

MacJava

Creado por Oscar Encabo, Raul Rodriguez , Diego Torres y Jaime Lozano

1.Introduccion

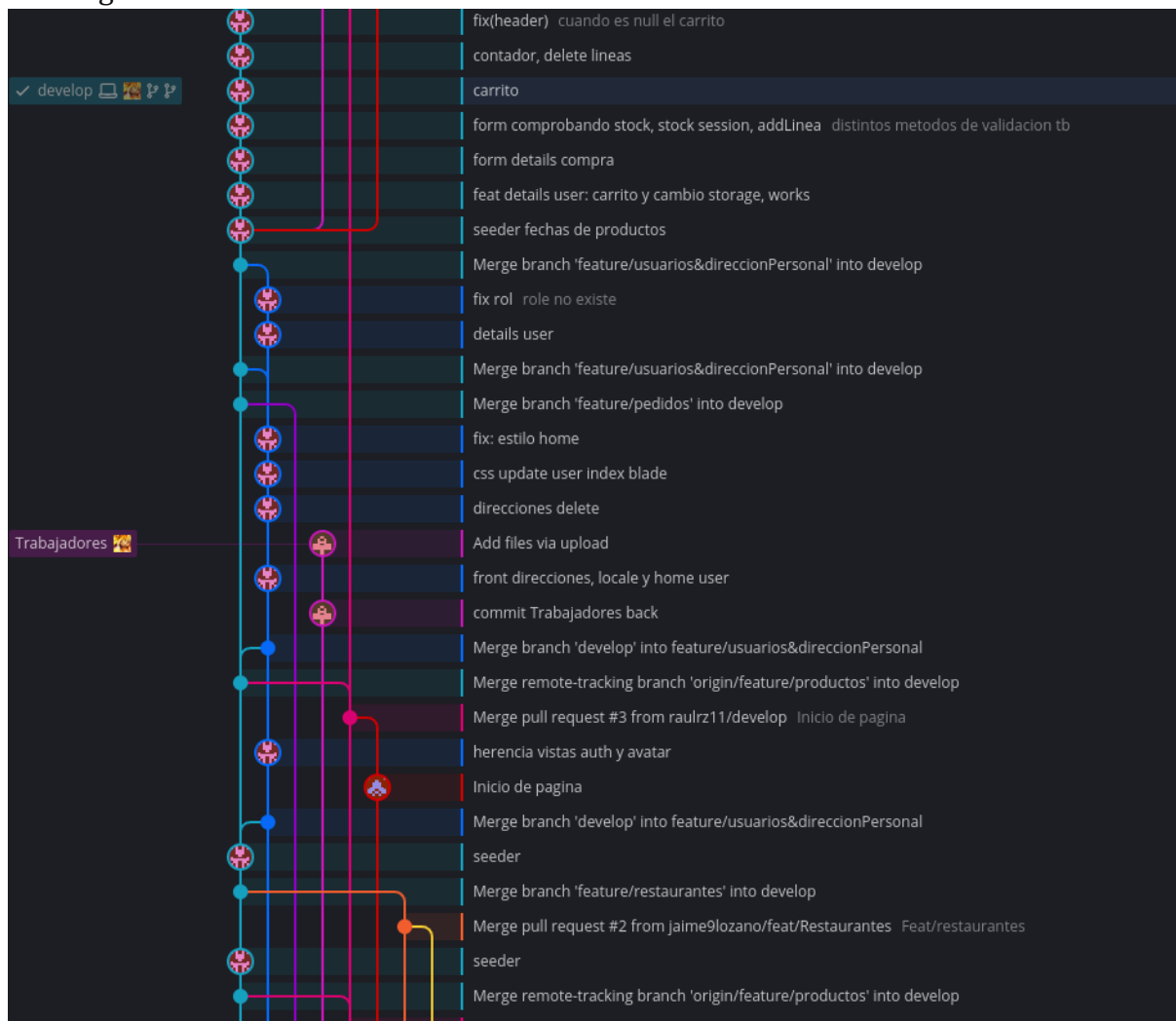
Bienvenido a nuestra API desarrollada por Jaime Lozano, Diego Torres, Oscar Encabo y Raul Rodriguez, que ofrece una gestión segura, completa y escalable de base de datos. MacJava, nuestra API, proporciona un conjunto de endpoints robustos que simplifican la administración de una tienda en línea, abarcando operaciones relacionadas con productos, pedidos, empleados, restaurantes, ubicaciones de los empleados y clientes. Elegimos Laravel por su elegante sintaxis y porque facilita las tareas comunes del desarrollo web, como la autenticación, el enrutamiento, las sesiones y el caché. Laravel se destaca por su capacidad para hacer el desarrollo más rápido y eficiente, ofreciendo una estructura de proyecto clara, un sistema de migración de base de datos potente y un ORM Eloquent para un manejo de datos relacional sencillo. Utiliza un enfoque de "convención sobre configuración", minimizando así la necesidad de una configuración extensiva. Además, Laravel facilita el desarrollo ágil al proporcionar herramientas integradas para tareas comunes, permitiendo a los desarrolladores enfocarse en las características únicas de sus proyectos sin preocuparse por la integración de componentes o configuraciones manuales.

