

EnsembleElegance



Carmelo Rodríguez Rosalina

EnsembleElegance

CAPÍTULO 1 – INTRODUCCIÓN.....	3
1.1 INTRODUCCIÓN.....	3
1.2 REQUISITOS NO FUNCIONALES.....	3
1.3 REQUISITOS FUNCIONALES.....	3
CAPÍTULO 2 – PLANIFICACIÓN.....	4
2.1 PLANIFICACIÓN DEL PROYECTO.....	4
2.1.3 SERVIDOR.....	6
2.3 TAREAS IMPORTANTES REALIZADAS DEL PROYECTO.....	7
CAPÍTULO 3 – ANÁLISIS DEL PROYECTO.....	10
3.Introducción.....	10
3.1 ROLES DE USUARIO.....	10
3.2 REQUISITOS FUNCIONALES.....	10
3.3 REQUISITOS NO FUNCIONALES.....	13
3.4 CASOS DE USO.....	14
3.5 DIAGRAMA DE SECUENCIA.....	16
3.5.4 ROL DIRECTOR.....	30
3.6 BASE DE DATOS.....	33
4. ARQUITECTURA Y TECNOLOGÍAS EMPLEADAS.....	39
5. LIBRERÍAS EMPLEADAS.....	41
4. PROYECTO LARAVEL + REACT, CÓDIGO.....	47
5. GITHUB (Control de versiones).....	101
6. MANUAL DE ESTILOS.....	101
Color y uso tipografía del color.....	101
Identidad visual.....	102
Uso y proporción de imagen.....	103
Uso de iconos y descripción.....	105
Descripción de la estructura.....	106
Estructura de las páginas de cada género.....	108
Sección productos.....	111
Detalle Producto.....	112
Inputs y filtros.....	114
Mensaje error.....	115
Botones.....	116
Estructura de sesión.....	117
Menú hamburguesa.....	118
Píe de página.....	119
Encabezado de administrador/director.....	120
Sidebar.....	121
Gráficas y tablas.....	122
Tablas.....	123
Paginación.....	124
Formulario.....	125

CAPÍTULO 1 – INTRODUCCIÓN

1.1 INTRODUCCIÓN

El proyecto consiste en el desarrollo de una tienda en línea de moda y accesorios, con el objetivo de ofrecer a los clientes una experiencia de compra conveniente y atractiva, similar a marcas reconocidas como Stradivarius o Zara. La plataforma permitirá a los usuarios explorar una amplia gama de prendas de moda y accesorios, con la posibilidad de realizar compras de manera accesible y conveniente.

1.2 REQUISITOS NO FUNCIONALES

- El sitio web debe ser rápido, con tiempos de carga mínimos.
- Garantizar la seguridad contra ataques de inyección de código.
- La interfaz de usuario debe ser intuitiva, fácil de usar y dinámica.
- Escribir un código limpio y modular para facilitar el mantenimiento y la incorporación de nuevas funcionalidades.
- Realizar pruebas después de cada implementación para garantizar el correcto funcionamiento del sistema.

1.3 REQUISITOS FUNCIONALES

- Visualización de Prendas:
 - Al seleccionar una prenda de ropa, mostrará opciones de talla, precio y otra información relevante.
 - Permitir agregar la prenda seleccionada al carrito de compras y procesar el pago.
 - Mostrar recomendaciones de prendas relacionadas después de seleccionar una prenda.
- Autenticación y Gestión de Usuario:
 - Programar el registro y el inicio de sesión para autenticar a los usuarios.
 - Desarrollar un sistema de carrito de compra que solo esté disponible para usuarios autenticados.
 - Permitir a los usuarios agregar prendas a una lista de deseos para futuras compras.
 - Crear perfiles de usuario para que los usuarios puedan ver y modificar sus datos personales.
- Gestión de Stock y Ofertas:
 - Mostrar el stock restante de cada prenda y la disponibilidad de tallas.
 - Implementar un sistema de ofertas que indique la duración, tipos de prendas y la cantidad de prendas incluidas en la oferta.
- Roles:
 - Definir rol de usuario, administrador, director e invitado.
 - Los usuarios podrán acceder a funciones como ver su carrito de compras, wishlist y perfil.
 - Los administradores podrán agregar y eliminar artículos del sitio web y permitir ver información sobre los niveles de stock de los artículos.
 - El director podrá eliminar a los administradores y editar sus datos.
 - El invitado el cuál solo podrá ver la página, sin comprar nada.

- API
 - Crear una API utilizando Laravel JWT para proporcionar información sobre todas las prendas disponibles en el sitio.
 - Permitir filtrar mediante categoría de prendas, precio, fecha de lanzamiento, etc.
 - Permitir consultar las 10 mejores prendas de una categoría.

Estas son solo ideas, que más adelante se matizarán y se expondrán con la implementación final..

CAPÍTULO 2 – PLANIFICACIÓN

2.1 PLANIFICACIÓN DEL PROYECTO

En este apartado se va a documentar las horas, fases y tiempos para la elaboración del proyecto así como la estimación de coste.

2.1.1 CICLO DE DESARROLLO

- Requisitos – Identificar, documentar y comprender las funciones específicas a implementar.
- Diseño – Elaboración de diagramas para crear la arquitectura y estructura general del proyecto.
- Implementación – Llevar a cabo la construcción del proyecto en base a los diseños y los requisitos en código ejecutable.
- Evaluación – Realizar pruebas para verificar si el proyecto cumple con los requisitos. El objetivo es identificar y corregir cualquier problema o error antes de entregarlo.
- Arreglos – Realizar correcciones finales y agregar características adicionales.

2.1.2 ESTIMACIÓN DEL TIEMPO A EMPLEAR

MES	A REALIZAR	DESCRIPCIÓN
Marzo (18/03/2024- 31/03/2024)	Instalación y configuración del proyecto Elaboración de requisitos funcionales y no funcionales, junto a la creación de los diagramas Diseño de la base de datos	Creación del proyecto de Laravel 9 y React. Junto a los contenedores Docker con un servidor MySQL y phpmyadmin. Instalación de librerías en React, como: axios y react-dom-router. Analizo páginas web para mi proyecto para decidir cuáles son los requisitos que quiero implementar, junto a la creación de los diagramas de dichos requisitos. Creación de la BBDD, abierta a modificación si lo es necesario, en el futuro.

Abril (01/04/2024- 30/04/2024)	Documentación del proyecto	Realizar la parte de la documentación para documentar el inicio del proyecto.
	Emplear tiempo para aprender React	Mira cursos y documentación para aprender React
	Implementar la gran parte de la BBDD	Tanto migraciones como modelos, las iré implementando cuando las vaya necesitando durante el proyecto.
	Desarrollo sistema de autenticación de usuarios y la creación del admin	Creación de componentes en React e instalación e implementación de la librería JWT para las sesiones de los usuarios.
	Usar laravel como una api para pasar los datos a React mediante axios	Gestionar todo a través de API e implementar las conexiones en todos los componentes para la recogida de datos.
	Desarrollo del carrito	Añadir producto al carrito, decidir su talla y la cantidad, junto al cálculo de pago.
	Desarrollo de la wishlist	Desarrollar un componente para almacenar los productos favoritos de los usuarios para una futura compra.
	Desarrollo de las categorías, vista de productos y vista de productos en detalle.	Implementación de las categorías, junto a la vista de todos los productos en la vista principal y su vista en detalle para la información de cada producto.
	Implementación del CRUD de productos	Desarrollo de crear, editar y eliminar un producto desde el panel de administrador.
	Implementación visualización y gestión de stock	Desarrollo de un panel de gestión de stock para el administrador.
MAYO	Implementación del método de pago	Implementar PayPal como método de pago, en función del tiempo que tenga, implementaré también Stripe.
	Desarrollo de sistema de ofertas	Desarrollo de un sistema de ofertas para los usuarios, si tengo suficiente tiempo restante, incluiré cupones de descuento, si no, solo se quedará en un simple sistema de ofertas, el cuál gestionará el administrador.
MAYO	Desarrollo de recomendaciones dinámica	Requisitos secundario para implementar de forma dinámica recomendaciones de productos.

(01/05/2024-
31/04/2024)

Desarrollo del rol de
director

Dar de alta y de baja
a los admin

Desarrollo e
implementación de la
interfaz gráfica

Desarrollar un sistema
de tareas mediante un
CRUD

Implementar la API
REST

Prueba de errores y
aplicar las mejoras
necesarias para el
usuario

Termina la
documentación del
proyecto junto a la
guía de estilo

Creación del rol de director, este es
el último rol que crearé.

Panel del director para insertar y
eliminar usuarios administradores de
la BBDD.

Creación de una interfaz sólida para
el usuario teniendo en cuenta todos
los temas de accesibilidad y
usabilidad.

Desarrollar un sistema de tarea, el
cual gestionará el director.

Crear una API REST para que pueda ser
utilizada por todo el público.

Periodo de fortalecer la seguridad de
la página web, junto a mejoras
visuales, técnicas, etc.

Dejar acabada la documentación del
proyecto junto a la guía de estilo.

JUNIO
(01/06/2024-
18/06/2024)

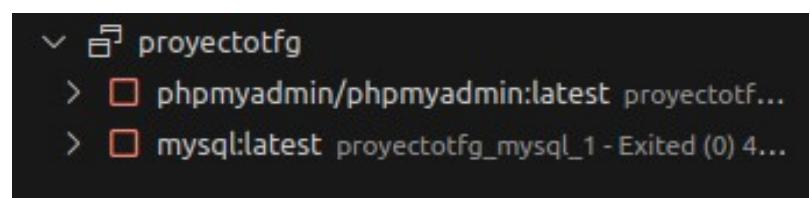
2.1.3 SERVIDOR

El servidor que utilizaré para soportar la base de datos es MySQL, con un cliente phpmyadmin, todo bajo dos contenedores lanzados en Docker, mediante un Docker Compose.

```
carmelo: ~ Escritorio ~ proyectoTfg ~ $ docker-compose.yml
version: '3.3'

services:
  mysql:
    image: mysql:latest
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: proyectoTfg
      MYSQL_USER: carmelo
      MYSQL_PASSWORD: 1234
    ports:
      - "3306:3306"
    volumes:
      - mysql_data:/var/lib/mysql:rw

  phpmyadmin:
    image: phpmyadmin/phpmyadmin:latest
    environment:
      PMA_HOST: mysql
    ports:
      - "8081:80"
    expose:
      - 80
    depends_on:
      - mysql
```



2.1.4 ENTORNOS DE PROGRAMACIÓN

Para la implementación del proyecto se han seleccionado los siguientes entornos de programación:

- React y Tailwind CSS para el frontend.
- Laravel y Laravel JWT para el backend.
- MySQL, PhpMyAdmin para la gestión de la base de datos con ejecución en Docker.

Se ha contado, con los siguientes entornos para la ayuda del desarrollo del proyecto:

- Visual Studio Code para el desarrollo del código de la aplicación.
- Docker para el lanzamiento de la BBDD.
- PostMan para la ayuda y comprobación de la API REST.
- GitHub para el control de versiones.
- StarUML para el diseño de los casos de uso, diagrama de clases,...
- PhotoShop para el diseño del logo de la aplicación y redimensión de algunas imágenes.
- Mockitt.wondershare para el desarrollo del diseño de la interfaz de usuario.
- En react la librería más importante que voy a usar es axios. Para la conexión frontend/backend.
- En laravel voy a utilizar spatie para la gestión de roles.

2.3 TAREAS IMPORTANTES REALIZADAS DEL PROYECTO

Aquí explicaré por encima, todas las tareas realizadas y su tiempo estimado, no voy a añadir muchos detalles, porque en posteriores apartados, voy a explicar cada una de las partes de forma detalladas. He de advertir, que todas las actividades de controladores, lleva consigo la creación de posibles nuevas columnas en la base de datos para almacenar nuevos datos, o creación de tablas nuevas con su correspondiente relación entre los modelos.

Simplemente es una guía resumen, de las partes más importantes del proyecto.

LUGAR	DESCRIPCIÓN	HORAS
Backend	Generar la API con enlace de autenticación con JWT.	2h
Backend	Generar migraciones y modelos en laravel.	3h
Frontend	Crear proyecto React e implementación de las librerías necesarias.	30min
Frontend	Creación de componente para el usuario sin sesión. Los componentes son: Home, Login, Register y Productos	6h
Frontend	Creación de componentes Guest y Auth, para diferenciar entre un invitado y un usuario.	2h

Frontend	Creación de componentes NavbarAuth, NavbarGuest y Footer para implementar una interfaz y rutas diferentes según tu sesión.	2h
Backend	Desarrollo de los controladores(AuthController) para la función de autenticación junto al enlace de token con la sesión(JWT).	12h
Frontend	Creación del helper pedirDatos y el componente AuthUser.jsx. PediDator.js pide los datos por axio al backend y AuthUser valida la sesión con la creación de un token.	2h
Backend	Desarrollo del controlador CategoriaController y ProductoController. Implementación de la vista de los productos y la diferenciación de ellos en categoría.	4h
Backend	Desarrollo de cada ruta en la api.php para enviar el controlador al frontend.	2h
Frontend	Creación del componente Carrito y WishList, para solo los usuarios. Implementación finalizada del carrito y wishlist.	3h
Backend	Creación de los controladores CarritoController y WishListController.	4h
Frontend	Creación del componente Detalles. Cuando pulsas un producto te lleva a la página en detalle de este con toda la información sobre este.	2h
Frontend	Creación de los componente AuthAdmin y AdminNavbar. Implementación del nuevo rol.	3h
Frontend	Creación de los componentes AdminProductos, AdminStock. Estos componentes contiene el CRUD de los controladores.	3h
Backend	Creación de los componentes StockController y TallaController. StockController contiene el crud	3h

	para el sistema de Stock y TallaController implementa el sistema de talla en los productos.	
Frontend	Implementación de plantillas con tailwind, de manera provisional.	3h
Backend	Creación del controlador OfertaController, para crear un sistema de gestión de ofertas, entre otras funcionalidades.	4h
Frontend	Ajuste en los controladores donde se ve el precio, para aplicar la lógica del sistema de ofertas.	1h
Backend	Desarrollo del controlador adminController para la gestión de la creación del usuario bajo el rol de "administrador".	2h
Frontend	Creación de los componentes en la carpeta Administrador y los componentes AdminPanel, CrearProducto, EliminarProducto, ListaTarea, CrearOferta, AsignarOfertas, ListDeOfertas y AsignarOfertasPorCategorias AdminStock para toda la funcionalidad respecto al panel de administrador.	10h
Backend	Creación del controlador FacturaController para crear la lógica en la creación de un recibo del cliente.	2h
Frontend	Creación del componente Facturas para sacar los datos al frontend de los datos de compra. Además de su correspondiente aplicación en el componente Perfil para visualizarlo desde ahí.	1h
Backend	Creación del controlador Tarea para la creación de un sistema de tareas del director hacia los administradores.	1h
Frontend	Creación de CrearTarea, VerTareas, AñadirForm y EditarAdmin para crear el sistema de administración para el rol "director"	5h
Backend	Creación del controlador PUBLICAPICONROLLER para la gestión de la API para desarrolladores de la página, más su comprobación en PostMan para comprobar su correcto funcionamiento.	2h
Frontend	Creación del componente APIRest para explicar el funcionamiento de la API.	2h
Frontend	Diseño de la página.	40h

CAPÍTULO 3 – ANÁLISIS DEL PROYECTO

3. Introducción

En este apartado se procederá a explicar en detalle las características del proyecto, esto implica tanto sus requisitos funcionales, como los no funcionales, además de la explicación y exposición de los casos de uso.

3.1 ROLES DE USUARIO

En la aplicación tendrás unas funciones u otras dependiendo del rol del usuario.

- Usuario: El usuario podrá comprar, añadir a la cesta, agregar a favoritos los producto que él deseé, ver los productos en la cesta, procesar pagos de los productos, ver su perfil y editar sus datos.
- Invitado: El invitado solo podrá únicamente y exclusivamente, ver lo que aparece en la tienda, es decir, no podrá realizar ninguna de las acciones del usuario, si el invitado desea realizar alguna de esas acciones, debe de registrarse.
- Administrador: Los administradores los cuales se consideran los empleados de la tienda, podrá añadir nuevos productos a la categoría de prenda que ellos deseen. También podrá eliminar productos de la página. Los administradores tendrá una panel con la información del stock de todos los productos y tendrán una opción para reponer el stock en caso de que exista este en la reserva.

Y por último, existirá una pestaña donde habrá tareas pendiente asignada por el director.

- Director: Tendrá un panel para eliminar administradores del sistema, editar los datos del administrador y dar de alta a nuevos administradores. Además, tendrá una pestaña para asignar actividades a los administradores, lo cual, puede eliminar o editar en un futuro.

3.2 REQUISITOS FUNCIONALES

3.2.1 GESTIÓN DE ADMINISTRADORES.

El director tiene en su panel, la posibilidad de dar de alta y de baja a los administradores, junto, a la posibilidad de editar los datos de los administradores en la BBDD.

3.2.2 AÑADIR TAREA EN LA LISTA

El director tendrá una pestaña para poder colgar tareas que deben de realizar los administradores como trabajo. Es decir, si el director necesita que un nuevo producto sea subido, esté escribirá una tarea, que será subido a una pestaña para asignárselo a un administrador en concreto.

Estas actividades podrán ser editadas por el director, en caso de querer corregir o añadir algo nuevo a la tarea.

Documentación – Análisis del proyecto

Por último, el director podrá eliminar la tarea, bien si la tarea ya está completada, o porque ya no hay necesidad de realizar esa actividad.

3.2.3 VISUALIZACIÓN PANEL DE TAREAS

Los administradores tendrá un panel para visualizar las tareas correspondiente a realizar. Una vez, se complete la tarea, el administrador tendrá la capacidad de marcar la tarea como realizada. Si esa actividad está realizada, le desaparecerá a los administradores y solo podrá ver que está completada el director.

3.2.4 AÑADIR/EDITAR/ELIMINAR PRODUCTOS

Los administradores tienen la capacidad de crear nuevos productos, editar esos productos, o editar lo ya existente y eliminar productos de la tienda.

3.2.5 VISUALIZAR STOCK EN TIENDA Y REPONER ESTE

Los administradores tendrán un panel con la información del stock que hay en tienda.

Dependiendo del nivel del stock que haya, los administradores podrán intervenir en el stock o no. Es decir, si el stock de algún producto es 0, los administradores puede añadir más stock, dependiendo si en la reserva de la BBDD hay existencias. Por el caso contrario, si hay stock en la tienda, simplemente no se podrá realizar ninguna acción.

3.2.6 CREACIÓN DE UN USUARIO.

Los visitante de la página tendrá la posibilidad de crear una cuenta que se almacenará en la BBDD, para acceder a todas las funcionalidades de un usuario.

3.2.7 COMPRA DE PRENDAS USUARIO

Los usuarios, una vez hayan iniciado sesión, podrán añadir al carrito o comprar directamente la prenda que necesiten. En el momento, que se proceda a comprar el producto, tendrás la opción de pagar con PayPal o Stripe.

3.2.8 GESTIÓN DE CARRITO

Los usuarios pueden añadir varios productos al carrito de compra antes de proceder al pago, y con ello, comprar varios productos al mismo tiempo. Los usuarios, tienen la capacidad de ver lo que hay en el carrito y eliminar una prenda si ya no la quieren comprar.

3.2.9 COMPRA DIRECTA

Los usuarios que no quieran comprar varias prendas, podrán comprar directamente, sin necesidad de acceder al carrito. Esto será montado de tal forma, que cuando le des a comprar prenda, te aparecerá la opción de comprar directamente, o de añadir al carrito de compras.

3.2.10 PROCESO DE PAGO.

Una vez, procesa con tu elección de pago, se informará en la misma página de tu compra realizada, y se actualizará en la base de datos el ingreso de dicho dinero. Además, de llevar un registro de las transacciones, incluyendo productos

comprados, método de pago utilizado y hora y fechas de las transacciones.

3.2.11 CANTIDAD DE PRENDAS A COMPRAR

El usuario tendrá la decisión de elegir la cantidad de prendas a comprar de un mismo producto. Para ello, se realizará una validación y se mostrará al usuario la cantidad de stock que haya en ese momento(si en el stock hay 10 camisetas, el usuario solo podrá comprar 10 camisetas). Posteriormente, se actualizará en la BBDD el stock actualizado.

Si el usuario trata de seleccionar una cantidad mayor o una negativa, se ocultará o deshabilitará el botón, para no permitir una interacción cliente-servidor.

3.2.12 ELECCIÓN AL COMPRAR UNA PRENDA

El usuario podrá seleccionar la talla(XS,S,L,XL). Si alguna de estas prendas no están disponible, no aparecerán como opción a la hora de elegir talla. Esto se hará en torno a si se encuentran tallas o no en la BBDD.

3.2.13 ACTUALIZACIÓN DINÁMICA DEL STOCK

De cara a esta página web, debemos de programar eventos en Laravel, los cuales escuchado en el frontend(React en este caso), para poder, actualizar todas las funciones de stock a tiempo real en la página web.

3.2.14 MOSTRAR RECOMENDACIONES

Cuando el usuario, esté en la página de una prenda seleccionada, aparecerá debajo de toda la información de la prenda, más productos relacionados con la prenda que está comprando. Para ello, necesitamos aplicar un algoritmo, este algoritmo será programado en función del producto que estás visualizando en este momento, es decir, si estás comprando camisas, se recomendarán camisas. Esto estará situada debajo de la información de una prenda la cual estará visualizando el usuario en ese momento.

3.2.15 WISHLIST

El usuario podrá añadir a favoritos una prenda que deseé comprar en el futuro. Tal y como la agrega a favoritos, también puede eliminarla de la lista.

El usuario podrá compartir su lista de deseo en redes sociales, lo cual, se hará gráficamente mediante un botón.

3.2.16 SISTEMA DE OFERTAS

Los administradores, tendrán una pestaña para crear una oferta de un producto, seleccionando de todos los existentes. Podrán definir los detalles de la oferta, como el descuento aplicado y la duración de la oferta. Una vez creado la oferta, el producto asociado tendrá un indicador de que tiene una oferta activa.

Cuando el producto tiene una oferta activa, el precio del producto se actualizará automáticamente, con ello, se le dará prioridad al precio de oferta ante el precio original.

3.2.17 CONFIGURAR REGISTRO E INICIO DE SESIÓN

A partir del registro e inicio de sesión por defecto de Laravel, modificarlo para ajustarlo al resto de características. Además, debo de validar y sanitizar

los datos ingresados por el usuario para prevenir inyecciones de código. Proteger todas las rutas relacionadas con el inicio de sesión mediante el middleware de autenticación.

Configurar el token de usuario para realizar la autenticación en las solicitudes de la API, asegurando un acceso seguro a los recursos protegidos.

3.2.18 PERFIL DE USUARIO

El usuario podrá acceder a su perfil para modificar y cambiar alguno de los datos de su perfil mediante un formulario. Antes de guardar los datos, se solicitará la confirmación del usuario para actualizar de forma definitiva el perfil.

3.2.19 API REST

Hacer una API REST para que los usuarios la puedan usar a su antojo. La API tendrá información sobre todos los productos más las mejores ofertas en ese momento de los productos.

3.3 REQUISITOS NO FUNCIONALES

3.3.1 SEGURIDAD

Garantizar que el sistema de autenticación y autorización sea robusto y seguro antes las inyecciones y el cifrado de contraseña y el manejo de tokens.

Proteger las rutas relacionadas con el inicio de sesión mediante middleware.

3.3.2 USABILIDAD

Desarrollar una interfaz de usuario intuitiva y fácil de usar que permita a los usuarios navegar y utilizar todas las funcionalidades del sistema de manera eficiente y sin confusiones.

Aseguramos que la aplicación sea accesible para todos los usuarios, incluyendo aquellos con discapacidades visuales o motoras, mediante accesibilidad como lectores de pantalla y soporte para navegación por teclado.

3.4 CASOS DE USO

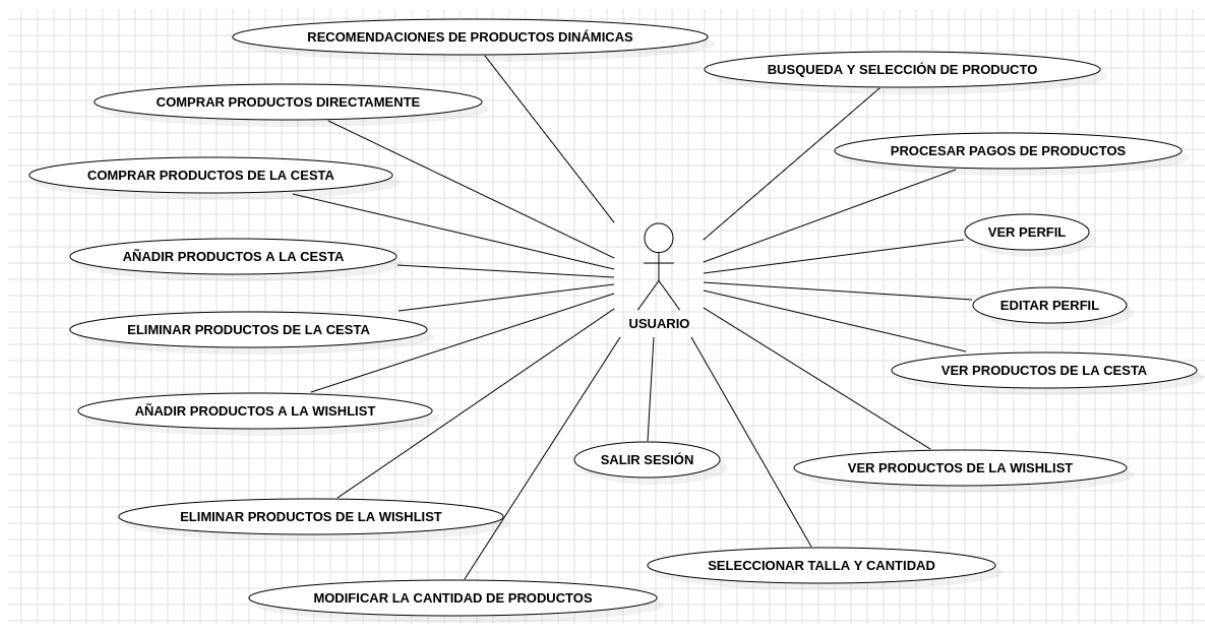
3.4.1 ROL DE INVITADO

El rol de invitado solo puede navegar por la tienda, visualizar en detalle los productos, registrarte y iniciar sesión en la web. No puedes almacenar nada en el carrito salvo que se inicie sesión, no como en otras páginas web.



