Unión Europea
Fondo Social Europeo
"El FSE invierte en tu futuro"

Proyecto de desarrollo de aplicaciones web
ALINA'S LUXURY OUTLET

CICLO FORMATIVO DE GRADO SUPERIOR
Desarrollo de Aplicaciones Web (IFCS03)

Curso 2023-24

Autor/a/es:
Joselyn Carolina Obando Fernández

Tutor/a:
Rafael Manjavacas Lara

Departamento de Informática y Comunicaciones
I.E.S. Luis Vives

1 INTRODUCCIÓN

El presente Trabajo de Fin de Grado (TFG) presenta el desarrollo de una tienda virtual denominada **Alina's Luxury Outlet**, utilizando las tecnologías de PHP con el framework Laravel y PostgreSQL como sistema de gestión de bases de datos, se despliega en DOKER. Este proyecto tiene como objetivo principal ofrecer una plataforma de comercio electrónico robusta y accesible, tanto para usuarios como para administradores, con funcionalidades específicas que mejoran la experiencia de compra y la gestión de productos.

Alina's Luxury Outlet permite a los usuarios navegar por una variedad de productos con sus respectivas descripciones y precios. Los visitantes no autenticados pueden visualizar los productos disponibles, pero para realizar una compra deben registrarse o iniciar sesión. El proceso de autenticación asegura que cada usuario registrado asuma el rol de "user", proporcionando acceso a funcionalidades adicionales como la adición de productos al carrito y la realización de compras.

Para la administración del sistema, se ha implementado un rol específico de "admin", que habilita la gestión completa de productos y categorías. Las funcionalidades de administración incluyen:

- **Productos**: Creación, modificación de detalles, actualización de imágenes y eliminar el producto.
- **Categorías**: Creación, modificación y eliminar la categoría.

Además, **Alina's Luxury Outlet** incluye características avanzadas como:

- **Filtrado de productos por categorías**: Permitiendo a los usuarios encontrar productos específicos basados en sus preferencias.
- **Clasificación de productos por género**: Opciones para filtrar por "man", "woman" o "unisex", mejorando la experiencia de usuario.
- **Paginación de productos**: Mostrando hasta 5 productos por página para una navegación más eficiente.
- **Formulario de contacto**: Para facilitar la comunicación entre los clientes y la administración de la tienda.
- **Buscador integrado**: Que permite a los usuarios realizar búsquedas precisas de productos por nombre.
- **Generación de pdf:** Cuando realiza la compra, se genera una factura con todos los datos.

Las vistas del proyecto han sido diseñadas utilizando **Bootstrap**, **CSS** y **JavaScript**, lo que garantiza una interfaz de usuario atractiva, responsiva y fácil de usar. Bootstrap proporciona un conjunto de herramientas que permiten un diseño coherente y adaptativo, mientras que CSS y JavaScript permiten personalizar y mejorar la interactividad de la aplicación.

El desarrollo de este proyecto ha sido guiado por las mejores prácticas en el diseño y desarrollo de aplicaciones web, garantizando un sistema seguro, escalable y fácil de mantener. Con **Alina's Luxury Outlet**, se busca no solo facilitar el proceso de compra para los usuarios finales, sino

también proporcionar una herramienta eficaz para la gestión integral del inventario por parte de los administradores.

1.1 OBJETIVO

Al inicio, los objetivos principales fueron establecer una plataforma de comercio electrónico robusta y eficiente, empleando tecnologías modernas y siguiendo las mejores prácticas en desarrollo web. Los objetivos específicos iniciales fueron los siguientes:

1. **Desarrollo de la Plataforma E-commerce**:
    - **Tecnologías utilizadas**: Implementar el proyecto utilizando el lenguaje PHP y el framework Laravel.
    - **Gestión de bases de datos**: Emplear PostgreSQL como sistema de gestión de bases de datos.
2. **Funcionalidades para Usuarios**:
    - **Registro y autenticación**: Permitir a los usuarios registrarse y autenticarse para acceder a funcionalidades avanzadas.
    - **Exploración de productos**: Visualizar productos con descripciones, precios y múltiples imágenes a través de un carrusel.
    - **Carrito de compras**: Añadir productos al carrito y proceder con el proceso de compra.
    - **Generación de PDF de compra**: Gestionar la generación de un PDF detallado de la compra, que se envía por correo electrónico tras la confirmación de la compra.
    - **Filtrado y clasificación de productos**: Filtrar productos por categorías y clasificarlos por género (hombre, mujer, unisex) para una mejor visibilidad.
    - **Paginación de productos**: Implementar paginación, mostrando 6 productos por página para una navegación más eficiente.
    - **Formulario de contacto**: Facilitar la comunicación directa entre clientes y administración.
3. **Funcionalidades para Administradores**:
    - **Gestión de productos**: Crear, modificar detalles y actualizar imágenes de productos. Además crear un carrusel de imágenes donde se puedan actualizar cada una de ellas, donde sea más visible para el usuario viendo más de una foto por cada producto.
    - **Gestión de categorías**: Crear y modificar categorías de productos.
4. **Interfaz de Usuario**:
    - **Diseño responsivo y atractivo**: Utilizar Bootstrap, CSS y JavaScript para crear una interfaz de usuario intuitiva y adaptable a diferentes dispositivos.
5. **Implementación de Infraestructura en la Nube**:
    - **Despliegue en AWS**: Implementar y desplegar la aplicación en Amazon Web Services para garantizar escalabilidad y alta disponibilidad.

Estos objetivos iniciales fueron fundamentales para desarrollar una plataforma de comercio electrónico que no solo facilita la experiencia de compra para los usuarios, sino que también optimiza la gestión de inventarios y productos para los administradores, garantizando una operación eficiente y efectiva.

