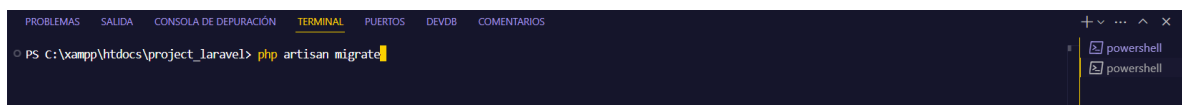


Una vez que verificamos de que este creada nuestra migración podemos realizar la estructura de nuestra tabla para que se almacenen los empleados.

```
1 <?php
2
3 use Illuminate\Database\Migrations\Migration;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 return new class extends Migration
8 {
9     /**
10      * Run the migrations.
11      */
12     public function up(): void
13     {
14         Schema::create('empleados', function (Blueprint $table) {
15             $table->id();
16             $table->string('nombre');
17             $table->string('primer_apellido');
18             $table->string('segundo_apellido');
19             $table->string('correo');
20             $table->string('foto');
21             $table->timestamps();
22         });
23     }
24
25     /**
26      * Reverse the migrations.
27      */
28     public function down(): void
29     {
30         Schema::dropIfExists('empleados');
31     }
32 };
33
```

En la “**public function up (): void**” dentro de “**Schema**” debemos agregar \$table y el tipo de dato que quiere que tenga esa columna con el nombre del campo que queremos que quede en nuestra tabla para lograr almacenar los datos de nuestros empleados.

Luego de hacer este abrimos una nueva terminal y digitamos el comando “**php artisan migrate**” para poder migrar la estructura realizada a nuestra base de datos.

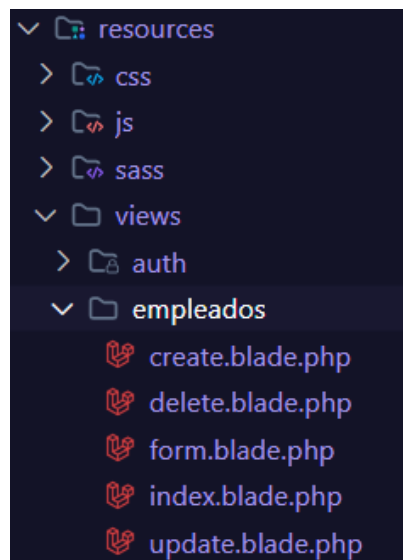


Nos debió crear una estructura de esta forma en nuestra tabla users con los campos que agregamos en nuestra migración.

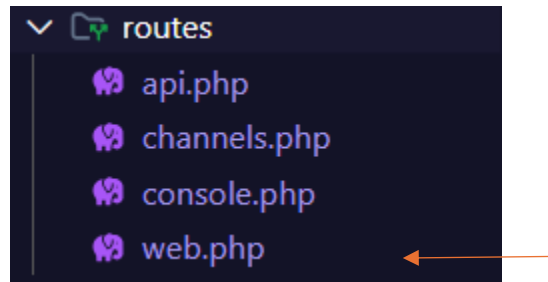
#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
<input type="checkbox"/> 1	id 🔑	bigint(20)		UNSIGNED	No	Ninguna		AUTO_INCREMENT	Cambiar Eliminar Más
<input type="checkbox"/> 2	name	varchar(255)	utf8mb4_unicode_ci		No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/> 3	email 📧	varchar(255)	utf8mb4_unicode_ci		No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/> 4	email_verified_at	timestamp			Sí	NULL			Cambiar Eliminar Más
<input type="checkbox"/> 5	password	varchar(255)	utf8mb4_unicode_ci		No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/> 6	role_id 🔑	bigint(20)		UNSIGNED	No	2			Cambiar Eliminar Más
<input type="checkbox"/> 7	remember_token	varchar(100)	utf8mb4_unicode_ci		Sí	NULL			Cambiar Eliminar Más
<input type="checkbox"/> 8	created_at	timestamp			Sí	NULL			Cambiar Eliminar Más
<input type="checkbox"/> 9	updated_at	timestamp			Sí	NULL			Cambiar Eliminar Más

Después de esto nos dirigimos a las vistas las cuales se encuentran en la carpeta “**resources**” en una subcarpeta llamada “**views**”.

Seguidamente dentro de views debemos crear una nueva carpeta llamada **empleados** con 5 nuevos archivos llamados **update.blade.php**, **index.blade.php**, **form.blade.php**, **delete.blade.php** y **create.blade.php**.