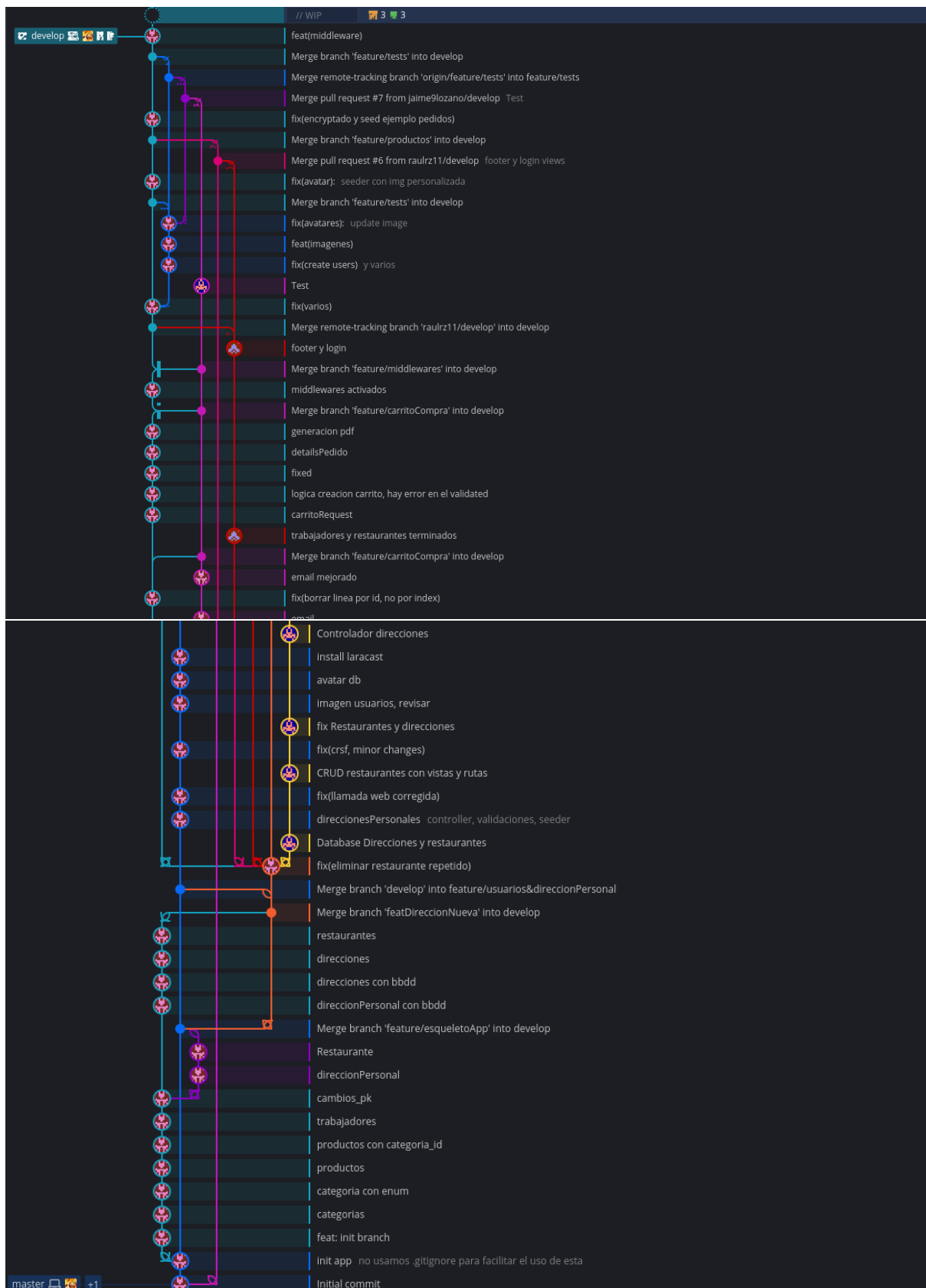
2.GitFlow

Siguiendo un esquema de organización eficaz, creamos una rama principal llamada develop, que sirvió como base para integrar otras ramas denominadas feature, cada una representando una funcionalidad específica del proyecto. Esto nos permitió mantener dos ramas estables y seguras, protegiendo el código verificado y correcto. Durante el desarrollo del proyecto, trabajamos en paralelo, con cada integrante enfocado en desarrollar distintas funcionalidades. Una vez que estas eran probadas y verificadas, se incorporaban a la rama develop para su revisión conjunta.

Casi al finalizar el proyecto, enfrentamos algunos inconvenientes con el código y ciertas clases, pero logramos solucionar estos problemas gracias al uso de ramas auxiliares denominadas fix. Finalmente, cuando consideramos que la aplicación estaba lista y funcionaba correctamente, realizamos el merge hacia la rama master. Sin embargo, detectamos algunos errores posteriores que fueron corregidos mediante parches de urgencia (hotfix).

A continuación, te presentaríamos algunas capturas de nuestro árbol de Git para ilustrar esta organización





3.Dependencias

En nuestro proyecto Laravel, hemos decidido utilizar **Laravel Sail** y **Redis** por varias razones clave que se alinean con nuestros objetivos de eficiencia, escalabilidad y simplicidad en el desarrollo.

Laravel Sail es una solución de desarrollo ligera diseñada para ejecutar aplicaciones Laravel utilizando Docker. Al elegir Sail, buscamos simplificar el proceso de configuración y gestión del entorno de desarrollo, asegurándonos de que todos los miembros del equipo trabajen en un entorno uniforme que imite fielmente nuestra configuración de producción. Esto elimina los problemas comunes de "funciona en mi máquina", facilitando una transición suave desde el desarrollo hasta la producción.