1.2 ALCANCE

El alcance del proyecto ha permitido desarrollar una tienda virtual que ofrece una experiencia de usuario completa y herramientas eficientes para la gestión de productos y categorías. Aunque algunos objetivos iniciales no se han implementado, las funcionalidades actuales proporcionan una base sólida y eficiente para futuras mejoras y ampliaciones. A continuación, se presenta el alcance del proyecto, destacando las funcionalidades implementadas y las que quedaron fuera del alcance actual:

Funcionalidades Implementadas

1. **Desarrollo de la Plataforma E-commerce**:
    - **Tecnologías Utilizadas**: El desarrollo se ha llevado a cabo con PHP y el framework Laravel.
    - **Gestión de Bases de Datos**: PostgreSQL ha sido seleccionado como el sistema de gestión de bases de datos.
2. **Características para Usuarios**:
    - **Registro y Autenticación**: Los usuarios pueden registrarse y autenticarse para acceder a funcionalidades adicionales.
    - **Visualización de Productos**: Los productos están disponibles con descripciones detalladas y precios.
    - **Carrito de Compras**: Los usuarios pueden añadir productos a su carrito y proceder con la compra.
    - **Filtrado y Clasificación**: Los productos pueden ser filtrados por categorías y clasificados por género (hombre, mujer, unisex) para facilitar la búsqueda.
    - **Paginación de Productos**: Se ha implementado la paginación, mostrando 5 productos por página para mejorar la navegación.
    - **Formulario de Contacto**: Está disponible un formulario de contacto para facilitar la comunicación entre clientes y administración.
    - Generación de pdf: Cuando el usuario finalice su compra se le descarga automáticamente su factura.
3. **Características para Administradores**:
    - **Gestión de Productos**: Los administradores pueden crear nuevos productos, actualizar detalles y modificar imágenes.
    - **Gestión de Categorías**: Los administradores tienen la capacidad de crear y actualizar categorías de productos.
4. **Interfaz de Usuario**:
    - **Diseño Responsivo**: La interfaz de usuario ha sido desarrollada con Bootstrap, CSS y JavaScript, proporcionando una experiencia adaptativa y atractiva en diferentes dispositivos.
5. **Despliegue e Infraestructura**:
    - **Despliegue con Docker**: La aplicación ha sido desplegada utilizando Docker, asegurando un entorno de desarrollo y producción consistente y eficiente.

Funcionalidades No Implementadas

Carrusel de Imágenes: No se ha implementado la funcionalidad de un carrusel para mostrar múltiples imágenes de los productos.

Despliegue en AWS: En lugar de Amazon Web Services, el despliegue se ha realizado utilizando Docker.

1.3 JUSTIFICACIÓN

La elección de tecnologías para el desarrollo de **Alina's Luxury Outlet** se basó en criterios de robustez, eficiencia, facilidad de uso y escalabilidad. A continuación, se detalla por qué se eligieron PHP, Laravel, PostgreSQL, Bootstrap, CSS, JavaScript y Docker en lugar de otras opciones disponibles en el mercado.

PHP con Laravel

PHP es un lenguaje de programación ampliamente utilizado en el desarrollo web, conocido por su facilidad de integración con diversos sistemas y su curva de aprendizaje relativamente baja. Laravel, un framework de PHP, fue seleccionado por las siguientes razones:

1. **Simplicidad y Elegancia**: Laravel proporciona una sintaxis limpia y legible, facilitando el desarrollo rápido y la comprensión del código.
2. **Ecosistema Rico**: Con una vasta cantidad de paquetes y herramientas, Laravel permite la implementación de funcionalidades avanzadas con menor esfuerzo.
3. **Comunidad Activa**: Laravel cuenta con una comunidad activa que proporciona soporte, recursos y actualizaciones frecuentes, asegurando que el proyecto se mantenga actualizado y seguro.
4. **Manejo Eficiente de Datos**: Laravel Eloquent ORM facilita la interacción con la base de datos, simplificando la consulta y manipulación de datos.

PostgreSQL

PostgreSQL fue elegido como el sistema de gestión de bases de datos por sus numerosas ventajas:

1. **Robustez y Confiabilidad**: PostgreSQL es conocido por su estabilidad y confiabilidad en el manejo de grandes volúmenes de datos.
2. **Funcionalidades Avanzadas**: Soporta características avanzadas como transacciones ACID, control de concurrencia multiversión (MVCC), y extensiones que permiten personalizar su comportamiento.
3. **Escalabilidad**: Es capaz de manejar una gran cantidad de conexiones simultáneas y puede ser escalado horizontal y verticalmente según las necesidades del proyecto.

Bootstrap, CSS y JavaScript

Para el diseño de la interfaz de usuario y la experiencia del usuario, se utilizaron **Bootstrap**, **CSS** y **JavaScript**:

1. **Bootstrap**:
   - **Diseño Responsivo**: Facilita la creación de interfaces responsive que se adaptan a diferentes tamaños de pantalla y dispositivos.
   - **Componentes Listos para Usar**: Ofrece una amplia variedad de componentes y utilidades que aceleran el desarrollo de interfaces atractivas y funcionales.
2. **CSS**:
   - **Estilos Personalizados**: Permite la personalización de la apariencia de la tienda para alinearse con la identidad visual de Alina's Luxury Outlet.
3. **JavaScript**:
   - **Interactividad**: Habilita funcionalidades interactivas en la página, mejorando la experiencia del usuario con elementos dinámicos y reacciones instantáneas a las acciones del usuario.

Docker

Docker fue elegido para el despliegue del proyecto debido a sus beneficios en términos de consistencia y portabilidad:

Entornos Aislados: Docker permite crear contenedores que aseguran que el entorno de desarrollo, prueba y producción sean idénticos, eliminando problemas de compatibilidad.

Facilidad de Despliegue: Simplifica el proceso de despliegue, permitiendo que la aplicación se ejecute de manera consistente en diferentes entornos.

Escalabilidad: Facilita la escalabilidad de la aplicación al permitir la fácil replicación de contenedores según sea necesario para manejar mayores cargas de tráfico.