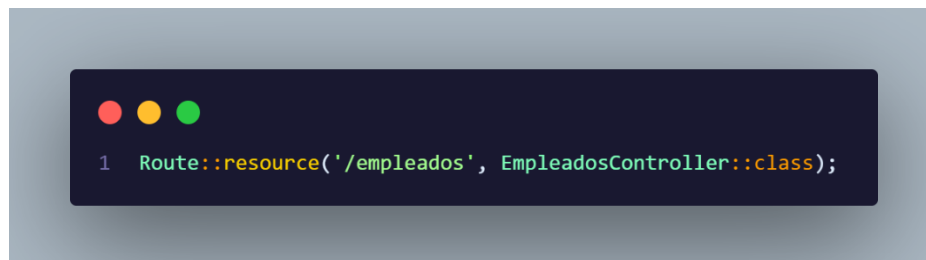
Ahora necesitamos acceder a las vistas del empleado que acabamos de crear. Las cuales se encuentran en el archivo **“web.php”** que esta ubicado en la carpeta **“routes”**



Una vez estemos en el archivo **web.php** procederemos a configurar las rutas



Esta es la ruta principal en la cual laravel nos hace acceder a la vista welcome.blade.php.



Esta es la ruta de recursos de empleados en donde nos lleva a la gestión de empleados y nos muestra la lista de empleados, agregar un nuevo empleado editar o eliminar sus datos, este `::class` nos admite para que podamos acceder a la carpeta empleados.

Seguidamente de esto nos dirigimos a el controlador `empleadosController.php` y empezaremos a configurar cada uno de nuestros controladores.

```

1 public function index()
2 {
3     $datos['empleados'] = empleados::paginate(5);
4     return view('empleados.index', $datos);
5 }

```

Esta función obtiene todos los empleados almacenados en la base de datos y los muestra paginados. Ósea que nos va a mostrar 5 empleados por página, y nos carga la vista `empleados.index` donde nos mostrará la lista.

```

1 public function create()
2 {
3     return view('empleados.create');
4 }

```

Esta devuelve la vista `empleados.create`, donde podemos llenar el formulario para agregar un nuevo empleado.

```

1 public function store(Request $request)
2 {
3     //
4     $datos_empleado = request()->except('_token');
5
6
7     if($request->hasFile('foto')){
8
9         $datos_empleado['foto'] = $request->file('foto')->store('uploads', 'public');
10    }
11    empleados::insert($datos_empleado);
12
13    return redirect('empleados')->with('mensaje', 'Empleado agregado exitosamente');
14 }

```

Esta recibe los datos de nuestro formulario y los guarda en la base de datos, si se sube una imagen o foto será almacenada en la carpeta **“uploads”** dentro de `storage/app/public`. En donde nos redirige a la lista de empleados con un mensaje de éxito.


```

1 public function edit($id)
2 {
3     $empleado = empleados::find($id);
4     return view('empleados.update', compact('empleado'));
5 }

```

Básicamente lo que hace es que por medio de el id con el que esta guardado el empleado en la base de datos nos envía una vista empleados.update donde se pueden actualizar o modificar los datos insertados anteriormente.

```

1 public function update(Request $request,$id)
2 {
3     //
4     $datos_empleado = request()->except('_token','_method');
5
6     if($request->hasFile('foto')){
7
8         $empleado= empleados::findOrFail($id);
9         Storage::delete('public/' . $empleado->foto);
10
11         $datos_empleado['foto'] = $request->file('foto')->store('uploads', 'public');
12     }
13
14     empleados::where('id', '=', $id)->update($datos_empleado);
15     $empleado= Empleados::findOrFail($id);
16     return view('empleados.update', compact('empleado'));
17 }

```

Acá se reciben los datos actualizados del formulario y los guarda en la base de datos si se modifica la imagen o foto también será actualizada automáticamente, después de esto nos retorna la vista con la actualización con los nuevos datos.

```

1 public function destroy($id)
2 {
3     $empleado = empleados::findOrFail($id);
4
5     // Eliminar la imagen asociada si existe
6     if ($empleado->foto) {
7         Storage::delete('public/' . $empleado->foto);
8     }
9
10    empleados::destroy($id);
11    return redirect('empleados')->with('mensaje', 'Empleado eliminado correctamente');
12 }

```

Busca el empleado en la base de datos en donde si el empleado tiene una foto almacenada, la elimina antes de borrar el registro. Después de esto elimina el empleado y redirige con un mensaje de confirmación.

```

1 <?php
2 namespace App\Http\Controllers;
3
4 use App\Models\empleados;
5 use Illuminate\Http\Request;
6 use Illuminate\Support\Facades\Storage;
7
8 class EmpleadosController extends Controller
9 {
10
11    public function index()
12    {
13        $datos['empleados'] = empleados::paginate();
14        return view('empleados.index', $datos);
15    }
16
17    public function create()
18    {
19        return view('empleados.create');
20    }
21
22    public function store(Request $request)
23    {
24        //
25        $datos_empleado = request()->except('_token');
26
27        if($request->hasFile('foto')){
28            $datos_empleado['foto'] = $request->file('foto')->store('uploads', 'public');
29        }
30        empleados::insert($datos_empleado);
31        return redirect('empleados')->with('mensaje', 'Empleado agregado exitosamente');
32    }
33
34    public function show(empleados $empleados)
35    {
36        //
37    }
38
39    public function edit($id)
40    {
41        $empleado = empleados::findOrFail($id);
42        return view('empleados.update', compact('empleado'));
43    }
44
45    public function update(Request $request, $id)
46    {
47        //
48        $datos_empleado = request()->except('_token', '_method');
49
50        if($request->hasFile('foto')){
51            $empleado = empleados::findOrFail($id);
52            Storage::delete('public/' . $empleado->foto);
53            $datos_empleado['foto'] = $request->file('foto')->store('uploads', 'public');
54        }
55        empleados::where('id', '=', $id)->update($datos_empleado);
56        $empleado = empleados::findOrFail($id);
57        return view('empleados.update', compact('empleado'));
58    }
59
60    public function destroy($id)
61    {
62        $empleado = empleados::findOrFail($id);
63
64        // Eliminar la imagen asociada si existe
65        if ($empleado->foto) {
66            Storage::delete('public/' . $empleado->foto);
67        }
68
69        empleados::destroy($id);
70        return redirect('empleados')->with('mensaje', 'Empleado eliminado correctamente');
71    }
72 }

