

Gestión de Carta de Restaurante con Laravel.

24/01/2025

Pablo García Espín

Desarrollo de aplicaciones web en entorno servidor

Índice

Introducción.....	3
Rutas para ver las vistas.....	3
Objetivos.....	3
Motivación.....	3
Funcionalidades.....	3
Desarrollo de Sprints.....	4
Sprint 1.....	4
Sprint 2.....	7
Sprint 3.....	8
Sprint 4.....	8
Esquema de Base de Datos.....	9
Guia de Usuario y Administrador.....	10
Guia de usuario.....	10
Guia de Administrador.....	10
Código Relevante.....	13
Conclusiones.....	13
Propuestas de Mejora.....	13
Dificultades encontradas.....	13
Anexos.....	13

Introducción

Rutas para ver las vistas

Vista pública: <http://localhost/TrabajoLaravel/laravel/public/products>

Vista privada: <http://localhost/TrabajoLaravel/laravel/public/login>

Objetivos

El objetivo de esta práctica es la realización mediante el uso de Laravel de dos vistas: una vista pública la cual será la carta del restaurante para los usuarios y una vista privada para los administradores en la que se podrán añadir nuevos productos y categorías a los productos.

Motivación

Lo que me motiva en este proyecto es aprender una nueva tecnología y saber desenvolverse con ella.

Desarrollo de Sprints

Sprint 1

Configuración Inicial:

Para la realización de la creación del proyecto de Laravel he seguido los pasos dados en las diapositivas. Siguiendo el siguiente orden: primero la creación del proyecto y la base de datos junto a la configuración del archivo .env, el cual queda de la siguiente manera:

```
DB_DATABASE=restaurante
DB_USERNAME=root
DB_PASSWORD=
```

Únicamente cambiando el nombre de la base de datos por la que hayamos creado. El siguiente paso ha sido la instalación del middleware que, suponiendo que tengamos ya instalado Node.js, crearemos la siguiente ruta e incluiremos dentro las rutas que queramos que se vean en la vista privada.

```
Route::middleware('auth')->group(function(){
    Route::get('products/create','ProductController@create')->name('products.create');
});
Route::get('products','ProductController@index')->name('products.index');
```

Dentro de la carpeta abrimos PowerShell y escribiremos los siguientes comandos:

- composer require laravel/ui
- php artisan ui vue --auth (nos genera las rutas necesarias)
- npm install
- npm install --save-dev vite laravel-vite-plugin
- npm install --save-dev @vitejs/plugin-vue
- npm run build

Ahora tenemos en nuestro archivo de rutas la siguiente ruta:

```
Auth::routes();
```

Genera otra ruta más pero esa se puede eliminar. Con esto tenemos la aplicación protegida.

El siguiente paso es crear los controladores para lo cual hace falta el siguiente comando:
php artisan make:controller (nombre de la tabla)Controller

En el archivo RouteServiceProvider que se encuentra en app/Providers debemos añadir lo siguiente:

```
protected $namespace = 'App\\Http\\Controllers';
```

Y en la función boot añadiremos dos líneas

```
public function boot()
{
    $this->configureRateLimiting();

    $this->routes(function () {
        Route::middleware('api')
            ->namespace($this->namespace) ←
            ->prefix('api')
            ->group(base_path('routes/api.php'));

        Route::middleware('web')
            ->namespace($this->namespace) ←
            ->group(base_path('routes/web.php'));
    });
}
```

Ahora dentro del controlador anteriormente creado meteremos la función de la ruta que quedaria asi

```
public function index(){
    $products= Product::orderBy('created_at','desc')->get();
    return view('products.index',compact('products'));
}

public function create(){
    $products= Product::orderBy('created_at','desc')->get();
    return view('products.create',compact('products'));
}
```

También es necesario añadir esta línea en el controlador

```
use App\Models\Product;
```

Ahora actualizamos el archivo de rutas para que use el controlador

```
Route::middleware('auth')->group(function(){

    Route::get('products/create','ProductController@create')->name('products.create');
});
Route::get('products','ProductController@index')->name('products.index');
```

Creación de modelos y migración: Para la creación de modelos necesitamos el siguiente comando: php artisan make:model (nombre de la tabla)

Laravel ya crea el modelo de usuarios. Para el tema de las migraciones las crearemos con el siguiente comando: php artisan make:migration create_(nombre de la tabla)_table

Dentro de la migración debemos poner los atributos que queramos que tenga nuestra tabla, en mi caso los siguientes:

```
Schema::create('products', function (Blueprint $table) {
    $table->id();
    $table->text('description');
    $table->integer('price');
    $table->timestamps();
});
```

Se necesita hacer una última configuración para que la migración no falle. Dentro de la carpeta config abrimos el archivo database y en la línea que pone engine después de la flecha ponemos 'InnoDB'. Una vez hecho esto escribimos este comando: php artisan migrate

Esto nos creará las tablas que nosotros hayamos creado

Tabla	Acción						
<input type="checkbox"/> categories	★	Examinar	Estructura	Buscar	Insertar	Vaciar	Eliminar
<input type="checkbox"/> failed_jobs	★	Examinar	Estructura	Buscar	Insertar	Vaciar	Eliminar
<input type="checkbox"/> migrations	★	Examinar	Estructura	Buscar	Insertar	Vaciar	Eliminar
<input type="checkbox"/> password_resets	★	Examinar	Estructura	Buscar	Insertar	Vaciar	Eliminar
<input type="checkbox"/> personal_access_tokens	★	Examinar	Estructura	Buscar	Insertar	Vaciar	Eliminar
<input type="checkbox"/> products	★	Examinar	Estructura	Buscar	Insertar	Vaciar	Eliminar
<input type="checkbox"/> relacion	★	Examinar	Estructura	Buscar	Insertar	Vaciar	Eliminar
<input type="checkbox"/> users	★	Examinar	Estructura	Buscar	Insertar	Vaciar	Eliminar

Sprint 2

CRUD de categorías:

Para la creación de categorías nos harán falta las siguientes rutas.

```
Route::get('admin/categorycreate', 'AdminController@categorycreate')->name('admin.category');
Route::post('admin/categorycreate', 'AdminController@addctg')->name('admin.addctg');
```

Una para ir a la vista de la categoría y la otra para añadirla. A su vez con las rutas nos hacen falta las siguientes funciones en el controlador.

```
public function categorycreate(){
    return view('admin.categorycreate');
}

public function addctg(Request $request){

    // Verificar si la categoría ya existe
    $existingCategory = Category::where('category_n', $request->input('category_n'))->first();
    if ($existingCategory) {
        return redirect()->back()->with('error', 'La categoría ya existe.');
```

En la función para añadir la categoría tenemos primero la verificación para saber si la categoría que vamos a crear ya existe en la base de datos.