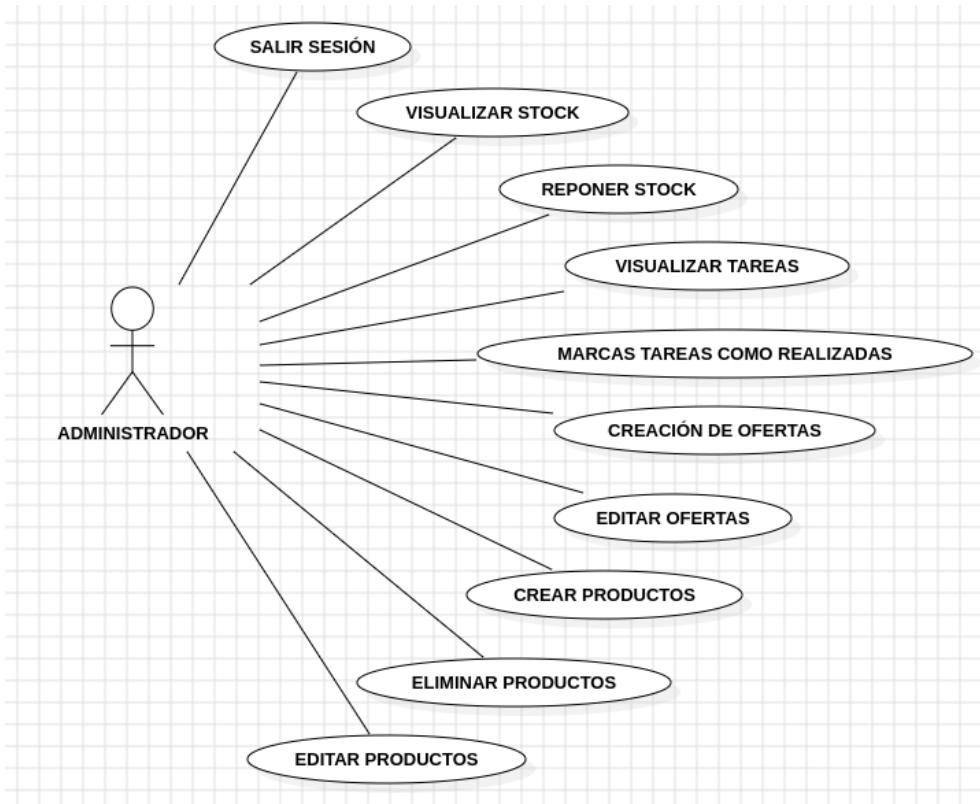
3.4.2 ROL DE USUARIO

EL rol de usuario puede añadir y eliminar productos en la lista de deseos y en la cesta. En la cesta puede seleccionar su talla y cantidad, además de procesar el pago. Por último, puede acceder a su perfil para modificar sus datos y visualizar otros datos, y cerrar su sesión. Todo esto lo diferencia del rol de invitado.



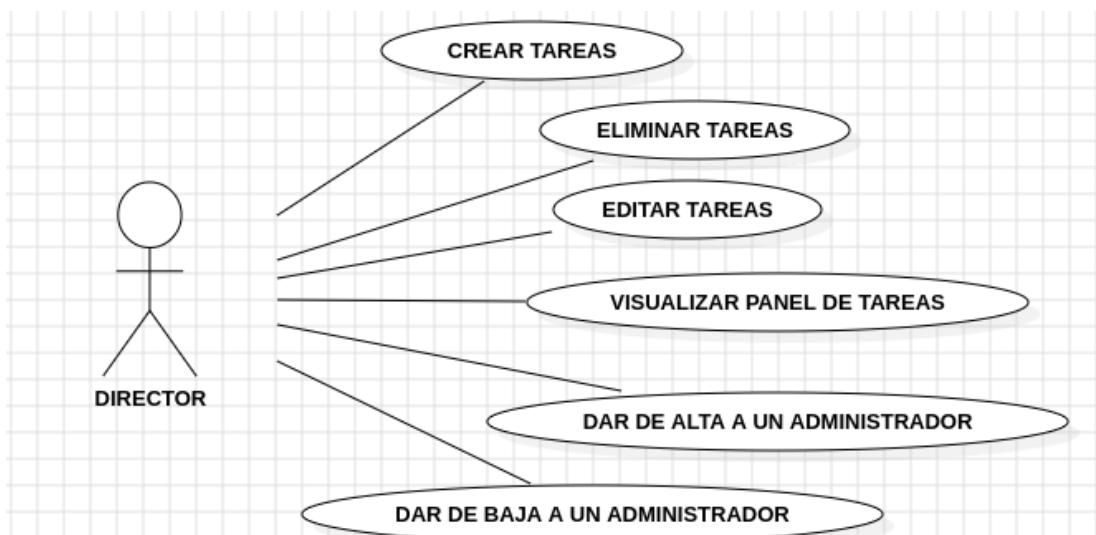
3.4.3 ROL DE ADMINISTRADOR

El rol de administrador puede visualizar y reponer el stock de todos los productos. También puede crear, editar y modificar ofertas para los productos. Puede crear productos, eliminar y editar productos en la página web. Además puede ver las tareas que le asigna su jefe para realizar.



3.4.4 ROL DE DIRECTOR

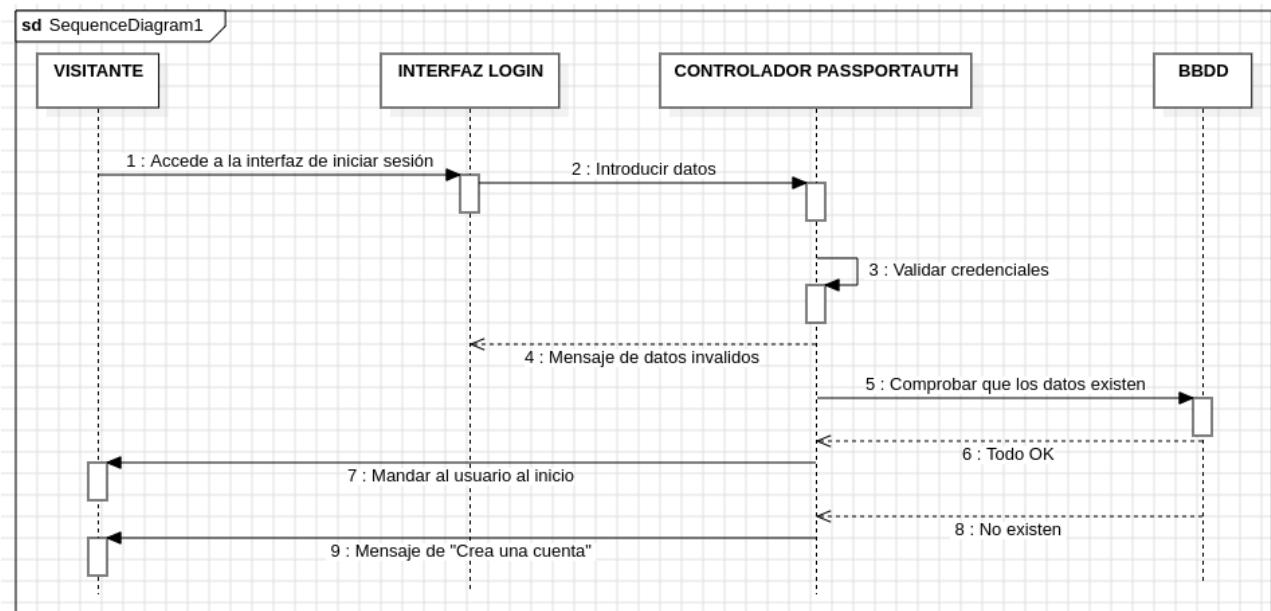
El rol de director puede crear, eliminar y editar tareas y visualizarlas. Tiene acceso a los datos de administrador y su correspondientes modificación, además de darles de baja en la página web.