```

Nos dirigimos a form.blade.php y insertamos nuestro formulario

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 <title>Formulario de Empleado</title>
7 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
8
9 </head>
10 <body class="bg-light">
11 <div class="container mt-5">
12 <div class="card shadow p-4">
13 <h2 class="text-center mb-4 text-primary">Formulario para crear empleados</h2>
14
15 <form>
16 <div class="row gx-3">
17 <div class="col-md-6 mb-3">
18 <label for="nombre" class="form-label">Nombre Empleado</label>
19 <input type="text" class="form-control" name="nombre" id="nombre" placeholder="Introduzca su nombre" value="{{ isset($empleado->nombre) ? $empleado->nombre : '' }}" required>
20 </div>
21 <div class="col-md-6 mb-3">
22 <label for="primer_apellido" class="form-label">Primer Apellido</label>
23 <input type="text" class="form-control" name="primer_apellido" id="primer_apellido" placeholder="Introduzca su primer apellido" value="{{ isset($empleado->primer_apellido) ? $empleado->primer_apellido : '' }}" required>
24 </div>
25 </div>
26
27 <div class="row gx-3">
28 <div class="col-md-6 mb-3">
29 <label for="segundo_apellido" class="form-label">Segundo Apellido</label>
30 <input type="text" class="form-control" name="segundo_apellido" id="segundo_apellido" placeholder="Introduzca su segundo apellido" value="{{ isset($empleado->segundo_apellido) ? $empleado->segundo_apellido : '' }}" required>
31 </div>
32 <div class="col-md-6 mb-3">
33 <label for="correo" class="form-label">Correo Electrónico</label>
34 <input type="email" class="form-control" name="correo" id="correo" placeholder="Introduzca su correo" value="{{ isset($empleado->correo) ? $empleado->correo : '' }}" required>
35 </div>
36 </div>
37
38 @if (isset($empleado->foto))
39 <div class="mb-3 text-center">
40 
41 </div>
42 @endif
43
44 <div class="mb-3">
45 <label for="foto" class="form-label">Foto</label>
46 <input type="file" class="form-control" name="foto" id="foto">
47 </div>
48
49 <button type="submit" class="btn btn-primary w-100 py-2 fw-bold">Guardar Empleado</button>
50 </form>
51 </div>
52 </div>
53
54 <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
55 </body>
56 </html>
```

```
1 @if (isset($empleado->foto))
```

Se usa @if para comprobar si la variable \$empleado->foto esta definida y contiene algún valor. Por lo tanto, si el empleado ya tiene una foto guardada en la base de datos, se ejecutará el código dentro del @if.

```

1 <div class="mb-3 text-center">
2   
3 </div>

```

La imagen se carga usando `src="{{ asset('storage').'/'. $empleado->foto }}"` en donde `asset('storage')` va a obtener la URL en la carpeta de almacenamiento y se concatena con el nombre de archivo guardado en `$empleado->foto` para mostrar la imagen correcta.

```

1 <div class="mb-3">
2   <label for="foto" class="form-label">Foto</label>
3   <input type="file" class="form-control" name="foto" id="foto">
4 </div>

```

Se crea un campo de entrada `<input>` para que el usuario pueda subir una nueva foto y un `type="file"` donde nos permite que se pueda hacer la selección de los archivos.

Formulario para crear empleados

Nombre Empleado

Primer Apellido

Segundo Apellido

Correo Electrónico

Foto

Guardar Empleado

	id	nombre	primer_apellido	segundo_apellido	correo	foto	created_at	updated_at
	2	Juan Sebastian	Duran	castellanos	jdurancastellanos21@gmail.com	uploads/4dJwHstmTuK8gooNck7iS9XjMjcEwLC5gpLc5mM a...	NULL	NULL

Acá podemos comprobar que si guarda los datos registrados en nuestra base de datos.

Ahora nos dirigimos a index.blade.php y vamos a crear la lista de empleados registrados