Para la edición de la categoría nos volverán a hacer falta otras dos rutas (una para la vista y otra para el update) y sus funciones correspondientes.

```
Route::get('admin/{id}/categoryedit', 'AdminController@Ceditview')->name('admin.categoryedit');
Route::put('admin/{id}', 'AdminController@categoryUpdate')->name('admin.categoryupdate');
```

```
public function Ceditview($id){
    $category= Category::findOrFail($id);
    return view('admin.categoryedit', compact('category'));
}

public function categoryUpdate(Request $request,$id){
    $category= Category::findOrFail($id);
    $category->category_n=$request->input('category_n');
    $category->save();
    return redirect()->route('admin.categorylist');
}
```

Para el listado de las categorias aparte de la ruta y su función

```
Route::get('admin/categorylist', 'AdminController@list')->name('admin.categorylist');
```

```
public function list(){
    $categories= Category::all();
    return view('admin.categorylist', compact('categories'));
}
```

Nos hace falta los siguiente en la vista

```
@foreach($categories as $categorias)
<tr>
    <td>
        {{ $categorias->category_n }}
    </td>
    <td>
        <a href="{{ route('admin.categoryedit', $categorias->id) }}" class="btn btn-warning ">Editar</a>
        <a href="javascript: document.getElementById('delete-{{ $categorias->id }}').submit()" class="btn btn-danger">Eliminar</a>
        <form id="delete-{{ $categorias->id }}" action="{{ route('admin.categorydelete', $categorias->id) }}" method="post">
            @method('delete')
            @csrf
        </form>
    </td>
</tr>
@endforeach
```

Este código se usa para que se muestren todas las categorías y que cada una tenga sus botones de editar y eliminar respectivamente. Para eliminar la categoría usaremos la siguiente función:


```

public function deleteCategory($id)
{
    // Encuentra la categoría o lanza un error si no existe
    $category = Category::findOrFail($id);

    // Comprueba si hay productos asociados a esta categoría
    $productsCount = Product::where('id_category', $id)->count();

    // Si hay productos asociados, redirige con un mensaje de error
    if ($productsCount > 0) {
        return redirect()->route('admin.categorylist')->with('error', 'No se puede eliminar la categoría porque hay productos asociados.');
```

En esta función llevaremos la cuenta de los productos que lleva asociados la categoría y si lleva algún producto no nos dejará eliminarla.

CRUD de productos:

Para el listado, la edición y la eliminación de los productos nos harán falta las siguientes vistas, rutas y funciones.

Listado:

```

<tbody>
    @foreach($products as $productos)
        <tr>
            <td>
                {{ $productos->description }}
            </td>
            <td>
                {{ $productos->price }}
            </td>
            <td>
                <a href="{{ route('admin.productedit', $productos->id) }}" class="btn btn-warning ">Editar</a>
                <a href="javascript: document.getElementById('delete-{{ $productos->id }}').submit()" class="btn btn-danger">Eliminar</a>
                <form id="delete-{{ $productos->id }}" action="{{ route('admin.productdelete', $productos->id) }}" method="post">
                    @method('delete')
                    @csrf
                </form>
            </td>
        </tr>
    @endforeach

```

```

public function productlist(){
    $products= Product::all();
    return view('admin.productlist', compact('products'));
}

```

Editar:

```
public function productedit($id){
    $product= product::findOrFail($id);
    return view('admin.productedit', compact('product'));
}

public function productUpdate(Request $request,$id){
    $product= product::findOrFail($id);
    $product->description=$request->input('description');
    $product->price=$request->input('price');
    $product->save();
    return redirect()->route('admin.productlist');
}
```

```
Route::get('admin/{id}/productedit', 'AdminController@productedit')->name('admin.productedit');
Route::put('admin/{id}', 'AdminController@productUpdate')->name('admin.productupdate');
```

Eliminar:

```
Route::delete('admin/productlist/{id}', 'AdminController@deleteproduct')->name('admin.productdelete');
```

```
public function deleteproduct($id){
    $product=product::findOrFail($id);
    $product->delete();
    return redirect()->route('admin.productlist');
}
```

Creación:

Para la creación de los productos se hace igual pero, al ser necesario la subida de una imagen, la he realizado con la API Spatie Media Library. Para el uso de esta API he seguido el siguiente video: <https://www.youtube.com/watch?v=E7dq7KMgQn0>

(El código se muestra en el apartado código relevante)

Sprint 3

Gestión de anuncios:

Para realizar el crud de los anuncios necesitamos las siguientes vistas, rutas y funciones:

Creación de productos:

```
<body>
  <div class="container">
    <div class="row">
      <div class="col-md-12">
        <div class="card">
          <div class="card-header">
            Nueva Anuncio
          </div>
          <div class="card-body">
            @if(session('error'))
              <div class="alert alert-danger">
                {{session('error')}}
              </div>
            @endif
            <form action="{{ route('admin.addanuncio') }}" method="post">
              @csrf
              <div class="form-group">
                <label for="title">Titulo:</label>
                <input type="text" name="title" id="title" class="form-control">
              </div>
              <div class="form-group">
                <label for="message">Mensaje:</label>
                <input type="text" name="message" id="message" class="form-control">
              </div>
              <div class="form-group">
                <label for="date_start">Fecha inicio:</label>
                <input type="date" name="date_start" id="date_start" class="form-control">
              </div>
              <div class="form-group">
                <label for="date_end">Fecha fin:</label>
                <input type="date" name="date_end" id="date_end" class="form-control">
              </div>
              <button type="submit" class="btn btn-primary">Guardar</button>
              <a href="{{ route('admin.index') }}" class="btn btn-danger">Cancelar</a>
            </form>
          </div>
        </div>
      </div>
    </div>
  </div>
```

```
Route::get('admin/createanuncio', 'AdminController@createanuncio')->name('admin.createanuncio');
Route::post('admin/createanuncio', 'AdminController@addanuncio')->name('admin.addanuncio');
```

```

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

public function addanuncio(Request $request){
    // Guardar nuevo anuncio
    $newAnuncio = new Anuncio();
    $newAnuncio->title = $request->input('title');
    $newAnuncio->message = $request->input('message');
    $newAnuncio->date_start = $request->input('date_start');
    $newAnuncio->date_end = $request->input('date_end');
    $newAnuncio->save();
    $text="<b>Nuevo Anuncio</b>\n\n"
    .$request->input('title')." \n\n". $request->input('message');

    \Telegram::sendMessage([
        'chat_id'=>env('TELEGRAM_CHANNEL_ID',''),
        'parse_mode'=>'HTML',
        'text'=>$text
    ]);
    return redirect()->route('admin.index');
}

```

En la función para añadir el anuncio se utiliza también el bot de telegram(se explicará la configuración y uso en 'código relevante') para mandar las notificaciones.

Listado:

En el listado de los anuncios tendremos también los botones para editar y eliminar el anuncio

```

<table class="table table-hover table-sm" border="1">
  <thead>
    <th>Titulo</th>
    <th>Mensaje</th>
    <th>Fecha inicio</th>
    <th>Fecha fin</th>
  </thead>
  <tbody>
    @foreach($anuncios as $anuncio)
    <tr>
      <td>
        {{ $anuncio->title }}
      </td>
      <td>
        {{ $anuncio->message }}
      </td>
      <td>
        {{ $anuncio->date_start }}
      </td>
      <td>
        {{ $anuncio->date_end }}
      </td>
      <td>
        <a href="{{ route('admin.anuncioedit', $anuncio->id) }}" class="btn btn-warning ">Editar</a>
        <a href="javascript: document.getElementById('delete-{{ $anuncio->id }}').submit()" class="btn btn-danger">Eliminar</a>
        <form id="delete-{{ $anuncio->id }}" action="{{ route('admin.anunciodelete', $anuncio->id) }}" method="post">
          @method('delete')
          @csrf
        </form>
      </td>
    </tr>
    @endforeach
  </tbody>
</table>

```

Editar:

Para editar el anuncio utilizamos las siguientes vistas,rutas y funciones

```

<div class="card-header">
  Editar producto
</div>
<div class="card-body">
  <form action="{{ route('admin.anuncioupdate', $anuncio->id) }}" method="post">
    @method('put')
    @csrf
    <div class="form-group">
      <label for="">Titulo:</label>
      <input type="text" name="title" class="form-control" value="{{ $anuncio->title }}">
    </div>
    <div class="form-group">
      <label for="">Mensaje:</label>
      <input type="text" name="message" class="form-control" value="{{ $anuncio->message }}">
    </div>
    <div class="form-group">
      <label for="">Fecha inicio:</label>
      <input type="date" name="date_start" class="form-control" value="{{ $anuncio->date_start }}">
    </div>
    <div class="form-group">
      <label for="">Fecha fin:</label>
      <input type="date" name="date_end" class="form-control" value="{{ $anuncio->date_end }}">
    </div>
    <button type="submit" class="btn btn-primary">Guardar</button>
    <a href="{{ route('admin.anunciolist') }}" class="btn btn-danger">Cancelar</a>
  </form>
</div>

```

La vista para editar el anuncio nos muestra los datos antiguos de ese mismo anuncio para cambiarlos.

```
// Rutas para editar un producto específico
Route::get('admin/{id}/anuncioedit', 'AdminController@anuncioedit')->name('admin.anuncioedit');
Route::put('admin/{id}', 'AdminController@anuncioUpdate')->name('admin.anuncioupdate');
```

```
// Muestra la vista para editar un producto específico.
public function anuncioedit($id){
    $anuncio= anuncio::findOrFail($id);
    return view('admin.anuncioedit', compact('anuncio'));
}
```

```
// Actualiza un producto existente en la base de datos.
public function anuncioUpdate(Request $request,$id){
    $anuncio= anuncio::findOrFail($id);
    $anuncio->title=$request->input('title');
    $anuncio->message=$request->input('message');
    $anuncio->date_start=$request->input('date_start');
    $anuncio->date_end=$request->input('date_end');
    $anuncio->save();
    return redirect()->route('admin.anunciolist');
}
```

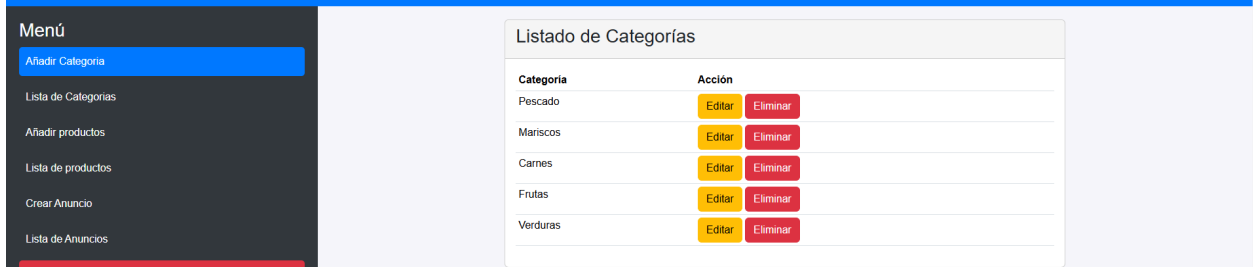
En la función para hacer el update recogemos los datos nuevos del anuncio para actualizarlo.