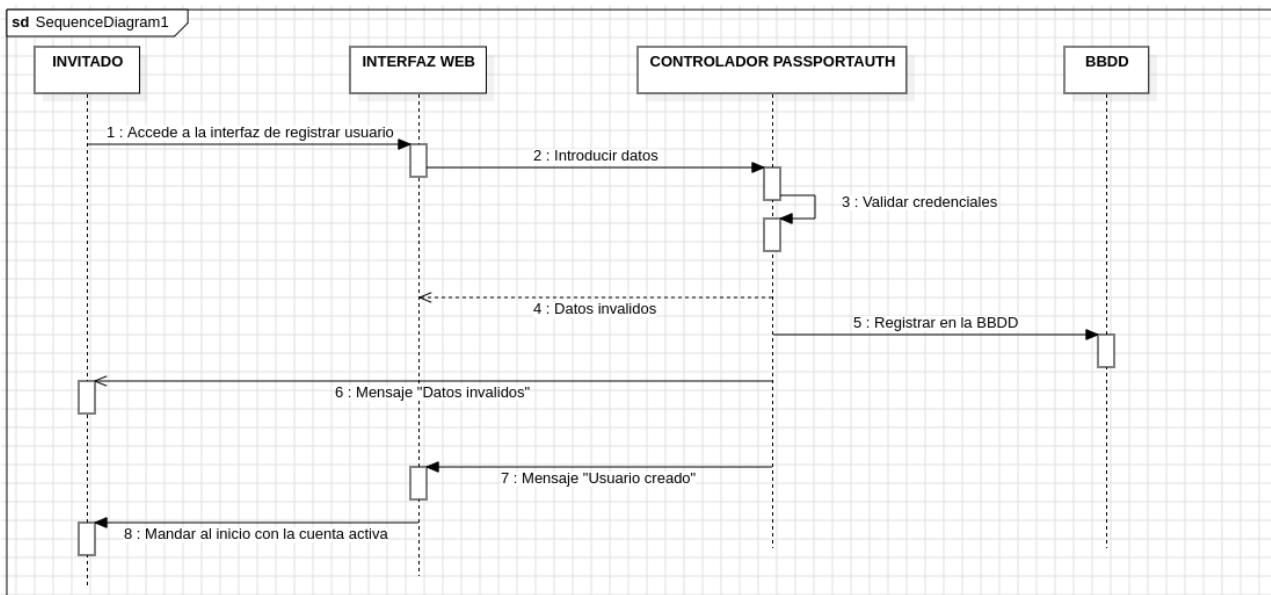
3.5 DIAGRAMA DE SECUENCIA

3.5.1 ROL INVITADO

3.5.1.1 INICIAR SESIÓN EN LA PÁGINA

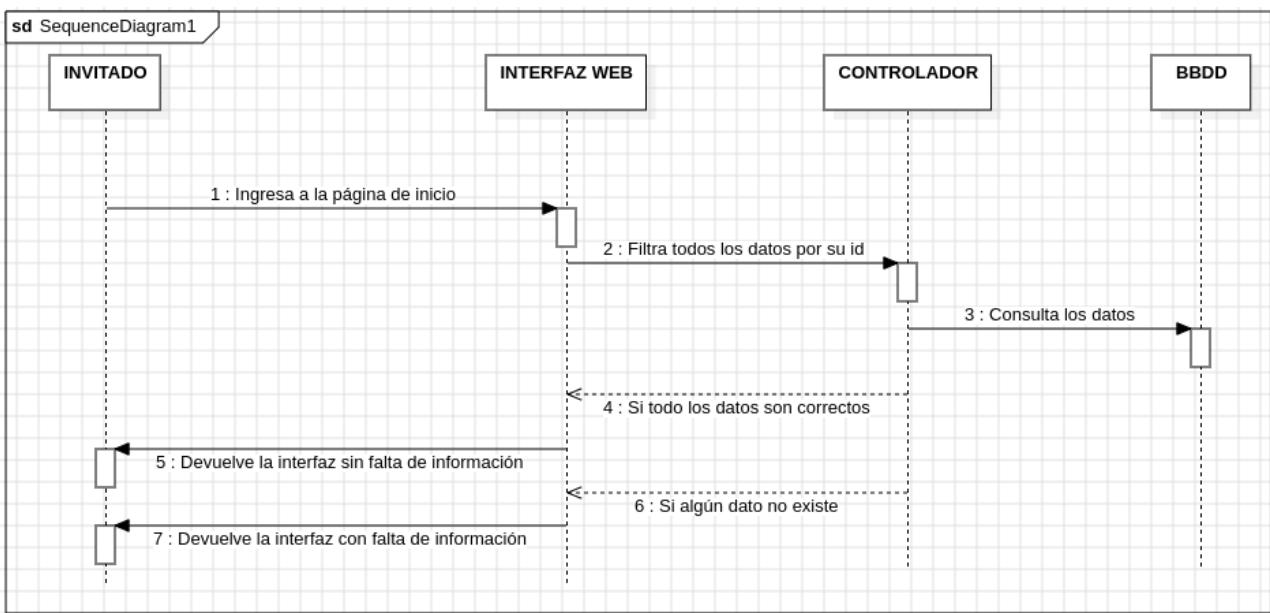


3.5.1.2 REGISTRO EN LA PÁGINA

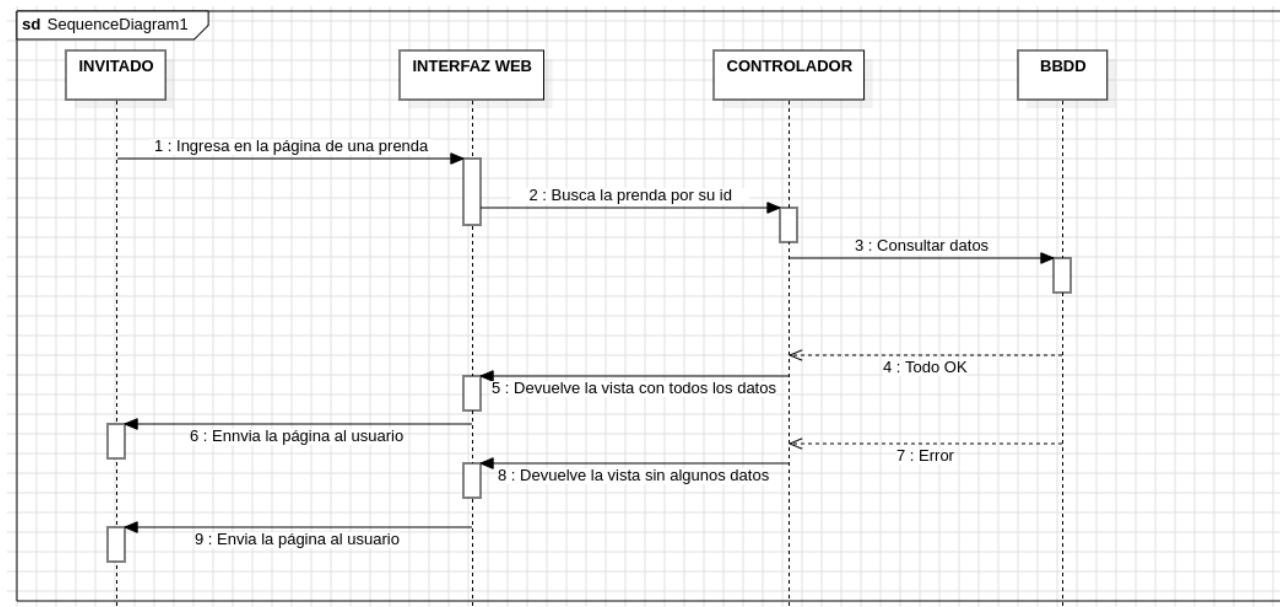


Documentación – Análisis del proyecto

3.5.1.3 NAVEGACIÓN POR LA TIENDA

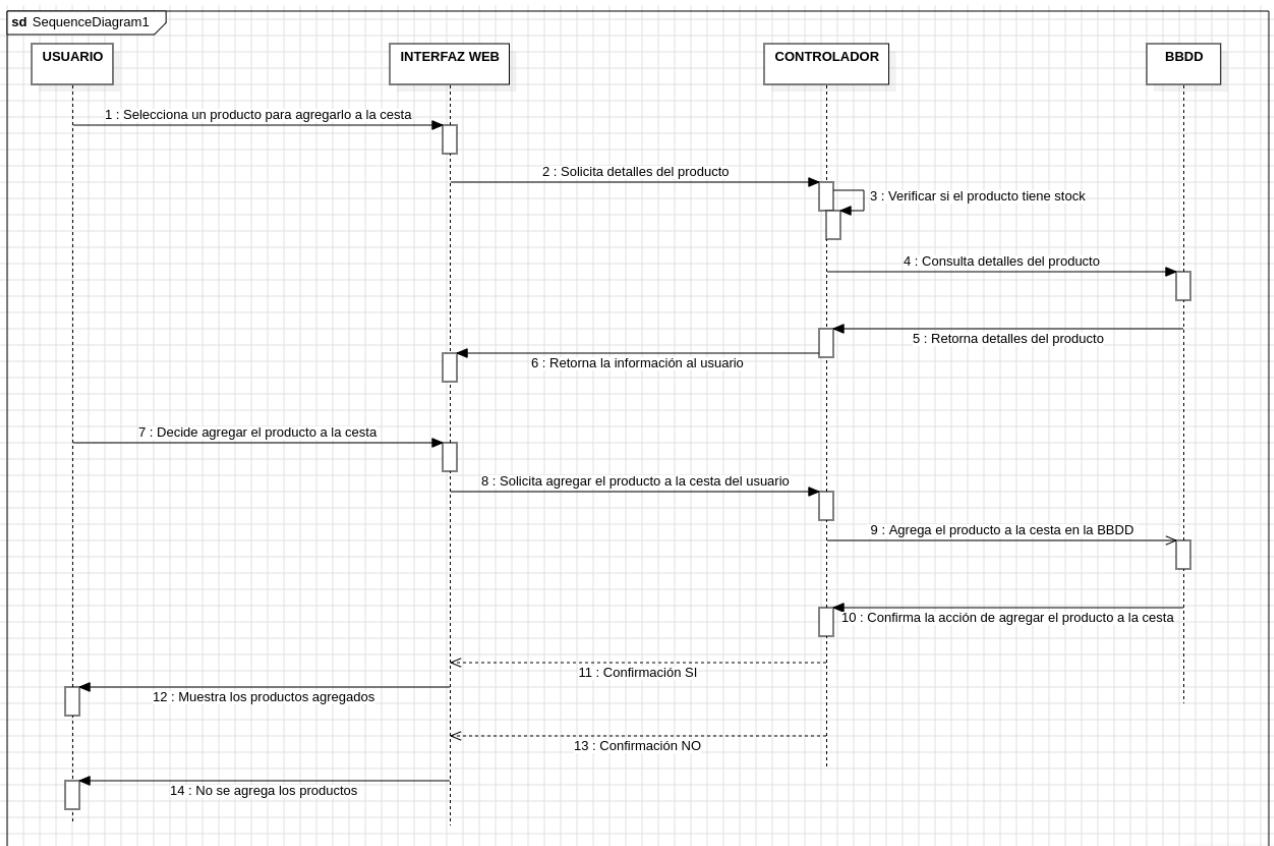


3.5.1.4 VISUALIZACIÓN EN DETALLE DEL PRODUCTO

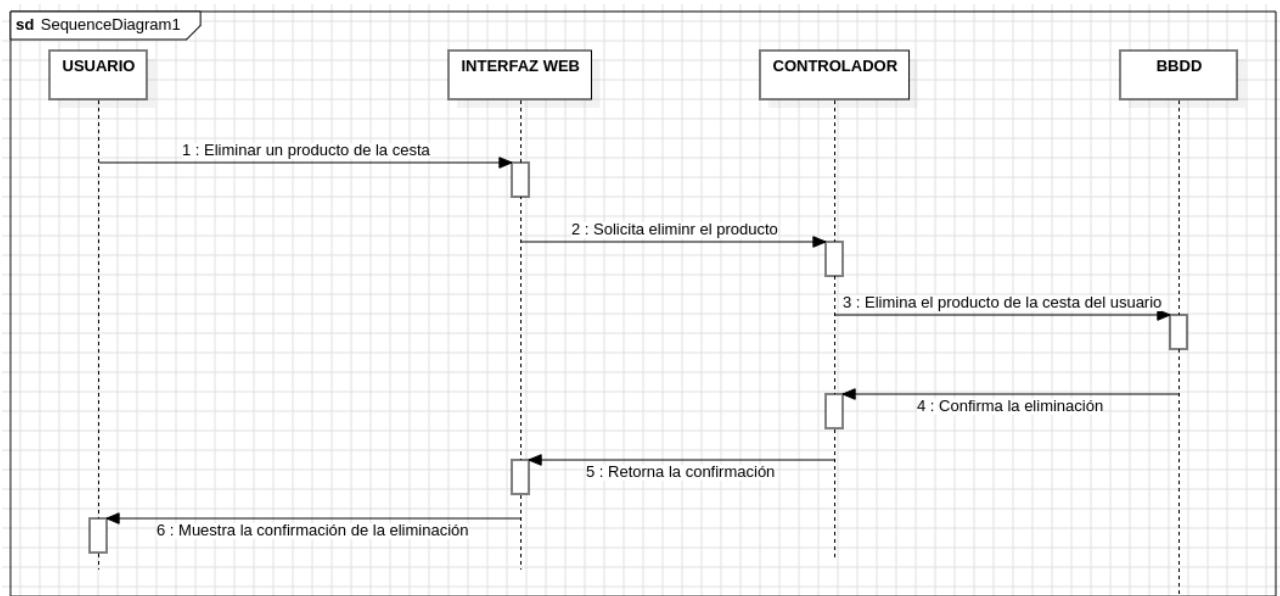


3.5.2 ROL USUARIO

3.5.2.1 AÑADIR PRODUCTOS A LA CESTA

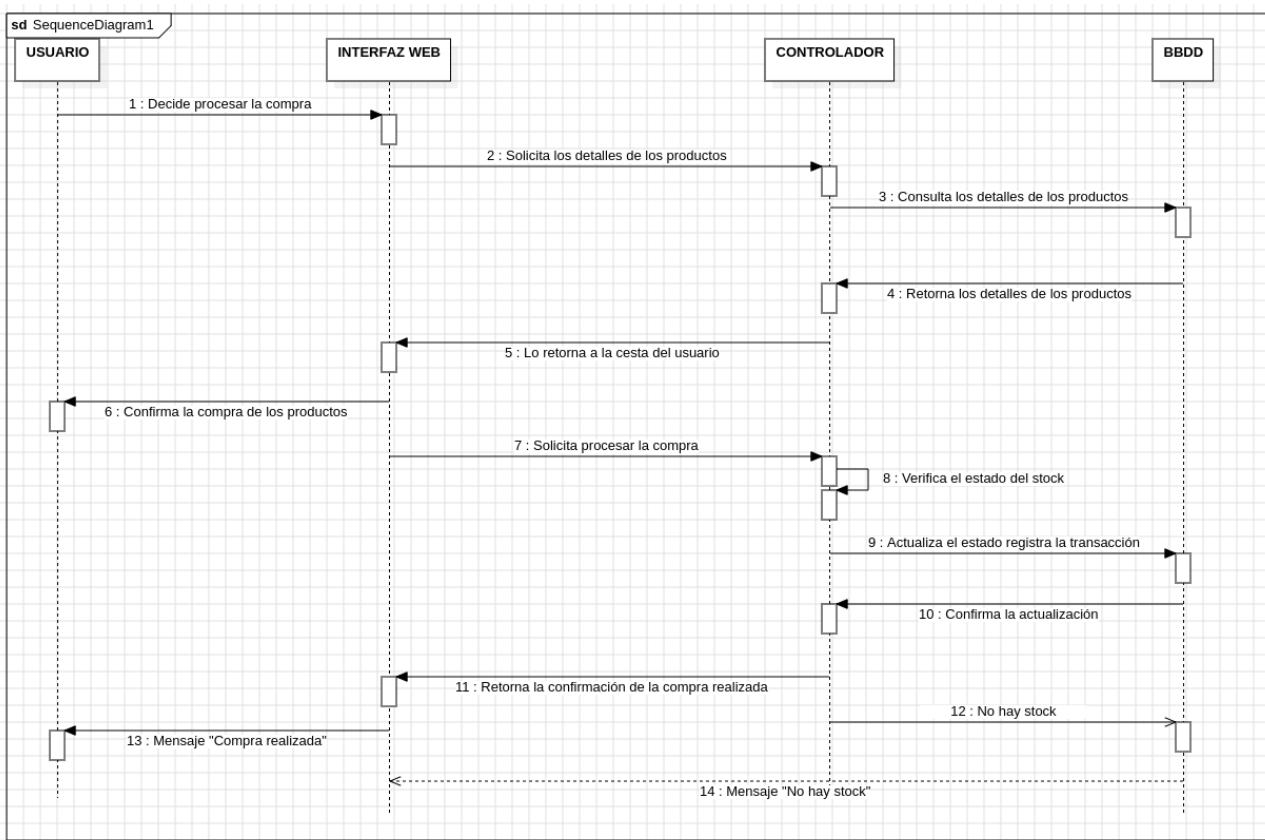


3.5.2.2 ELIMINAR PRODUCTOS DE LA CESTA

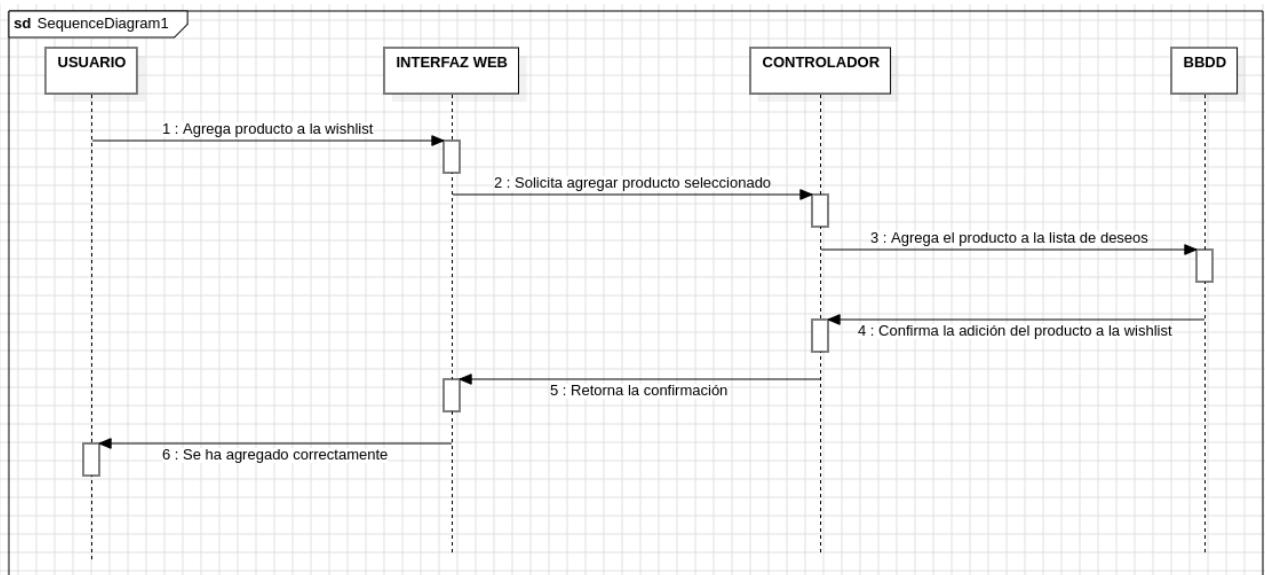


Documentación – Análisis del proyecto

3.5.2.3 COMPRAR PRODUCTOS DE LA CESTA

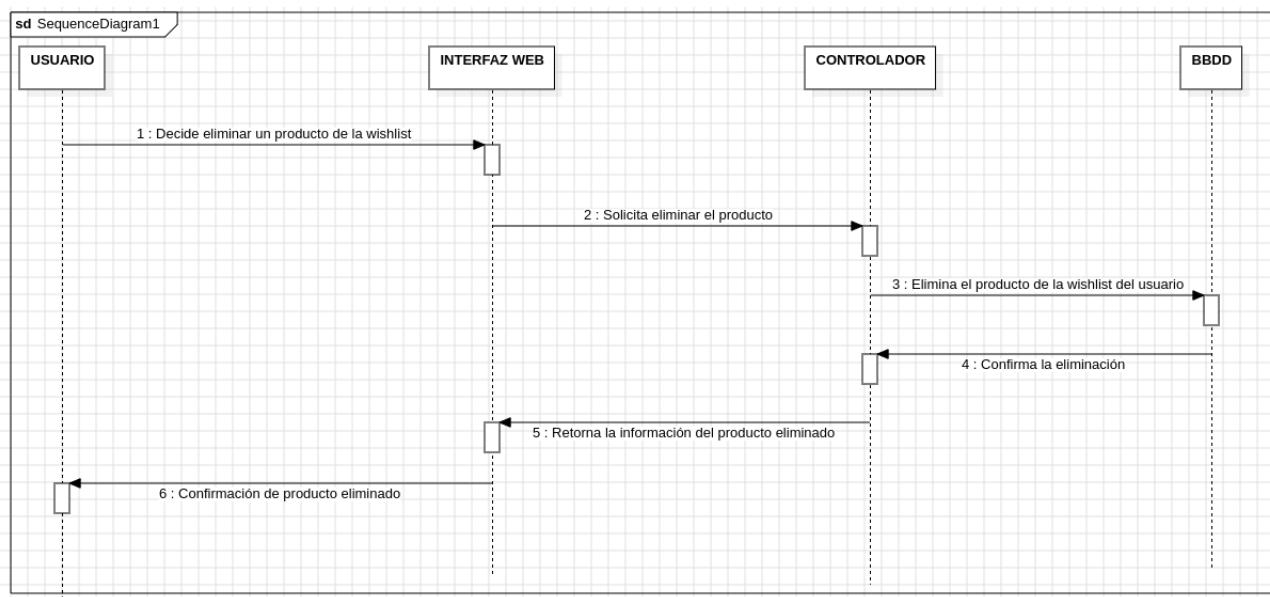


3.5.2.4 AÑADIR PRODUCTOS A LA WISHLIST

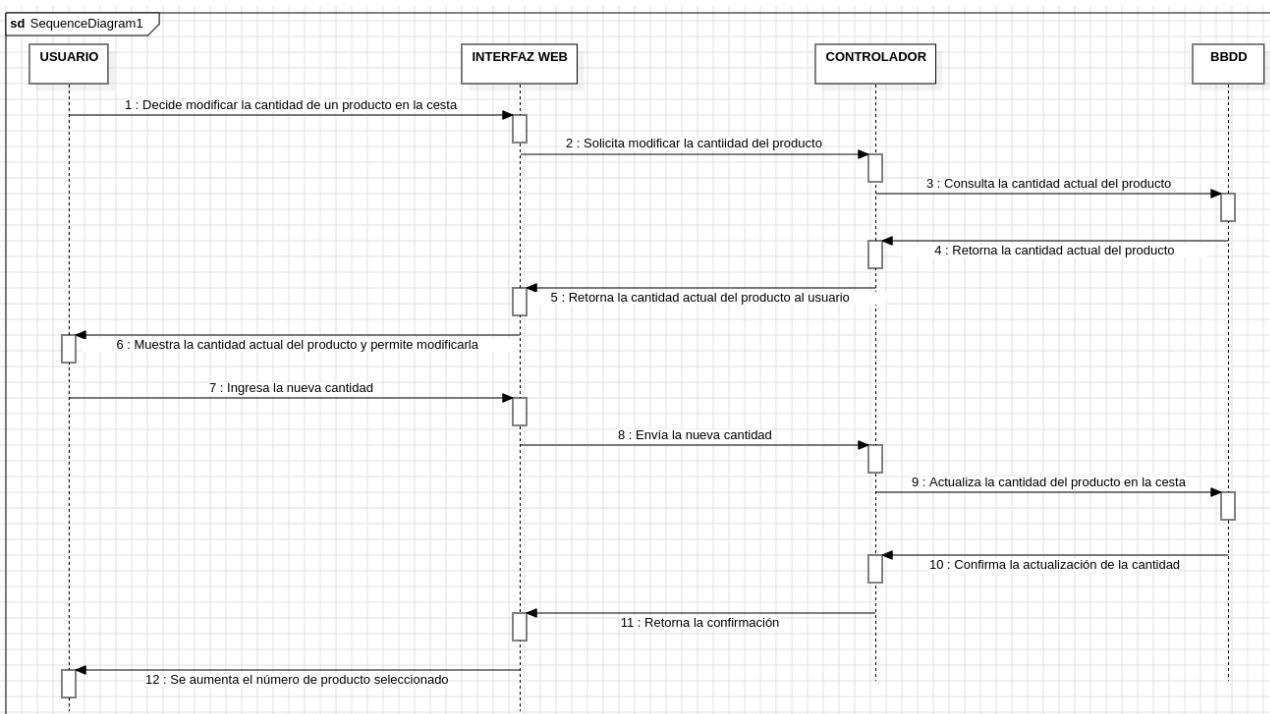


Documentación – Análisis del proyecto

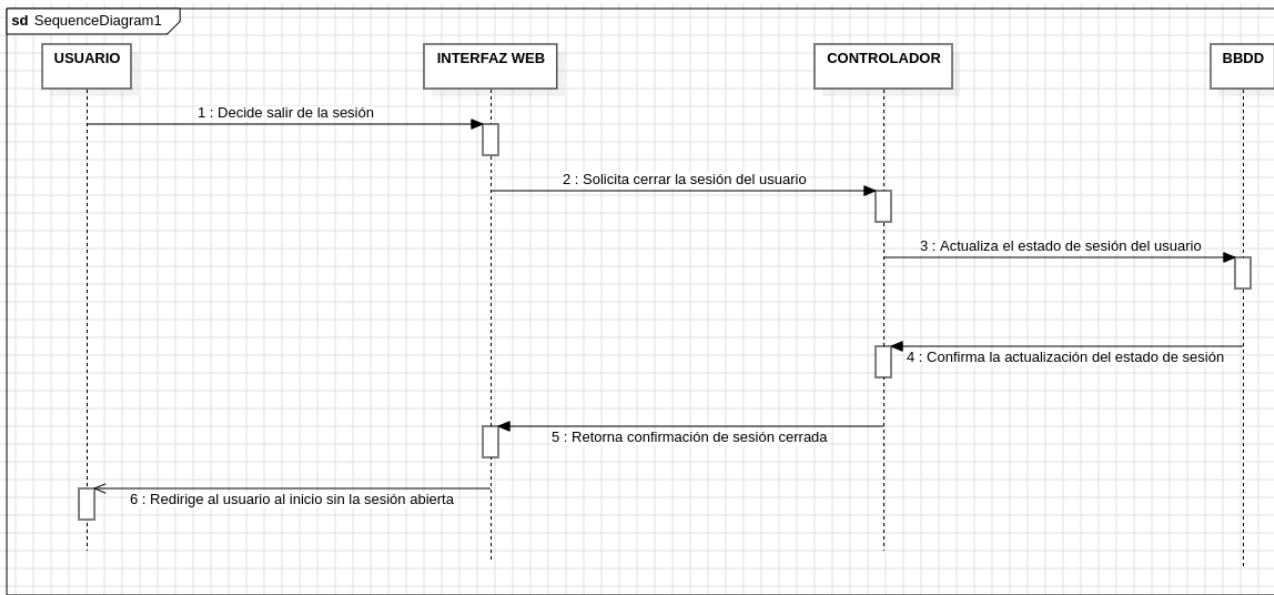
3.5.2.5 ELIMINAR PRODUCTOS DE LA WISHLIST



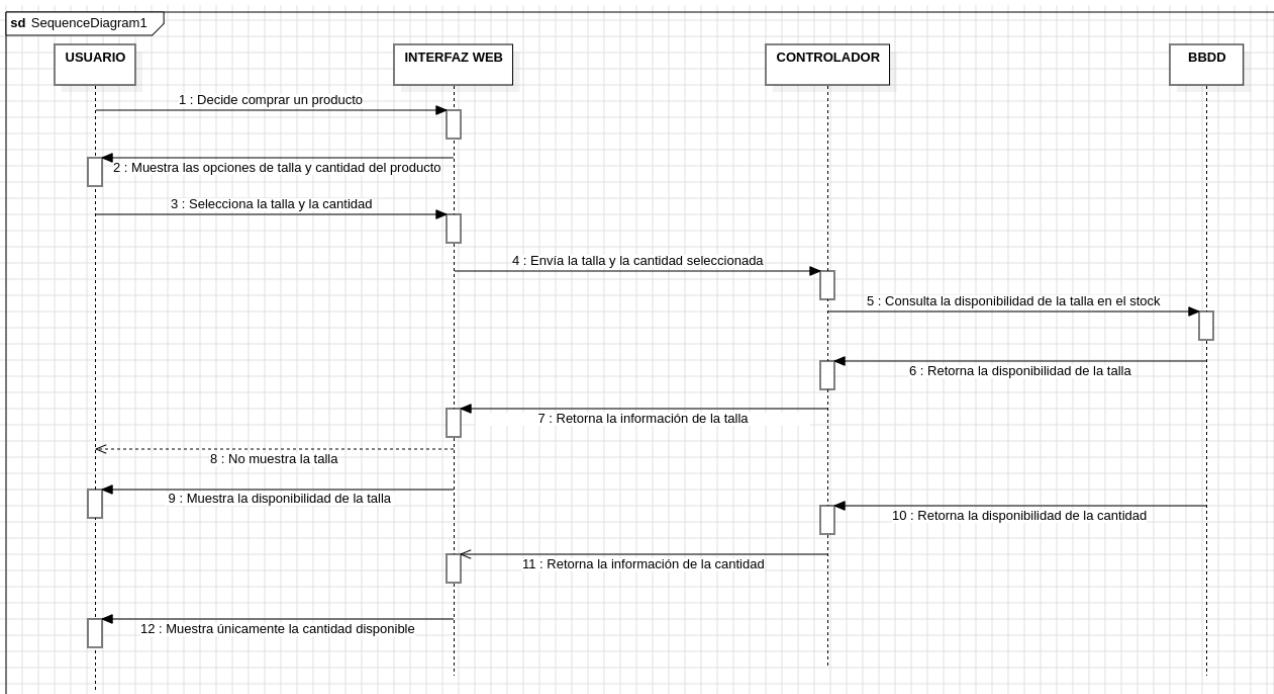
3.5.2.6 MODIFICAR LA CANTIDAD DE PRODUCTOS