Por otro lado, hemos integrado **Redis** en nuestro proyecto como nuestro almacén de datos en memoria para las operaciones de caché y las colas de trabajos. Elegimos Redis debido a su excepcional rendimiento, confiabilidad y soporte para estructuras de datos complejas como strings, hashes, listas, sets y sorted sets. Utilizar Redis nos permite reducir significativamente los tiempos de respuesta de nuestra aplicación al almacenar datos frecuentemente accedidos en la caché.

4. Características principales de nuestros endpoints

Nuestros endpoints juegan un papel crucial en la interacción entre la interfaz de usuario y la base de datos de nuestra aplicación Laravel, facilitando una amplia gama de operaciones relacionadas con el manejo de trabajadores, productos, y categorías. Estas son las características principales de algunos de nuestros endpoints más importantes:

Productos:

- **Creacion de producto:** Permite a los administradores agregar nuevos productos al sistema, especificando detalles como nombre, descripción, precio, y cantidad en stock.
- **Mostrar producto:** Facilita la visualización de todos los productos disponibles en la plataforma, pudiendo incluir filtros por categoría, precio, o popularidad.
- **Update de producto:** Usuarios autorizados pueden actualizar la información de un producto existente, como modificar su precio, descripción, o cantidad en stock.
- **Update de la imagen del producto:** Permite la actualización y edición de la imagen del producto asociado