Eliminar

Para eliminar el anuncio recogemos la id desde el listado el cual nos lleva mediante la ruta a la función para eliminar el anuncio.

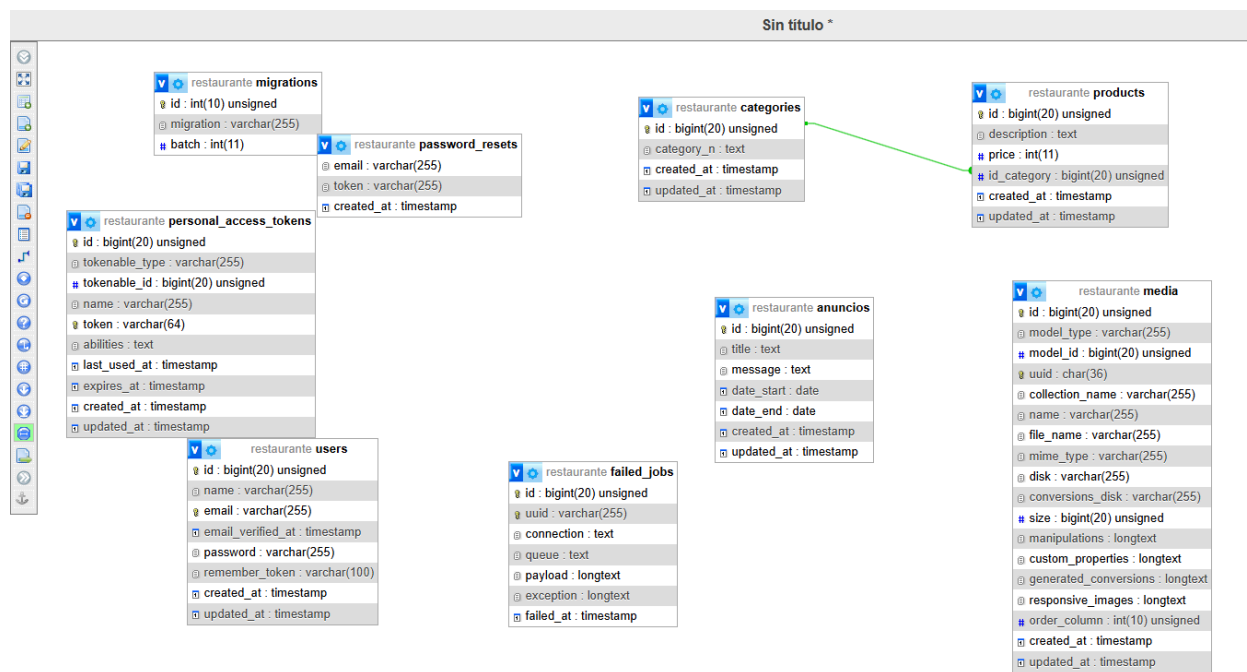
```
Route::delete('admin/anunciolist/{id}', 'AdminController@deleteanuncio')->name('admin.anunciodelete');
```

```
// Elimina un producto de la base de datos.
public function deleteanuncio($id){
    $anuncio=Anuncio::findOrFail($id);
    $anuncio->delete();
    return redirect()->route('admin.anunciolist');
}
```



```
@extends('layouts.main')
@section('contenido')
    <div class="container">
    <div class="card">
        <div class="card-header">
            <h1 class="text-center">Listado de Productos</h1>
        </div>
        <div class="card-body">
            <table class="table table-hover table-sm">
                <thead>
                    <tr>
                        <th>Producto</th>
                        <th>Precio</th>
                        <th>Acciones</th>
                    </tr>
                </thead>
                <tbody>
                    @foreach($products as $productos)
                        <tr>
                            <td>{{ $productos->description }}</td>
                            <td>{{ $productos->price }}</td>
                            <td>
                                <a href="{{ route('admin.productedit', $productos->id) }}" class="btn btn-warning">Editar</a>
                                <a href="javascript: document.getElementById('delete-{{ $productos->id }}').submit()" class="btn btn-danger">Eliminar</a>
                                <form id="delete-{{ $productos->id }}" action="{{ route('admin.productdelete', $productos->id) }}" method="post" style="display: none;"
                                    @method('delete')
                                    @csrf
                                </form>
                            </td>
                        </tr>
                    @endforeach
                </tbody>
            </table>
        </div>
    </div>
@endsection
```

Esquema de Base de Datos



En la base de datos tenemos las tablas para los productos y las categorías las cuales relacionamos con el uso de una tabla de relación.

En la tabla productos tenemos los siguientes atributos: id, descripción del producto, precio del producto y los atributos de creación.

En la tabla categorías utilizamos estos atributos: id, tipo de categoría y los atributos de creación.

La tabla media es creada al hacer la instalación de la API.

En la tabla anuncios utilizamos los siguientes atributos: title(el titulo del anuncio), message(el contenido que lleva el anuncio), date_start(fecha en la que comienza el anuncio) y date_end (fecha en la que termina el anuncio).

Guia de Usuario y Administrador

Guia de usuario

Nada más entrar el usuario a la página se encontrará con la carta del restaurante en la cual aparecen los productos y en la parte superior un buscador para poder filtrar los productos por categorías.

Carta del Restaurante

Buscar por Categoría:

Todas

solomillo

Precio: \$4.00

patatas

Precio: \$1.00

cocacola

Precio: \$1.00

Cuando se le da al desplegable aparecerán las categorías disponibles

Todas

Todas

entrante

bebidas

Y cuando le demos click a una de ellas solo nos mostrará los productos de esa categoría.