3.5.2.7 SALIR SESIÓN

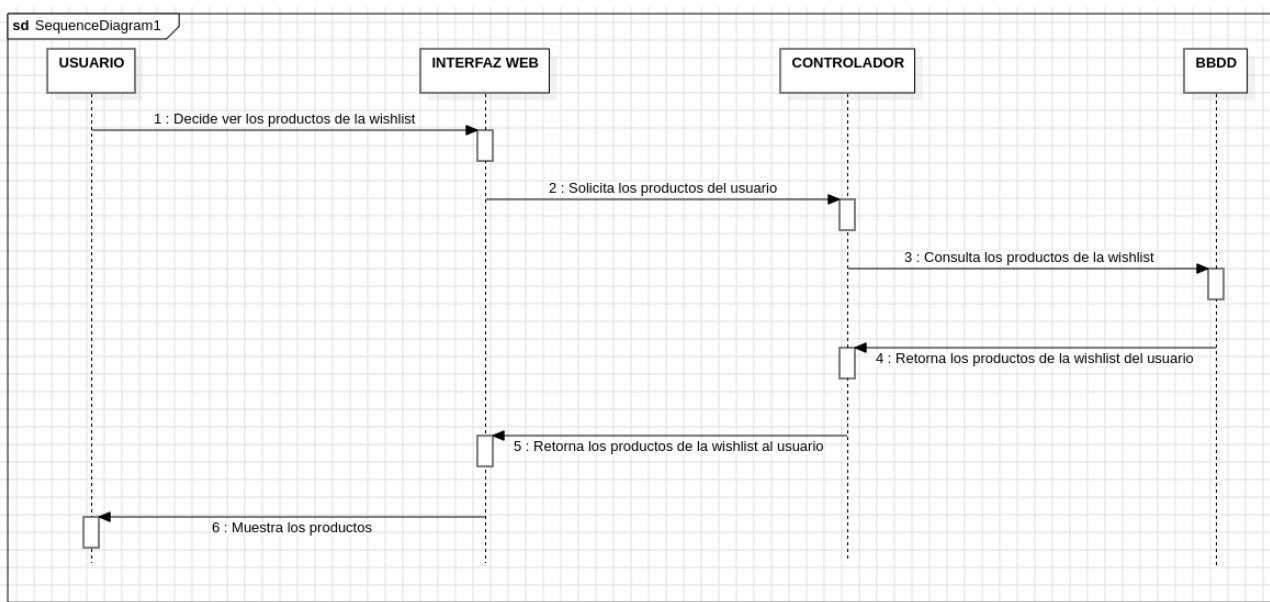


3.5.2.8 SELECCIONAR TALLA Y CANTIDAD PARA COMPRAR

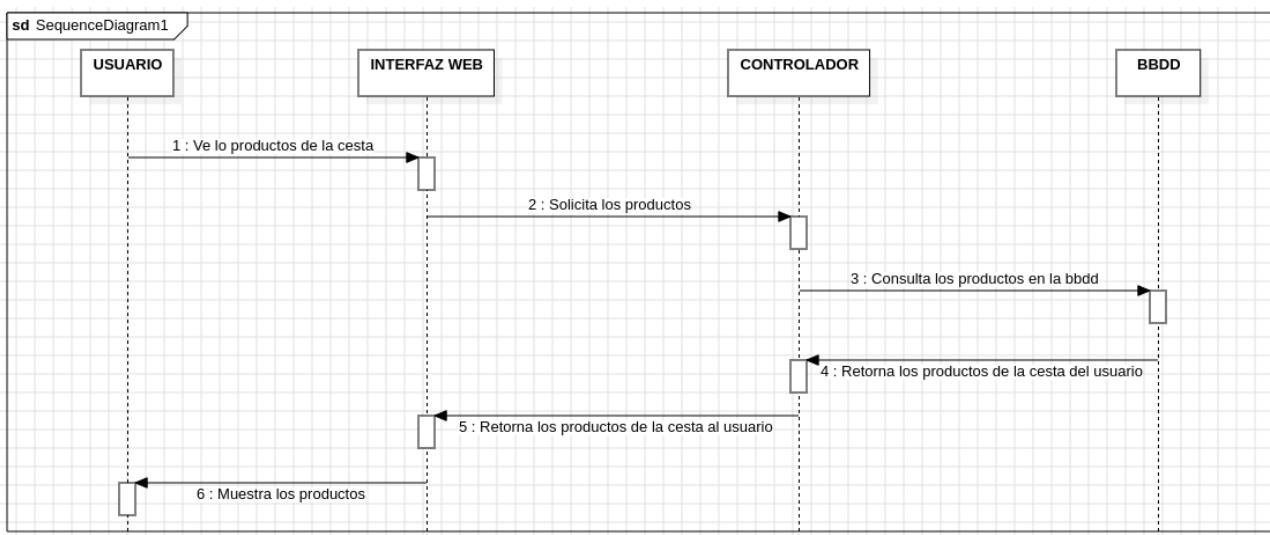


Documentación – Análisis del proyecto

3.5.2.9 VER PRODUCTOS DE LA WISHLIST

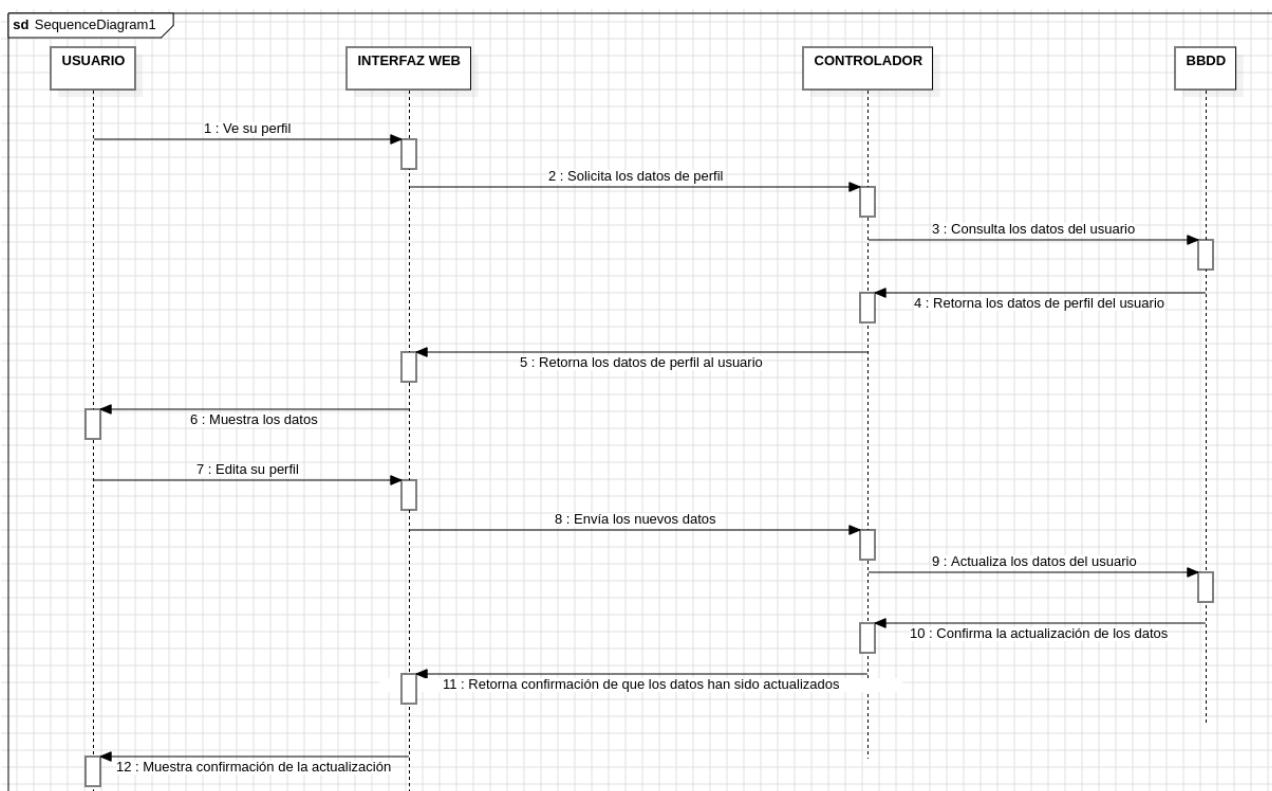


3.5.2.10 VER PRODUCTOS DE LA CESTA

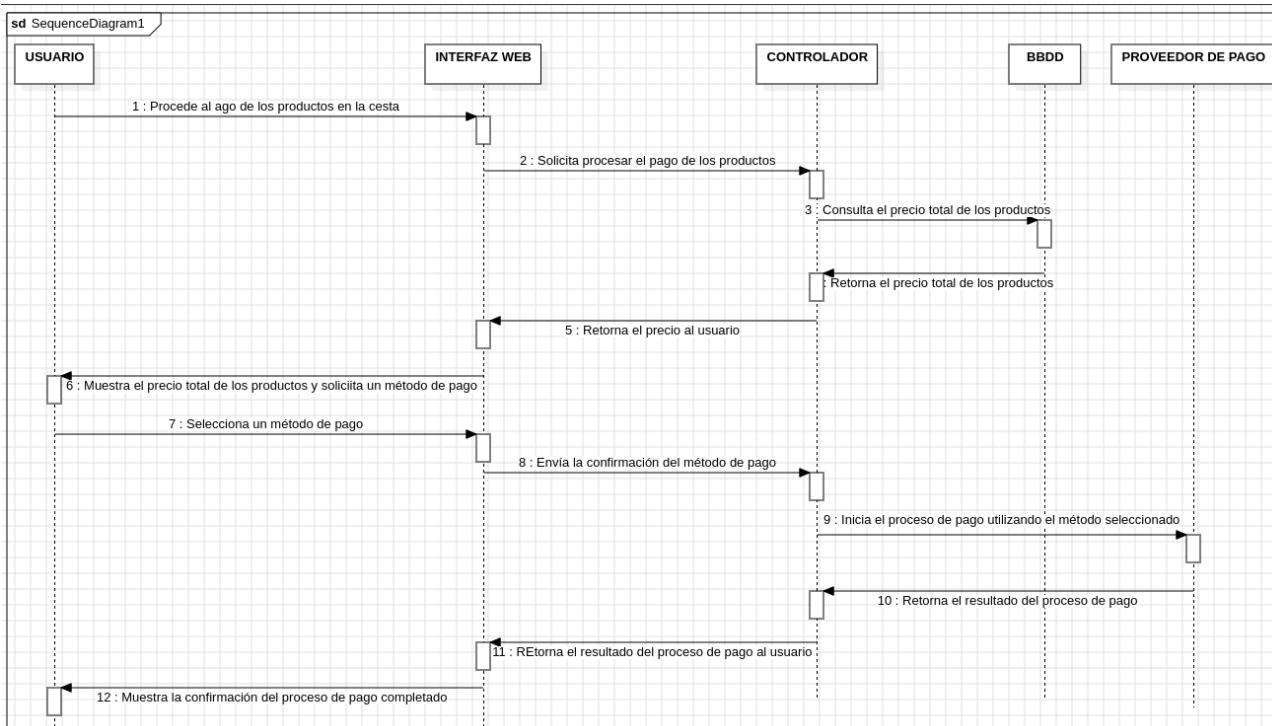


Documentación – Análisis del proyecto

3.5.2.11 VER Y EDITAR PERFIL

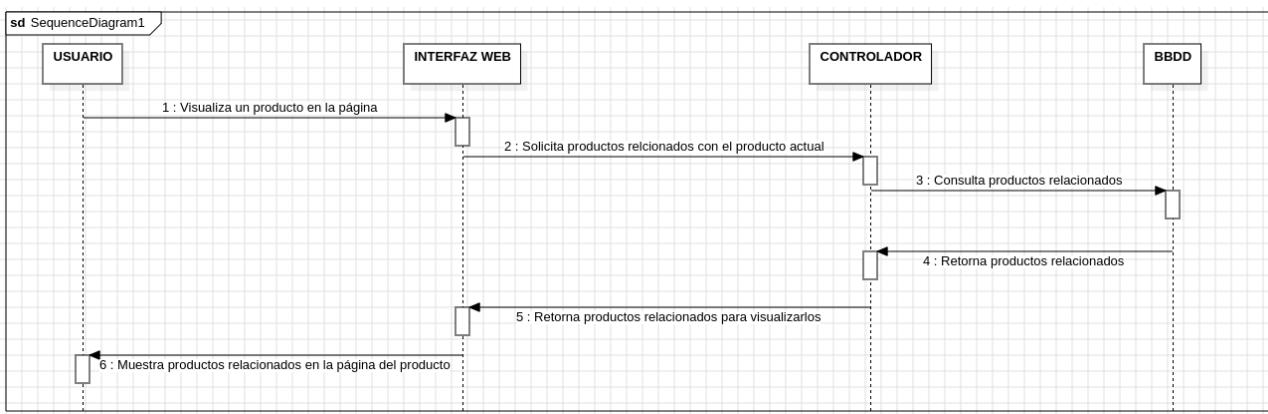


3.5.2.12 PROCESAR PAGOS DE PRODUCTOS

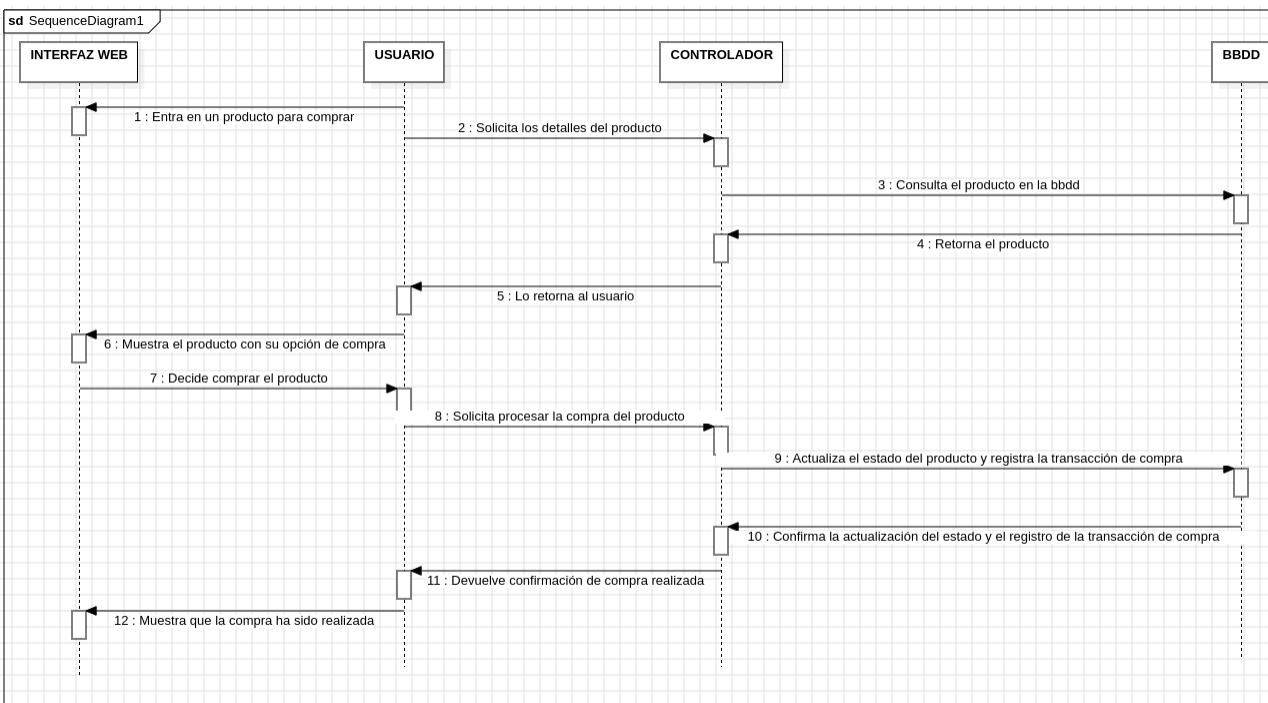


Documentación – Análisis del proyecto

3.5.2.13 RECOMENDACIONES DE PRODUCTOS DINÁMICAS



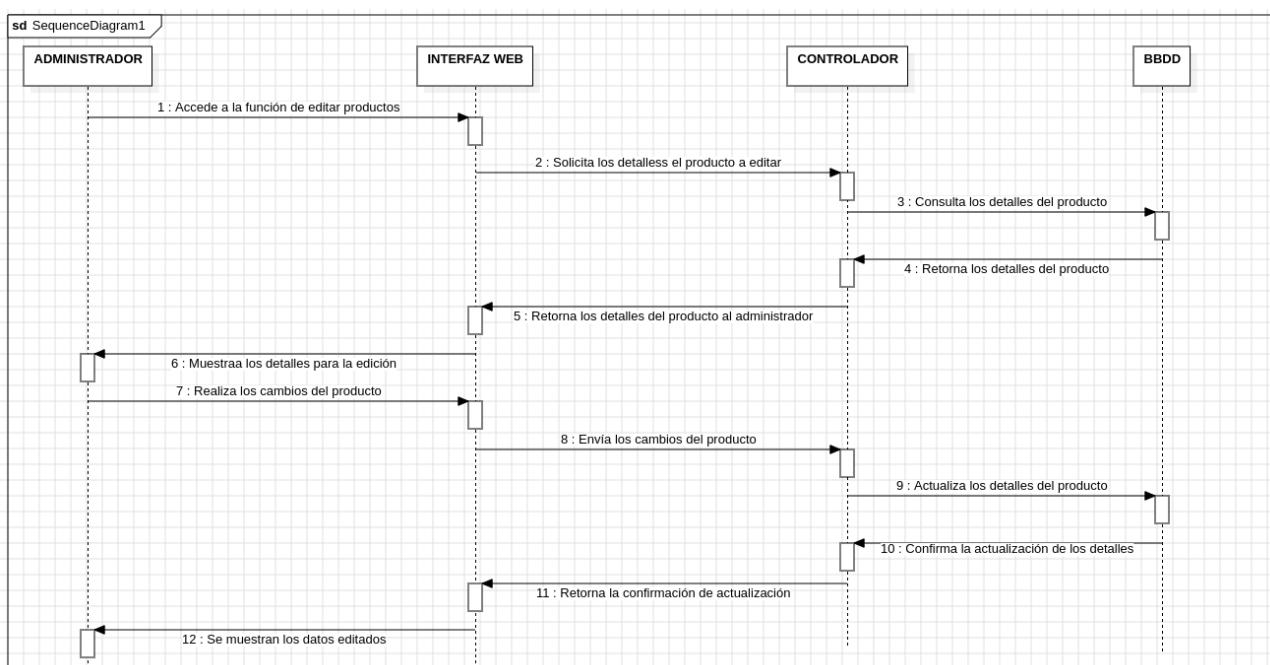
3.5.2.14 COMPRAR PRODUCTOS DIRECTAMENTE



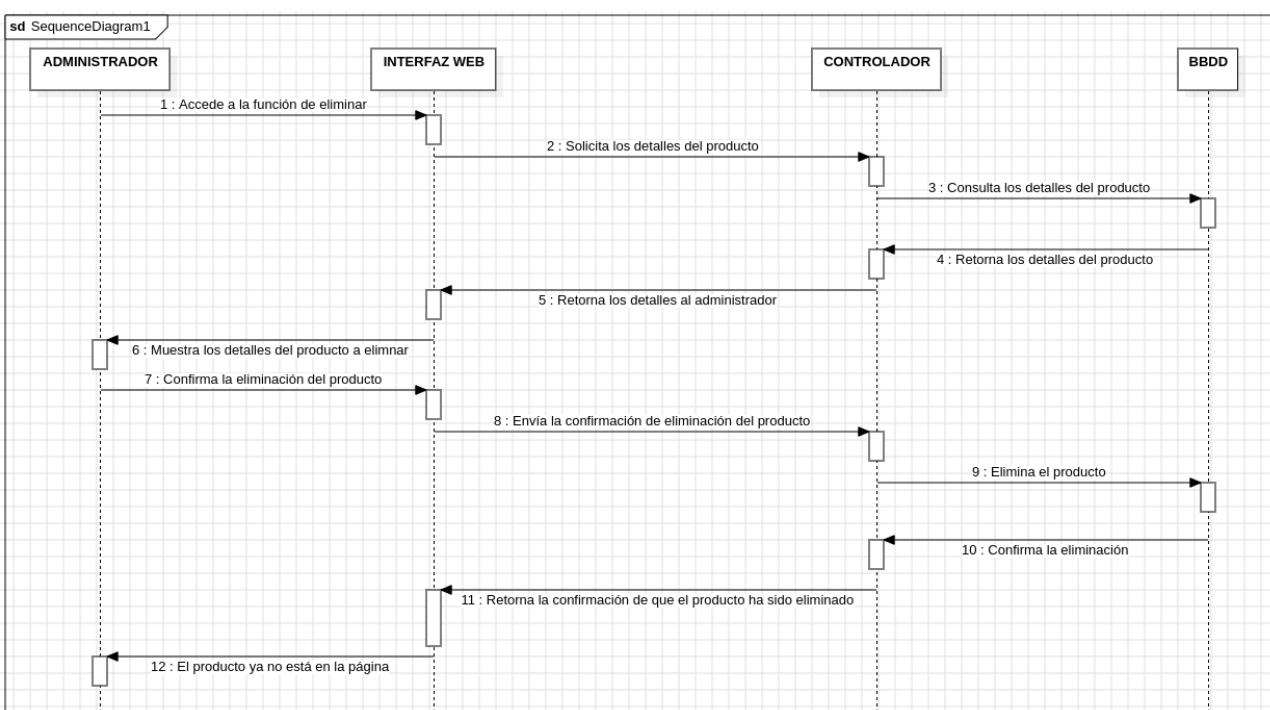
Documentación – Análisis del proyecto

3.5.3 ROL ADMINISTRADOR

3.5.3.1 EDITAR PRODUCTOS

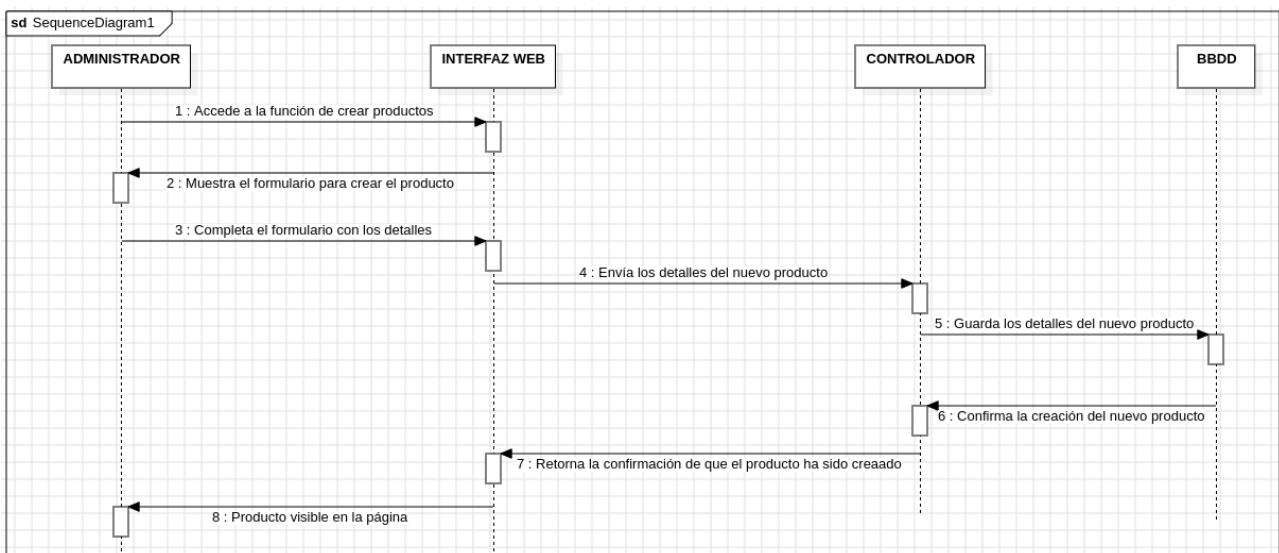


3.5.3.2 ELIMINAR PRODUCTOS

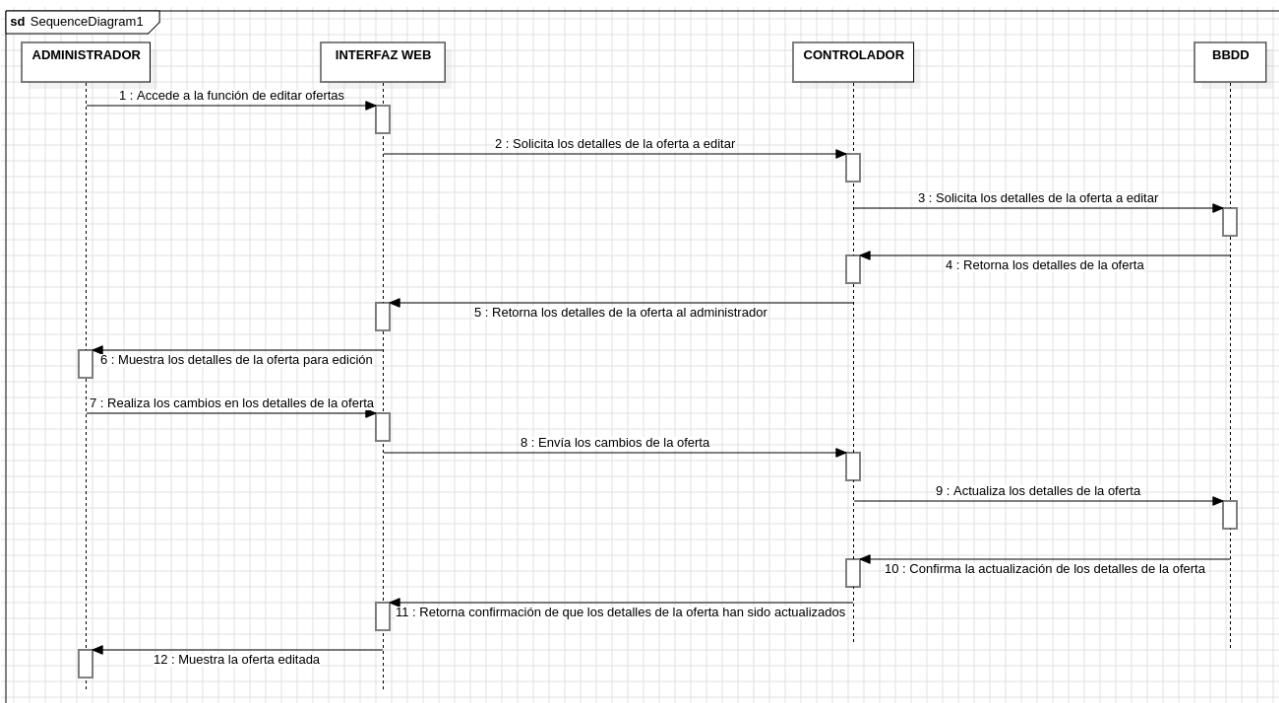


Documentación – Análisis del proyecto

3.5.3.3 CREAR PRODUCTOS

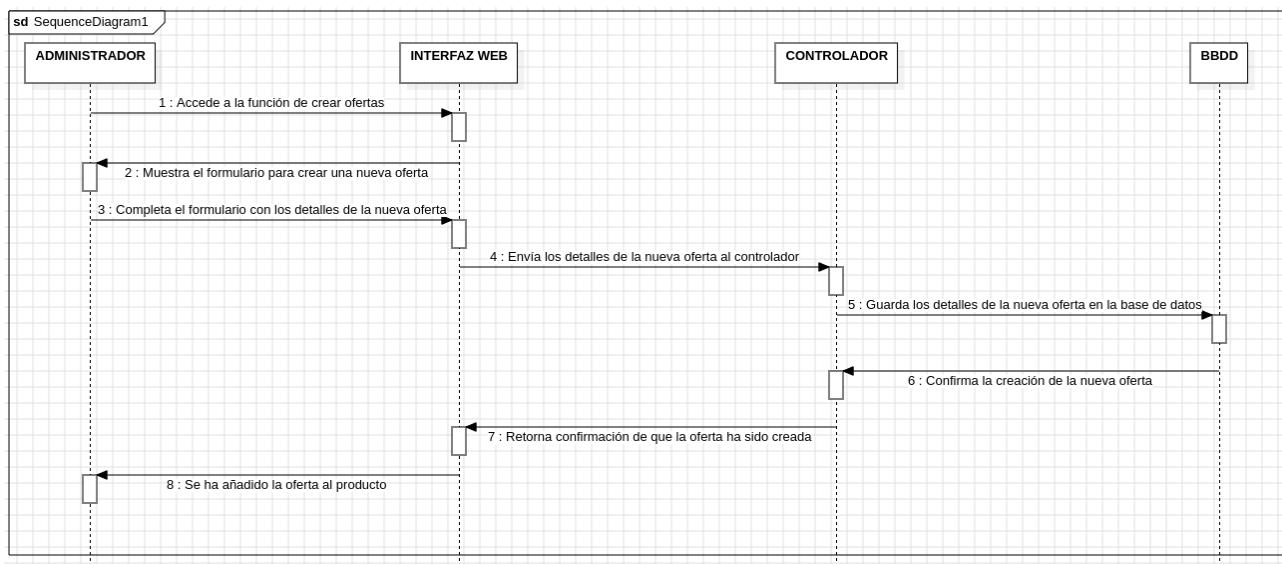


3.5.3.4 EDITAR OFERTAS

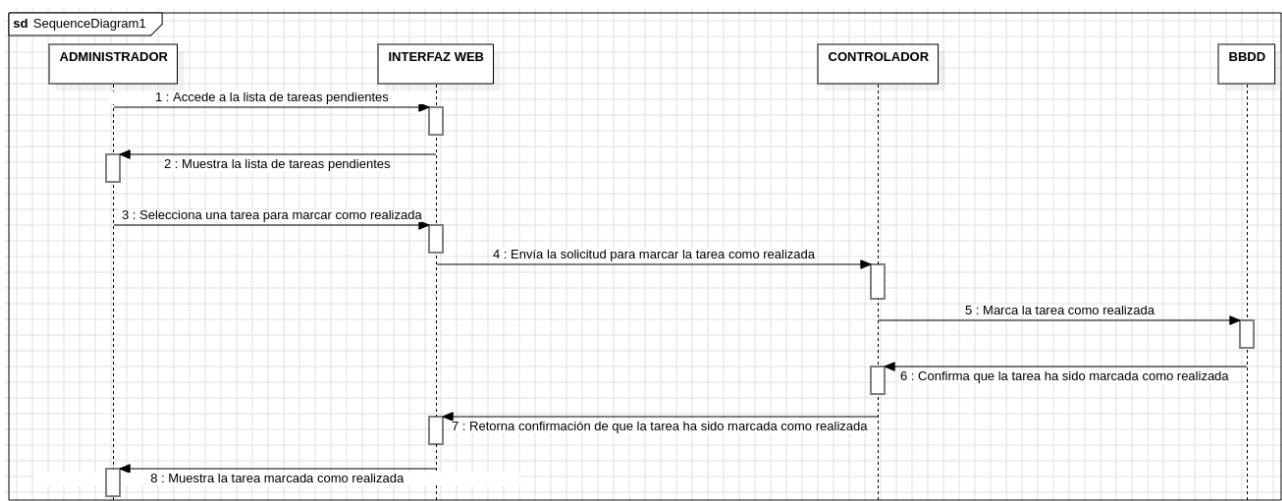


Documentación – Análisis del proyecto

3.5.3.5 CREACIÓN DE OFERTAS

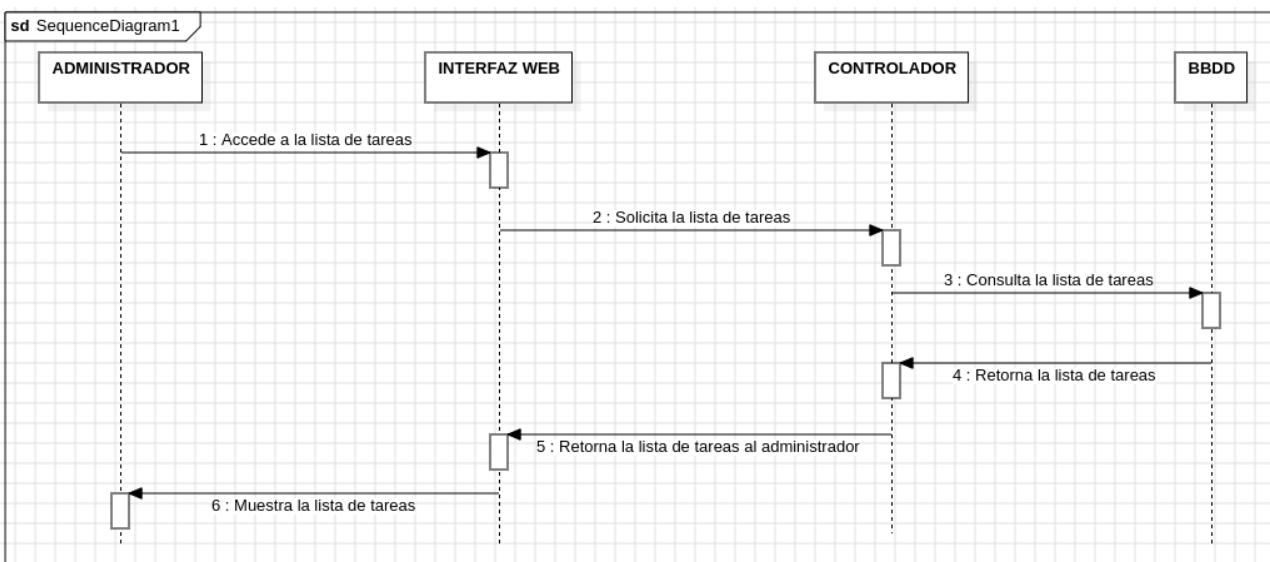


3.5.3.6 MARCAR TAREAS COMO REALIZADAS

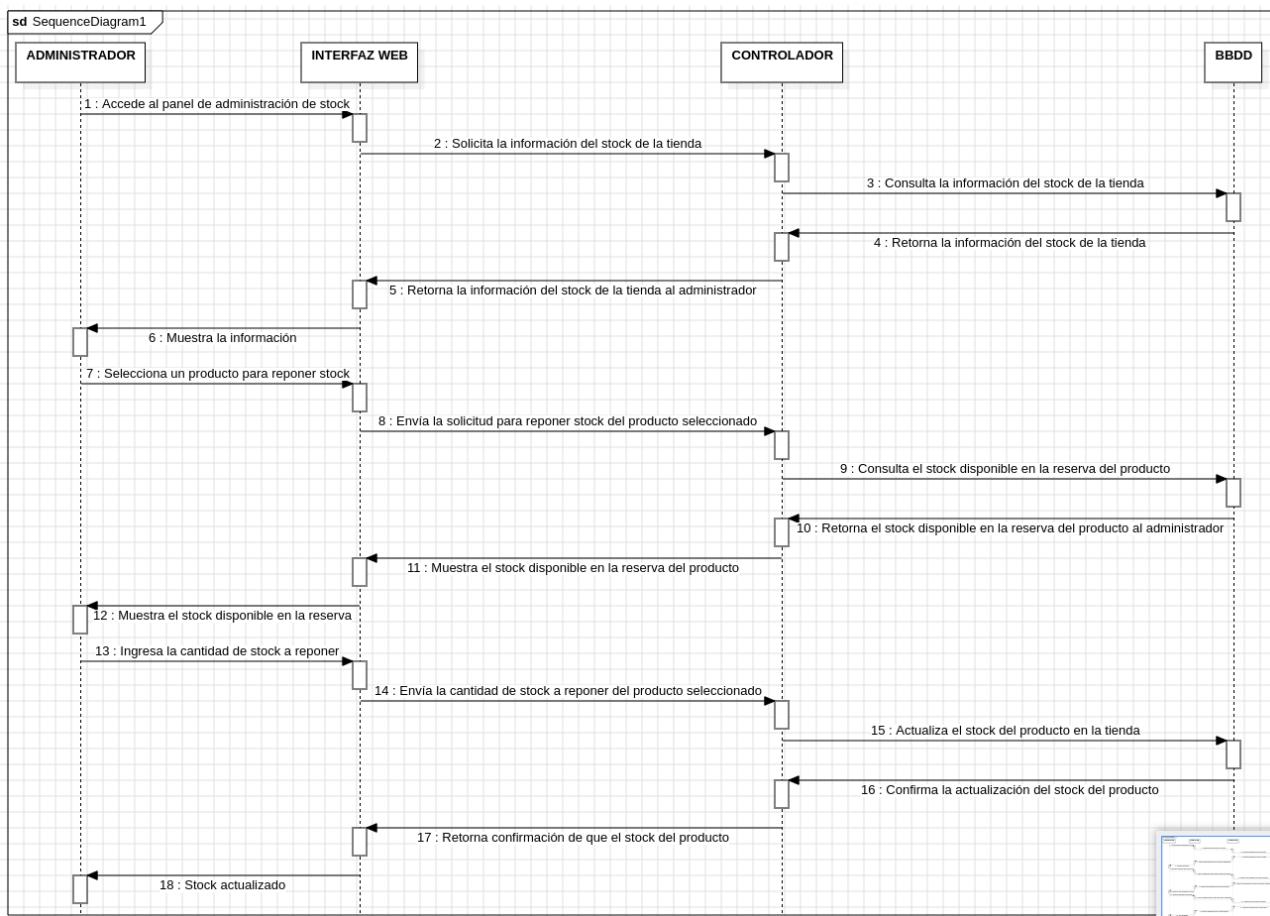


Documentación – Análisis del proyecto

3.5.3.7 VISUALIZAR TAREAS

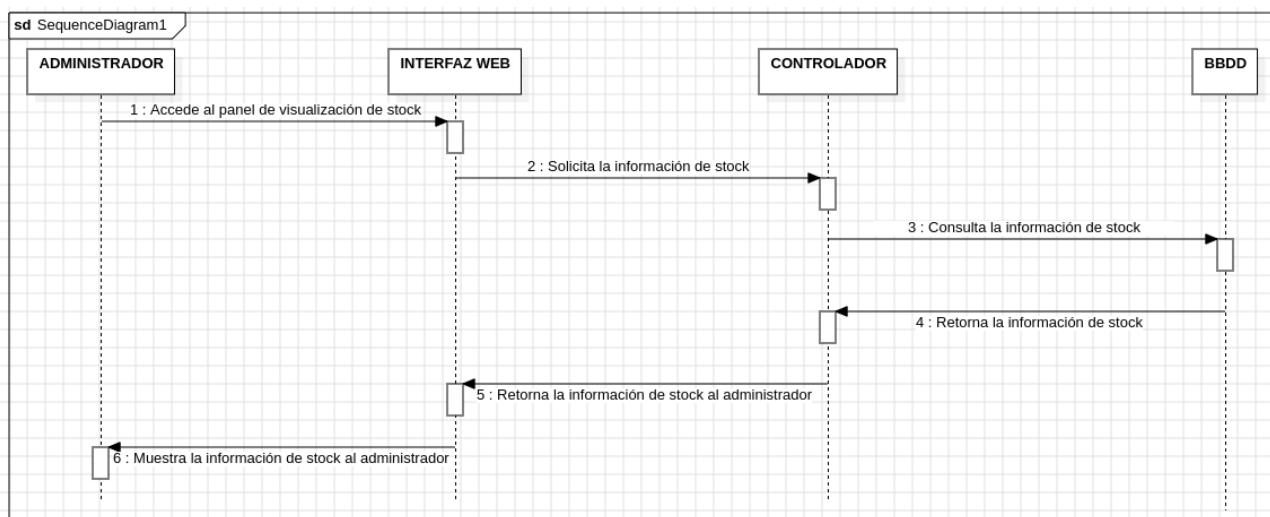


3.5.3.8 REPONER STOCK



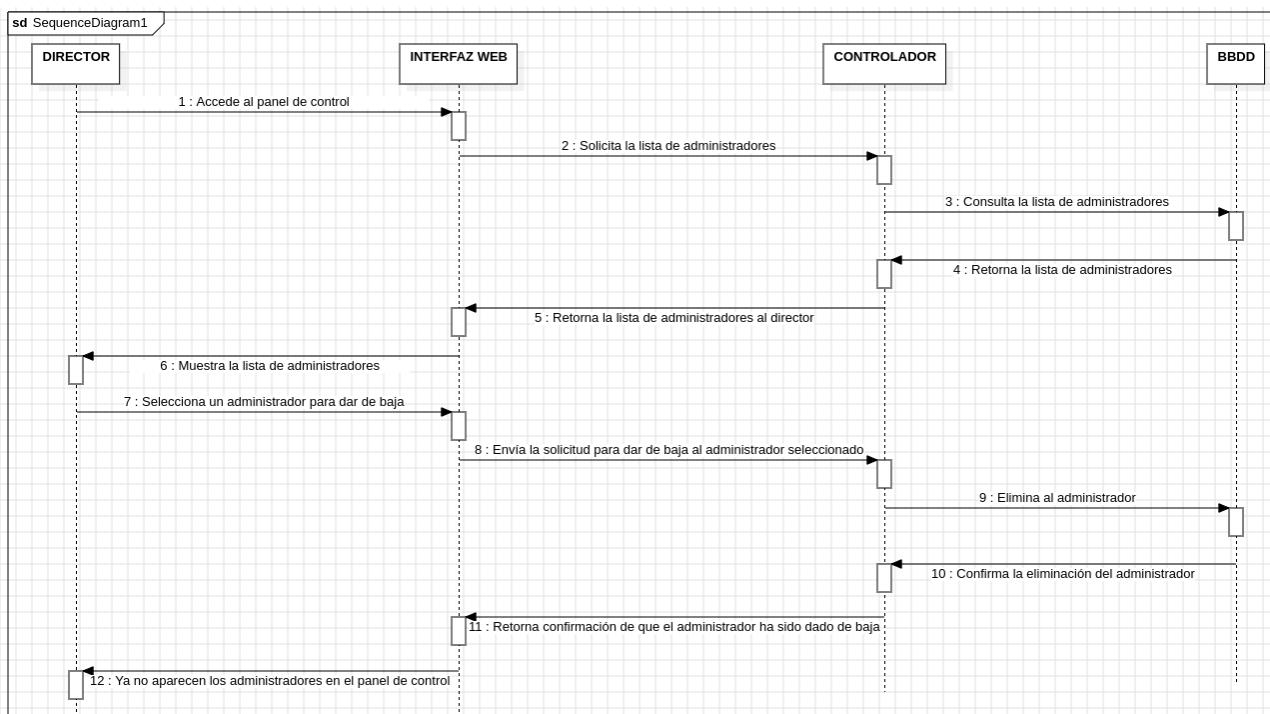
Documentación – Análisis del proyecto

3.5.3.9 VISUALIZAR STOCK



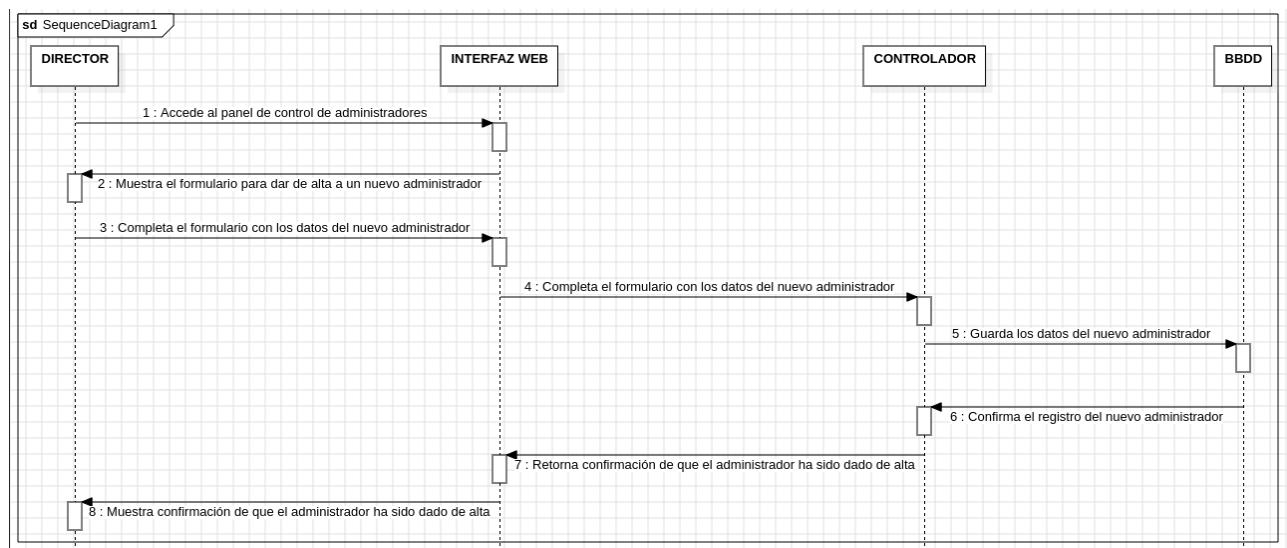
3.5.4 ROL DIRECTOR

3.5.4.1 DAR DE BAJA A UN ADMINISTRADOR

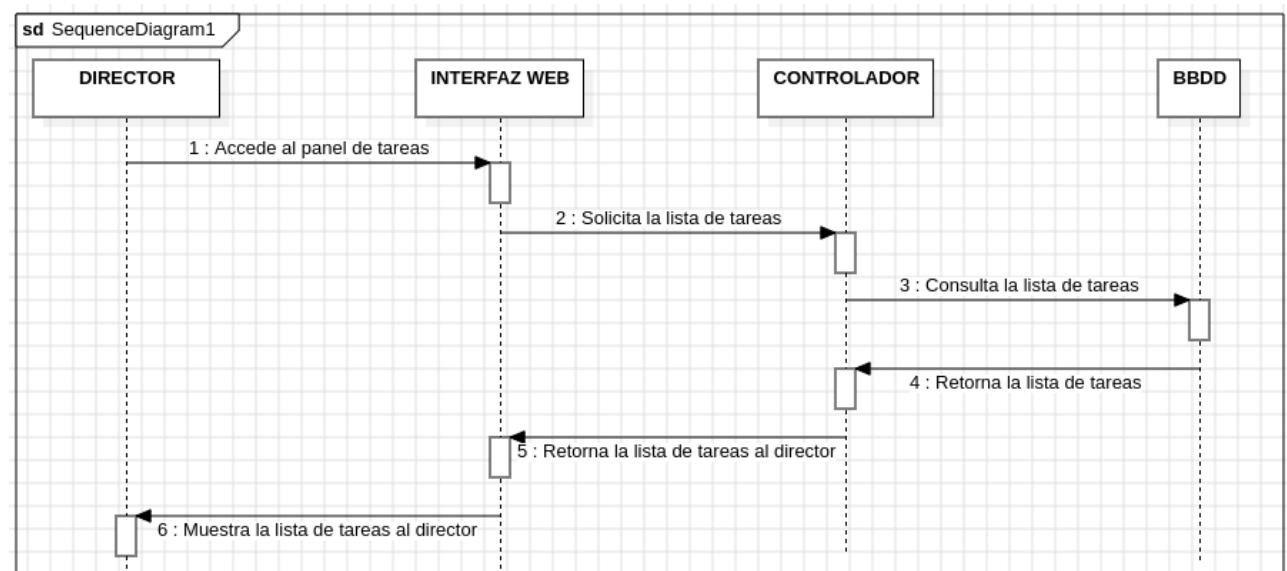


Documentación – Análisis del proyecto

3.5.4.2 DAR DE ALTA A UN ADMINISTRADOR

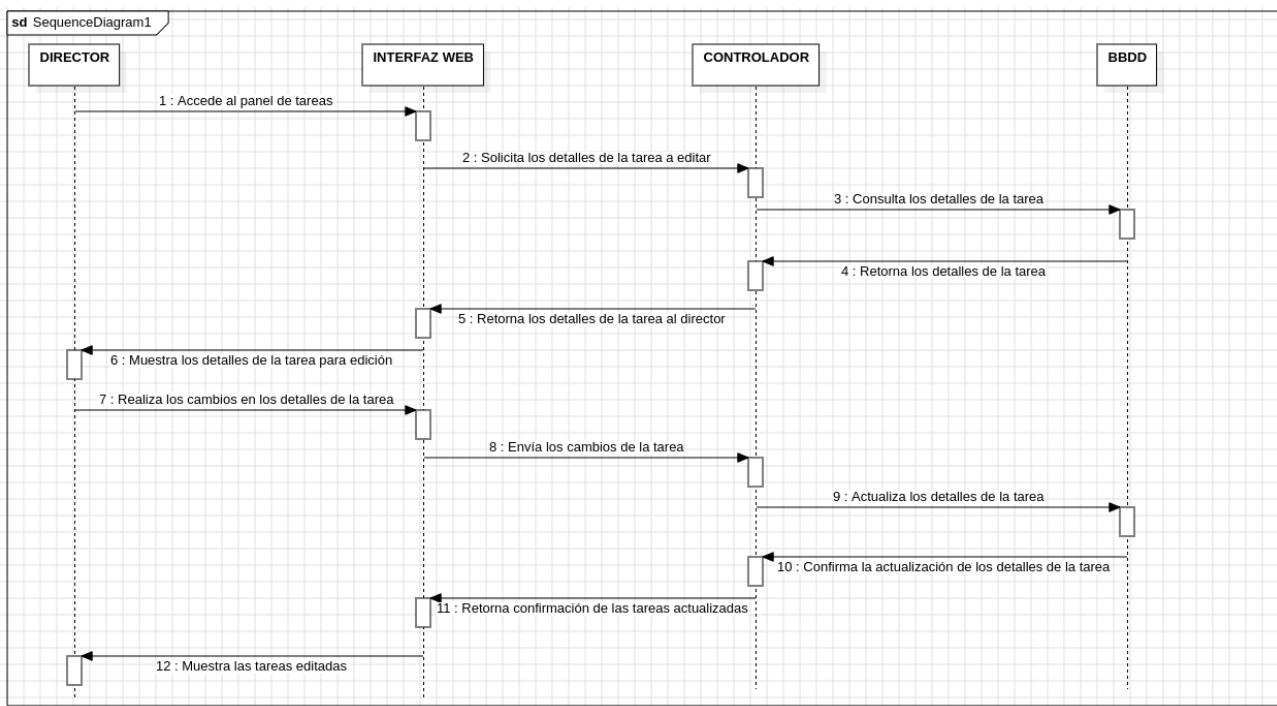


3.5.4.3 VISUALIZAR PANEL DE TAREAS

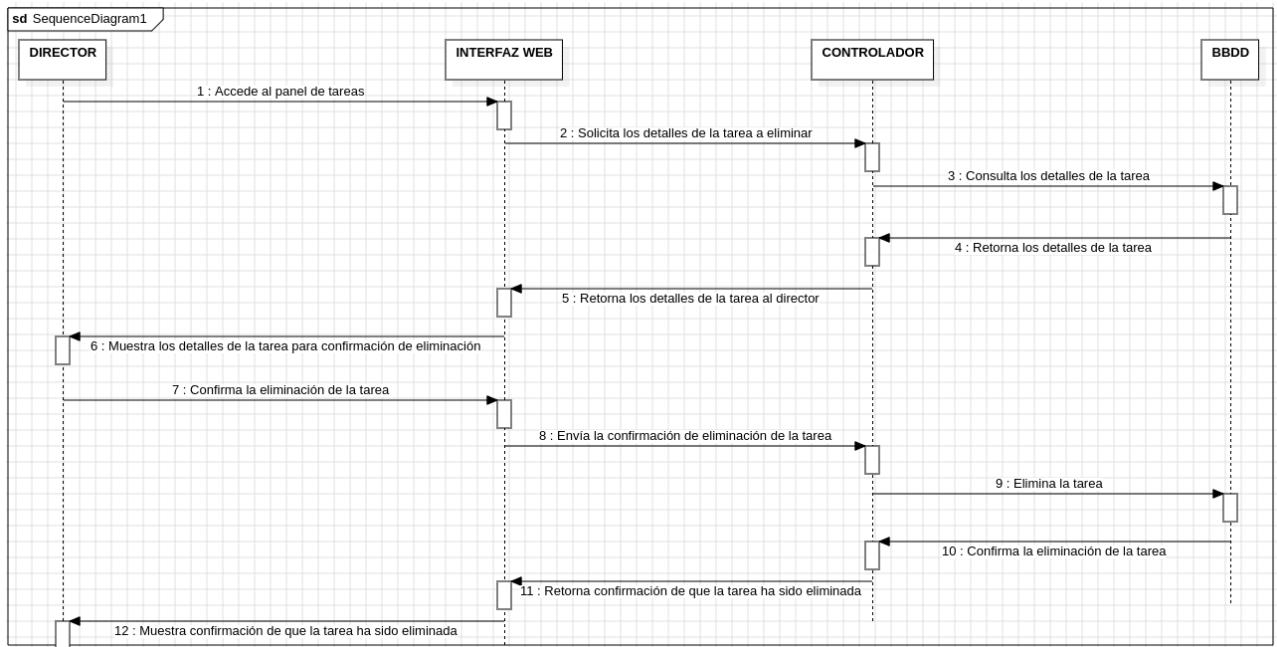


Documentación – Análisis del proyecto

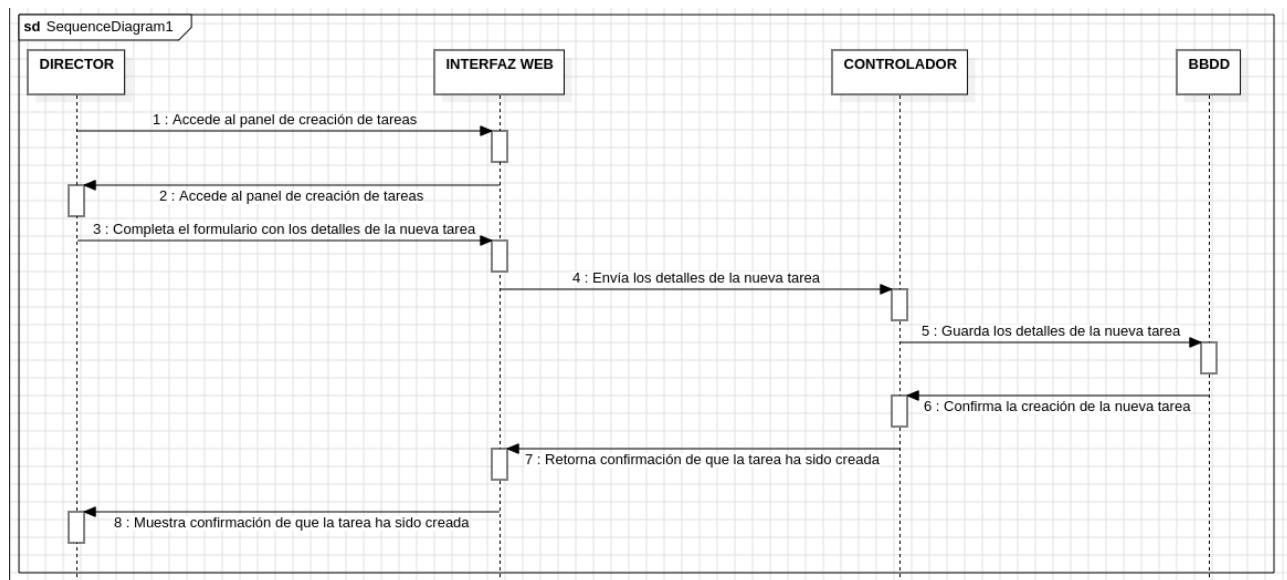
3.5.4.4 EDITAR TAREAS



3.5.4.5 ELIMINAR TAREAS



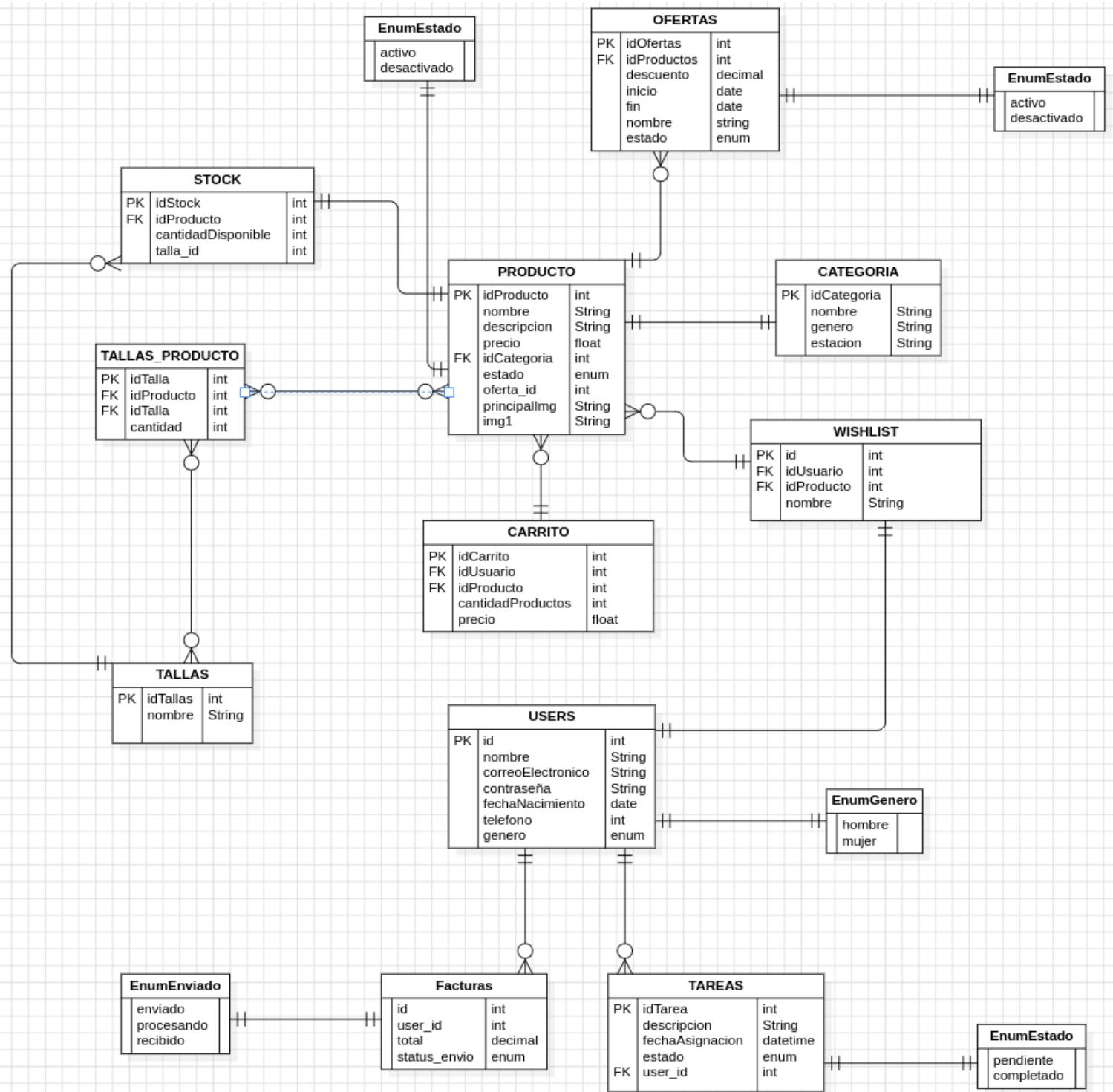
3.5.4.6 CREAR TAREAS



3.6 BASE DE DATOS

En este apartado de procederá a explicar todo el diseño de la base de datos.

3.6.1 DIAGRAMA E/R



Documentación – Análisis del proyecto

3.6.2 DETALLE DE LAS TABLAS

3.6.2.1 TABLA USER

Atributo	Tipo	Descripción
Id	Int	Refleja el id del elemento en la BBDD. Es único.
Nombre	String	Guarda el nombre del usuario.