```

1  @extends('layouts.app')
2  @section('content')
3
4  <div class='container'>
5
6      <!DOCTYPE html>
7      <html lang="es">
8      <head>
9          <meta charset="UTF-8">
10         <meta name="viewport" content="width=device-width, initial-scale=1.0">
11         <meta http-equiv="X-UA-Compatible" content="ie=edge">
12         <title>Lista de Empleados</title>
13         <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
14     </head>
15     <body>
16         <div class="container mt-4">
17             <h1 class="text-center">Lista de los Empleados</h1>
18             <a href="{{ url('/empleados/create') }}" class="btn btn-primary mb-3">Registrar Nuevo Empleado</a>
19             <table class="table table-light table-bordered">
20                 <thead class="thead-light">
21                     <tr>
22                         <th>#</th>
23                         <th>Foto</th>
24                         <th>Nombre</th>
25                         <th>Primer Apellido</th>
26                         <th>Segundo Apellido</th>
27                         <th>Correo</th>
28                         <th>Acción</th>
29                     </tr>
30                 </thead>
31                 <tbody>
32                     @foreach ($empleados as $datos)
33                         <tr>
34                             <td>{{ $datos->id }}</td>
35                             <td></td>
36                             <td>{{ $datos->nombre }}</td>
37                             <td>{{ $datos->primer_apellido }}</td>
38                             <td>{{ $datos->segundo_apellido }}</td>
39                             <td>{{ $datos->correo }}</td>
40
41                             <td>
42                                 <a href="{{ route('empleados.edit', $datos->id) }}" class="btn btn-primary btn-sm">Editar</a>
43                                 <form action="{{ route('empleados.destroy', $datos->id) }}" method="POST" style="display:inline;">
44                                     @csrf
45                                     @method('DELETE')
46                                     <button type="submit" class="btn btn-danger btn-sm onclick="return confirm('¿Estás seguro?')">Borrar</button>
47                                 </form>
48                             </td>
49                         </tr>
50                     @endforeach
51                 </tbody>
52             </table>
53             {!! $empleados->links() !!}
54         </div>
55     </body>
56 </html>
57 </div>
58 @endsection
59

```

```
1 @extends('layouts.app')
2 @section('content')
```

Laravel permite extender una plantilla base para mantener una estructura común en la aplicación. Aquí, la vista de layouts.app donde se asegura que se mantenga la misma cabecera y pie de página en todas las páginas de la aplicación.

```
1 <a href="{{ url('/empleados/create') }}" class="btn btn-primary mb-3">Registrar Nuevo Empleado</a>
```

Este botón dirige a la vista de creación de empleados usando la URL /empleados/create.

```
1 <table class="table table-light table-bordered">
2     <thead class="thead-light">
3         <tr>
4             <th>#</th>
5             <th>Foto</th>
6             <th>Nombre</th>
7             <th>Primer Apellido</th>
8             <th>Segundo Apellido</th>
9             <th>Correo</th>
10            <th>Acción</th>
11        </tr>
12    </thead>
```

Se crea una tabla con las columnas necesarias para mostrar la información de los empleados.

```
1 <tbody>
2     @foreach ($empleados as $datos)
3     <tr>
4         <td>{{ $datos->id }}</td>
5         <td></td>
6         <td>{{ $datos->nombre }}</td>
7         <td>{{ $datos->primer_apellido }}</td>
8         <td>{{ $datos->segundo_apellido }}</td>
9         <td>{{ $datos->correo }}</td>
```

Se recorre la colección \$empleados y se muestra cada registro en una fila de la tabla. La imagen del empleado se recupera de la carpeta storage.

```
1 <td>
2     <a href="{{ route('empleados.edit', $datos->id) }}" class="btn btn-primary btn-sm">Editar</a>
3     <form action="{{ route('empleados.destroy', $datos->id) }}" method="POST" style="display:inline;"
4         @csrf
5         @method('DELETE')
6         <button type="submit" class="btn btn-danger btn-sm" onclick="return confirm('¿Estás seguro?');">Borrar</button>
7     </form>
8 </td>
```

Este contiene un botón de editar que dirige a la ruta empleados.edit con el id del empleado en donde maneja un método delete para eliminar un empleado, protegiendo la acción con @csrf y @method('DELETE') y nos va a pedir una confirmación de que si desea eliminar ese empleado.

```
1 {!! $empleados->links() !!}
```

Muestra los controles de paginación proporcionados por Laravel cuando la cantidad de empleados supera el número permitido por página.

Lista de los Empleados

#	Foto	Nombre	Primer Apellido	Segundo Apellido	Correo	Acción
1		Juan Sebastian	Duran	Castellanos	jdurancastellanos21@gmail.com	Editar Borrar
2		Brian Stiven	Rocha	Poveda	rocha@gmail.com	Editar Borrar
3		Edwar Farid	Gomez	Sanchez	edwar@gmail.com	Editar Borrar
4		Juan Sebastian	Aranda	Lozano	aranda@gmail.com	Editar Borrar
5		Edier Santiago	Moyano	Betancourt	moyano@gmail.com	Editar Borrar

[<](#) [1](#) [2](#) [>](#)

Después de esto nos dirigimos al archivo create.blade.php y editaremos su configuración

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <meta http-equiv="X-UA-Compatible" content="ie=edge">
7     <title>Formulario Registro</title>
8     <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
9 </head>
10 <body>
11     <form action="{{ url('empleados') }}" method="POST" enctype="multipart/form-data">
12         @csrf
13         @include('empleados.form');
14     </form>
15 </body>
16 </html>
```

```
1 <body>
2     <form action="{{ url('empleados') }}" method="POST" enctype="multipart/form-data">
3         @csrf
4         @include('empleados.form');
5     </form>
6 </body>
7 </html>
```

la vista empleados.form, se facilita el mantenimiento y modularidad del sistema.