2 IMPLEMENTACIÓN

2.1 ANÁLISIS DE LA APLICACIÓN

En esta sección se presenta un análisis exhaustivo detallando los casos de uso, el diagrama entidad relación y diagrama de clases:

Casos de Uso

Los casos de uso describen las interacciones entre los usuarios y el sistema para lograr objetivos específicos. A continuación se presentan los principales casos de uso:

Registrarse: El usuario proporciona su nombre, correo electrónico y contraseña para crear una cuenta. El resultado esperado "El usuario se registra exitosamente y puede iniciar sesión en el sistema."

Iniciar sesión: El usuario proporciona sus credenciales para iniciar sesión. El resultado esperado "El usuario accede a su cuenta con el rol de 'user' ".

Visualizar Productos: El usuario navega y ve los productos disponibles, con opciones de filtrado por categoría y género. El resultado esperado "El usuario visualiza una lista de productos con sus detalles."

Agregar Producto al Carrito: El usuario selecciona un producto y lo agrega a su carrito de compras. El resultado esperado " El producto se añade al carrito del usuario".

Realizar Compra: El usuario procede al pago de los productos en su carrito. El resultado esperado "La compra se completa y se genera una confirmación".

Gestionar Productos y Categorías: El administrador puede crear, editar y eliminar productos y categorías. El resultado esperado "Los productos y categorías se gestionan correctamente en la tienda".

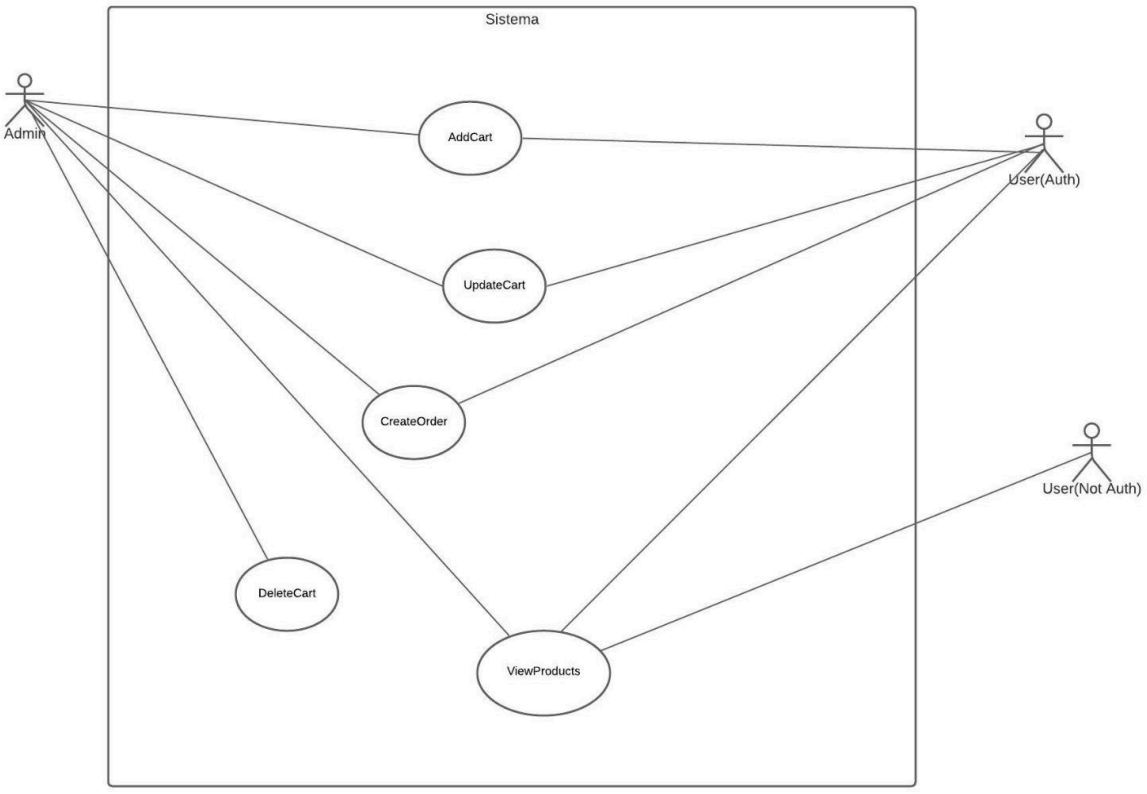


Diagrama de Entidad-Relación

El diagrama de entidad-relación muestra la estructura de la base de datos y las relaciones entre las entidades principales. A continuación se presenta el diagrama correspondiente:

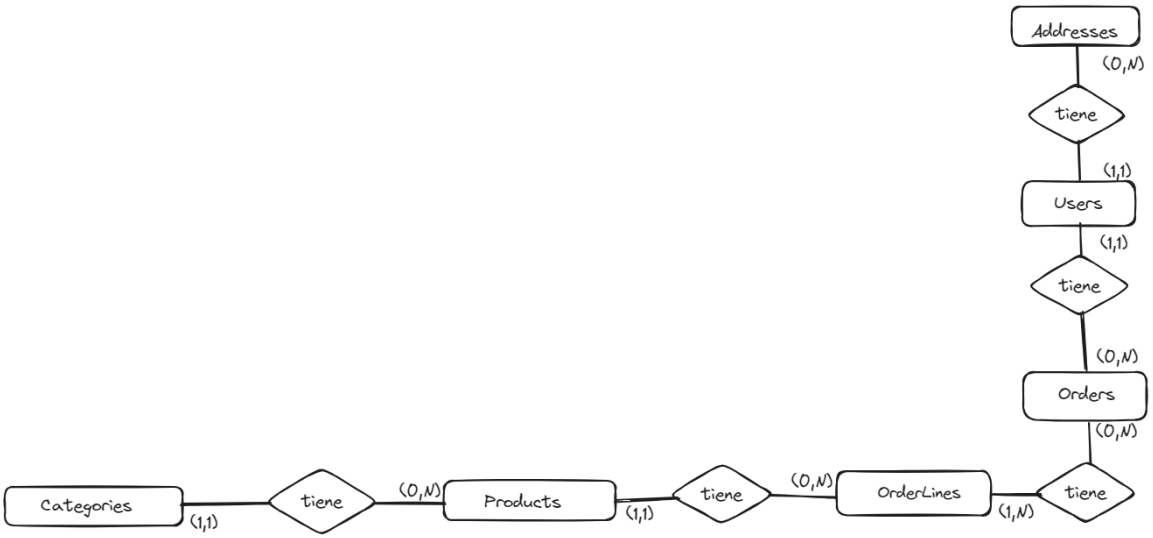
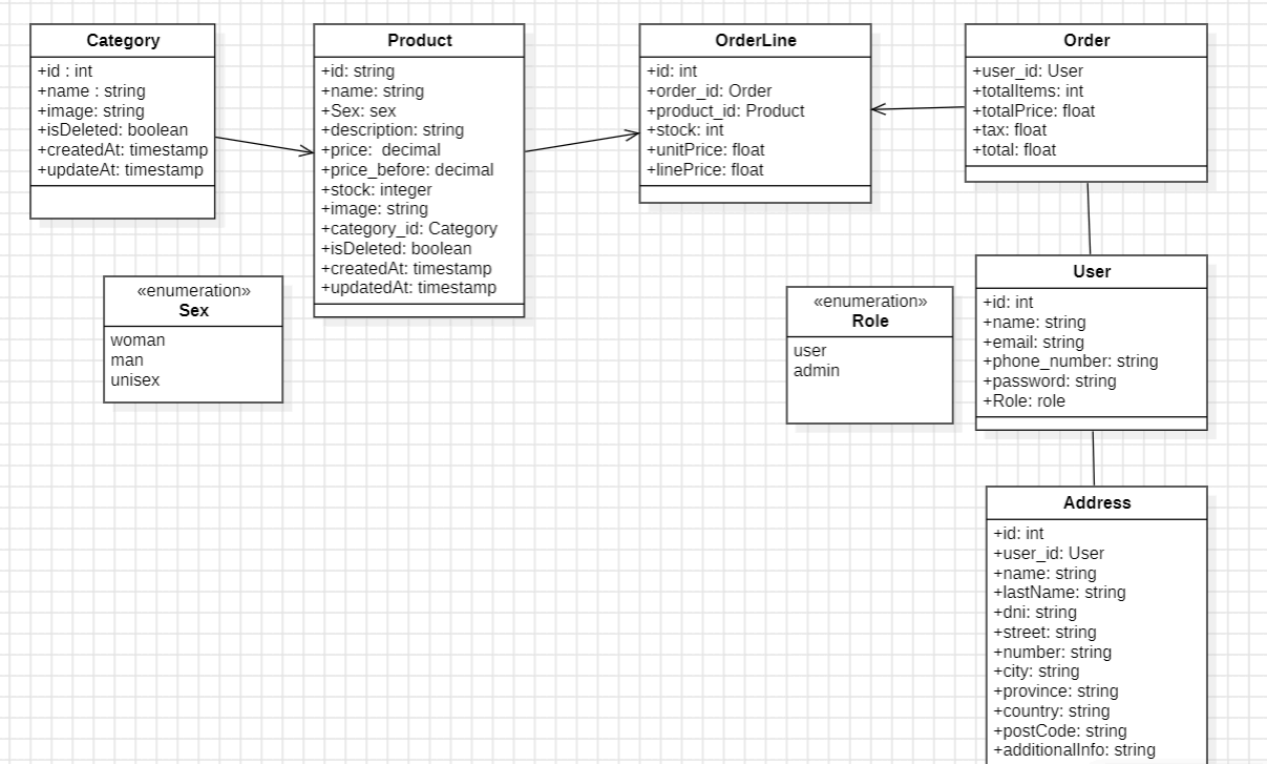


Diagrama de Clases

El diagrama de clases representa la estructura del sistema en términos de clases y sus relaciones. A continuación se presenta una descripción de las clases principales, junto con el diagrama correspondiente:



2.2 DISEÑO

Manual de usuario

2.3 IMPLEMENTACIÓN

La implementación de **Alina's Luxury Outlet** abarcó diversas etapas y tecnologías para asegurar una aplicación funcional. A continuación, se detalla el proceso de desarrollo y las tecnologías empleadas.

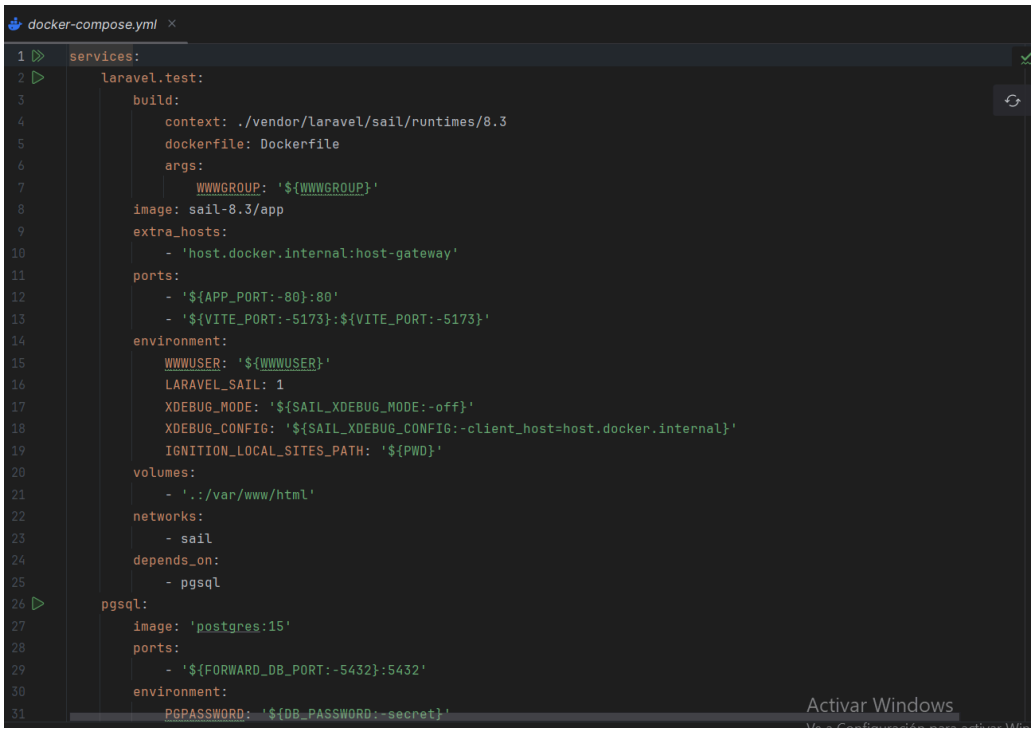
Tecnologías Utilizadas

- **Lenguaje de Programación**: PHP
- **Framework**: Laravel
- **Base de Datos**: PostgreSQL
- **Frontend**: Bootstrap, CSS, JavaScript
- **Control de Versiones**: Git
- **Contenedores**: Docker