CorreoElectronico	String	Guarda el correcto del usuario. Es único.
contraseña	String	Guarda la contraseña del usuario.
FechaNacimiento	Date	Guarda la fecha de nacimiento.
Telefono	Int	Guarda el teléfono del usuario.
Genero	Enum	Guarda el género del usuario, entre hombre y mujer.

3.6.2.2 TABLA CARRITO

Atributo	Tipo	Descripción
idCarrito	Int	Refleja el id del elemento en la BBDD. Es único.
idUsuario	Int	Refleja el id del elemento en la BBDD. Es único.
idProducto	Int	Refleja el id del elemento en la BBDD. Es único.
cantidadProductos	Int	Guarda el número de productos que hay de uno mismo.
precio	float	Guarda el precio de los productos.

3.6.2.3 TABLA PRODUCTO

Atributo	Tipo	Descripción

idProducto	Int	Refleja el id del elemento en la BBDD. Es único.
Nombre	String	Guarda el nombre del producto.
Descripción	String	Guarda la descripción del producto.
Precio	Float	Guarda el precio del producto.
idCategoria	Int	Refleja el id del elemento en la BBDD. Es único.
estado	Enum	Guarda el estado del usuario, entre activo y desactivado.
oferta_id	Int	Refleja el id del elemento en la BBDD. Es único
PrincipalImg	String	Guarda una url de la imagen asociada.
Img1	String	Guarda una url de la imagen asociada.

3.6.2.4 TABLA STOCK

Atributo	Tipo	Descripción
idStock	Int	Refleja el id del elemento en la BBDD. Es único.
idProducto	Int	Refleja el id del elemento en la BBDD. Es único
cantidadDisponible	Int	Guarda la cantidad total disponible de cada producto.
talla_id	Int	Refleja el id del elemento en la BBDD. Es único.

3.6.2.5 TABLA TALLAS

Atributo	Tipo	Descripción
idTallas	Int	Refleja el id del elemento en la BBDD. Es único.
Nombre	String	Guarda el nombre de la talla.

3.6.2.6 TABLA OFERTAS

Atributo	Tipo	Descripción
idOfertas	Int	Refleja el id del elemento en la BBDD. Es único.
idProductos	Int	Refleja el id del elemento en la BBDD. Es único.
descuento	Decimal	Guarda el porcentaje del descuento.
Inicio	Date	Guarda la fecha de inicio de la oferta.
Fin	Date	Guarda la fecha del final de la oferta.
Nombre	String	Guarda el nombre de la oferta.
Estado	Enum	Guarda el estado del usuario, entre activo y desactivado.

3.6.2.7 TABLA CATEGORIA

Atributo	Tipo	Descripción
idCategoria	Int	Refleja el id del elemento en la BBDD. Es único.
Nombre	String	Guarda el nombre de la categoria.
Genero	String	Guarda el género de la categoria.
Estación	String	Guarda la estación de la categoria, entre verano, primavera, invierno, otoño.