Destroy: Permite remover productos del catálogo, generalmente reservado para administradores o usuarios con roles específicos.

```

<?php
use ...

return new class extends Migration {
    // Diego +1
    public function up(): void
    {
        Schema::create( table: 'productos', function (Blueprint $table) {
            $table->id();
            $table->string( column: 'nombre' )->unique( indexName: 'productos_nombre_unico');
            $table->string( column: 'descripcion');
            $table->string( column: 'imagen' )->default(Producto::$IMAGE_DEFAULT);
            $table->integer( column: 'stock' )->default( value: 0);
            $table->decimal( column: 'precio' )->default( value: 0);
            $table->boolean( column: 'oferta' )->default( value: false);
            $table->foreignId( column: 'categoria_id' )->constrained( table: 'categorias' )->onDelete( action: 'cascade');
            $table->softDeletes();
            $table->timestamps();
        });
    }

    // Diego
    public function down(): void
    {
        Schema::dropIfExists( table: 'productos');
    }
};

```

Categorías:

- **Creación de categoría:** Habilita la adición de nuevas categorías al sistema, para una mejor organización de los productos según su tipo o función.
- **Listado de categoría:** Proporciona una lista de todas las categorías existentes, ayudando a los usuarios a navegar y filtrar productos con mayor eficacia.
- **Update de categoría:** Permite modificar los detalles de una categoría, como su nombre o descripción.
- **Destroy:** Facilita la gestión de categorías, permitiendo su eliminación cuando sea necesario.

```

<?php
use ...

return new class extends Migration {
    ⤴ Diego
    public function up(): void
    {
        Schema::create( table: 'categorias', function (Blueprint $table) {
            $table->id();
            $table->string( column: 'nombre' )->unique( indexName: 'categoria_unique_name' );
            $table->softDeletes();
            $table->timestamps();
        });
    }

    ⤴ Diego
    public function down(): void
    {
        Schema::dropIfExists( table: 'categorias' );
    }
};

```

Usuarios

- **Registro:** Permite a los nuevos usuarios crear una cuenta en la plataforma, proporcionando datos personales y de contacto.
- **Autenticación:** Gestiona el inicio de sesión de los usuarios, verificando credenciales y proporcionando tokens de acceso para sesiones seguras.
- **Actualización de Perfil:** Usuarios pueden actualizar su información personal, como correo electrónico, contraseña, y datos de contacto.
- **Eliminación de Cuenta:** Permite a los usuarios eliminar su cuenta, eliminando sus datos personales de la base de datos de forma segura.

Metodos principales:

create

- **Descripción:** Prepara y muestra un formulario para la creación de un nuevo recurso, como un producto o categoría. En una API REST, este método podría no ser implementado directamente, pero conceptualmente representa la acción de preparar la creación de un recurso.
- **Parámetros:** No aplicable para APIs REST, ya que este paso es generalmente manejado en el frontend.
- **Retorno:** Vista o formulario para la creación del recurso.

store

- **Descripción:** Guarda un nuevo recurso en la base de datos.
- **Parámetros: request:** Contiene la información del nuevo recurso a guardar.
- **Retorno:** Objeto guardado con su información persistente.

- **Excepción:** Validación fallida o error de guardado.
- show**
 - **Descripción:** Recupera un recurso por su ID y lo muestra.
- **Parámetro:** **id** del recurso a buscar.
- **Retorno:** Recurso con el ID especificado.
- **Excepción:** NotFound si el recurso no se encuentra.
- edit**
 - **Descripción:** Prepara y muestra un formulario para editar un recurso existente, basado en su ID.
- **Parámetro:** **id** del recurso a editar.
- **Retorno:** Vista o formulario prellenado con la información del recurso para su edición.
- **Excepción:** NotFound si el recurso no se encuentra.
- update**
 - **Descripción:** Actualiza un recurso existente en la base de datos, según su ID.
- **Parámetros:**
 - **id:** ID del recurso a actualizar.
 - **request:** Contiene la información actualizada del recurso.
- **Retorno:** Objeto actualizado.
- **Excepción:** NotFound si el recurso no se encuentra, o validación fallida.
- destroy**
 - **Descripción:** Elimina (o realiza un borrado lógico de) un recurso específico, cambiando su estado o removiéndolo de la base de datos.
- **Parámetro:** **id** del recurso a eliminar.
- **Retorno:** Confirmación de la eliminación o cambio de estado.
- **Excepción:** NotFound si el recurso no se encuentra.