Proceso de Desarrollo

Configuración del Entorno de Desarrollo

Se configuró el entorno de desarrollo utilizando Docker para asegurar la consistencia entre los entornos de desarrollo y producción. El archivo docker-compose-yml se creó para definir los servicios necesarios, incluyendo la base de datos PostgreSQL y la aplicación Laravel.



```yaml
services:
    laravel.test:
        build:
            context: ./vendor/laravel/sail/runtimes/8.3
            dockerfile: Dockerfile
            args:
                WWWGROUP: '${WWWGROUP}'
        image: sail-8.3/app
        extra_hosts:
            - 'host.docker.internal:host-gateway'
        ports:
            - '${APP_PORT:-80}:80'
            - '${VITE_PORT:-5173}:${VITE_PORT:-5173}'
        environment:
            WWWUSER: '${WWWUSER}'
            LARAVEL_SAIL: 1
            XDEBUG_MODE: '${SAIL_XDEBUG_MODE:-off}'
            XDEBUG_CONFIG: '${SAIL_XDEBUG_CONFIG:-client_host=host.docker.internal}'
            IGNITION_LOCAL_SITES_PATH: '${PWD}'
        volumes:
            - '.:/var/www/html'
        networks:
            - sail
        depends_on:
            - pgsql
    pgsql:
        image: 'postgres:15'
        ports:
            - '${FORWARD_DB_PORT:-5432}:5432'
        environment:
            PGPASSWORD: '${DB_PASSWORD:-secret}'
```


Estructura del Proyecto

Se creó un nuevo proyecto Laravel utilizando Composer, y se estructuró de la siguiente manera:

- **Modelos**: Representan las entidades principales (User, Product, Category, Order, OrderLine, Address).
- **Controladores**: Manejan la lógica de negocio y las interacciones del usuario.
- **Vistas**: Utilizando Blade Templates para la generación dinámica de HTML.
- **Rutas**: Definen las URL y asignan controladores y métodos correspondientes.

Autenticación y Roles de Usuario

Se implementó el sistema de autenticación utilizando Laravel Breeze para facilitar el registro e inicio de sesión de usuarios. Se configuraron dos roles principales: **user** y **admin**.

```php
public function up(): void
{
    Schema::create( table: 'users', function (Blueprint $table) {
        $table->id();
        $table->string( column: 'name');
        $table->string( column: 'email')->unique();
        $table->string( column: 'phone_number', length: 9);
        $table->timestamp( column: 'email_verified_at')->nullable();
        $table->string( column: 'password');
        $table->enum( column: 'role', ['user', 'admin'])->default( value: 'user');
        $table->rememberToken();
        $table->timestamps();
    });
}
```

Gestión de Productos y Categorías

Los administradores tienen la capacidad de gestionar productos y categorías. Se crearon controladores y vistas específicos para estas acciones:

Product:

```php
Joselyn Obando
public function store(Request $request){
    $request->validate([
        'name' => 'min:3|max:120|required|unique:products,name',
        'sex' => 'required',
        'description' => 'max:255|sometimes',
        'price' => 'required|regex:/^\d{1,6}(\.\d{1,2})?$/',
        'price_before' => 'sometimes|regex:/^\d{1,6}(\.\d{1,2})?$/',
        'stock' => 'required|integer|max:10000',
        'category' => 'sometimes|exists:categories,id',
    ]);

    try {
        $product = new Product($request->all());
        $product->id = Str::uuid();
        $product->category_id = $request->category ?? 1;
        $product->save();
        flash( message: 'Product ' . $product->name . ' created successfully.')->success()->important();
        return redirect()->route( route: 'products.index');
    } catch (Exception $e) {
        flash( message: 'Error creating the product. ' . $e->getMessage())->error()->important();
        return redirect()->back();
    }
}
```

Category:

```
Joselyn Obando
public function store(Request $request){
    $request->validate([
        'name' => 'min:4|max:120|required|unique:categories,name',
    ]);
    try {
        $category = new Category();
        $category->name = strtoupper($request->name);
        $category->save();
        flash( message: 'Category ' . $category->name . ' successfully created.')->success()->important();
        return redirect()->route( route: 'categories.index');
    } catch (Exception $e) {
        flash( message: 'There is already another category with the same name')->error()->important();
        return redirect()->back();
    }
}
```

Funcionalidades del Carrito y Compra

Se implementan funcionalidades para que los usuarios puedan agregar productos al carrito, gestionar su carrito y realizar compras. La finalización de la compra genera un pedido y actualiza el inventario.

