

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.

Funcionalidades

(Se añadirá en los siguientes Sprints)

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 los productos 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 se hará como con las categorías.

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:

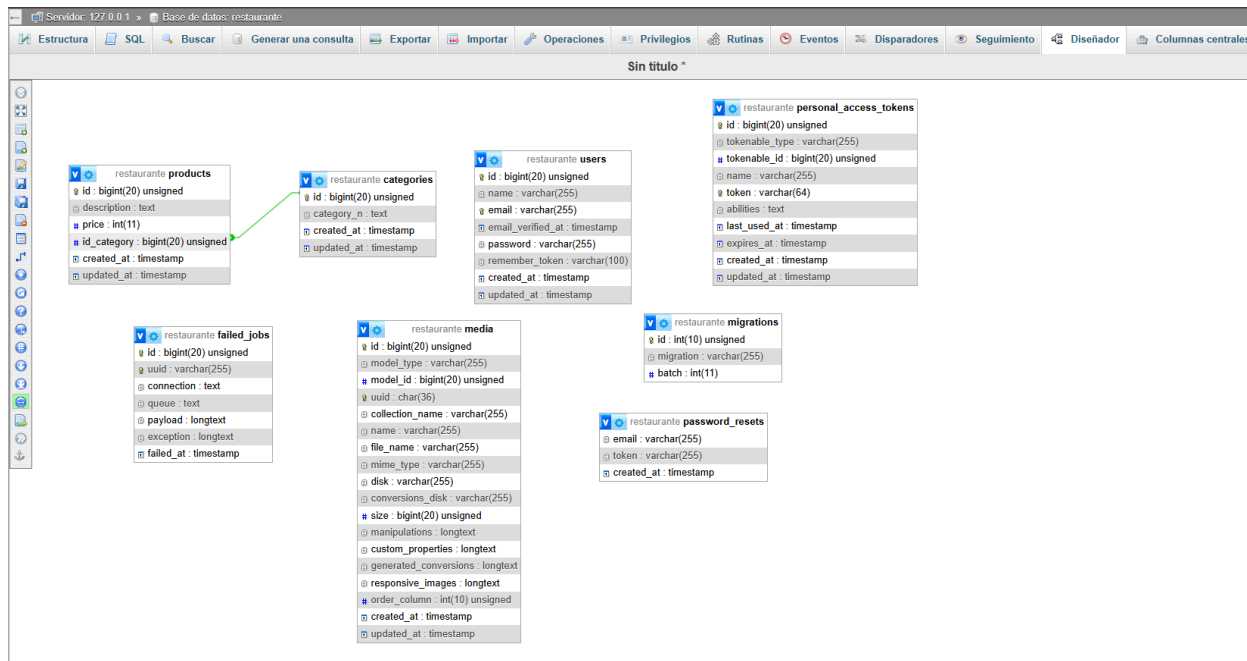
Para la creación de los productos se hace igual pero, al ser necesario la subida de una imagen, la he realizado con la API Spatie Media Library. Para el uso de esta API he seguido el siguiente video: <https://www.youtube.com/watch?v=E7dq7KMgQn0>

(El código se muestra en el apartado código relevante)

Sprint 3

Sprint 4

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.

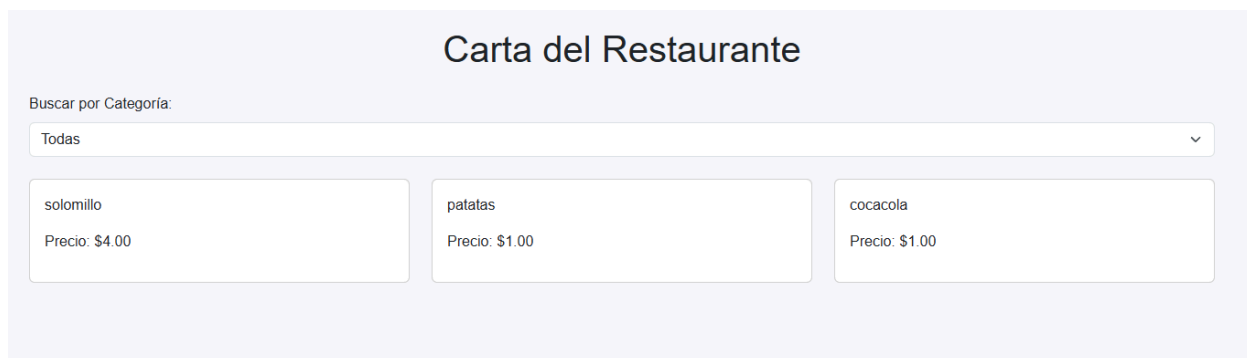
La tabla media es creada al hacer la instalación de la API.

(Falta la tabla de las anuncios para el uso de la API)

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:

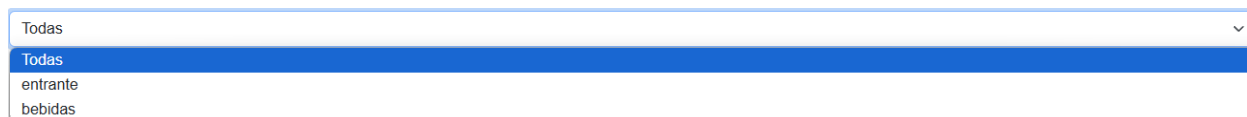
Todas

solomillo
Precio: \$4.00

patatas
Precio: \$1.00

cocacola
Precio: \$1.00

Cuando se le da al desplegable aparecerán las categorías disponibles



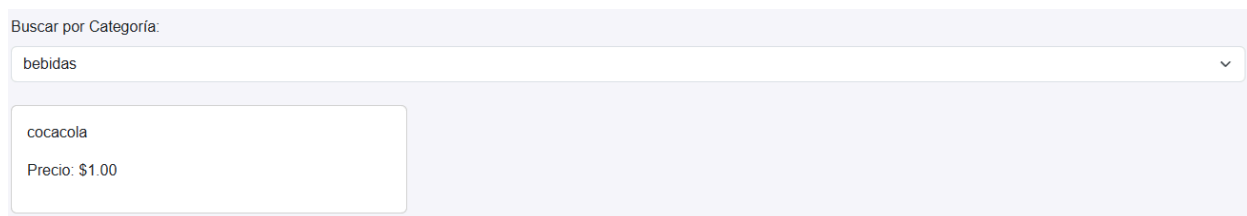
Todas

Todas

entrante

bebidas

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



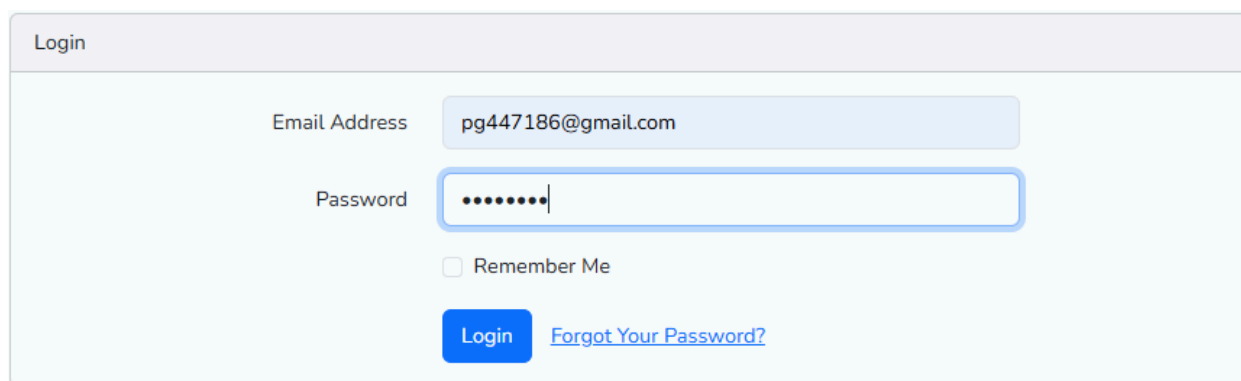
Buscar por Categoría:

bebidas

cocacola
Precio: \$1.00

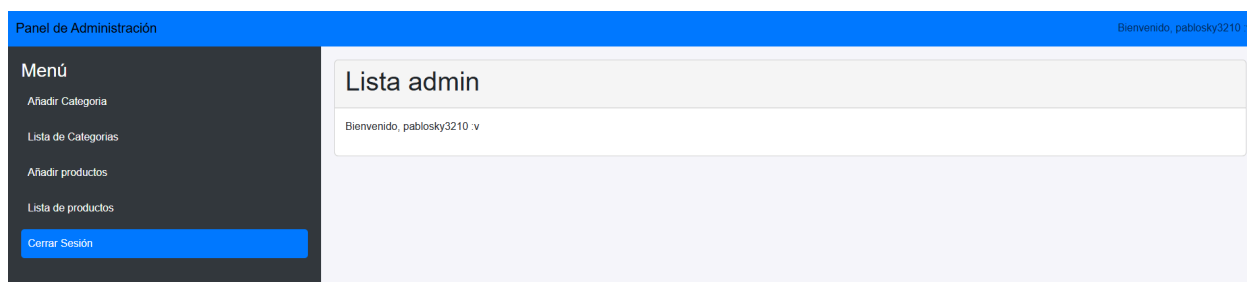
Guía de Administrador

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



The login form is titled "Login" and is contained within a light blue box. It features 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

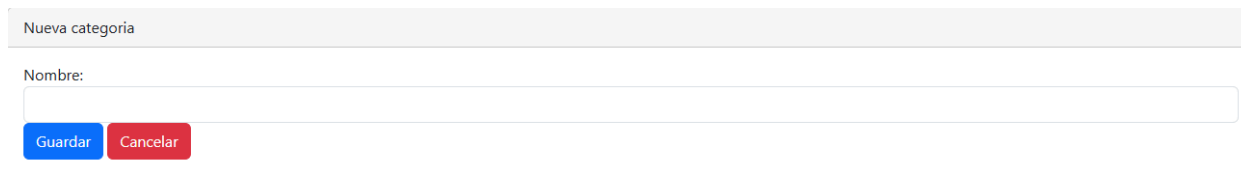


The administrator panel has a blue header bar with "Panel de Administración" on the left and "Bienvenido, pablosky3210 v" on the right. A dark sidebar on the left contains a "Menú" with options: "Añadir Categoría", "Lista de Categorías", "Añadir productos", "Lista de productos", and a highlighted "Cerrar 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 is a light gray box. It contains a label "Nombre:" followed by 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 categorias		
Categorias	Accion	
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.

Productos:

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

Nuevo Producto

Descripción del Producto:

Precio:

Categoría:

Imagen del Producto:

Seleccionar archivo

Ningún archivo seleccionado

Guardar

Cancelar

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	<button>Editar</button> <button>Eliminar</button>
patatas	1	<button>Editar</button> <button>Eliminar</button>
cocacola	1	<button>Editar</button> <button>Eliminar</button>
<button>Volver</button>		

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:



Conclusiones

Propuestas de Mejora

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.

Anexos