Seguridad y autenticación:

Este servicio gestiona las operaciones fundamentales de autenticación: registrarse y loguearse. Ambas acciones se llevan a cabo a través de métodos definidos en el controlador de autenticación, asegurando una interacción clara y segura con el sistema.

- **Registrarse:** Permite a los nuevos usuarios crear una cuenta en la plataforma, proporcionando información necesaria como nombre de usuario, correo electrónico y contraseña. Este proceso incluye validaciones para asegurar que los datos sean correctos y únicos.
- **Iniciar Sesión:** Facilita el acceso de usuarios registrados, verificando sus credenciales (correo electrónico/usuario y contraseña). Si la autenticación es exitosa, el usuario recibe acceso a la plataforma.

Roles:

- **Administradores:** Tienen acceso completo a todas las funcionalidades relacionadas con los usuarios. Esto incluye la capacidad de ver todos los usuarios registrados en el sistema, crear nuevos usuarios, modificar información de usuarios existentes y eliminar usuarios.

- **Usuarios Comunes:** Pueden interactuar solo con su propia información. Esto abarca ver su perfil, actualizarlo, y eliminar su cuenta. Además, tienen acceso a gestionar sus pedidos, incluyendo la visualización de todos sus pedidos, la creación y actualización de pedidos, y la eliminación de un pedido en particular.

Hemos añadido un nuevo usuario para este proyecto, que será el rol de empleados, a continuación mostraremos las posibilidades de estos:

Rol de Empleado

- **Funcionalidades Básicas Compartidas:** Al igual que los usuarios comunes, los empleados pueden ver y editar su perfil personal, lo que incluye cambiar su contraseña, actualizar su dirección de correo electrónico, y modificar otros detalles personales. También pueden eliminar su cuenta si así lo desean.
- **Gestión de Información de Empleados:** Una diferencia clave es la capacidad de los empleados para acceder y gestionar información específica de empleados, como su horario de trabajo, historial de tareas asignadas, y posiblemente sus evaluaciones de desempeño o información sobre capacitaciones. Esto requiere que el sistema diferencie claramente entre información de usuario general y datos específicos de empleados.
- **Acceso Restringido:** Aunque los empleados pueden tener acceso a funciones adicionales en comparación con los usuarios comunes, este acceso sigue estando restringido en función de las políticas de la empresa y los permisos establecidos en el sistema. Por ejemplo, un empleado podría tener acceso a herramientas internas o aplicaciones específicas para su trabajo que no están disponibles para usuarios generales.


```
DB::table('users')->insert([
    [
        'name' => 'admin',
        'email' => 'admin@admin.es',
        'password' => bcrypt( value: 'Admin2024'),
        'rol' => 'ADMIN',
        'avatar'=>'8e37c6fa-1667-498e-ac63-d06de4d52f83.webp',
        'email_verified_at' => now(),
        'created_at' => now(),
        'updated_at' => now(),
    ],
    [
        'name' => 'empleado',
        'email' => 'empleado@macjava.es',
        'password' => bcrypt( value: 'empleado'),
        'rol' => 'USER',
        'avatar'=>User::$AVATAR_DEFAULT,
        'email_verified_at' => now(),
        'created_at' => now(),
        'updated_at' => now(),
    ],
    [
        'name' => 'user',
        'email' => 'user@user.es',
        'password' => bcrypt( value: 'User2024'),
        'rol' => 'USER',
        'avatar'=>User::$AVATAR_DEFAULT,
        'email_verified_at' => now(),
        'created_at' => now(),
        'updated_at' => now(),
    ],
],
```