Muestra el carrito de compra actual, recupera el carrito de compras actual de la sesión. Si no hay ningún carrito en la sesión, inicializa un array vacío como el carrito. Devuelve la vista del carrito con los datos del carrito para mostrar los elementos actualmente en el carrito de compras y devuelve una vista que muestra el carrito de compras.

```
1 usage  Joselyn Obando
public function showCart(){
    $cart = Session::get( key: 'cart', []);
    $cartItems = [];
    $totalPrice = 0;
    foreach ($cart as $item) {
        $product = $this->getProduct($item['product_id']);
        $totalPrice += $product->price * $item['stock'];
        $cartItems[] = [
            'product' => $product,
            'quantity' => $item['stock'],
        ];
    }

    return view( view: 'cart.cart')->with('cartItems', $cartItems)->with('totalPrice', $totalPrice);
}
```

Muestra la página de pago, obtiene el usuario actual y la dirección de envío del usuario desde la base de datos. Calcula el precio total, el ahorro y el impuesto sobre el precio total de los productos en el carrito y devuelve la vista de pago con los datos calculados y la dirección de envío.

```php
public function payment(){
    $user = User::find(Auth::id());
    $address = $user->addresses()->orderBy('id', 'desc')->first();
    $cart = Session::get( key: 'cart', []);
    $totalPrice = 0;
    $save = 0;
    foreach ($cart as $item) {
        $product = $this->getProduct($item['product_id']);
        $totalPrice += $product->price * $item['stock'];
        if($product->price_before && $product->price_before > 0){
            $save += ($product->price_before - $product->price) * $item['stock'];
        }
    }
    $tax = $totalPrice * 0.21;
    $total = $totalPrice + $tax;
    return view( view: 'layouts.payment')
        ->with('totalPrice', $totalPrice)
        ->with('save', $save)
        ->with('tax', $tax)
        ->with('total', $total)
        ->with('address', $address);
}
```

Actualiza la cantidad de un producto específico en el carrito de compras, validando la cantidad solicitada en relación con el stock disponible del producto. Si la validación es exitosa, actualiza la cantidad del producto especificado en el carrito de compras almacenado en la sesión, también actualiza el número total de elementos en el carrito. Redirecciona a la página anterior tras una actualización exitosa o en caso de un error con un mensaje flash apropiado.

```php
1 usage  Joselyn Obando
public function updateCartLine(Request $request){
    try {
        $product = $this->getProduct($request->id);

        $request->validate([
            'stock' => 'required|gt:0|lte:' . $product->stock,
        ]);

        $cart = Session::get( key: 'cart', []);
        $totalItems = Session::get( key: 'totalItems',  default: 0);

        foreach ($cart as $key => $item) {
            if ($key == $request->key) {
                $totalItems -= $cart[$key]['stock'];
                $cart[$key]['stock'] = $request->stock;
                $totalItems += $cart[$key]['stock'];
                break;
            }
        }
        Session::put('cart', $cart);
        Session::put('totalItems', $totalItems);
        return redirect()->back();
    } catch (Exception $e) {
        flash( message: 'Error updating cart line ' . $e->getMessage())->error()->important();
        return redirect()->back();
    }
}
```

Elimina una línea de producto específica del carrito de compras, a través de los elementos del carrito almacenados en la sesión para encontrar la línea de producto que coincida con la clave proporcionada en la solicitud. Al encontrar el elemento coincidente, deduce la cantidad de ese elemento del recuento total de elementos y elimina el elemento del carrito.

Actualiza el carrito y el recuento total de elementos en la sesión, redirecciona a la página anterior tras la eliminación exitosa o en caso de un error con un mensaje flash apropiado, (Request) la solicitud HTTP que contiene la clave de la línea de producto a eliminar. RedirectResponse redirecciona a la página anterior con un mensaje de éxito o error.

```php
1 usage  Joselyn Obando
public function destroyCartLine(Request $request){
    try {
        $cart = Session::get( key: 'cart', []);
        $totalItems = Session::get( key: 'totalItems', default: 0);

        foreach ($cart as $key => $item) {
            if ($key == $request->key) {
                $totalItems = max( value: 0, ...values: $totalItems - $cart[$key]['stock']);
                unset($cart[$key]);
                break;
            }
        }
        $cart = array_values($cart);
        Session::put('cart', $cart);
        Session::put('totalItems', $totalItems);
        return redirect()->back();
    } catch (Exception $e) {
        flash( message: 'Error updating cart line ' . $e->getMessage())->error()->important();
        return redirect()->back();
    }
}
```

Obtiene un producto de la caché si está disponible; de lo contrario, lo recupera de la base de datos. El id del producto que se va a obtener, el producto recuperado de la caché o de la base de datos.

```php
3 usages  Joselyn Obando
private function getProduct($id){
    return Cache::has($id) ? Cache::get($id) : Product::find($id);
}
```

Vistas y Estilos

Las vistas fueron desarrolladas utilizando Blade Templates, Bootstrap y CSS para una interfaz de usuario atractiva y responsiva.

```
ction('title', 'Alina Luxury - Products')

ction('content')
 @include('normalhead')
 <section class="py-2">
     <div class=" px-4 px-lg-5 mt-5 row">
         <div class="col-8 my-3 ms-4"><h1 class="fw-bolder">Products</h1></div>
         <div class="col-3 my-3 ms-5"><h2>Filter by:</h2></div>
         <div class="row gx-4 gx-lg-5 row-cols-1 row-cols-md-2 row-cols-xl-3 justify-content-center col-md-9">
             @foreach($products as $product)
                 <div class="col mb-5">
                     <div class="card h-100">
                         @if($product->image != Product::$IMAGE_DEFAULT)
                             <img class="card-img-top" src="{{ asset('storage/' . $product->image) }}" alt="{{ $product->name
                         @else
                             <img class="card-img-top" src="{{ Product::$IMAGE_DEFAULT }}" alt="{{ $product->name }}" style="
                         @endif

                         <div class="card-body p-4">
                             <div class="text-center">
                                 <h5 class="fw-bolder">{{ $product->name }}</h5>
                                 @if($product->price_before && $product->price_before != 0)
                                     <span style="...">€{{ $product->price }} </span> <del>€{{ $product->price_before }}</del
                                 @else
                                     €{{ $product->price }}
                                 @endif
                             </div>
                 </div>
```


Paginación de Productos

Se implementó la paginación para mostrar cinco productos por página, mejorando así la navegación y usabilidad.

```
Joselyn Obando
public function products(){
    $products = Product::orderBy('id', 'asc')->paginate(6);
    return view( view: 'products.gestion')->with('products', $products);
}
```

Filtrado y Búsqueda

Los usuarios pueden filtrar productos por categoría y género, y buscar productos por nombre.

