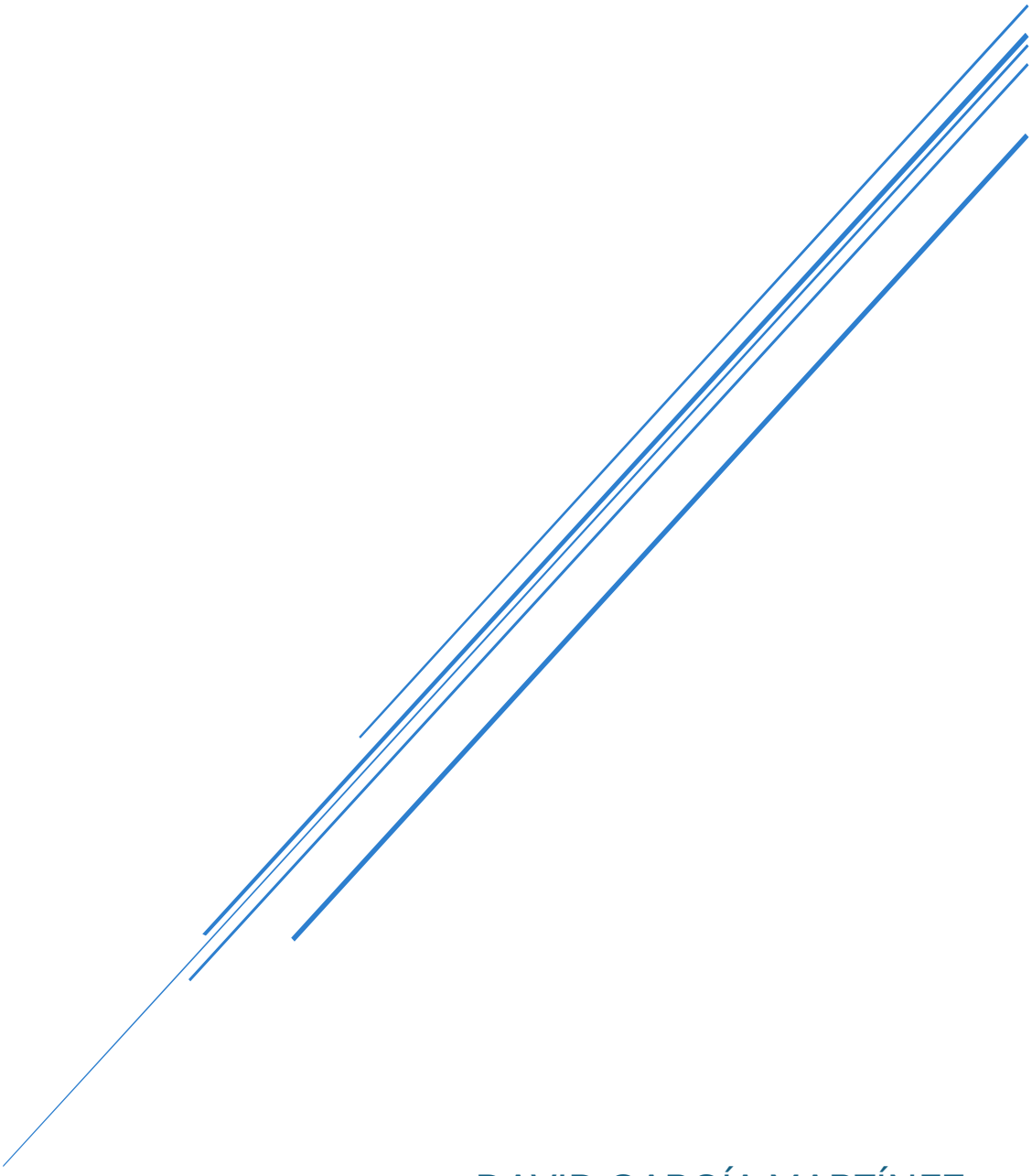
MEMORIA PROYECTO IW

LUXURY PARFUM



DAVID GARCÍA MARTÍNEZ

ENTREGA 2 DE LA ASIGNATURA

ÍNDICE

INTRODUCCIÓN

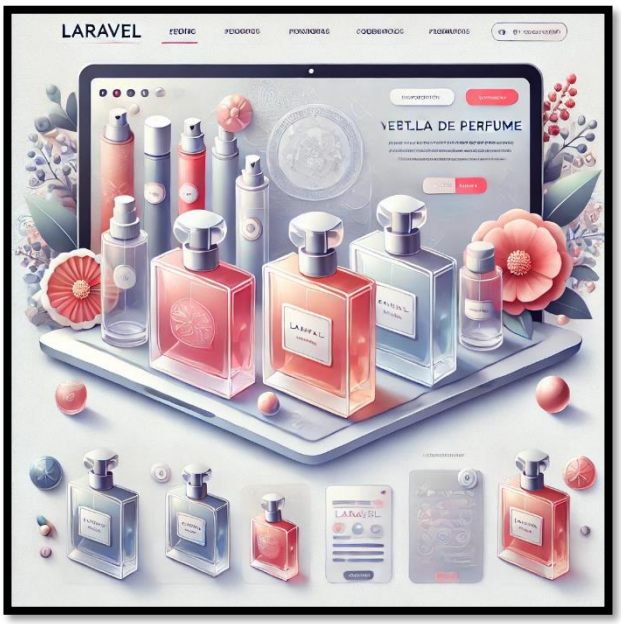
El proyecto tiene como objetivo la creación de una aplicación web que gestione la venta en línea de una tienda de perfumes, la vista de administrador, etc. Utilizando el framework Laravel y el motor de plantillas Blade. La base de datos se almacena en MySQL.