Buscar por Categoría:

bebidas

cocacola

Precio: \$1.00

Para poder ver los anuncios del restaurante le daremos click al botón que se encuentra encima del buscador

Listado de Productos

Anuncios

Buscar por Categoría:

pescado

Salmon

Precio: \$5.00

En esa página se nos mostrara todos los anuncios activos

Listado de Anuncios

☐ Filtrar por Fecha Actual

Aplicar Filtro

sehgds
hfgsdgfsdgsfsg
2025-02-09
2025-02-10

Anuncio prueba
Este es un anuncio de prueba
2025-02-11
2025-02-21

También podemos filtrar los anuncios para mostrar los anuncios activos durante el día de hoy.

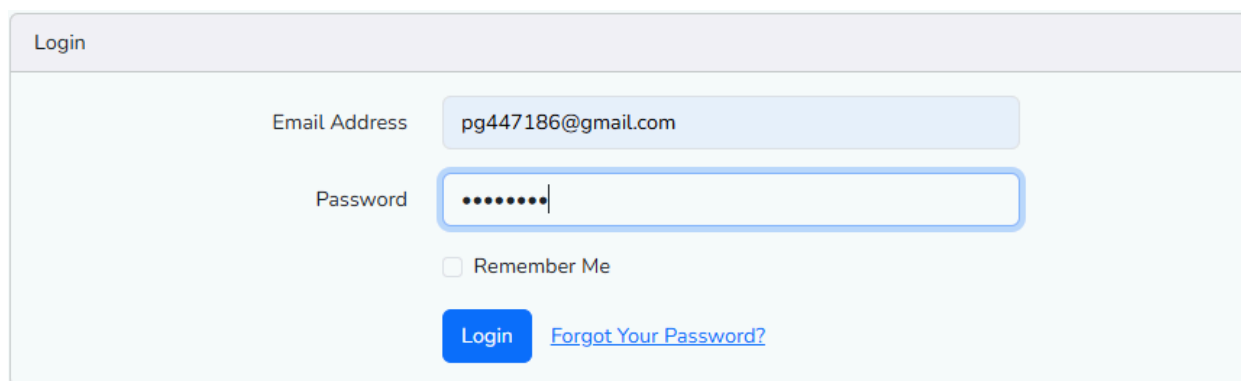
☒ Filtrar por Fecha Actual

Aplicar Filtro

sehgds
hfgsdgfsdgsfsg
2025-02-09
2025-02-10

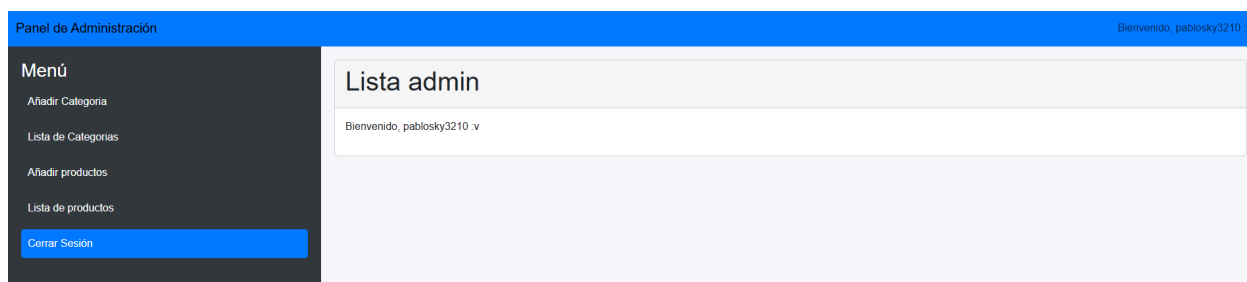
Guía de Administrador

Para la guía del administrador el primer paso es iniciar sesión.



The login form is titled "Login" and is set against a light blue background. It contains two input fields: "Email Address" with the value "pg447186@gmail.com" and "Password" with masked characters ".....". Below the password field is a checkbox labeled "Remember Me". At the bottom, there is a blue "Login" button and a blue link labeled "Forgot Your Password?".

Una vez iniciada la sesión te mandara al panel de administrador



The administrator dashboard has a blue header bar with "Panel de Administración" on the left and "Bienvenido, pablosky3210 v" on the right. A dark sidebar on the left contains a "Menú" with options: "Añadir Categoría", "Lista de Categorías", "Añadir productos", "Lista de productos", and a highlighted "Cerrar Sesión" button. The main content area is titled "Lista admin" and displays a welcome message "Bienvenido, pablosky3210 v".

Donde en la parte izquierda encontrarás el menú con las distintas funcionalidades que tiene el administrador.

Categorías:

Para crear las categorías le daremos en la parte del menú donde pone añadir categoría que nos mandará a la siguiente pantalla.



The "Nueva categoría" form has a light gray header. It features a "Nombre:" label above a text input field. At the bottom, there are two buttons: a blue "Guardar" button and a red "Cancelar" button.

Mediante este formulario podemos añadir una nueva categoría solo introduciendo el nombre y dándole al botón de guardar.

Para la siguiente parte nos meteremos en el apartado que pone lista de categorías donde allí podremos ver todas las categorías y podremos editarlas y eliminarlas.

Listado de categorías	
Categorías	Acción
entrante	<button>Editar</button> <button>Eliminar</button>
bebidas	<button>Editar</button> <button>Eliminar</button>
carne	<button>Editar</button> <button>Eliminar</button>
<button>Volver</button>	

Una vez en esta pantalla podremos editar el nombre de la categoría dándole al botón editar y podremos eliminarla dándole al botón eliminar.

Editar Categoría
<p>Nombre:</p> <input type="text" value="pescado"/>
<div> <div>Guardar</div> <div>Cancelar</div> </div>

Productos:

Para la creación de productos le daremos a la opción añadir productos.

Nuevo Producto
<p>Descripción del Producto:</p> <input type="text"/>
<p>Precio:</p> <input type="text"/>
<p>Categoría:</p> <input type="text" value="entrante"/>
<p>Imagen del Producto:</p> <div> <div> <div>Seleccionar archivo</div> <div>Ningún archivo seleccionado</div> </div> </div>
<div> <div>Guardar</div> <div>Cancelar</div> </div>

Para poder crearlo nos hará falta el nombre del producto, el precio, la categoría la cual se añadirá mediante un desplegable donde estarán todas las categorías y una imagen del producto.