Obtiene los productos por categoría y los muestra paginados, La vista que muestra los productos de una categoría específica.

```
no usages  Joselyn Obando
public function getProductsByCategory($id){
    $products = Product::where('category_id', "=", $id)->orderBy('id', 'asc')->paginate(5);
    return view( view: 'products.index')->with('products', $products);
}
```

Obtiene los productos por género y los muestra paginados, la vista muestra los productos de un genero especifico.

```
1 usage  Joselyn Obando
public function getProductsBySex($sex){
    $products = Product::where('sex', "=", $sex)->orderBy('id', 'asc')->paginate(5);
    $categories = Category::where('id', '<>', 1)->get();
    return view( view: 'products.index')->with('products', $products)->with('categories', $categories);
}
```

Test

Se desarrollaron pruebas unitarias y funcionales para asegurar la calidad del código y el correcto funcionamiento de la aplicación.

```
class ProductsControllerTest extends TestCase{

    use RefreshDatabase;
    no usages  Joselyn Obando
    protected function setUp(): void{
    parent::setUp();
    $this->artisan( command: 'migrate:fresh');
    $this->artisan( command: 'db:seed');
    }

    Joselyn Obando
    public function test_index() {
        $response = $this->get( uri: '/products');
        $response->assertViewIs( value: 'products.index');
        $response->assertViewHas('products');
        $response->assertStatus( status: 200);
    }

    Joselyn Obando
    public function test_show_view(){
        $product = Product::first();
        $response = $this->get(route( name: 'products.show', $product->id));
        $response->assertStatus( status: 200);
        $response->assertViewIs( value: 'products.show');
        $response->assertViewHas('product', $product);
    }
```

```php
class CategoriesControllerTest extends TestCase
{
    use RefreshDatabase;

    3 usages
    private $categoriesController;

    no usages  Joselyn Obando
    public function setUp(): void
    {
        parent::setUp();
        $this->categoriesController = new CategoriesController();
    }

    Joselyn Obando
    public function testIndex(){
        $request = new Request();
        $request->search = null;
        $categories = $this->categoriesController->index($request);
        $this->assertIsObject($categories);
    }

    Joselyn Obando
    public function testStore(){
        $request = new Request();
        $request->name = 'BAGS';
        $categories = $this->categoriesController->create($request);
        $this->assertIsObject($categories);
    }
```

2.4 IMPLANTACIÓN

El despliegue e instalación se realiza utilizando Docker, una plataforma que facilita la creación y el despliegue de aplicaciones en contenedores, garantizando un entorno controlado y consistente. A continuación, se describe detalladamente cómo llevar a cabo este proceso utilizando Docker y Laravel Sail.

Pre-requisitos

Antes de comenzar, asegúrate de tener Docker y Docker Compose instalados en tu sistema. Estas herramientas permiten ejecutar la aplicación en contenedores, asegurando que todas las dependencias y configuraciones sean gestionadas de manera eficiente.

Configuración del entorno Docker

El archivo docker-compose-yml define los servicios necesarios para ejecutar la aplicación. A continuación, se muestra y explica cada sección del archivo.

```yaml
services:
    laravel.test: <8 keys>
    pgsql:
        image: 'postgres:15'
        ports:
            - '${FORWARD_DB_PORT:-5432}:5432'
        environment:
            PGPASSWORD: '${DB_PASSWORD:-secret}'
            POSTGRES_DB: '${DB_DATABASE}'
            POSTGRES_USER: '${DB_USERNAME}'
            POSTGRES_PASSWORD: '${DB_PASSWORD:-secret}'
        volumes:
            - 'sail-pgsql:/var/lib/postgresql/data'
            - './vendor/laravel/sail/database/pgsql/create-testing-database.sql:/docker-entrypoint-initdb.d/10-create-testing-
        networks:
            - sail
        healthcheck:
            test:
                - CMD
                - pg_isready
                - '-q'
                - '-d'
                - '${DB_DATABASE}'
                - '-U'
                - '${DB_USERNAME}'
            retries: 3
            timeout: 5s
networks:
    sail:
        driver: bridge
volumes:
```

1.- Clonar el repositorio: Clona el repositorio del proyecto en tu máquina local.

```
git clone https://github.com/usuario/alinas-luxury-outlet.git
```

```
cd alinas-luxury-outlet
```

2.-Copiar el archivo .env.example a .env : Copia el archivo de ejemplo de configuración de entorno y ajusta los parámetros según tus necesidades.

```
cp .env.example .env
```

3.-**Configurar las variables de entorno:** Asegúrate de configurar las variables de entorno en el archivo .env para que coincidan con tu configuración local. Presta especial atención a las configuraciones de la base de datos, usuario y contraseñas.

4.-Instalar las dependencias: Utiliza Composer para instalar las dependencias del proyecto.

```
composer install
```

5.- Iniciar los contenedores Docker: Utiliza Docker Compose para construir e iniciar los servicios definidos en docker-compose.yml .

```
./vendor/bin/sail up -d
```

6.-Migrar la base de datos: Ejecuta las migraciones de la base de datos para crear las tablas necesarias.

```
./vendor/bin/sail artisan migrate:fresh
```

7.- Seeders: Si tienes seeders configurados para poblar la base de datos con datos de prueba, ejecutados.

```
./vendor/bin/sail artisan db:seed
```

El uso de Docker y Laravel Sail simplifica significativamente el proceso de despliegue, proporcionando un entorno coherente y fácil de gestionar tanto para desarrollo como para producción. Esta configuración asegura que todos los desarrolladores trabajen en un entorno idéntico, minimizando problemas de configuración y mejorando la eficiencia del desarrollo y despliegue de la aplicación.