Seguido de esto nos dirigimos al update.blade.php y editamos esta configuración

```
1 <!DOCTYPE html>
2 <html Lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <meta http-equiv="X-UA-Compatible" content="ie=edge">
7   <title>Formulario Actualizar</title>
8 </head>
9 <body>
10   @if (Session::has('mensaje'))
11     {{Session::get('mensaje')}}
12
13   @endif
14
15   <form action="{{url('/empleados/'. $empleado->id)}}" method="POST" enctype="multipart/form-data">
16     @csrf
17     {{method_field('PATCH')}}
18
19     @include('empleados.form', ['modo'=>'EDITAR'])
20
21   </form>
22 </body>
23 </html>
```

```
1 @if (Session::has('mensaje'))
2   {{Session::get('mensaje')}}
3
4 @endif
```

Esto comprueba si existe un mensaje en la sesión (Session::has('mensaje')) y si hay un mensaje lo mostrara con Session::get('mensaje') , esto nos ayuda a mostrar confirmaciones o errores tras actualizar un empleado.

```
1 <form action="{{url('/empleados/'. $empleado->id)}}" method="POST" enctype="multipart/form-data">
2   @csrf
3   {{method_field('PATCH')}}
```

El action="{{url('/empleados/'. \$empleado->id)}}" Especifica la URL con el ID del empleado para que el formulario actualice los datos.



Se importa la vista parcial `empleados.form` en donde la variable `modo` con el valor `EDITAR`, permite que el formulario se adapte según su uso (registro o edición).

The form is titled "Formulario Para Registrar Empleados" and is set against a light purple background. It contains four input fields, each with a user icon and a label: "Introduzca Nombre", "Introduzca Primer Apellido", "Introduzca Segundo Apellido", and "Introduzca Email". Below these fields is a square image placeholder showing a man's face, labeled "Foto". Under the photo, there is a file selection interface with the text "Seleccionar archivo" and "Ningún archivo seleccionado". At the bottom of the form is a large blue button labeled "Guardar Empleado".

Creamos una nueva migración en la cual vamos a crear la tabla roles

```
1  <?php
2
3  use Illuminate\Database\Migrations\Migration;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Support\Facades\Schema;
6
7  return new class extends Migration
8  {
9      /**
10       * Run the migrations.
11       */
12     public function up(): void
13     {
14         Schema::create('roles', function (Blueprint $table) {
15             $table->id();
16             $table->string('name') ->unique();
17             $table->timestamps();
18         });
19     }
20
21     /**
22      * Reverse the migrations.
23      */
24     public function down(): void
25     {
26         Schema::dropIfExists('roles');
27     }
28 };
```

Debemos dirigirnos hacia la terminal de visual studio code y vamos a digitar el comando

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS  DEVD  COMENTARIOS
PS C:\xampp\htdocs\project_laravel> php artisan make:migration roles
```

Diríjase a la carpeta models y entre a el archivo Rol.php

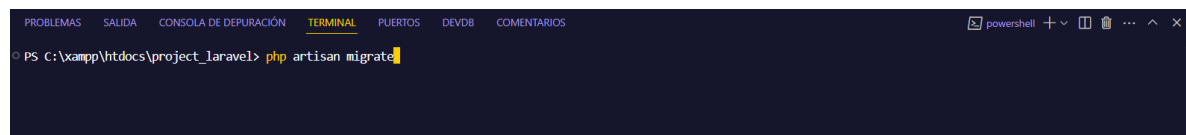
```

1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  class Rol extends Model
9  {
10     use HasFactory;
11     protected $fillable = ['name'];
12     protected $table = 'roles';
13     public function users()
14     {
15         return $this->hasMany(User::class, 'role_id');
16     }
17 }
18

```

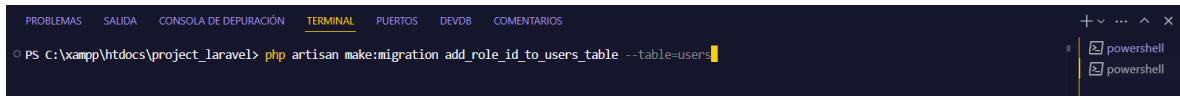
En este apartado asignaremos los campos que llevara nuestra tabla rol

Después de realizar esto deberemos entrar nuevamente a nuestra terminal y digitar el comando php artisan migrate para cargar nuestra tabla rol en la base de datos



The image shows a terminal window with a dark background. At the top, there are tabs for 'PROBLEMAS', 'SALIDA', 'CONSOLA DE DEPURACIÓN', 'TERMINAL' (which is active), 'PUERTOS', 'DEVDB', and 'COMENTARIOS'. Below the tabs, the command prompt shows 'PS C:\xampp\htdocs\project_laravel> php artisan migrate' with the cursor at the end of the line.

Una vez ejecutado ese comando nos creara la tabla roles en nuestra base de datos