Pedidos:

Este es el endpoint que acumula y muestra los detalles finales de la compra de productos:

Teniendo en cuenta eso para un pedido pedimos, los detalles de los productos, las direccion , restaurante, usuario y demas para a continuacion añadirlo a nuestro carrito

```

public function index()
{
    return PedidoResource::collection(Pedido::all());
}

± Diego
public function store(PedidoRequest $request)
{
    return new PedidoResource(Pedido::create($request->validated()));
}

± Diego
public function show($id)
{
    $pedido = $this->getById($id);
    $pedidoResource = new PedidoResource($pedido);

    return view('pedidos.show')->with('pedido', $pedidoResource->data());
}

± Diego
public function update(PedidoRequest $request, Pedido $pedido)
{
    $pedido->update($request->validated());

    return new PedidoResource($pedido);
}

```

Carrito:

El carrito de compras juega un papel crucial en la experiencia de compra en línea, actuando como un puente entre la selección de productos y el proceso de checkout. Una vez que los usuarios han terminado de seleccionar productos, el carrito de compras ofrece una vista previa detallada de su selección antes de proceder con la compra. A continuación, se describen los componentes y funcionalidades clave del carrito de compras en nuestro sistema:

Vista General del Carrito de Compras



- **Listado de Productos Seleccionados:** Muestra todos los productos que el usuario ha agregado al carrito, incluyendo información esencial como el nombre del producto, la cantidad seleccionada, el precio unitario y el precio total por producto. Esto permite a los usuarios revisar su elección antes de realizar la compra.
- **Actualización de Cantidad:** Los usuarios pueden modificar la cantidad de cada producto directamente desde el carrito de compras. Esto recalcula automáticamente el precio total por producto y el subtotal del carrito, asegurando que los usuarios tengan control total sobre su pedido.
- **Eliminación de Productos:** Si un usuario decide no comprar un artículo, puede eliminarlo fácilmente de su carrito. Esta acción actualiza instantáneamente la lista de productos y el total del carrito, manteniendo la precisión de la selección del usuario.

Resumen del Pedido

- **Subtotal:** Calcula el costo total de todos los productos en el carrito sin incluir impuestos adicionales o costos de envío. Este subtotal ofrece a los usuarios una estimación inicial del monto de su compra.

- **Cálculo de Impuestos y Envío:** Dependiendo de la ubicación del usuario y de las políticas del sistema, el carrito puede calcular automáticamente los impuestos aplicables y los costos de envío, proporcionando así una visión completa del costo total del pedido.
- **Total Final:** Presenta el monto total a pagar, incluyendo el subtotal del carrito, impuestos y costos de envío. Este total final permite a los usuarios ver el costo completo de su pedido antes de proceder al pago.

Carrito de Compra
×

Producto	Imagen	Precio	Cantidad	Subtotal	
Tarta de queso		5.99	2	11.98	<button>Eliminar</button>
MacJava		10.99	2	21.98	<button>Eliminar</button>
Total				33.96	

Direccion de Compra

Casa

Detalles
Seleccionar

Chaletazo

Detalles
Seleccionar

Nueva Dirección

Datos de pago

Número de Tarjeta

Número de Tarjeta

- Tambien podemos seleccionar las direcciones en las que se enviara el pedido y finalmente podremos ingresar nuestro numero de tarjeta y nuestro codigo de seguridad

Una vez completado el pago se daran los detalles del pedido y podremos descargar una cversion pdf de nuestro pedido

Pedido creado correctamente



Detalles del Pedido

ID del Pedido: 9b820392-f688-45b8-859b-ba0412773481
Fecha: 2024-03-07 15:58:00
Precio Total: 33.96
Stock Total: 4
Estado: CREADO
Tarjeta: ****_****_****-7888

Dirección

Nombre: Chaletazo
País: España
Municipio: Fuenla
Calle: Calle del Ejemplo 2
Portal:
Información Adicional:

Apellidos: de Fuenla
Provincia: Madrid
Código Postal: 28911
Número: 123
Piso:

Líneas de Pedido:

ID	Precio	Stock	SubTotal
Tarta de queso	5.99	2	11.98
MacJava	10.99	2	21.98

Descargar en pdf

Volver

ub

MacJava

Conti

Aquí el pdf con los detalles del pedido



¡Gracias por comprar en nuestra tienda!