Objetivos principales

- Desarrollar la aplicación web con Laravel, garantizando una arquitectura limpia y escalable.
- Utilizar MySQL como base de datos para almacenar la información de forma segura.
- Implementar vistas dinámicas con el sistema de plantillas Blade.
- Asegurar un flujo de trabajo colaborativo con GitHub como repositorio de código y herramienta de gestión de tareas.

Alcance esperado del sistema

El sistema abarca la creación de migraciones, controladores y modelos en Laravel, así como la configuración de vistas Blade para la parte de interfaz de usuario. También se definió una metodología de trabajo en equipo, usando GitHub como soporte para la comunicación y la asignación de tareas.



DESCRIPCIÓN TÉCNICA

TECNOLOGÍAS

Laravel (versión 11): Framework de PHP que facilita la organización de la aplicación, la generación de rutas y la implementación de un patrón MVC.

MySQL: Sistema de gestión de bases de datos relacional que almacena la información del proyecto (por ejemplo, usuarios, pedidos, productos, etc.).

Blade: Motor de plantillas propio de Laravel que se utiliza para generar las vistas de forma elegante y con la separación lógica de la capa de presentación.

GitHub: Plataforma de alojamiento de repositorios Git que provee control de versiones, seguimiento de issues y herramientas de colaboración (pull requests, revisión de código).

ARQUITECTURA

Modelo-Vista-Controlador (MVC):

- Modelos: Contienen la lógica de acceso a datos (ORM Eloquent de Laravel).
- Vistas: Se generan con Blade, separando la lógica de la presentación.
- Controladores: Gestionan la lógica de negocio y coordinan la comunicación entre las vistas y los modelos.

Estructura de carpetas de Laravel:

- app/Models: Modelos Eloquent.
- app/Http/Controllers: Controladores que procesan las peticiones y retornan vistas o respuestas JSON.
- resources/views: Vistas Blade.
- database/migrations: Definición de la estructura de tablas en MySQL.
- database/seeders: Código que rellena las tablas automáticamente con datos.
- routes/web.php: Definición de rutas web y endpoints principales.
- public/images: Imágenes que utiliza la página web
- public/storage/images: Donde se guardan las imágenes de los productos y las categorías subidas por los administradores.
- resources/views: Donde se almacenan las plantillas del frontend de la página web.

INTEROPERACIÓN

Conexión a MySQL: A través del archivo .env se configura la conexión a la base de datos y diferentes opciones de Laravel. Si se quisiera crear un servicio SMTP de correo también se utilizaría este archivo para implementarlo.

Control de dependencias: Se utiliza Composer para instalar las librerías necesarias.

Comunicación interna (frontend-backend): Las rutas definen endpoints que devuelven vistas Blade o datos en JSON. Blade se encarga de procesar variables y renderizar contenido en HTML.

MOCKUPS

Los mockups se encuentran en el repositorio de GitHub en el archivo 'Mockups IW.pdf'.

La ruta a los Mockups en el repositorio de GitHub es:

https://github.com/dgm91-ua/IngenieriaWeb/tree/main/Mockups

Los mockups han sido generados con Figma y se ha guardado una copia en el repositorio del archivo que te genera dicho proyecto importándolo.

PATRONES DE DISEÑO WEB APLICADOS

Layout Master (Layout Pattern): Uso de un layout base en Blade (por ejemplo, masterusers.blade.php) que define cabecera, footer y secciones comunes; cada vista concreta extiende este layout.

Pagination Pattern: Implementación de paginación en listados de registros (categorias, productos) para mejorar la usabilidad y el rendimiento.

Partial Views: Componentes o "partials" para fragmentos de HTML reutilizables, como menús de navegación o formularios de búsqueda.

MVC propiamente dicho: Favorece la separación de responsabilidades y la claridad del código.

METODOLOGÍA

HERRAMIENTAS

- Se utilizó GitHub como repositorio Git.
- Se crearon Issues para cada tarea y se asignaron a los miembros del equipo.
- Se realizaron pull requests para controlar la calidad del código.
- Se creo un sistema de control de calidad de código con approvers de pull request: Rama Development → Rama Pre-Production → Rama main

METODOLOGÍA DE TRANAJO

- Scrum para iterar en sprints quincenales con la gestión de las tareas para la entrega del proyecto o gestionar el flujo de trabajo con un tablero.
- Reuniones semanales de seguimiento, donde se revisan avances y se definen las próximas tareas.