```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS  DEVD8  COMENTARIOS
PS C:\xampp\htdocs\project_laravel> php artisan make:migration add_role_id_to_users_table --table=users
```

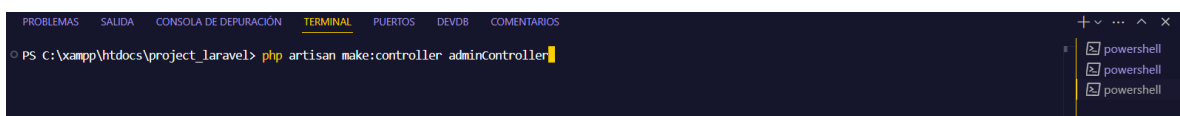
Entramos a la terminal y digitamos este comando para crear una nueva migración que es donde vamos a trabajar en la relación de la foreign key y que quede role_id en la tabla users.



```
1 <?php
2 use Illuminate\Database\Migrations\Migration;
3 use Illuminate\Database\Schema\Blueprint;
4 use Illuminate\Support\Facades\Schema;
5
6 return new class extends Migration {
7     public function up(): void
8     {
9         Schema::table('users', function (Blueprint $table) {
10             $table->unsignedBigInteger('role_id')->default(2)->after('password'); // Valor predeterminado 2
11             $table->foreign('role_id')->references('id')->on('roles')->onDelete('cascade');
12         });
13     }
14
15     public function down(): void
16     {
17         Schema::table('users', function (Blueprint $table) {
18             $table->dropForeign(['role_id']);
19             $table->dropColumn('role_id');
20         });
21     }
22 };
23
```

Una vez ejecutemos este comando se creará la migración en donde vamos a configurar la relación entre la tabla roles y users en donde vamos a traer la primary key de roles a users.

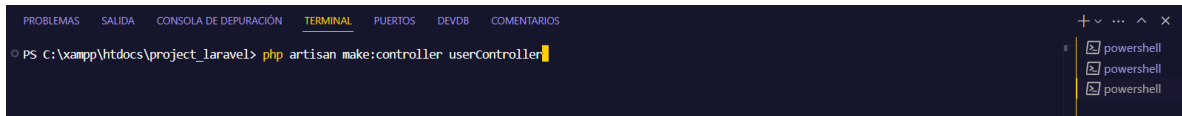
Luego debemos crear un nuevo controlador llamado adminController.php, entramos a la terminal y ejecutamos el comando php artisan make:controller adminController



```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS  DEVD8  COMENTARIOS
PS C:\xampp\htdocs\project_laravel> php artisan make:controller adminController
```

Con este comando se creará nuestro controlador admin

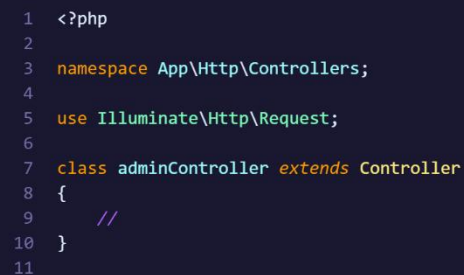
Ahora debemos hacer lo mismo solo que agregamos el userController.php donde debemos ejecutar el comando `php artisan make:controller userController`



```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS  DEVD  COMENTARIOS
PS C:\xampp\htdocs\project_laravel> php artisan make:controller userController
```

Ya con esto creamos nuestros dos controladores llamados `adminController.php` y `userController.php`.

Controlador admin:



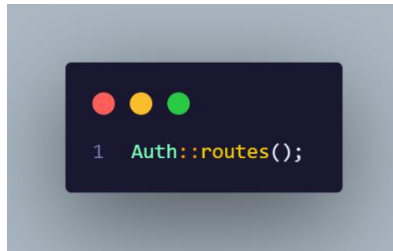
```
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6
7  class adminController extends Controller
8  {
9      //
10 }
11
```

Controlador user:

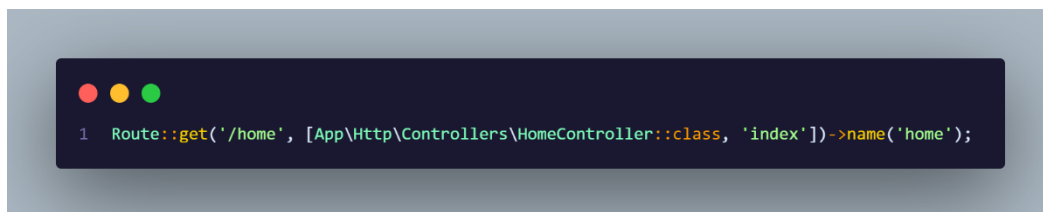