2.5 DOCUMENTACIÓN

Manual técnico con los recursos técnicos utilizados
Documentación de los procesos, Backend & Frontend – servicios… Javadoc

3 RESULTADOS Y DISCUSIÓN

El desarrollo de **Alina's Luxury Outlet** ha sido un viaje lleno de aprendizajes y logros. Uno de los mayores éxitos ha sido la implementación de un sistema de autenticación y gestión de usuarios, que permite a los visitantes registrarse, iniciar sesión y navegar por la tienda con roles claramente definidos. Los usuarios no autenticados son dirigidos a la página de inicio de sesión al intentar realizar una compra, asegurando que solo los usuarios registrados puedan completar transacciones.

En cuanto a la gestión de productos y categorías, la plataforma permite a los administradores añadir, modificar y eliminar productos, así como gestionar las categorías de manera eficiente. Esto asegura que el catálogo de la tienda esté siempre actualizado y relevante para los clientes.

La funcionalidad de carrito de compras ha sido implementada con éxito, permitiendo a los usuarios agregar productos al carrito y realizar compras fácilmente. Además, la paginación de productos, mostrando seis productos por página, mejora significativamente la navegación y la experiencia de usuario. Los usuarios también pueden filtrar productos por categoría y género, y buscar productos por nombre, lo que facilita encontrar exactamente lo que buscan.

La interfaz de usuario, creada con Bootstrap, CSS y JavaScript, ofrece una experiencia visual atractiva y una navegación intuitiva. Estas tecnologías permiten validaciones en el cliente y una interfaz responsiva, lo que mejora la experiencia general de compra.

Además, se ha implementado un formulario de contacto para que los clientes puedan enviar sus consultas, mejorando así la comunicación con la tienda.

Sin embargo, el desarrollo no estuvo exento de desafíos. Implementar y gestionar roles y permisos para asegurar que solo los administradores tengan acceso a ciertas funcionalidades fue un reto significativo. También hubo complicaciones en la configuración de la paginación y el filtrado de productos, que requirieron optimizaciones en las consultas de la base de datos.

En lugar de desplegar la aplicación en Amazon Web Services, optamos por Docker para simplificar la gestión del entorno, lo cual presentó sus propios desafíos, especialmente en la sincronización de datos y la gestión de volúmenes.

No todos los objetivos iniciales se alcanzaron. No se implementó el carrusel de imágenes para los productos. A pesar de estos elementos pendientes, la plataforma resultante es funcional y ofrece una experiencia de usuario satisfactoria.

4 TRABAJO FUTURO (OPCIONAL)

El proyecto actual de la tienda virtual proporciona una base sólida para futuras expansiones y mejoras, a pesar de que algunas funcionalidades planeadas, como el carrusel de imágenes y el despliegue en AWS, aún no se han implementado. La elección de tecnologías y la arquitectura del sistema están diseñadas para facilitar la incorporación de nuevas características y optimizaciones.

Trabajos Futuros

Despliegue en AWS El objetivo es migrar la aplicación a la nube para mejorar la escalabilidad, disponibilidad y rendimiento. Para lograr esto, se configurará una instancia de EC2 para alojar la aplicación, una base de datos RDS para PostgreSQL, y S3 para el almacenamiento de archivos estáticos. Además, se implementará un balanceador de carga (ELB) y se tomarán medidas de seguridad adecuadas, como la configuración de grupos de seguridad y políticas de IAM.

Carrusel de Imágenes Mejorar la experiencia del usuario en la página de inicio con un carrusel de imágenes destacadas es una prioridad. Esto implicará la integración de una biblioteca de carrusel de imágenes, como Slick o Owl Carousel, y su configuración para mostrar promociones y productos destacados. Se asegurará que el carrusel sea responsive y funcione bien en dispositivos móviles.

Cupones de Descuento La implementación de un sistema de cupones de descuento permitirá ofrecer promociones a los clientes. Se creará una interfaz para que los administradores puedan generar y gestionar cupones, y se modificará el proceso de compra para permitir la entrada y validación de cupones, asegurando que los descuentos se apliquen correctamente y se reflejen en el total del pedido.

Gestión de Perfil de Usuario Se permitirá a los usuarios gestionar su perfil y actualizar su información personal a través de una nueva vista. Esta vista permitirá ver y editar detalles como nombre, dirección y correo electrónico, con validaciones para asegurar que los datos ingresados sean correctos y seguros.

Histórico de Pedidos Proveer a los usuarios una vista para gestionar su histórico de pedidos mejorará significativamente la experiencia de usuario. Esta vista mostrará el historial de pedidos pasados, incluyendo detalles como el número de pedido, fecha de compra, productos adquiridos, estado del pedido y total pagado. También se permitirá la descarga de facturas o resúmenes de pedidos.

5 CONCLUSIONES

El proyecto "Alina 's Luxury Outlet" ha culminado en el desarrollo de una tienda virtual robusta y eficiente, utilizando tecnologías avanzadas como PHP con el framework Laravel y PostgreSQL como base de datos. La plataforma resultante no solo facilita la compra de productos, sino que también optimiza la gestión de los mismos. Los usuarios pueden registrarse e iniciar sesión para acceder a sus cuentas, ver productos, añadirlos al carrito y realizar compras. La interfaz de usuario, diseñada con Bootstrap, CSS y JavaScript, ofrece una experiencia visual atractiva y fácil de usar, con funcionalidades de paginación para mejorar la navegación.

Además, los administradores pueden gestionar productos y categorías de manera eficaz, incluyendo la creación, modificación y eliminación de productos, así como la clasificación por género (hombre, mujer, unisex) y filtrado por categorías. La generación de facturas en PDF al momento de la compra ha sido implementada, mejorando la transparencia y la confianza del cliente. Aunque algunas características planificadas, como el carrusel de imágenes y el despliegue en Amazon Web Services, no se implementaron, el proyecto se desplegó con éxito en Docker, proporcionando una base sólida para futuras mejoras. En resumen, "Alina 's Luxury Outlet" representa un paso significativo hacia la modernización y digitalización del comercio de lujo, ofreciendo una plataforma escalable y adaptable a las necesidades del mercado.

ANEXOS

Link de proyecto GitHub:

https://github.com/JoselynO/Alina-s_Outlet-/tree/main/php-outletLaravel