3.6.2.8 TABLA WISHLIST

Atributo	Tipo	Descripción
Id	Int	Refleja el id del elemento en la BBDD. Es único.
idUsuario	Int	Refleja el id del elemento en la BBDD. Es único.

idProducto	Int	Refleja el id del elemento en la BBDD. Es único.
Nombre	String	Guarda el nombre del producto en la wishlist.

3.6.2.9 TABLA FACTURAS

Atributo	Tipo	Descripción
Id	Int	Refleja el id del elemento en la BBDD. Es único.
user_id	Int	Refleja el id del elemento en la BBDD. Es único.
Total	String	Guarda el precio total de todos los productos comprados.
Status_envio	Enum	Guarda el envío de la factura entre enviado, procesando y recibido.

3.6.2.10 TABLA TAREAS

Atributo	Tipo	Descripción
idTarea	Int	Refleja el id del elemento en la BBDD. Es único.
Descripcion	String	Guarda la descripción de la tarea.
fechaAsignacion	DateTime	Guarda la fecha de asignación al administrador correspondiente.
Estado	Enum	Guarda el estado del usuario, entre activo y desactivado.
user_id	Int	Refleja el id del elemento en la BBDD. Es único.

3.6.2.11 TABLA TALLAS_PRODUCTO

Atributo	Tipo	Descripción
idTallas	Int	Refleja el id del elemento en la BBDD. Es único.

idProducto	Int	BBDD. Es único.
idTalla	Int	Refleja el id del elemento en la BBDD. Es único.
Cantidad	Int	Guarda la cantidad de cada talla en el producto.

3.6.3 RELACIONES

La creación de la base de datos consta de las siguientes tablas y relaciones (los campos de cada tabla puedes observarla en el diagrama):

- **Producto:**
 - Relación muchos a muchos con tallas_productos
 - Relación uno a uno con EnumEstado
 - Relación muchos a uno con carrito
 - Relación muchos a uno con wishlist
 - Relación uno a uno con categoria
 - Relación uno a muchos con ofertas
- **Categoria:**
 - Relación mucho uno a uno con producto
- **Wishlist:**
 - Relación uno a muchos con producto
 - Relación uno a uno con users
- **Carrito:**
 - Relación uno a muchos con producto
- **Ofertas:**
 - Relación uno a uno con EnumEstado
 - Relación muchos a uno
- **Stock:**
 - Relación uno a uno con productos
 - Relación muchos a unos con tallas
- **Tallas:**
 - Relación muchos a muchos con tallas_productos
- **Tallas_producto:**
 - Relación muchos a muchos con producto
- **Users:**
 - Relación uno a uno con enumGenero
 - Relación uno a muchos con tareas
 - Relación uno a muchos con facturas
- **Facturas:** Relación muchos a uno con users
 - Relación uno a uno con enumEnviado
- **Tareas:** Relación muchos a uno con users
 - Relación uno a uno con EnumEstado

4. ARQUITECTURA Y TECNOLOGÍAS EMPLEADAS

4.1 ARQUITECTURA DE PROGRAMACIÓN

En esta sección, se detalla la arquitectura del software y las tecnologías de programación seleccionadas para el desarrollo del proyecto.

4.1.1 API REST

Laravel se ha elegido como el framework principal para la construcción del API REST, debido a su capacidad para crear aplicaciones web robustas y seguras. Laravel simplifica la creación de APIs RESTful al proporcionar una estructura clara y módulos integrados, como autenticación y manejo de errores. Las comunicaciones se basan en el protocolo HTTP, con JSON como el formato de datos estándar. Esta API REST es fundamental para la lógica del back-end, proporcionando a la aplicación frontend información sobre productos, usuarios, y órdenes.

4.1.2 React para el Cliente

React se ha seleccionado para el desarrollo del cliente por su capacidad para crear interfaces de usuario interactivas y dinámicas. React permite el desarrollo de Single Page Applications (SPA), mejorando la experiencia del usuario al proporcionar una navegación fluida y rápida sin recargas completas de la página. Su enfoque basado en componentes facilita la reutilización de código y la organización modular, esencial para mantener el proyecto escalable y mantenable.

4.1.3 Tailwind CSS

Tailwind CSS es el framework CSS utilizado para estilizar la aplicación. A diferencia de otros frameworks, Tailwind CSS adopta un enfoque utility-first, que permite aplicar estilos directamente en los elementos HTML a través de clases utilitarias. Esto proporciona un alto grado de personalización y un flujo de trabajo eficiente, ya que permite construir componentes visualmente coherentes y únicos sin necesidad de escribir CSS personalizado. Esta flexibilidad se adapta bien a la creación de una tienda en línea moderna y visualmente atractiva.

4.1.4 MySQL en contenedor Docker

MySQL ha sido la elección para la gestión de la base de datos debido a su robustez y rendimiento en el manejo de datos relacionales. En este proyecto, MySQL se implementa dentro de un contenedor Docker, lo que facilita su despliegue, escalabilidad y gestión. Docker permite que MySQL opere en un entorno aislado, garantizando la consistencia y la portabilidad entre diferentes entornos de desarrollo y producción.

Además, se ha integrado la interfaz de phpMyAdmin en otro contenedor Docker.

phpMyAdmin proporciona una interfaz gráfica accesible para la administración y consulta de la base de datos MySQL. Esta configuración permite la gestión eficiente de los datos relacionados con productos, usuarios, órdenes y otros aspectos críticos de la tienda en línea. La combinación de MySQL y phpMyAdmin en contenedores Docker simplifica la administración, mejora la seguridad y asegura una implementación coherente y fiable de la base de datos.

5.2 ENTORNOS DE PROGRAMACIÓN

En esta sección, se detalla la arquitectura del software y las tecnologías de programación seleccionadas para el desarrollo del proyecto.

5.2.1 Visual Studio Code

Visual Studio Code (VS Code) es un editor de código fuente altamente personalizable y extensible. Su uso en este proyecto se debe a sus características avanzadas como la depuración integrada, control de versiones mediante Git, IntelliSense para la autocompletación de código, y un amplio ecosistema de extensiones que facilitan el desarrollo en diferentes lenguajes y plataformas.

5.2.2 Docker

Docker se ha empleado para la virtualización y gestión de entornos mediante contenedores. Esto incluye el despliegue de MySQL y phpMyAdmin, lo que garantiza un entorno de desarrollo y producción coherente, aislado y fácil de escalar. Docker facilita la configuración y despliegue de la infraestructura del proyecto, asegurando la reproducibilidad del entorno.

5.2.3 PostMan

Postman es una herramienta para la prueba de APIs que ha sido utilizada para diseñar, probar y documentar la API REST de Laravel. Permite simular peticiones HTTP, automatizar pruebas, y generar documentación de la API, lo que mejora la calidad y confiabilidad de la comunicación entre el frontend y el backend.

5.2.4 GitHub

GitHub ha sido utilizado como plataforma de control de versiones y colaboración. Facilita la gestión del código fuente, seguimiento de cambios, manejo de ramas, y revisión de código. GitHub también permite la integración continua y despliegue continuo (CI/CD), facilitando un flujo de trabajo eficiente y colaborativo.

5.2.5 StarUML

StarUML es una herramienta de modelado UML utilizada para diseñar la arquitectura y diagramas de flujo del sistema. Permite la creación de diagramas detallados para visualizar la estructura de datos, la lógica del sistema, y las interacciones entre componentes, ayudando en la planificación y documentación del proyecto.

5.2.6 Mockitt.wondershare

Wondershare Mockitt es una herramienta de diseño y prototipado de interfaces que

se ha utilizado para crear prototipos de la interfaz de usuario. Permite el diseño de maquetas interactivas y la validación de la experiencia del usuario antes de la implementación, lo que asegura una interfaz atractiva y funcional.

5. LIBRERÍAS EMPLEADAS

Librerías Usadas en React

- `@heroicons/react`: Proporciona iconos SVG para React, compatibles con Tailwind CSS.
 - `npm install @heroicons/react`
- `axios`: Librería para realizar peticiones HTTP desde React. Simplifica la obtención y el envío de datos al servidor.
 - `npm install axios`
- `react`: Biblioteca principal para la construcción de la UI en componentes reutilizables. Facilita la creación de interfaces interactivas.
 - `npm install react`
- `react-dom`: Proporciona métodos específicos del DOM que se utilizan junto con React. Permite la interacción con el DOM en aplicaciones React.
 - `npm install react-dom`
- `react-router-dom`: Biblioteca para la gestión de rutas en aplicaciones React. Facilita la navegación y el enrutamiento dinámico.
 - `npm install react-router-dom`
- `react-slick`: Componente de carrusel para React. Permite la creación de carruseles y sliders con funcionalidad avanzada.
 - `npm install react-slick`
- `slick-carousel`: Biblioteca de carrusel utilizada por `react-slick` para manejar carruseles en la UI.
 - `npm install slick-carousel`

Librería Usada en Laravel API

- `spatie/laravel-permission`: Paquete para la gestión de roles y permisos en Laravel. Facilita la asignación de permisos y roles a usuarios, permitiendo un control de acceso flexible.
 - `composer require spatie/laravel-permission`
 - `php artisan vendor:publish --provider="Spatie\Permission\PermissionServiceProvider"`
 - `php artisan migrate`

6 . PRUEBA

6.1 Introducción.

Este apartado presenta las pruebas diseñadas para validar la funcionalidad de mi aplicación, enfocándose en aspectos clave como la autenticación, gestión de usuarios, administración de productos y ofertas, así como la asignación y seguimiento de tareas según los roles de usuario, administrador y director. Cada prueba detalla los pasos seguidos, el resultado esperado y el obtenido, asegurando así la estabilidad y funcionalidad del sistema antes de su implementación final.

6.2 Pruebas de autenticación.

Vista	Nº	Prueba	Pasos	Resultado esperado	Resultado obtenido
Login	1	Acceso al login	Acceder al login, insertar datos y entrar en la web	El rol debe visualizar la vista Home que le corresponda.	OK
Registro	2	Registro de usuario	Acceder y registrarse	El nuevo registro tendrá el rol usuario y accederá después del registro a la vista Home, y posteriormente podrá iniciar sesión para entrar como verificado.	OK

6.3 Pruebas para el Rol de Usuario.

Vista	Nº	Prueba	Pasos	Resultado esperado	Resultado obtenido
Editar Mi Perfil	1	Actualización de perfil	Logarse, acceder al menú de usuario y seleccionar "Editar correctamente en mi perfil". Actualizar los campos y guardar	Los datos del perfil deben actualizarse en la BBDD y reflejarse en la interfaz	OK

Añadir al Carrito	Navegar por la tienda, seleccionar un producto y pulsar sobre "Añadir al Carrito".			El producto debe añadirse al carrito y debe ser visible en la lista del carrito.	OK
	Acceder al menú de usuario y seleccionar "Carrito".			Deben visualizarse todos los productos añadidos al carrito.	
Ver Carrito	Acceder al menú de usuario y seleccionar "Carrito".			Deben visualizarse todos los productos añadidos al carrito.	OK
	Acceder al carrito, seleccionar un producto y pulsar sobre "Eliminar".			El producto debe eliminarse del carrito y no debe aparecer en la lista del carrito.	
Eliminar del Carrito	Acceder al carrito, seleccionar un producto y pulsar sobre "Eliminar".			El producto debe eliminarse del carrito y no debe aparecer en la lista del carrito.	OK
	Navegar por la tienda, seleccionar un producto y pulsar sobre "Añadir a la Lista de Deseos".			El producto debe añadirse a la lista de deseos y debe ser visible en la lista.	
Añadir a la Lista de Deseos	Acceder al menú de usuario y seleccionar "Lista de Deseos".			Deben visualizarse todos los productos añadidos a la lista de deseos.	OK
	Acceder a la lista de deseos, seleccionar un producto y pulsar sobre "Eliminar".			El producto debe eliminarse de la lista de deseos y no debe aparecer en la lista.	
Ver Lista de Deseos	Acceder a la lista de deseos, seleccionar un producto y pulsar sobre "Eliminar".			El producto debe eliminarse de la lista de deseos y no debe aparecer en la lista.	OK
	Acceder al carrito, revisar productos, y pulsar sobre "Comprar". Completar el proceso de pago.			Los productos deben ser comprados correctamente y una factura debe generarse en el perfil del usuario.	
Realizar Compra	Acceder al carrito, revisar productos, y pulsar sobre "Comprar". Completar el proceso de pago.			Los productos deben ser comprados correctamente y una factura debe generarse en el perfil del usuario.	OK

Ver Facturas	9	Visualización de facturas	Acceder al menú de usuario y seleccionar "Mis Facturas".	Deben visualizarse todas las facturas generadas tras las compras.	OK
--------------	---	---------------------------	--	---	----

6.3 Pruebas para el Rol de Administrador .

Vista	Nº	Prueba	Pasos	Resultado esperado	Resultado obtenido
Crear Producto	1	Creación de producto	Acceder a la lista de productos, pulsar "Nuevo Producto" y completar el formulario	El nuevo producto debe ser registrado correctamente y debe aparecer en la lista	OK
Editar Producto	2	Modificación de producto	Acceder a la lista de productos, seleccionar un producto, pulsar "Editar" y actualizar los datos	El producto debe actualizarse correctamente en la BBDD y reflejarse en la lista	OK
Eliminar Producto	3	Eliminación de producto	Acceder a la lista de productos, seleccionar un producto y pulsar "Eliminar"	El producto debe ser eliminado correctamente de la BBDD y no debe aparecer en la lista	OK
Desactivar Producto	4	Desactivación de producto	Acceder a la lista de productos, seleccionar un producto y pulsar "Desactivar"	El producto debe ser desactivado correctamente y no debe estar disponible para la venta	OK
Crear Oferta	5	Creación de oferta	Acceder a la sección de ofertas, pulsar "Nueva Oferta" y completar el formulario	La nueva oferta debe ser registrada correctamente y debe aparecer en la lista	OK
			Acceder a la lista	La oferta debe	

Editar Oferta	6	Modificación de oferta	de ofertas, seleccionar una oferta, pulsar "Editar" y actualizar los datos	actualizarse correctamente en la BBDD y reflejarse en la lista	OK
Asignar Oferta a Producto	7	Asignación de oferta	Acceder a la lista de productos, seleccionar un producto, pulsar "Asignar Oferta" y elegir una oferta disponible	La oferta debe ser asignada correctamente al producto y reflejarse en la interfaz	OK
Incrementar Stock	8	Incrementar stock	Acceder a la lista de productos, seleccionar un producto, pulsar "Incrementar Stock" y agregar cantidad	La cantidad de stock debe actualizarse correctamente en la BBDD y reflejarse en la lista	OK
Disminuir Stock	9	Disminuir stock	Acceder a la lista de productos, seleccionar un producto, pulsar "Disminuir Stock" y restar cantidad	La cantidad de stock debe disminuirse correctamente en la BBDD y reflejarse en la lista	OK
Visualizar Stock	10	Listar stock	Acceder a la lista de productos	Deben mostrarse correctamente las cantidades de stock para cada producto	OK
Completar Tarea Asignada	11	Completar tarea	Acceder a la lista de tareas asignadas, seleccionar una tarea y marcar como completada	La tarea debe marcarse como completada y reflejarse en la BBDD	OK

Documentación – Prueba

6.4 Pruebas para el Rol de Director.

Vista	Nº	Prueba	Pasos	Resultado esperado	Resultado obtenido
Crear Administrador	1	Creación de administrador	Acceder a la lista de usuarios, pulsar sobre "Nuevo usuario" y seleccionar rol de administrador. Completar el formulario	El nuevo administrador debe ser registrado correctamente y debe aparecer en la lista de usuarios	OK
Editar Administrador	2	Modificación de datos de administrador	Acceder a la lista de usuarios, seleccionar un administrador, pulsar sobre "Editar" y actualizar los datos	Los datos del administrador deben actualizarse correctamente en la BBDD y reflejarse en la lista	OK
Eliminar Administrador	3	Eliminación de administrador	Acceder a la lista de usuarios, seleccionar un administrador y pulsar sobre "Eliminar"	El administrador debe ser eliminado correctamente de la BBDD y no debe aparecer en la lista de usuarios	OK
Crear Tarea	4	Creación de tarea	Acceder a la sección de tareas, pulsar sobre "Nueva Tarea" y completar el formulario	La nueva tarea debe ser registrada correctamente y debe aparecer en la lista de tareas	OK
Editar Tarea	5	Modificación de tarea	Acceder a la lista de tareas, seleccionar una tarea, pulsar sobre "Editar" y actualizar los datos	La tarea debe actualizarse correctamente en la BBDD y reflejarse en la lista	OK
Visualizar Tareas	6	Listar tareas	Acceder a la lista de tareas	Deben listarse todas las tareas registradas en la BBDD	OK

4. PROYECTO LARAVEL + REACT, CÓDIGO.

4.1 MIGRACIONES Y MODELOS.

En este apartado voy a explicar la formación de la base de datos mediante laravel con el uso de migraciones y los modelos para crear toda la estructura entre tablas.

4.1.1 Users migración

Los campos que componen esta tabla son:

- Id: Clave primaria.
- Name: Nombre del usuario.
- Email: Correo electrónico único del usuario.
- Email_verified_at: Marca de tiempo para verificar el correo electrónico del Usuario(viene por defecto en laravel).
- Password: Contraseña del usuario (puede ser nulo).
- Fecha_nacimiento: Fecha de nacimiento del usuario.
- Telefono: Número de teléfono del usuario.
- Genero: Género del usuario (puede ser hombre, mujer o nulo).
- RememberToken: Token para recordar la sesión del usuario.

4.1.2 Ofertas migración

Los campos que componen esta tabla son:

- id: Clave primaria auto incremental.
- nombre: Nombre de la oferta.
- descuento: Descuento ofrecido por la oferta.
- inicio: Fecha y hora de inicio de la oferta.
- fin: Fecha y hora de finalización de la oferta.
- estado: Estado de la oferta (activa o expirada).

4.1.3 Categorias migración

Los campos que componen esta tabla son:

- id: Clave primaria autoincremental.
- nombre: Nombre de la categoría.
- genero: Género asociado a la categoría (puede ser nulo).
- estación: Estación del año asociada a la categoría (puede ser nulo).

4.1.4 Productos migración

Los campos que componen esta tabla son:

- id: Clave primaria autoincremental.
- nombre: Nombre del producto.
- descripcion: Descripción del producto.
- precio: Precio del producto.
- principalImg: Nombre del archivo de imagen principal del producto (puede ser nulo).
- img1: Nombre del archivo de imagen adicional del producto (puede ser nulo).
- activo: Estado de disponibilidad del producto.
- categoria_id: Clave foránea que referencia la categoría a la que pertenece

el producto.

- oferta_id: Clave foránea que referencia la oferta asociada al producto (puede ser nulo).

4.1.5 Wish_lists migración

Los campos que componen esta tabla son:

- id: Clave primaria autoincremental.
- nombre: Nombre de la lista de deseos.
- producto_id: Clave foránea que referencia el producto guardado en la lista de deseos.
- user_id: Clave foránea que referencia el usuario propietario de la lista de deseos.

4.1.6 Tallas migración

Los campos que componen esta tabla son:

- id: Clave primaria autoincremental.
- nombre: Nombre de la talla.

4.1.7 Producto_talla migración

Los campos que componen esta tabla son:

- id: Clave primaria autoincremental.
- producto_id: Clave foránea que referencia el producto.
- talla_id: Clave foránea que referencia la talla.

4.1.8 Carritos migración

Los campos que componen esta tabla son:

- id: Clave primaria autoincremental.
- producto_id: Clave foránea que referencia el producto agregado al carrito.
- user_id: Clave foránea que referencia el usuario propietario del carrito.
- talla_id: Clave foránea que referencia la talla seleccionada del producto.
- cantidadProducto: Cantidad del producto agregado al carrito.
- precio: Precio del producto en el momento de agregarlo al carrito.

4.1.9 Stocks migración

Los campos que componen esta tabla son:

- id: Clave primaria autoincremental.
- producto_id: Clave foránea que referencia el producto.
- talla_id: Clave foránea que referencia la talla.
- cantidad: Cantidad disponible en stock.

4.1.10 Facturas migración

Los campos que componen esta tabla son:

- id: Clave primaria autoincremental.
- user_id: Clave foránea que referencia el usuario que realizó la compra.
- total: Total de la factura.
- detalle: Detalle de los productos comprados.
- status_envio: Estado del envío de la factura.

4.1.11 Tareas migración

Los campos que componen esta tabla son:

- id: Clave primaria autoincremental.
- assigned_to: Clave foránea que referencia el usuario al que se asigna la tarea.
- title: Título de la tarea.
- description: Descripción de la tarea.
- status: Estado de la tarea (pendiente o completada).

4.1.12 Users modelo

- Relaciones:
 - carritos(): Define una relación uno a muchos con la tabla carritos, indicando que un usuario puede tener varios elementos en su carrito de compras.
 - facturas(): Establece una relación uno a muchos con la tabla facturas, lo que significa que un usuario puede tener varias facturas asociadas.
 - tareas(): Define una relación uno a muchos con la tabla tareas, lo que indica que un usuario puede tener múltiples tareas asignadas.
 - belongsTo: Este modelo también tiene una relación de pertenencia (belongsTo) con los modelos relacionados mediante las claves foráneas.

4.1.13 Carritos modelo

- Relaciones:
 - producto(): Define una relación de pertenencia (belongsTo) con el modelo Producto, indicando que un elemento del carrito pertenece a un producto.
 - usuario(): Establece una relación de pertenencia (belongsTo) con el modelo User, indicando el usuario propietario del carrito.
 - talla(): Define una relación de pertenencia (belongsTo) con el modelo Talla, indicando la talla del producto en el carrito.

4.1.14 Categorias modelo

- Relaciones:
 - productos(): Establece una relación uno a muchos con el modelo Producto, indicando que una categoría puede tener varios productos asociados.

4.1.15 Wish_lists modelo

- Relaciones:
 - producto(): Define una relación de pertenencia (belongsTo) con el modelo Producto, indicando el producto guardado en la lista de deseos.

4.1.16 Productos modelo

- Relaciones:
 - carritos(): Establece una relación uno a muchos con el modelo Carrito, indicando que un producto puede estar presente en varios carritos.
 - categoria(): Define una relación de pertenencia (belongsTo) con el modelo Categoria, indicando la categoría a la que pertenece el producto.
 - wishList(): Establece una relación uno a muchos con el modelo WishList,

- indicando que un producto puede estar en varias listas de deseos.
- `tallas()`: Define una relación de muchos a muchos con el modelo Talla a través de la tabla pivot `producto_talla`.
- `stocks()`: Establece una relación uno a muchos con el modelo Stock, indicando la cantidad de stock disponible para el producto.
- `oferta()`: Define una relación de pertenencia (`belongsTo`) con el modelo Oferta, indicando si el producto está asociado a alguna oferta.

4.1.17 Tallas modelo

- Relaciones:
 - `productos()`: Define una relación de muchos a muchos con el modelo Producto a través de la tabla pivot `producto_talla`.
 - `carritos()`: Establece una relación uno a muchos con el modelo Carrito,
 - indicando que una talla puede estar presente en varios carritos.
 - `stocks()`: Define una relación uno a muchos con el modelo Stock, indicando la cantidad de stock disponible para la talla.

4.1.18 Ofertas modelo

- Relaciones:
 - `productos()`: Establece una relación uno a muchos con el modelo Producto, indicando que una oferta puede estar asociada a varios productos.

4.1.19 Stocks modelo

- Relaciones:
 - `producto()`: Define una relación de pertenencia (`belongsTo`) con el modelo Producto, indicando el producto al que pertenece el stock.
 - `talla()`: Establece una relación de pertenencia (`belongsTo`) con el modelo Talla, indicando la talla asociada al stock.

4.1.20 Facturas modelo

- Relaciones:
 - `user()`: Define una relación de pertenencia (`belongsTo`) con el modelo User, indicando el usuario que realizó la compra.

4.1.21 Tareas modelo

- Relaciones:
 - `user()`: Define una relación de pertenencia (`belongsTo`) con el modelo User, indicando el usuario al que se asigna la tarea.

4.2 LOGIN, REGISTRO Y ROLES.

4.2.1 ROLESEEDER

La finalidad de este Seeder es crear en la base de datos y en el controlador de Roles de spattie, dos roles diferentes, llamados “administrador” y “director” con un usuario asignado a cada rol. Con ello, podremos utilizar estos roles para

crear más usuarios del tipo de rol que deseemos. Pero su finalidad, es que el usuario normal y corriente no tenga ningún rol asociado para diferencia entre las partes administrativas de las páginas y la interfaz de los clientes.

Explicación del código:

1. Se crea un nuevo rol llamado "admin" utilizando el modelo Role. Este rol se asignará al usuario.
2. Se crea un nuevo usuario administrador utilizando el modelo User.
3. Se crea otro rol llamado "director" utilizando el modelo Role. Este rol se asignará al usuario director.
4. Se crea un nuevo usuario director utilizando el modelo User.
5. Se asigna el rol "admin" y el rol "director" al usuario administrador utilizando el método assignRole() del modelo User.

4.2.2 AUTHCONTROLLER

Este controlador maneja la autenticación y el registro de usuarios en la página, a través de la información que se envía a la API creada en laravel.