```
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6
7  class userController extends Controller
8  {
9      public function index()
10     {
11         return view('usuario.index');
12     }
13 }
14
```

Luego de realizar esto nos dirigimos a el archivo web.php y agregamos las rutas que nos van a llevar a el apartado de usuario y administrador.



Este fragmento de Auth lo que hace es que verifica y autentifica de que la información o datos sean validos.

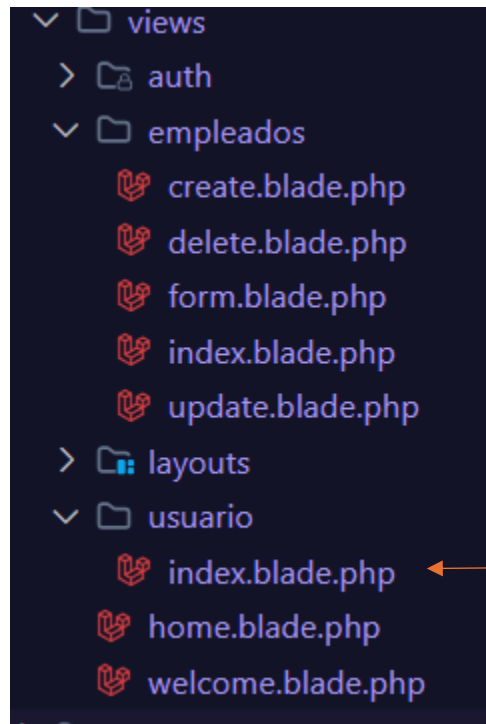


Route::get('/home', ...) Define una ruta HTTP GET para la URL /home y [App\Http\Controllers\HomeController::class, 'index'] lo que hace es que especifica que la petición sea manejada por el método index del controlador HomeController y el ->name('home') asigna un nombre a la ruta (home), lo que permite ser referenciada fácilmente en otras partes de la aplicación.



Route::get('/usuario/index',)Define una ruta HTTP GET para la URL /usuario/index mientras que el [UserController::class, 'index'] llama al método index del UserController cuando se accede a esta ruta y el ->name('usuario.index') nos va a asignar el nombre usuario.index a la ruta, facilitando su uso en redirecciones o enlaces.


Seguidamente de esto nos vamos a dirigir nuevamente a las vistas en la carpeta views y vamos a crear una carpeta llamada usuario la cual va a contener un archivo llamado index.blade.php.



```
1  @extends('layouts.app')
2  @section('content')
3
4  <div class='container'>
5
6
7      <html Lang="en">
8      <head>
9          <meta charset="UTF-8">
10         <meta name="viewport" content="width=device-width, initial-scale=1.0">
11         <meta http-equiv="X-UA-Compatible" content="ie=edge">
12         <title>Document</title>
13     </head>
14     <body>
15         <h1>Hello World</h1>
16     </body>
17
18 </div>
19 @endsection
20
```


Por último, nos vamos a dirigir a la carpeta controllers en la subcarpeta de que dice Auth e ingresamos a el archivo LoginController.php y vamos a editar su configuración para su correcto funcionamiento.


```
1  <?php
2
3  namespace App\Http\Controllers\Auth;
4
5  use App\Http\Controllers\Controller;
6  use Illuminate\Foundation\Auth\AuthenticatesUsers;
7  use Illuminate\Support\Facades\Auth;
8
9  class LoginController extends Controller
10 {
11     /*
12     |-----
13     | Login Controller
14     |-----
15     |
16     | This controller handles authenticating users for the application and
17     | redirecting them to your home screen. The controller uses a trait
18     | to conveniently provide its functionality to your applications.
19     |
20     */
21
22     use AuthenticatesUsers;
23
24     /**
25     * Where to redirect users after login.
26     *
27     * @var string
28     */
29     protected function redirectTo ()
30     {
31         $user = Auth::user();
32         if ($user->role->name === 'Administrador'){
33             return '/empleados';
34         } elseif ($user->role->name === 'Empleado') {
35             return 'usuario/index';
36         }
37         return '/home';
38     }
39     /**
40     * Create a new controller instance.
41     *
42     * @return void
43     */
44     public function __construct()
45     {
46         $this->middleware('guest')->except('logout');
47         $this->middleware('auth')->only('logout');
48     }
49 }
50
```



```
1 namespace App\Http\Controllers\Auth;
2
3 use App\Http\Controllers\Controller;
4 use Illuminate\Foundation\Auth\AuthenticatesUsers;
5 use Illuminate\Support\Facades\Auth;
```

Lo que hace el namespace `App\Http\Controllers\Auth;` es prácticamente que define que ese controlador pertenezca al espacio de nombres de autenticación mientras que el `use App\Http\Controllers\Controller;` adapta la funcionalidad base de los controladores de laravel.

El `use Illuminate\Foundation\Auth\AuthenticatesUsers;` lo que hace es que incluye el trait que proporciona las funciones básicas de autenticación y el `use Illuminate\Support\Facades\Auth;` nos permite acceder a la autenticación de usuarios.




```
1 use AuthenticatesUsers;
```

Esto proporciona métodos esenciales como el inicio de sesión, cierre de sesión y verificación de credenciales.