Para los siguientes apartados nos iremos al listado de productos donde desde allí podremos editarlos y eliminarlos.

Listado de productos		
Productos	Precio	Accion
solomillo	4	Editar Eliminar
patatas	1	Editar Eliminar
cocacola	1	Editar Eliminar

[Volver](#)

Al editar el producto podremos cambiar tanto el nombre como el precio.

Editar producto

Nombre:

Precio:

[Guardar](#) [Cancelar](#)

Y para eliminarlos solo es necesario pulsar el botón eliminar.

Anuncios:

Para crear un anuncio le daremos a la opcion crear anuncio desde la pagina inicial.

Panel de Administración
 Bienvenido, pablosky3210 v.

Menú

- Añadir Categoría
- Lista de Categorías
- Añadir productos
- Lista de productos
- Crear Anuncio
- Lista de Anuncios
- Cerrar Sesión

Lista admin

Bienvenido, pablosky3210 v

Una vez allí tendremos que introducir el título del anuncio, la descripción y las fechas de inicio y de fin.

Nueva Anuncio

Titulo:

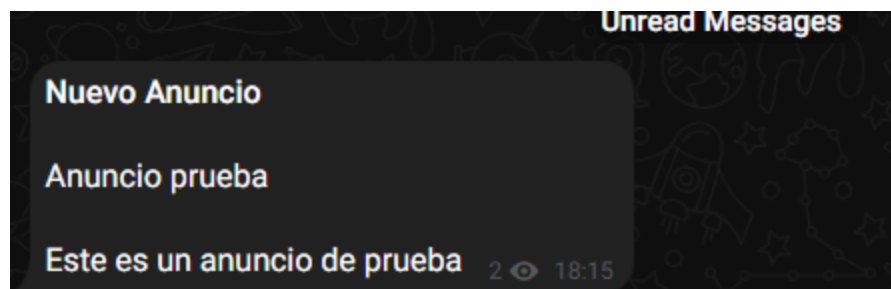
Mensaje:

Fecha inicio:

Fecha fin:

[Guardar](#) [Cancelar](#)

Nada más crear el anuncio nos mandará una notificación por telegram con el título y la descripción del anuncio.



Código Relevante

Spatie Media Library:

Primero instalamos la API: `composer require "spatie/laravel-medialibrary"`

Generamos y ejecutamos la migración: `php artisan vendor:publish --provider="Spatie\MediaLibrary\MediaLibraryServiceProvider"`

`php artisan migrate`

Generamos el archivo de configuración: `php artisan vendor:publish --provider="Spatie\MediaLibrary\MediaLibraryServiceProvider"`

:

En el modelo añadimos lo siguiente:

```
class Product extends Model implements HasMedia
{
    use InteractsWithMedia;

    // Personalizar la ruta de almacenamiento para evitar subcarpetas
    public function getMediaPathAttribute(): string
    {
        return public_path('uploads') . '/' . $this->getFirstMedia('images')->file_name;
    }

    protected $fillable = [
        'description',
        'price',
        'id_category',
    ];

    public function category()
    {
        return $this->belongsTo(Category::class, 'id_category');
    }
}
```

```
use Spatie\MediaLibrary\HasMedia;
use Spatie\MediaLibrary\InteractsWithMedia;
```

En el controlador dejamos la función de la siguiente manera:

```
public function addproduct(Request $request)
{
    $request->validate([
        'description' => 'required|string|max:255',
        'price' => 'required|numeric',
        'id_category' => 'required|exists:categories,id',
        'image' => 'required|image|mimes:jpg,png,jpeg,gif|max:2048',
    ]);

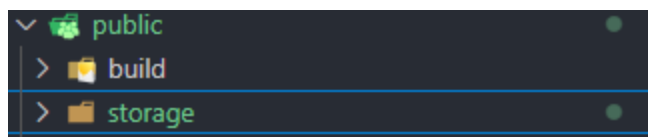
    // Crear el producto
    $product = Product::create($request->only('description', 'price', 'id_category'));

    if ($request->hasFile('image')) {
        $product->addMedia($request->file('image'))
            ->usingFileName($request->file('image')->getClientOriginalName())
            ->toMediaCollection('images', 'public');
    }
}
```

En la carpeta config entramos en filesystems y cambiamos lo siguiente:

```
'public' => [
    'driver' => 'local',
    'root' => public_path('storage'),
    'url' => env('APP_URL').'/storage',
    'visibility' => 'public',
],
```

En la carpeta public crearemos una carpeta donde queramos que se guarden las imágenes que subamos:

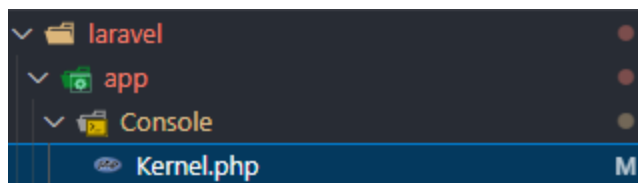


Laravel Scheduler:

Para el uso de laravel scheduler he seguido el siguiente video

<https://www.youtube.com/watch?v=UEsjzOvOWao>

Para la utilización del scheduler entraremos en el siguiente archivo



Dentro del archivo kernel introduciremos lo siguiente dentro de la función schedule

```
$schedule->call(function(){
    $fechaActual= now();
    Anuncio::where('date_end','<',$fechaActual)->delete();
})->everyMinute();
```

Con esta función controlo que los anuncios los cuales su fecha de fin es anterior a la fecha actual sean eliminados.

Para que se ponga en marcha el schedule necesitamos el siguiente comando en el powershell

```
php artisan schedule:work
```

Telegram:

Para enviar los mensajes por un bot en telegram he seguido las diapositivas dadas por el profesor.