- *Método register()*

Utiliza un validador para validar los datos de entrada del usuario, como nombre, correo electrónico, contraseña, fecha de nacimiento, teléfono y género. Si la validación falla, devuelve una respuesta JSON con los errores de validación. Si la validación es exitosa, crea un nuevo usuario en la base de datos con la contraseña encriptada y devuelve una respuesta JSON con el usuario creado.

```
public function register()
{
    $validator = validator(request()->all(), [
        'name' => 'required|string|max:255',
        'email' => 'required|string|email|max:255|unique:users',
        'password' => [
            'required',
            'string',
            'min:6',
            'regex:/[A-Z]/',
            'regex:/[0-9]/'
        ],
        'fecha_nacimiento' => [
            'required',
            'date',
            'before_or_equal:' . now()->subYears(18)->toDateString()
        ],
        'telefono' => [
            'required',
            'digits:9',
            'regex:/' . [67][0-9]{8} . '/'
        ],
        'genero' => 'required|in:hombre,mujer',
    ]);

    $messages = [
        'email.unique' => 'El email ya existe.',
        'password.regex' => 'La contraseña necesita tener al menos 6 caracteres, 1 mayúscula y 1 número.',
        'fecha_nacimiento.before_or_equal' => 'Debes tener al menos 18 años.',
        'telefono.digits' => 'El teléfono debe tener 9 dígitos.',
        'telefono.regex' => 'Debe de empezar por 6 y 7.',
        'genero.required' => 'El género es obligatorio.',
        'genero.in' => 'El género debe ser hombre o mujer.'
    ];

    $validator->setCustomMessages($messages);

    if ($validator->fails()) {
        return response()->json($validator->errors(), 400);
    }

    $credentials = $validator->validated();
    $credentials['password'] = bcrypt($credentials['password']);

    $user = User::create($credentials);
    return response()->json(['user' => $user], 201);
}
```

- *Método login()*

Obtiene las credenciales de inicio de sesión del cuerpo de la solicitud y las pasa al método attempt() de la clase Auth. Si las credenciales son válidas, devuelve una respuesta JSON con un token JWT (JSON Web Token) válido. Si las credenciales no son válidas, devuelve una respuesta JSON con un mensaje de error

```
public function login()
{
    $credentials = request(['email', 'password']);

    if (! $token = auth()->attempt($credentials)) {
        return response()->json(['error' => 'Unauthorized'], 401);
    }

    return $this->respondWithToken($token);
}
```

- *Método respondWithToken()*

Este método devuelve una respuesta JSON estructurada con el token de acceso, tipo de token, tiempo de expiración del token, datos del usuario y sus roles.

4.2.3 AUTHUSER.JSX

Este módulo define un hook personalizado de React para manejar la autenticación del usuario y las autorizaciones dentro de una aplicación. También nos ayuda con las recuperaciones de tokens y datos de usuarios, así como la verificación de roles específicos.

- *Verificación de Roles.*

IsAdministrator: Verifica si el usuario autenticado tiene el rol de administrador. Para ello, obtiene el usuario desde sessionStorage. Parsea el string JSON almacenado para obtener el objeto user y comprueba si el usuario tiene el rol de 'admin'.

IsDirector: Verifica si el usuario tiene el rol de director, similar a isAdministrator, pero busca el rol 'director'.

- *Tokens y Usuario*

getToken: Obtiene y devuelve el token del usuario desde sessionStorage y parsea el string JSON almacenado para obtener el token.

getUser: Obtiene y devuelve los detalles del usuario desde sessionStorage y parsea el string JSON almacenado para obtener el objeto user_detail.

- *Funciones de Gestión de Tokens*

saveToken: Guarda el token y los detalles del usuario en sessionStorage. Actualiza los estados token y user y redirige a la ruta raíz (/).

logout: Limpia el sessionStorage y redirige a la ruta raíz (/).

Documentación – Código

- *Configuración de axios*

http: Instancia de axios configurada con la URL base de la API y los headers necesarios. Incluye el token en los headers de autorización.

- *Retorno del Hook*

La función AuthUser retorna un objeto con las siguientes propiedades:

setToken: La función saveToken para guardar el token.
token: El token actual.
user: Los detalles del usuario actual.
getToken: La función para obtener el token.
http: La instancia configurada de axios.
logout: La función para cerrar sesión.
isAdministrator: La función para verificar si el usuario es administrador.
isDirector: La función para verificar si el usuario es director.

```
export default function AuthUser(){
  const navigate = useNavigate();

  const isAdministrator = () => {
    const userString = sessionStorage.getItem('user');
    const user = userString ? JSON.parse(userString) : null;

    if (!user || !user.roles) return false;

    const isAdmin = user.roles.some(role => role.name === 'admin');
    return isAdmin;
  }

  const isDirector = () => {
    const userString = sessionStorage.getItem('user');
    const user = userString ? JSON.parse(userString) : null;

    if (!user || !user.roles) return false;

    const isDirector = user.roles.some(role => role.name === 'director');
    return isDirector;
  }

  const getToken = () =>{
    const tokenString = sessionStorage.getItem('token');
    const userToken = JSON.parse(tokenString);
    return userToken;
  }

  const getUser = () =>{
    const userString = sessionStorage.getItem('user');
    const user_detail = JSON.parse(userString);
    return user_detail;
  }

  const [token, setToken] = useState(getToken());
  const [user, setUser] = useState(getUser());

  const saveToken = (user, token) =>{
    console.log(user);
    sessionStorage.setItem('token', JSON.stringify(token));
    sessionStorage.setItem('user', JSON.stringify(user));

    setToken(token);
    setUser(user);
    navigate('/');
  }

  const logout = () => {
    sessionStorage.clear();
    navigate('/');
  }
}
```

```
const http = axios.create({
  baseURL: "http://localhost:8000/api",
  headers: {
    "Content-type": "application/json",
    "Authorization": `Bearer ${token}`
  }
});
return {
  setToken: saveToken,
  token,
  user,
  getToken,
  http,
  logout,
  isAdministrator,
  isDirector,
}
}
```

4.2.4 APP.JSX

Hooks y estado

Se utiliza el hook useEffect para realizar operaciones secundarias después de que el componente se monta o actualiza. Se utiliza el hook useState para mantener el estado del rol del usuario. Se actualiza el estado del rol del usuario en función de su autenticación y rol dentro de la aplicación. Se renderizan los componentes de las diferentes páginas según el rol del usuario.

Documentación – Código

Si el usuario no está autenticado, se muestra el componente Guest.

Suspense

Se utiliza el componente Suspense para manejar la carga diferida de los componentes y proporcionar una experiencia de carga suave. Mientras se cargan los componentes, se muestra un mensaje de carga.

```
import React, { Suspense, lazy, useEffect, useState } from 'react';
import AuthUser from './components/AuthUser';

const Guest = lazy(() => import('./Pages/navbar/Guest'));
const Auth = lazy(() => import('./Pages/navbar/Auth'));
const AuthAdmin = lazy(() => import('./Pages/navbar/AuthAdmin'));
const AuthDirector = lazy(() => import('./Pages/navbar/AuthDirector'));

function App() {
  const { getToken, isAdministrator, isDirector } = AuthUser();
  const [role, setRole] = useState(null);

  useEffect(() => {
    const token = getToken();
    const updateRole = () => {
      if (!token) {
        setRole(null);
      } else if (isDirector()) {
        setRole('director');
      } else if (isAdministrator()) {
        setRole('admin');
      } else {
        setRole('user');
      }
    };
    updateRole();
  }, [getToken, isAdministrator, isDirector]);

  return (
    <Suspense fallback={<div>Loading...</div>}>
      {role === 'director' ? <AuthDirector /> :
       role === 'admin' ? <AuthAdmin /> :
       role === 'user' ? <Auth /> : <Guest />}
    </Suspense>
  );
}

export default App;
```

4.2.4 REGISTER.JSX

¡¡ALERT!! NO VOY A ENSEÑAR LOS RETURNS, DEBIDO A QUE ES DONDE SE APLICAN TODOS LOS MÉTODOS CREADOS DE LOS COMPONENTES Y ADEMÁS, LOS RETURN OCUPAN DEMASIADAS LÍNEAS DE CÓDIGO.

Este componente maneja el registro de usuarios en la página web.

Props

- onClose: Propiedad que se utiliza para cerrar el formulario de registro.

Estados

- name, email, password, fechaNacimiento, telefono, genero: Estados que almacenan los valores de los campos del formulario de registro.
- errors: Estado que almacena los errores de validación del formulario.
- isLoginOpen: Estado que gestiona la visibilidad del componente de inicio de sesión.
- generoOptions: Estado que almacena las opciones disponibles para el campo género.

Documentación – Código

UseEffect

- Se utiliza para inicializar las opciones del campo género cuando el componente se monta.
- `setGeneroOptions(['Selecciona tu género', 'Hombre', 'Mujer']);`

Función submitForm

- Maneja el envío del formulario de registro.
- `event.preventDefault()`: Previene el comportamiento por defecto del formulario (recarga de la página).
- Realiza una solicitud POST a `/register` utilizando la instancia http de axios proporcionada por AuthUser.
- Si la solicitud es exitosa, llama a `onClose` para cerrar el formulario.

Función handleLoginOpen

- Abre el componente de inicio de sesión estableciendo `isLoginOpen` en true.

```
import { useState, useEffect } from 'react';
import AuthUser from './AuthUser';
import menuNav from '../Pages/Image/register.webp';
import LoginSection from './LoginSection';

export default function Register({ onClose }) {
  const [http, setToken] = AuthUser();
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [fechaNacimiento, setFechaNacimiento] = useState('');
  const [telefono, setTelefono] = useState('');
  const [errors, setErrors] = useState({});
  const [isLoginOpen, setIsLoginOpen] = useState(false);
  const [genero, setGenero] = useState('');
  const [generoOptions, setGeneroOptions] = useState([]);

  useEffect(() => {
    setGeneroOptions(['Selecciona tu género', 'Hombre', 'Mujer']);
  }, []);

  const submitForm = (event) => {
    event.preventDefault();

    http.post('/register', {
      name,
      email,
      password,
      fecha_nacimiento: fechaNacimiento,
      telefono: telefono,
      genero: genero
    }).then((res) => {
      onClose();
    }).catch(error => {
      if (error.response && error.response.status === 400) {
        setErrors(error.response.data);
      } else {
        setErrors({ general: "Ha ocurrido un error inesperado. Por favor, intenta de nuevo más tarde." });
      }
    });
  };

  const handleLoginOpen = () => {
    setIsLoginOpen(true);
  };
}
```

4.2.5 LOGINSECTION.JSX

Props

- `onClose`: Propiedad que se utiliza para cerrar el componente de inicio de sesión.

Documentación – Código

Estados

- email, password: Estados que almacenan los valores de los campos de correo electrónico y contraseña del formulario de inicio de sesión.
- isRegisterOpen: Estado que gestiona la visibilidad del componente de registro.
- isMenuOpen: Estado que gestiona la visibilidad de un menú.
- loginError: Estado que almacena un mensaje de error en caso de que falle el inicio de sesión.

Función submitForm

- Maneja el envío del formulario de inicio de sesión.
- event.preventDefault(): Previene el comportamiento por defecto del formulario (recarga de la página).
- Realiza una solicitud POST a /login utilizando la instancia http de axios proporcionada por AuthUser.
- Si la solicitud es exitosa, llama a setToken con los datos de usuario y el token de acceso recibidos en la respuesta.

useEffect

- Gestiona el desplazamiento del cuerpo del documento según el estado de isMenuOpen.
- Si isMenuOpen es true, se bloquea el desplazamiento.
- Cuando el componente se desmonta, se restaura el desplazamiento del cuerpo.

Función handleRegisterClose

- Cierra el componente de registro y el componente de inicio de sesión llamando a onClose.

```
import { useEffect, useState } from 'react';
import AuthUser from './AuthUser';
import Register from './Register';
import menuNav from '../Pages/image/login.webp';

export default function LoginSection({ onClose }) {
    const { http, setToken } = AuthUser();
    const [email, setEmail] = useState('');
    const [password, setPassword] = useState('');
    const [isRegisterOpen, setIsRegisterOpen] = useState(false);
    const [isMenuOpen, setIsMenuOpen] = useState(false);
    const [loginError, setLoginError] = useState('');

    const submitForm = (e) => {
        e.preventDefault();
        http.post('/login', { email: email, password: password })
            .then((res) => {
                setToken(res.data.user, res.data.access_token);
            })
            .catch((error) => {
                setLoginError('Error: No se pudo iniciar sesión. Por favor, verifica tus credenciales.');
            });
    };

    useEffect(() => {
        if (isMenuOpen) {
            document.body.style.overflow = 'hidden';
        } else {
            document.body.style.overflow = 'auto';
        }

        return () => {
            document.body.style.overflow = 'auto';
        };
    }, [isMenuOpen]);

    const handleRegisterClose = () => {
        setIsRegisterOpen(false);
        onClose();
    };
}
```

4.2.6 GUEST.JSX

Esta función define un componente funcional en React llamado Guest, que representa la interfaz de usuario para usuarios que no han iniciado sesión.

Estos enlaces lo usa la librería Navigate para mandar al usuario a lugar u otro.

4.2.7 AUTH.JSX

Este componente representa la interfaz de usuario para usuarios autenticados. Es decir, se encarga de los enlaces en los que puede entrar el usuario y cada uno de ello conectado a un componente.

4.2.8 AUTHADMIN.JSX

Este componente se encarga de mostrar la interfaz de usuario para el administrador una vez que ha iniciado sesión y se ha verificado su rol.

Utiliza el hook AuthUser para obtener la función isAdministrator, que verifica si el usuario actual es un administrador. Si el usuario no es un administrador, redirige al usuario a la página de inicio.

4.2.9 AUTHDIRECTOR.JSX

Este componente se encarga de mostrar la interfaz de usuario para el director una vez que ha iniciado sesión y se ha verificado su rol.

Utiliza el hook AuthUser para obtener la función isDirector, que verifica si el usuario actual es un director. Si el usuario no es un director, redirige al usuario a la página de inicio.

4.3 ADMINCONTROLLER

Este controlador maneja las operaciones relacionadas con los AdminController.

Función index()

Este método devuelve una lista de todos los administradores en la base de datos. Utiliza la relación roles en el modelo User para filtrar los usuarios que tienen el rol de "admin".

Función añadir(Request \$request)

Este método se utiliza para agregar un nuevo administrador a la base de datos. Valida los datos recibidos del formulario utilizando las reglas de validación definidas en \$validatedData. Luego crea un nuevo usuario en la base de datos con los datos validados y le asigna el rol de administrador.

Documentación – Código

```
public function añadir(Request $request)
{
    $validatedData = $request->validate([
        'name' => 'required|string|max:255',
        'email' => 'required|string|email|max:255|unique:users',
        'password' => [
            'required',
            'string',
            'min:6',
            'regex:/[A-Z]/',
            'regex:/[0-9]/'
        ],
        'fecha_nacimiento' => [
            'required',
            'date',
            'before_or_equal:' . now()->subYears(18)->toDateString()
        ],
        'telefono' => [
            'required',
            'digits:9',
            'regex:/^([67][0-9]{8})$/'
        ],
        'genero' => 'required|in:hombre,mujer',
    ]);

    $user = User::create([
        'name' => $validatedData['name'],
        'email' => $validatedData['email'],
        'password' => Hash::make($validatedData['password']),
        'fecha_nacimiento' => $validatedData['fecha_nacimiento'],
        'telefono' => $validatedData['telefono'],
        'genero' => $validatedData['genero']
    ]);

    $adminRole = Role::findByName('admin');
    $user->assignRole($adminRole);

    return response()->json([
        'message' => 'Admin creado correctamente',
        'user' => $user
    ], 201);
}
```

Función editar(\$id)

Este método devuelve los detalles de un administrador específico basado en su ID.

Función eliminar(\$id)

Este método elimina un administrador específico de la base de datos basado en su ID.

Función actualizar(Request \$request, \$id)

Este método actualiza la información de un administrador específico en la base de datos. Primero encuentra al usuario basado en su ID. Luego valida los datos recibidos del formulario y actualiza los datos del usuario en la base de datos. Si se proporciona una nueva contraseña, la encripta antes de actualizarla en la base de datos.

Documentación – Código

```
public function actualizar(Request $request, $id)
{
    $user = User::findOrFail($id);

    $validatedData = $request->validate([
        'name' => 'required|string|max:255',
        'email' => 'required|string|email|max:255|unique:users,email,' . $id,
        'fecha_nacimiento' => 'required|date|before_or_equal:' . now()->subYears(18)->toDateString(),
        'telefono' => 'required|digits:9|regex:/[67][0-9]{8}$/',
        'password' => 'nullable|string|min:6|regex:/[A-Z]/|regex:/[0-9]/',
        'genero' => 'nullable|in:hombre,mujer',
    ]);

    $userData = [
        'name' => $validatedData['name'],
        'email' => $validatedData['email'],
        'fecha_nacimiento' => $validatedData['fecha_nacimiento'],
        'telefono' => $validatedData['telefono'],
    ];

    if ($request->has('password')) {
        $userData['password'] = Hash::make($validatedData['password']);
    }

    if ($request->has('genero')) {
        $userData['genero'] = $validatedData['genero'];
    }

    $user->update($userData);

    return response()->json(['message' => 'Admin actualizado correctamente', 'user' => $user]);
}
```

4.4 CARRITOCONTROLLERS

Este controlador gestiona las operaciones relacionadas con el carrito de compras en una aplicación web.

Función agregarProducto()

Este método se utiliza para agregar un producto al carrito de compras. Primero, valida los datos recibidos del formulario para garantizar que cumplan con las reglas especificadas. Luego, verifica si el producto existe y si hay suficiente stock disponible. Si el producto ya está en el carrito, se actualiza la cantidad; de lo contrario, se crea un nuevo registro en el carrito.

```
public function agregarProducto(Request $request)
{
    $request->validate([
        'user_id' => 'required|integer',
        'producto_id' => 'required|integer',
        'cantidad' => 'required|integer|min:1',
        'talla_id' => 'required|integer',
    ]);

    $producto = Producto::with('oferta')->find($request->producto_id);
    if (!$producto) {
        return response()->json(['message' => 'Producto no encontrado'], 404);
    }

    $precio = $producto->precio;
    if ($producto->oferta && now() ->between(new \DateTime($producto->oferta->inicio), new \DateTime($producto->oferta->fin))) {
        $precio = round($producto->precio * (1 + $producto->oferta->descuento / 100), 2);
    }

    $stock = Stock::where('producto_id', $request->producto_id)->where('talla_id', $request->talla_id)->first();
    if (!$stock || $request->cantidad > $stock->cantidad) {
        return response()->json(['message' => 'Stock insuficiente para la cantidad solicitada'], 400);
    }

    $carrito = Carrito::where('user_id', $request->user_id)
        ->where('producto_id', $request->producto_id)
        ->where('talla_id', $request->talla_id)
        ->first();

    if ($carrito) {
        $carrito->cantidadProducto += $request->cantidad;
    } else {
        $carrito = new Carrito();
        $carrito->user_id = $request->user_id;
        $carrito->producto_id = $request->producto_id;
        $carrito->talla_id = $request->talla_id;
        $carrito->cantidadProducto = $request->cantidad;
    }

    $carrito->precio = $precio;
    $carrito->save();

    return response()->json(['message' => 'Producto agregado al carrito correctamente'], 200);
}
```

Función mostrar(\$userId)

Este método muestra los productos actualmente en el carrito para un usuario específico. Recupera los productos del carrito y, si hay una oferta válida, calcula el precio con descuento. Luego, devuelve los productos en formato JSON.

Documentación – Código

```
public function mostrar($userId)
{
    $productosEnCarrito = Carrito::where('user_id', $userId)
    ->with(['producto.oferta', 'talla'])
    ->get()
    ->map(function ($item) {
        if ($item->producto->oferta && now()->between(new \DateTime($item->producto->oferta->inicio), new \DateTime($item->producto->oferta->fin))) {
            $descuento = $item->producto->oferta->descuento;
            $item->precio = round($item->producto->precio * (1 - $descuento / 100), 2);
        } else {
            $item->precio = $item->producto->precio;
        }
        return $item;
    });
    $numeroDeElementos = $productosEnCarrito->count();
    return response()->json($productosEnCarrito);
}

public function eliminarProducto($userId, $productId)
{
    try {
        $carrito = Carrito::where('user_id', $userId)
            ->where('producto_id', $productId)
            ->first();

        $carrito->delete();
        return response()->json(['message' => 'Producto eliminado del carrito correctamente'], 200);
    } catch (\Exception $e) {
        return response()->json(['message' => 'Error al eliminar el producto del carrito', 'error' => $e->getMessage()], 500);
    }
}
```

Función eliminarProducto(\$userId, \$productId)

Este método elimina un producto específico del carrito de un usuario. Busca y elimina el producto del carrito según el ID del usuario y el ID del producto.

Función incrementarCantidad(\$userId, \$productId)

Este método incrementa la cantidad de un producto en el carrito. Primero, busca el producto en el carrito y luego verifica si hay suficiente stock disponible. Si es así, aumenta la cantidad del producto en uno.

Función disminuirCantidad(\$userId, \$productId)

Este método disminuye la cantidad de un producto en el carrito. Busca el producto en el carrito y reduce su cantidad en uno si es mayor que uno.

Función cambiarTalla(Request \$request, \$userId)

Este método se utiliza para cambiar la talla de un producto en el carrito. Valida los datos recibidos del formulario y actualiza la talla del producto en el carrito. Si se encuentra algún error durante el proceso, devuelve un mensaje de error correspondiente.

```
public function cambiarTalla(Request $request, $userId)
{
    $request->validate([
        'producto_id' => 'required|integer|exists:productos,id',
        'nueva_talla_id' => 'required|integer|exists:tallas,id'
    ]);

    try {
        $carritoItem = Carrito::where('user_id', $userId)
            ->where('producto_id', $request->producto_id)
            ->firstOrFail();

        $carritoItem->talla_id = $request->nueva_talla_id;
        $carritoItem->cantidadProducto = 1;
        $carritoItem->save();

        return response()->json(['message' => 'Talla y cantidad actualizadas correctamente'], 200);
    } catch (\Exception $e) {
        return response()->json(['message' => 'Error al actualizar la talla del producto en el carrito', 'error' => $e->getMessage()], 404);
    }
}
```

4.5 CATEGORIACONTROLLER

Este controlador maneja las operaciones relacionadas con las categorías y productos.

Función index()

Este método recupera todas las categorías disponibles en la base de datos y las devuelve en formato JSON.

Función productosPorCategoria(\$categoriaId)

Este método recupera todos los productos que pertenecen a una categoría específica, identificada por el ID de la categoría, y los devuelve en formato JSON.

Función getCategoriaByProducto(\$id)

Este método recupera la categoría asociada a un producto específico, identificado por el ID del producto. Devuelve la categoría en formato JSON, o un mensaje de error si la categoría no se encuentra.

Función productosPorNombreYGenero(\$nombre, \$genero)

Este método recupera todos los productos de una categoría específica basada en el nombre y género. También maneja las ofertas asociadas a los productos, ajustando el precio si hay una oferta válida y actualizando el estado de la oferta si ha expirado.

```
public function productosPorNombreYGenero($nombre, $genero)
{
    $categoria = Categorias::where('nombre', $nombre) ->where('genero', $genero) ->first();

    if ($categoria) {
        $productos = Producto::with('oferta')
            ->where('categoria_id', $categoria->id)
            ->where('activo', true)
            ->get();

        $now = Carbon::now();
        $productos->transform(function ($producto) use ($now) {
            if ($producto->oferta) {
                $inicio = new Carbon($producto->oferta->inicio);
                $fin = new Carbon($producto->oferta->fin);
                if ($now->greaterThan($fin) && $producto->oferta->estado !== 'expirada') {
                    $producto->oferta->estado = 'expirada';
                    $producto->oferta->save();
                    $producto->oferta_id = null;
                    $producto->save();
                } elseif ($now->between($inicio, $fin)) {
                    $producto->precio_con_descuento = round($producto->precio * (1 - $producto->oferta->descuento / 100), 2);
                }
            }
            return $producto;
        });
        return response()->json($productos);
    } else []
        return response()->json(['message' => 'Categoría no encontrada'], 404);
}
```

Función productosPorNombreYEstacion(\$nombre, \$estacion)

Este método recupera todos los productos de una categoría específica basada en el nombre y la estación del año. Similar al método anterior, maneja las ofertas asociadas y ajusta los precios y estados de las ofertas según corresponda.

Documentación – Código

```
public function productosPorNombreYEstacion($nombre, $estacion)
{
    $categoria = Categorias::where('nombre', $nombre)->where('estacion', $estacion)->first();

    if ($categoria) {
        $productos = Producto::with('oferta')
            ->where('categoria_id', $categoria->id)
            ->where('activo', true)
            ->get();

        $now = Carbon::now();
        $productos->transform(function ($producto) use ($now) {
            if ($producto->oferta) {
                $inicio = new Carbon($producto->oferta->inicio);
                $fin = new Carbon($producto->oferta->fin);
                if ($now->greaterThan($fin) && $producto->oferta->estado !== 'expirada') {
                    $producto->oferta->estado = 'expirada';
                    $producto->oferta->save();
                    $producto->oferta_id = null;
                    $producto->save();
                } elseif ($now->between($inicio, $fin)) {
                    $producto->precio_con_descuento = round($producto->precio * (1 - $producto->oferta->descuento / 100), 2);
                }
            }
            return $producto;
        });
        return response()->json($productos);
    } else {
        return response()->json(['message' => 'Categoría no encontrada'], 404);
    }
}
```

4.6 FACTURACONTROLLER

Este controlador maneja las operaciones relacionadas con las facturas.

Función añadir(Request \$request)

Este método agrega una nueva factura a la base de datos. Valida los datos del request, crea una nueva factura y la guarda en la base de datos.

Función obtenerFacturas(\$userId)

Este método recupera todas las facturas de un usuario específico, identificado por el ID del usuario, y las devuelve en formato JSON.

Función totalFacturas()

Este método calcula la suma total de todas las facturas y la devuelve en formato JSON.