Detalles del Pedido

ID del Pedido:

9b820392-f688-45b8-859b-ba0412773481

Fecha:

2024-03-07 15:58:00

Precio Total:

33.96

Stock Total:

4

Estado:

CREADO

Tarjeta:

****_****_****-7888

Dirección

Nombre:

Chaletazo

Apellidos:

Test:

Para nuestros test adjuntaremos distintas imagenes sobre la estructura y los distintos test que hemos realizado de la aplicacion

```

11
12 >> class ProductoControllerTest extends TestCase
13 {
14     use RefreshDatabase;
15
16     protected $productos;
17     protected $categorias;
18
19     protected User $user;
20     protected User $admin;
21
22     no usages  Jaime9lozano
23     public function setUp(): void
24     {
25         parent::setUp();
26
27         $this->categorias = Categoria::factory()->count( count: 3)->create();
28         $this->productos = Producto::factory()->count( count: 3)->create();
29
30         $this->user = User::factory()->create();
31         $this->admin = User::factory()->create();
32         $this->admin->rol = 'ADMIN';
33     }

```

```

71 >> public function testCreateProducto()
72 {
73     $this->actingAs($this->admin);
74
75     $data = [
76         'nombre' => 'Nuevo Producto',
77         'precio' => 19.99,
78         'stock' => 50,
79         'descripcion' => 'Descripción del nuevo Producto',
80         'categoria_id' => $this->categorias[0]->id,
81     ];
82
83     $response = $this->post(route( name: 'productos.store'), $data);
84
85     $response->assertRedirect(route( name: 'productos.index'));
86     $this->assertDatabaseHas( table: 'productos', $data);
87 }

```

```

9
no usages  👤 Diego +1
10 class ProductoFactory extends Factory
11 {
12     ⚙️ protected $model = Producto::class;
13
14     ⚙️ no usages  👤 Diego +1
15     public function definition(): array
16     {
17         return [
18             'created_at' => Carbon::now(),
19             'updated_at' => Carbon::now(),
20             'nombre' => $this->faker->unique()->name(),
21             'descripcion' => $this->faker->word(),
22             'imagen' => $this->faker->word(),
23             'stock' => $this->faker->randomNumber( nbDigits: 2),
24             'precio' => $this->faker->randomFloat( nbMaxDecimals: 2, min: 0, max: 100),
25             'categoria_id' => Categoria::all()->random()->id,
26         ];
27     }
28 }

jaime9lozano
53 ▶ public function testCreateForbidden()
54 {
55     $this->actingAs($this->user);
56
57     $response = $this->get(route( name: 'productos.create'));
58
59     $response->assertStatus( status: 403);
60 }
61
jaime9lozano
62 ▶ public function testCreateAccessForm()
63 {
64     $this->actingAs($this->admin);
65
66     $response = $this->get(route( name: 'productos.create'));
67
68     $response->assertStatus( status: 200);
69 }
70

```

Son test para comprobar validaciones y metodos distintos

Presupuesto de la aplicación:

- **Número de semanas trabajadas:** 2
 - **Sueldo medio por hora de un trabajador:** 7,5 euros
 - **Horas de trabajo por semana por programador:** 40 horas
 - **Número de programadores:** 4
- Horas de trabajo total = 4 programadores × 40 horas/semana × 2 semanas*
- Horas de trabajo total = 4 programadores × 40 horas/semana × 2 semanas

Ahora, calculemos el presupuesto total con esta nueva cantidad de horas de trabajo. Con el ajuste a 2 semanas de trabajo, el nuevo presupuesto total para desarrollar una aplicación básica de Laravel sería de 2400 €.