Suponiendo que ya se ha creado el bot mediante BotFather en telegram escribiremos estos comandos para la utilización de la Api.

```
composer require guzzlehttp/psr7
```

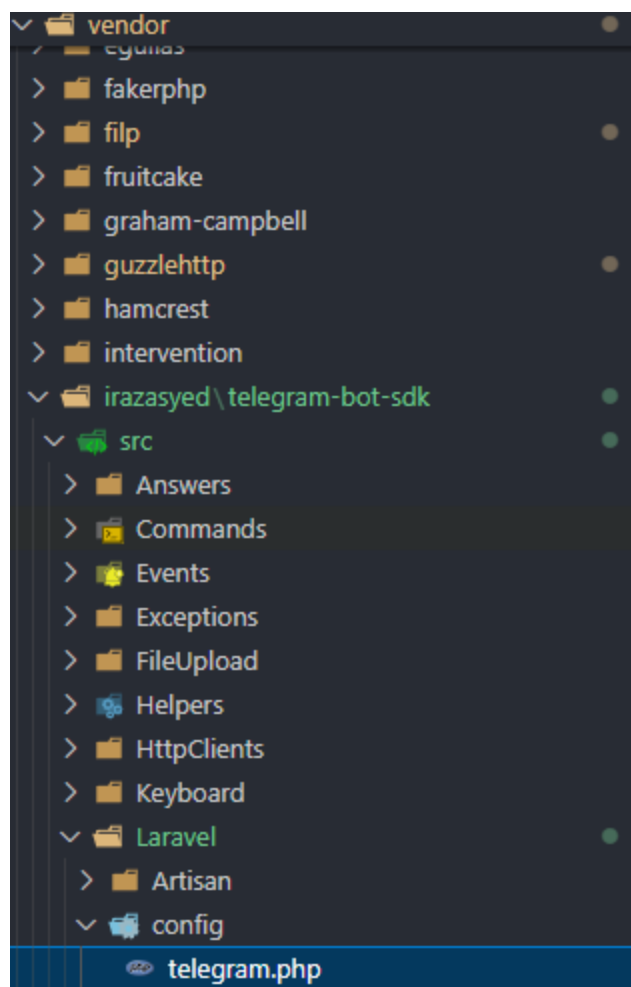
```
composer require irazasyed/telegram-bot-sdk
```

Una vez escritos los comandos entraremos en el archivo env y escribiremos lo siguiente

```
TELEGRAM_BOT_TOKEN=8191788129:AAHMi4v2Mq0aeYUO-knneRKcqpVcouxVKw
```

Una vez hecho eso, creamos un canal de telegram con el bot.

Nos metemos en el archivo telegram.php



y lo dejamos de la siguiente manera

```
'bots' => [
    'mybot' => [
        'username' => 'VegaRestauranteBot',
        'token' => env('TELEGRAM_BOT_TOKEN', ''),
        'certificate_path' => env('TELEGRAM_CERTIFICATE_PATH', 'YOUR-CERTIFICATE-PATH'),
        'webhook_url' => env('TELEGRAM_WEBHOOK_URL', 'YOUR-BOT-WEBHOOK-URL'),
        /*
         * @see https://core.telegram.org/bots/api#update
         */
        'allowed_updates' => null,
        'commands' => [
            //Acme\Project\Commands\MyTelegramBot\BotCommand::class
        ],
    ],
],
```

Necesitaremos una ruta y función nueva para comprobar los mensajes nuevos y así obtener la ip del canal.

Una vez obtenida la ip del canal lo introduciremos en el archivo env

```
TELEGRAM_CHANNEL_ID=-1002418805857
```

Después de configurar el archivo env escribiremos este comando

```
php artisan config:cache
```

Una vez hecho eso ponemos lo siguiente en la función para crear un anuncio la cual nos enviará la notificación nada más crear el anuncio.

```
public function addanuncio(Request $request){
    // Guardar nuevo anuncio
    $newAnuncio = new Anuncio();
    $newAnuncio->title = $request->input('title');
    $newAnuncio->message = $request->input('message');
    $newAnuncio->date_start = $request->input('date_start');
    $newAnuncio->date_end = $request->input('date_end');
    $newAnuncio->save();
    $text="<b>Nuevo Anuncio</b>\n\n"
    .$request->input('title')." \n\n". $request->input('message');

    \Telegram::sendMessage([
        'chat_id'=>env('TELEGRAM_CHANNEL_ID',''),
        'parse_mode'=>'HTML',
        'text'=>$text
    ]);
    return redirect()->route('admin.index');
}
```

E introducimos los siguientes comandos para limpiar el cache

```
php artisan config:cache
```

```
php artisan cache:clear
```

Conclusiones

En conclusión, para mí este proyecto ha sido un proyecto interesante con el que me ha quedado bastante claro el uso base de laravel y saber donde se ubica cada archivo, sabiendo separar las rutas de controladores.

Propuestas de Mejora

Para próximas actualizaciones de este proyecto yo añadiría la posibilidad de que los clientes pudieran realizar los pedidos desde la vista pública ingresando únicamente el número de su mesa.

Dificultades encontradas

Relación de las tablas: Tuve problemas al hacer la relación de las tablas products y categories al querer hacer la relación directamente haciendo la foreign key en la tabla products. Este problema lo arregle realizando primero la migración de la tabla categorías y después la migración de la tabla products.