CRONOGRAMA

- Fase de análisis de requisitos.
- Diseño de mockups y diagramas.
- Implementación inicial (migraciones y modelos).
- Implementación de controladores y rutas.
- Desarrollo de vistas con Blade.
- Desarrollo de modelo administrador.
- Pruebas y refactorización.
- Creación de la documentación.

IMPLEMENTACIÓN

Las partes más complejas del código son:

- **Middleware Auth**: Middleware de Laravel que te permite gestionar si un usuario ha iniciado sesión. Este Mddleware se ha modificado para que permita controlar ciertos mecanismos de administrador.
- **Migraciones**: Gestión de las tablas de la base de datos con las relaciones entre ellas.
- **Seeders**: Permiten generar datos automáticamente en las tablas de la base de datos.
- **Controladores**: Permiten implementar la lógica del backend (generar los CRUD de productos y categorías), almacenamiento de imágenes, obtener ciertas vistas filtradas de productos, etc.
- **Modelos Eloquent**: Definición de las relaciones entre las diferentes tablas de la base de datos.
- **Blade Templates:** Uso de secciones dentro del código (HTML) que te permiten generar condicionales o recorrer listas para mostrar solo ciertas secciones si estas logueado o recorrer toda una lista para mostrar los productos.
- **Validaciones:** Uso de modelos FormRequest o validar directamente los datos desde el controlador para asegurarnos de que son correctos.

Unos ejemplos de esta complejidad de código podrían ser:

- La suma de los productos del carrito a tiempo real mientras actualizas las cantidades de estos (script que te permite actualizar a tiempo real la página si actualizas las cantidades, también actualiza el total del carrito y los subtotales).
- Mostrar ciertos enlaces dependiendo del tipo de usuario (no registrado, comprador, administrador).
- Mantener el stock de los productos actualizado y no permitir comprar nunca más productos de los que aparecen como límite de stock.
- Permitir a los administradores subir imágenes de los diferentes productos mediante un vínculo que tiene Laravel con la carpeta public/storage.
- Manejar diferentes filtros simultáneamente en los productos los cuáles pueden ser todos utilizados o no simultáneamente (buscar, filtrar por nombre, filtrar por precio, ordenar).

PROBLEMAS

- Permitir a los usuarios tener imágenes de perfil sin que afecte al rendimiento o almacenamiento si aumentase la cantidad exponencialmente:

  Para solucionar este problema se optó por utilizar enlaces a https://ui-avatars.com, esta página te genera imágenes utilizando el nombre del usuario y después se almacena esta ruta como ruta a la imagen del usuario.

- Permitir a los administradores subir imágenes de productos permitiendo enlace entre la subida de estas imágenes y el enlace en la base de datos:

  Para solucionar este problema se utilizó un comando de artisan que te permite hacer una conexión para acceder a la ruta storage y así obtener las imágenes o subirlas.

  ```
  php artisan storage:link
  ```

- Para generar datos de prueba en la base de datos y poder tener siempre datos en las tablas para hacer pruebas:

  Se optó por utilizar Factories y Seeders que te generan los datos con un solo comando.

  ```
  php artisan migrate:fresh --seed
  ```

- Para poder actualizar de manera automática las cantidades de productos que se desean comprar:

  Se optó por utilizar un script de Ajax que te permite actualizar la página, así como la base de datos (cantidades del carrito) a medida que se cambia. Esto también generó la necesidad de crear un script que actualice los importes de subtotales como los de totales a medida que cambian sin refrescar la página ni darle a un botón.

- Para gestionar si un usuario se ha logueado o qué tipo de usuario se ha logueado:

  Se optó por utilizar el Middleware de Laravel Auth que permite controlar este tipo de cosas, además se modificaron algunas funciones para poder implementar direcciones en los usuarios y controlar ciertos campos y accesos restringidos.

MEJORAS

- **Internacionalizar idioma**: Soporte multilenguaje.
- **Modularización**: Añadir paquetes de Laravel para roles y permisos avanzados.
- **Tests automatizados**: Implementar pruebas unitarias con PHPUnit o Pest, e integración continua.
- **Servicios externos**: Integrar un servicio de pago o un sistema de notificaciones en tiempo real.
- **SMTP de Correo**: Implementar un servicio de corre que te permita recuperar la contraseña.
- **Localizador de Envíos**: Diferenciar los diferentes estados de un envío y actualizar junto con la agencia de paquetería el estado del pedido.
- **Implementar métodos de pago**: Permitir a los usuarios pagar con diferentes métodos (PayPal, Criptomonedas, Pago contrarrembolso, etc).
- **Internacionalizar moneda**: Permitir cambiar la moneda y cambiar los precios de la página con esto.

REFERENCIAS

- **Documentación oficial de Laravel**: https://laravel.com/docs
- **Documentación de MySQL**: https://dev.mysql.com/doc/
- **Tutoriales oficiales de Blade**: https://laravel.com/docs/blade
- **Guías de GitHub**: https://docs.github.com

ENLACES

GITHUB

https://github.com/dgm91-ua/IngenieriaWeb

PLANIFICACIÓN

https://github.com/users/dgm91-ua/projects/4