```
1 protected function redirectTo ()
2 {
3     $user = Auth::user();
4     if ($user->role->name === 'Administrador'){
5         return '/empleados';
6     } elseif ($user->role->name === 'Empleado') {
7         return 'usuario/index';
8     }
9     return '/home';
10 }
```

Básicamente el Auth::user obtiene al usuario autenticado y se encarga de verificar el rol del usuario para redirigirlo.




```
1 public function __construct()
2 {
3     $this->middleware('guest')->except('logout');
4     $this->middleware('auth')->only('logout');
5 }
```

El `$this->middleware('guest')->except('logout');` funciona para que los usuarios no autenticados puedan acceder a las funciones de iniciar sesión y el `$this->middleware('auth')->only('logout');` asegura que solo usuarios autenticados puedan cerrar sesión.

Luego hay mismo en la carpeta Auth nos dirigimos a el archivo RegisterController.php y también editamos su configuración.

```
1 <?php
2
3 namespace App\Http\Controllers\Auth;
4
5 use App\Http\Controllers\Controller;
6 use App\Models\User;
7 use Illuminate\Foundation\Auth\RegistersUsers;
8 use Illuminate\Support\Facades\Hash;
9 use Illuminate\Support\Facades\Validator;
10 use Illuminate\Support\Facades\Auth;
11
12 class RegisterController extends Controller
13 {
14     /**
15      * Register Controller
16      *
17      * This controller handles the registration of new users as well as their
18      * validation and creation. By default this controller uses a trait to
19      * provide this functionality without requiring any additional code.
20      */
21
22     use RegistersUsers;
23
24     /**
25      * Where to redirect users after registration.
26      *
27      * @var string
28      */
29     protected function redirectTo ()
30     {
31         $user = Auth::user();
32         if ($user->role->name === 'Administrador'){
33             return '/empleados';
34         } elseif ($user->role->name === 'Empleado') {
35             return 'usuario/index';
36         }
37         return '/home';
38     }
39
40     /**
41      * Create a new controller instance.
42      *
43      * @return void
44      */
45     public function __construct()
46     {
47         $this->middleware('guest');
48     }
49
50     /**
51      * Get a validator for an incoming registration request.
52      *
53      * @param array $data
54      * @return \Illuminate\Contracts\Validation\Validator
55      */
56     protected function validator(array $data)
57     {
58         return Validator::make($data, [
59             'name' => ['required', 'string', 'max:255'],
60             'email' => ['required', 'string', 'email', 'max:255', 'unique:users'],
61             'password' => ['required', 'string', 'min:8', 'confirmed'],
62         ]);
63     }
64
65     /**
66      * Create a new user instance after a valid registration.
67      *
68      * @param array $data
69      * @return \App\Models\User
70      */
71     protected function create(array $data)
72     {
73         return User::create([
74             'name' => $data['name'],
75             'email' => $data['email'],
76             'password' => Hash::make($data['password']),
77             'role_id' => $data['role_id'] ?? 2,
78         ]);
79     }
80 }
81
82 }
```



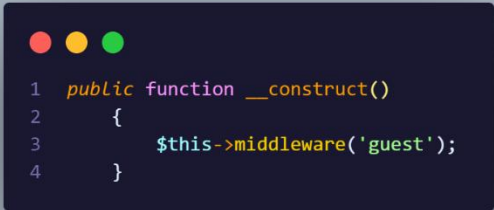
```
1 use RegistersUsers;
```

Esto maneja la lógica del registro y valida los datos del usuario antes de ser guardados y lo que hace es crear un nuevo usuario en la base de datos.



```
1 protected function redirectTo ()
2 {
3     $user = Auth::user();
4     if ($user->role->name === 'Administrador'){
5         return '/empleados';
6     } elseif ($user->role->name === 'Empleado') {
7         return 'usuario/index';
8     }
9     return '/home';
10 }
```

Obtiene el usuario recién registrado y verifica el rol del usuario para redirigirlo al sitio correcto.



```
1 public function __construct()
2 {
3     $this->middleware('guest');
4 }
```

Esta función se basa en que solo los usuarios no autenticados pueden acceder al registro.

```

1  protected function validator(array $data)
2  {
3      return Validator::make($data, [
4          'name' => ['required', 'string', 'max:255'],
5          'email' => ['required', 'string', 'email', 'max:255', 'unique:users'],
6          'password' => ['required', 'string', 'min:8', 'confirmed'],
7      ]);
8  }

```

Esta función valida los campos del formulario dependiendo de las limitaciones que tengamos en cada campo el va a verificar que se cumplan con esos requisitos.

```

1  protected function create(array $data)
2  {
3      return User::create([
4          'name' => $data['name'],
5          'email' => $data['email'],
6          'password' => Hash::make($data['password']),
7          'role_id' => $data['role_id'] ?? 2,
8      ]);
9  }

```

Esta función crea un nuevo usuario en la base de datos con los datos proporcionados, el Hash::make(\$data['password']) encripta la contraseña antes de guardarla y con el role_id es asignado un rol por defecto que es el 2 (Usuario).