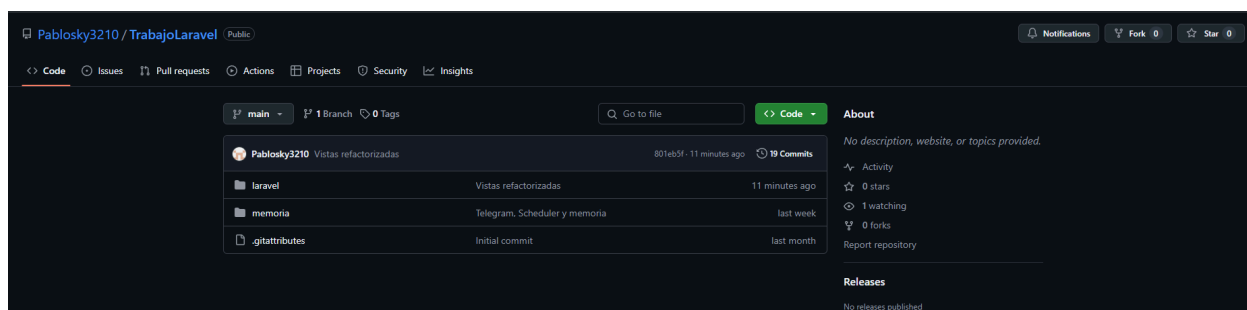
Migrantes: Al hacer los migrantes en clase no podía hacer migrantes por tener una versión anterior a la que usaba en casa. Lo solucione reinstalando XAMPP a la versión 8.2 en clase.

Mostrado de imágenes: Al querer mostrar las imagenes he tenido problemas ya que no podía mostrarlas, pero con la implementación de código en la vista pública y un cambio al subir las imagenes las consigo mostrar, salvo por un detalle que es cuando generas los productos mediante los seeders en los cuales no he conseguido que se muestre ninguna imagen.

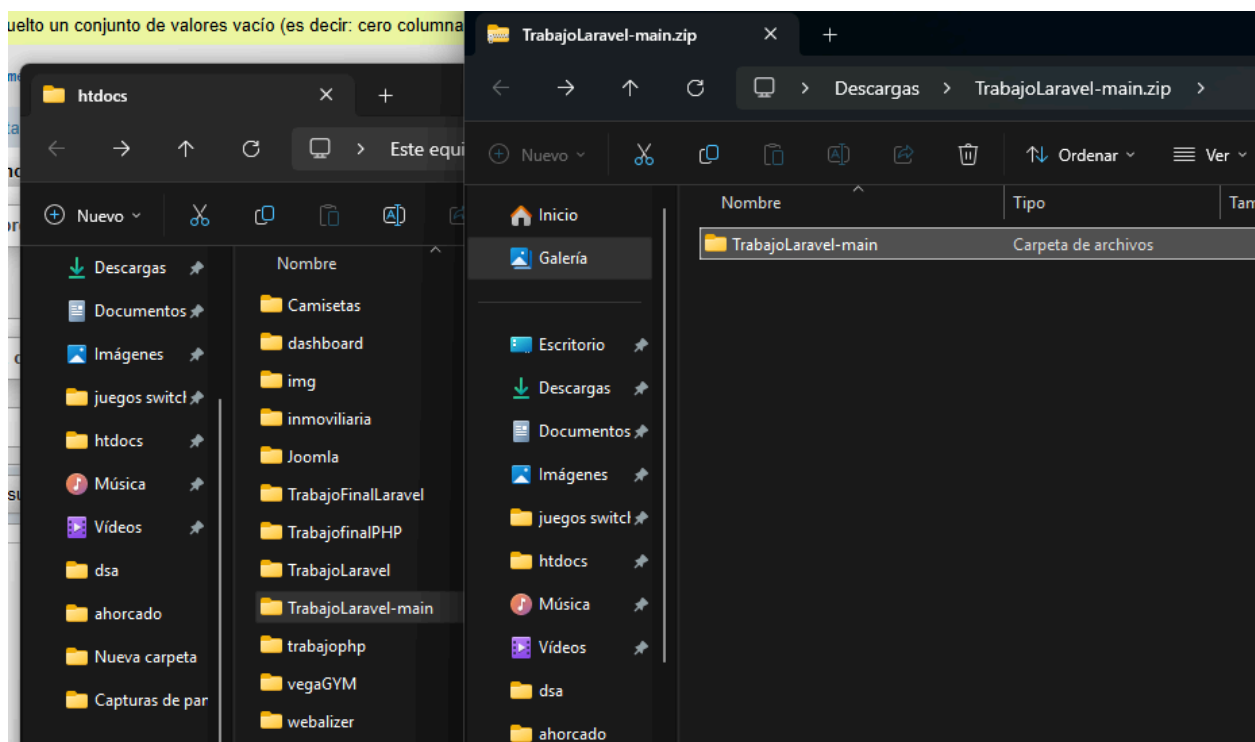
Anexos

Manual de instalación:

Primero de todo entramos a github y nos descargamos el proyecto



Una vez descargado extraemos el zip en nuestra carpeta de htdocs



Luego entramos en la carpeta laravel y abrimos powershell e introducimos en siguiente código

```
php artisan migrate:refresh --seed
```

```

Windows PowerShell
php:70
PDO::__construct("mysql:host=127.0.0.1;port=3306;dbname=restaurante", "root", Object(SensitiveParameterValue), [])
PS C:\xampp\htdocs\TrabajoLaravel-main\laravel> php artisan migrate:refresh --seed

WARN Migration table not found.

INFO Preparing database.

Creating migration table ..... 8ms DONE

INFO Running migrations.

2014_10_12_000000_create_users_table ..... 27ms DONE
2014_10_12_100000_create_password_resets_table ..... 17ms DONE
2019_08_19_000000_create_failed_jobs_table ..... 13ms DONE
2019_12_14_000001_create_personal_access_tokens_table ..... 22ms DONE
2025_01_21_171431_create_categories_table ..... 4ms DONE
2025_01_30_074151_create_products_table ..... 31ms DONE
2025_01_30_164243_create_media_table ..... 31ms DONE
2025_02_05_170651_create_anuncios_table ..... 4ms DONE

INFO Seeding database.

Database\Seeders\CategorySeeder ..... RUNNING
Database\Seeders\CategorySeeder ..... 15.56 ms DONE

Database\Seeders\ProductsSeeder ..... RUNNING
Database\Seeders\ProductsSeeder ..... 12.88 ms DONE

PS C:\xampp\htdocs\TrabajoLaravel-main\laravel>

```

Ese código nos sirve para realizar las migraciones e introducir los seeders poblando la base de datos.