Función facturasHoy()

Este método cuenta el número de facturas creadas en el día actual y devuelve el resultado en formato JSON.

Función todasFacturas()

Este método recupera todas las facturas en la base de datos y las devuelve en formato JSON.

Función actualizarEstadoEnvio(Request \$request, \$id)

Este método actualiza el estado de envío de una factura específica. Valida el estado del envío y actualiza el estado en la base de datos.

Función obtenerFacturasConNombreCliente()

Documentación – Código

Este método recupera todas las facturas junto con los nombres de los clientes y las devuelve en formato JSON.

Función todasFacturasGenero()

Este método recupera todas las facturas junto con la información de los usuarios asociados para contar y categorizar el número de facturas según el género del usuario (hombre o mujer). Tras obtener los datos, se organiza esta información en un formato específico para la respuesta JSON.

```
public function todasFacturasGenero()
{
    try {
        $facturas = Factura::with('user')->get();

        $generoCounts = [
            'hombre' => 0,
            'mujer' => 0,
        ];

        foreach ($facturas as $factura) {
            if ($factura->user) {
                $genero = $factura->user->genero;
                if (isset($generoCounts[$genero])) {
                    $generoCounts[$genero]++;
                }
            }
        }

        $data = [];
        foreach ($generoCounts as $genero => $count) {
            $data[] = [
                'name' => ucfirst($genero),
                'value' => $count,
            ];
        }

        return response()->json($data);
    } catch (\Exception $e) {
        return response()->json(['error' => 'Error fetching invoices', 'message' => $e->getMessage()]);
    }
}
```

Función todasFacturasEdades()

Este método obtiene todas las facturas junto con la información de los usuarios, calcula las edades de los usuarios en base a sus fechas de nacimiento, y agrupa las facturas en rangos de edad: 18-29, 30-39, 40-49, 50-59 y 60+. El resultado es un arreglo con el conteo de facturas en cada rango de edad, el cual se devuelve en formato JSON.

Documentación – Código

```
public function todasFacturasEdades()
{
    try {
        $facturas = Factura::with('user')->get();
        $fechaActual = Carbon::now();
        $edadesCounts = [
            '+18' => 0,
            '+30' => 0,
            '+40' => 0,
            '+50' => 0,
            '+60' => 0,
        ];
        foreach ($facturas as $factura) {
            if ($factura->user && $factura->user->fecha_nacimiento) {
                $fechaNacimiento = Carbon::parse($factura->user->fecha_nacimiento);
                $edad = $fechaActual->diffInYears($fechaNacimiento);

                if ($edad >= 18 && $edad < 30) {
                    $edadesCounts['+18']++;
                } elseif ($edad >= 30 && $edad < 40) {
                    $edadesCounts['+30']++;
                } elseif ($edad >= 40 && $edad < 50) {
                    $edadesCounts['+40']++;
                } elseif ($edad >= 50 && $edad < 60) {
                    $edadesCounts['+50']++;
                } elseif ($edad >= 60) {
                    $edadesCounts['+60']++;
                }
            }
        }
        $data = [];
        foreach ($edadesCounts as $rangoEdad => $count) {
            $data[] = [
                'name' => $rangoEdad,
                'value' => $count,
            ];
        }
        return response()->json($data);
    } catch (\Exception $e) {
        return response()->json(['error' => 'Error fetching invoices', 'message' => $e->getMessage()]);
    }
}
```

Función topClientes()

Este método obtiene los 10 clientes que más han gastado en total. Primero, consulta la tabla de facturas para agruparlas por user_id y calcular la suma del total gastado por cada usuario. Luego, ordena estos resultados en orden descendente y selecciona los 10 principales. Para cada uno de estos clientes, recupera los detalles del usuario como el ID, nombre y correo electrónico, y devuelve esta información en formato JSON.

```

public function topClientes()
{
    try {
        $topCustomers = Factura::selectRaw('user_id, SUM(total) as total_spent')
            ->groupBy('user_id')
            ->orderByDesc('total_spent')
            ->limit(10)
            ->get();

        $detailedCustomers = [];
        foreach ($topCustomers as $customer) {
            $user = User::find($customer->user_id);
            if ($user) {
                $detailedCustomers[] = [
                    'id' => $user->id,
                    'name' => $user->name,
                    'email' => $user->email,
                    'totalSpent' => $customer->total_spent,
                ];
            }
        }

        return response()->json($detailedCustomers);
    } catch (\Exception $e) {
        return response()->json(['error' => 'Error fetching top customers', 'message' => $e->getMessage()]);
    }
}

```

4.7 OFERTACONTROLLER

Este controlador permiten a los administradores crear, asignar, actualizar y eliminar ofertas, así como asociar ofertas a productos y categorías de productos.

Función crearOferta(Request \$request)

Este método permite crear una nueva oferta promocional. La solicitud debe contener el nombre de la oferta, el porcentaje de descuento, y las fechas de inicio y fin de la oferta. El método valida estos datos y, si son correctos, guarda la nueva oferta en la base de datos.

Función asignarOferta(Request \$request)

Este método asigna una oferta existente a un producto específico. Se asegura de que los IDs del producto y de la oferta sean válidos y que las fechas de la oferta sean coherentes. Esto permite aplicar descuentos a productos individuales según las promociones vigentes.

Función listarOfertas()

Este método recupera todas las ofertas disponibles en la base de datos y las devuelve en formato JSON.

Función obtenerOferta(\$id)

Recupera una oferta específica por su ID. Si la oferta existe, la devuelve en formato JSON.

Función actualizarOferta(Request \$request, \$id)

Permite actualizar los detalles de una oferta existente. Valida la información

de la solicitud (nombre, descuento, fechas) y, si es correcta, actualiza la oferta en la base de datos.

Función asignarOfertaCat(Request \$request)

Este método asigna una oferta a todos los productos dentro de una categoría específica. Valida los IDs de la categoría y la oferta, y verifica las fechas. Luego, aplica la oferta a cada producto de la categoría.

```
public function asignarOfertaCat(Request $request)
{
    $request->validate([
        'categoria_id' => 'required|exists:categorias,id',
        'oferta_id' => 'nullable|exists:ofertas,id'
    ]);

    $productos = Producto::where('categoria_id', $request->categoria_id)->get();

    if ($request->oferta_id) {
        $oferta = Oferta::find($request->oferta_id);
        if ($oferta && $oferta->fin <= $oferta->inicio) {
            return response()->json(['message' => 'La fecha de fin de la oferta no puede ser menor o igual que la fecha de inicio'], 400);
        }
    }

    foreach ($productos as $producto) {
        $producto->oferta_id = $request->oferta_id;
        $producto->save();
    }

    return response()->json(['message' => 'Oferta asignada a todos los productos de la categoría exitosamente']);
}
```

Función quitarOfertaCat(Request \$request)

Permite eliminar una oferta de todos los productos de una categoría. Valida el ID de la categoría y luego elimina la oferta de cada producto en esa categoría(pasándola a null).

4.8 PAYMENTCONTROLLER

Este controlador maneja las operaciones relacionadas con los pagos, específicamente utilizando el servicio de PayPal.

Función construct(PaypalService \$paypalService)

El constructor inyecta una instancia del PaypalService y la asigna a la propiedad protegida \$paypalService.

Función createPayment()

Este método inicia el proceso de creación de un pago. Obtiene el contexto de la API de PayPal a través del servicio de PayPal (\$this->paypalService->getApiContext()).

PAYPAL_CLIENT_ID Este es el ID del cliente proporcionado por PayPal cuando registras tu aplicación en la consola de desarrolladores de PayPal.

PAYPAL_SECRET Este es el secreto del cliente asociado con tu ID de cliente. Es utilizado junto con el ID del cliente para autenticar las solicitudes a la API de PayPal.

```
PAYPAL_CLIENT_ID=AduEizYvT8L-ev_seo3aysBuRYeVLxPQfW3aE5vkGKNktsffwXptc5NcvuTAUEEvUmNtlDq2xmVEmMlUm  
PAYPAL_SECRET=EJSfM72Xah6o2DnKtty6zpWAjMPz6uDS-25JklsVUrD-UoBgUg5UjksYiovxRsvB07e38cN0futRAWWN  
PAYPAL_MODE=sandbox
```

4.9 PUBLICAPICONTRROLLER

Este controlador gestiona varias operaciones relacionadas con los productos, categorías, tallas y ofertas para los desarrolladores que quieran una API pública relacionada con la prendas de ropa.

Función getAllProductos(Request \$request)

Este método recupera y devuelve todos los productos activos, permitiendo filtrar por nombre y disponibilidad de stock si se especifican en la solicitud. Además, procesa las ofertas asociadas a cada producto: si la oferta ha expirado, la marca como expirada y la disocia del producto; si está vigente, calcula el precio con el descuento aplicado. Finalmente, retorna los productos procesados en una respuesta JSON.

Función getProducto(\$id)

Obtiene un producto específico por su ID, junto con su oferta asociada si está vigente, y calcula el precio con descuento si la oferta está activa.

Función getProductosByCategoria(\$categoriaId)

Recupera todos los productos activos pertenecientes a una categoría específica identificada por su ID.

Función getProductosByTalla()

Obtiene y devuelve todas las tallas disponibles en la base de datos.

Función getAllCategorias()

Recupera y devuelve todas las categorías de productos disponibles en la base de datos.

Función getAllOfertas()

Recupera y devuelve todas las ofertas disponibles en la base de datos.

Función getOferta(\$id)

Obtiene una oferta específica por su ID y la devuelve.

Función getTallaById(\$tallaId)

Recupera y devuelve una talla específica por su ID.

Función getTallaByName(\$nombre)

Recupera y devuelve una talla específica por su nombre.

Función getProductosByName(\$nombre)

Busca y devuelve el primer producto que coincide con el nombre proporcionado.

Función getProductosByPrecio(\$precio)

Busca y devuelve el primer producto que tiene el precio especificado.

4.10 PRODUCTOSCONTROLLERS

Este controlador realiza operaciones como la creación, modificación, eliminación y visualización de productos, así como la gestión de sus ofertas asociadas.

Además, también maneja la activación y desactivación de productos, así como la obtención de información estadística, como el recuento de productos activos

Función index()

Recupera todos los productos, incluyendo sus ofertas asociadas.

Verifica si las ofertas están activas, expiradas o vigentes y actualiza el estado de las ofertas y los precios con descuento si es necesario.

Devuelve una lista de productos en formato JSON.

```
public function index()
{
    $now = Carbon::now();
    $productos = Producto::with(['oferta'])->get()->map(function ($producto) use ($now) {
        if ($producto->oferta) {
            $inicio = new Carbon($producto->oferta->inicio);
            $fin = new Carbon($producto->oferta->fin);
            if ($now->greaterThan($fin) && $producto->oferta->estado !== 'expirada') {
                $producto->oferta->estado = 'expirada';
                $producto->oferta->save();

                $producto->oferta_id = null;
                $producto->save();
            } elseif ($now->between($inicio, $fin)) {
                $producto->precio_con_descuento = round($producto->precio * (1 - $producto->oferta->descuento / 100), 2);
            }
        }
        return $producto;
    });
    return response()->json($productos);
}
```

Función mostrar(\$id)

Recupera un producto específico por su ID junto con su oferta asociada.

Calcula y asigna el precio con descuento si la oferta está activa.

Devuelve el producto en formato JSON.

```
public function mostrar($id)
{
    $producto = Producto::with('oferta')->findOrFail($id);

    if ($producto->oferta && now()->between(new \DateTime($producto->oferta->inicio), new \DateTime($producto->oferta->fin))) {
        $producto->precio_con_descuento = round($producto->precio * (1 - $producto->oferta->descuento / 100), 2);
    }

    return response()->json($producto);
}
```

Función añadir(Request \$request)

Valida los datos de entrada y crea un nuevo producto con la información proporcionada. Asigna imágenes si están presentes y las guarda en el almacenamiento público. Inicializa el stock del producto con una cantidad de 1 para todas las tallas disponibles.

Devuelve el producto creado en formato JSON con un estado HTTP 201 (Creado).

Documentación – Código

```
public function añadir(Request $request)
{
    $request->validate([
        'nombre' => 'required|string|max:255',
        'descripcion' => 'required|string',
        'precio' => 'required|numeric',
        'principalImg' => 'nullable|image',
        'img1' => 'nullable|image',
        'categoria_id' => 'required|exists:categorias,id'
    ]);

    $producto = new Producto();
    $producto->nombre = $request->nombre;
    $producto->descripcion = $request->descripcion;
    $producto->precio = $request->precio;
    $producto->principalImg = $request->principalImg ? $request->principalImg->store('productos', 'public') : null;
    $producto->img1 = $request->img1 ? $request->img1->store('productos', 'public') : null;
    $producto->categoria_id = $request->categoria_id;
    $producto->save();

    $tallas = Talla::all();
    foreach ($tallas as $talla) {
        $stock = new Stock();
        $stock->producto_id = $producto->id;
        $stock->talla_id = $talla->id;
        $stock->cantidad = 1;
        $stock->save();
    }

    $defaultSizes = Talla::all();
    foreach ($defaultSizes as $size) {
        $producto->tallas()->attach($size->id);
    }
}

return response()->json($producto, 201);
}
```

Función eliminar(\$id)

Busca un producto por su ID y lo elimina si existe.
Devuelve un mensaje de éxito o un error 404 si el producto no se encuentra.

Función actualizar(Request \$request, \$id)

Busca un producto por su ID y actualiza sus datos con la información proporcionada en la solicitud. Actualiza las imágenes si se proporcionan nuevas imágenes. Devuelve el producto actualizado en formato JSON.

```
public function actualizar(Request $request, $id)
{
    $producto = Producto::findOrFail($id);

    $data = $request->all();

    if ($request->hasFile('principalImg')) {
        $data['principalImg'] = $request->file('principalImg')->store('productos', 'public');
    }

    if ($request->hasFile('img1')) {
        $data['img1'] = $request->file('img1')->store('productos', 'public');
    }

    $producto->update($data);

    return response()->json($producto);
}
```

Documentación – Código

Función productosRecomendados(\$id)

Encuentra un producto por su ID y recupera productos recomendados de la misma categoría. Filtra los productos recomendados para excluir el producto actual y solo incluye productos activos. Devuelve los productos recomendados en formato JSON.

```
public function productosRecomendados($id)
{
    $productoActual = Producto::findOrFail($id);
    $productosRecomendados = Producto::where('categoria_id', $productoActual->categoria_id)
        ->where('id', '!=', $id)
        ->where('activo', true)
        ->inRandomOrder()
        ->limit(100)
        ->get();

    return response()->json($productosRecomendados);
}

public function activar($id)
{
    try {
        $producto = Producto::findOrFail($id);
        $producto->activo = true;
        $producto->save();

        return response()->json(['message' => 'Producto activado con éxito'], 200);
    } catch (\Exception $e) {
        Log::error("Error al activar el producto: {$e->getMessage()}");
        return response()->json(['error' => 'Error interno del servidor'], 500);
    }
}
```

Función activar(\$id)

Intenta encontrar el producto correspondiente en la base de datos utilizando el ID proporcionado. Si el producto se encuentra, se establece su estado como activo (activo = true) y se guarda en la base de datos. Se devuelve un mensaje de éxito en formato JSON con un código de estado HTTP 200 si el proceso se realiza correctamente.

```
public function activar($id)
{
    try {
        $producto = Producto::findOrFail($id);
        $producto->activo = true;
        $producto->save();

        return response()->json(['message' => 'Producto activado con éxito'], 200);
    } catch (\Exception $e) {
        Log::error("Error al activar el producto: {$e->getMessage()}");
        return response()->json(['error' => 'Error interno del servidor'], 500);
    }
}
```

Función desactivar(\$id)

Intenta encontrar el producto correspondiente en la base de datos utilizando el ID proporcionado. Si el producto se encuentra, se establece su estado como

inactivo (activo = false) y se guarda en la base de datos. Se devuelve un mensaje de éxito en formato JSON con un código de estado HTTP 200 si el proceso se realiza correctamente.

```
public function desactivar($id)
{
    try {
        $producto = Producto::findOrFail($id);
        $producto->activo = false;
        $producto->save();

        return response()->json(['message' => 'Producto desactivado con éxito'], 200);
    } catch (\Exception $e) {
        Log::error("Error al desactivar el producto: {$e->getMessage()}");
        return response()->json(['error' => 'Error interno del servidor'], 500);
    }
}
```

Función countActiveProducts()

Cuenta el número de productos activos. Devuelve el conteo de productos activos en formato JSON.

4.11 STOCKCONTROLLER

Este controlador proporciona una serie de métodos que permiten realizar operaciones relacionadas con el stock, como obtener el stock para un producto y una talla específicos, incrementar o disminuir el stock disponible, y obtener una lista completa de todos los stocks disponibles en la base de datos

Función getStockPorProductoTalla(\$productId, \$tallaId)

Este método busca un registro de stock específico para un producto y talla dados. Si el stock no se encuentra, devuelve un mensaje de error con un código de estado 404. Si se encuentra, devuelve los detalles del stock en formato JSON.

Función getTodoStock()

Este método recupera todos los registros de stock disponibles en la base de datos, incluyendo información sobre el producto y la talla asociados. Si no hay stock disponible, devuelve un mensaje de error con un código de estado 404. Si hay stock disponible, devuelve los detalles de todos los registros de stock en formato JSON.

Función getProductoStock(\$productId)

Este método recupera todos los registros de stock para un producto específico, incluyendo información sobre la talla asociada. Si no hay stock disponible para el producto, devuelve un mensaje de error con un código de estado 404. Si hay stock disponible, devuelve los detalles de los registros de stock en formato JSON.

Función incrementarStock(Request \$request, \$stockId)

Este método incrementa la cantidad de stock para un registro específico en la base de datos. Utiliza la cantidad proporcionada en la solicitud para aumentar

Documentación – Código

la cantidad de stock. Devuelve un mensaje de éxito con los detalles actualizados del stock en formato JSON.

```
public function incrementarStock(Request $request, $stockId)
{
    $cantidad = $request->input('cantidad', 1);
    $stock = Stock::findOrFail($stockId);
    $stock->cantidad += $cantidad;
    $stock->save();

    return response()->json(['message' => 'Stock incrementado con éxito', 'stock' => $stock]);
}
```

Función disminuirStock(Request \$request)

Este método disminuye la cantidad de stock para un producto y talla específicos en la base de datos. Utiliza la cantidad proporcionada en la solicitud para disminuir la cantidad de stock. Si no hay suficiente stock disponible para realizar la operación, devuelve un mensaje de error con un código de estado 400. Si la operación se realiza con éxito, devuelve un mensaje de éxito con los detalles actualizados del stock en formato JSON.

```
public function disminuirStock(Request $request)
{
    $productoId = $request->input('producto_id');
    $tallaId = $request->input('talla_id');
    $cantidad = $request->input('cantidad');

    $stock = Stock::where('producto_id', $productoId)->where('talla_id', $tallaId)->first();

    if (!$stock || $stock->cantidad < $cantidad) {
        return response()->json(['message' => 'No hay suficiente stock para realizar la operación'], 400);
    }

    $stock->cantidad -= $cantidad;
    $stock->save();

    return response()->json(['message' => 'Stock actualizado con éxito', 'stock' => $stock]);
}
```

Función decrementarStock(Request \$request, \$stockId)

Similar al método disminuirStock, este método disminuye la cantidad de stock para un registro específico. Si la cantidad a disminuir es mayor que el stock disponible, devuelve un mensaje de error con un código de estado 400. Si la operación se realiza con éxito, devuelve un mensaje de éxito con los detalles actualizados del stock en formato JSON.

```
public function decrementarStock(Request $request, $stockId)
{
    $cantidad = $request->input('cantidad', 1);
    $stock = Stock::findOrFail($stockId);

    if ($stock->cantidad < $cantidad) {
        return response()->json(['error' => 'La cantidad a disminuir es mayor que el stock disponible'], 400);
    }

    $stock->cantidad -= $cantidad;
    $stock->save();

    return response()->json(['message' => 'Stock decrementado con éxito', 'stock' => $stock]);
}
```

4.12 TALLACONTROLLER

Este controlador se encarga de manejar las operaciones relacionadas con las tallas de los productos.

Función getTallasPorProducto(\$productId)

Este método espera que se proporcione el ID del producto como parámetro. Luego, el controlador busca el producto correspondiente en la base de datos junto con todas las tallas asociadas a ese producto. Utiliza la relación definida en el modelo Producto para cargar las tallas asociadas. Una vez que se obtienen las tallas, se devuelven como una respuesta JSON.

4.13 TAREACONTROLLER

Este controlador maneja operaciones CRUD (Crear, Leer, Actualizar y Eliminar) para la gestión de tareas de los administradores y director.

Función index()

Este método obtiene todas las tareas con información de usuario asociada y las devuelve como una respuesta JSON.

Función añadir(Request \$request)

Este método crea una nueva tarea utilizando los datos validados proporcionados en la solicitud. Los datos válidos incluyen el usuario asignado (assigned_to), el título (title), la descripción (description) y el estado (status). Si la validación es exitosa, se crea la tarea y se devuelve como una respuesta JSON.

Función actualizar(Request \$request, Tarea \$tarea)

Actualiza una tarea existente utilizando los datos validados proporcionados en la solicitud. Los datos válidos incluyen el usuario asignado (assigned_to), el título (title), la descripción (description) y el estado (status). Si la validación es exitosa, se actualiza la tarea y se devuelve como una respuesta JSON.

Función mostrar(Tarea \$tarea)

Devuelve los detalles de una tarea específica en formato JSON.

Función eliminar(Tarea \$tarea)

Elimina una tarea específica. No devuelve ningún contenido en la respuesta, solo un código de estado HTTP 204 (sin contenido) para indicar que la tarea se eliminó con éxito.

Función admin()

Este método está destinado a un usuario administrador. Obtiene todas las tareas asignadas al usuario autenticado actualmente que no están completadas y las devuelve como una respuesta JSON.

4.14 USERCONTROLLER

Este controlador maneja las solicitudes relacionadas con los usuarios, como obtener detalles de usuario, actualizar información de usuario, y mostrar información de usuario.

Función getUser(Request \$request)

Este método devuelve los detalles del usuario autenticado actualmente en formato JSON. Utiliza el método user() proporcionado por la solicitud para obtener el usuario.

Función mostrar(\$id)

Devuelve los detalles de un usuario específico según su ID. Si el usuario no existe, devuelve un mensaje JSON indicando que el usuario no se encontró con un código de estado HTTP 404.

Función actualizar(Request \$request)

Valida los datos proporcionados en la solicitud para actualizar el perfil del usuario. Se valida el nombre (name), el correo electrónico (email), y el teléfono (telefono). Si la validación falla, devuelve los errores de validación en formato JSON con un código de estado HTTP 422. Si la validación es exitosa, actualiza los detalles del usuario y devuelve los detalles actualizados en formato JSON.

4.15 WISHLISTCONTROLLER

Este controlador permite a los usuarios ver, agregar y eliminar productos de su lista de deseos.

Función mostrar(\$userId)

Este método recupera y devuelve todos los elementos de la lista de deseos para un usuario específico. Utiliza el modelo WishList para buscar los elementos de la lista de deseos asociados al ID de usuario proporcionado y carga los detalles de los productos asociados utilizando la relación definida en el modelo. Luego, devuelve estos elementos de la lista de deseos en formato JSON.

Función agregar(Request \$request)

Este método permite agregar un nuevo producto a la lista de deseos de un usuario. Primero, valida los datos recibidos en la solicitud para asegurarse de que se proporcionen el ID del producto y el ID del usuario. Luego, verifica si el producto ya existe en la lista de deseos del usuario. Si el producto ya está en la lista de deseos, devuelve un mensaje de conflicto. Si no, crea un nuevo registro en la tabla WishList con los detalles del usuario y el producto proporcionados en la solicitud y devuelve un mensaje de éxito.

Documentación – Código

```
public function agregar(Request $request)
{
    $request->validate([
        'producto_id' => 'required|exists:productos,id',
        'user_id' => 'required|exists:users,id'
    ]);

    $existingItem = WishList::where('user_id', $request->user_id)
        ->where('producto_id', $request->producto_id)
        ->first();

    if ($existingItem) {
        return response()->json(['message' => 'El producto ya está en la wishlist.'], 409);
    }

    try {
        $wishlist = new WishList();
        $wishlist->nombre = 'Mi Wishlist';
        $wishlist->user_id = $request->user_id;
        $wishlist->producto_id = $request->producto_id;
        $wishlist->save();

        return response()->json(['message' => 'Producto agregado a la wishlist con éxito.']);
    } catch (\Exception $e) {
        return response()->json(['message' => 'No se pudo agregar el producto a la wishlist.', 'error' => $e->getMessage()], 500);
    }
}
```

Función eliminar(\$userId, \$productoId)

Este método maneja la eliminación de un producto específico de la lista de deseos de un usuario. Primero, busca el elemento de la lista de deseos correspondiente al usuario y al producto proporcionados. Si se encuentra, elimina el registro y devuelve un mensaje de éxito. Si no se encuentra, devuelve

un mensaje indicando que el producto no está en la lista de deseos. En caso de cualquier error interno durante el proceso, devuelve un mensaje de error junto con el código de estado HTTP 500.

4.16 WishList.jsx

Función useEffect()

Se utiliza para cargar los elementos de la lista de deseos del usuario actual cada vez que hay un cambio en la identificación del usuario autenticado.

Función fetchWishlistItems()

Realiza una solicitud HTTP para recuperar los elementos de la lista de deseos del servidor y actualiza el estado local con los datos recibidos.

Función removeFromWishlist()

Permite eliminar un elemento específico de la lista de deseos del usuario. Realiza una solicitud HTTP de eliminación al servidor y vuelve a cargar los elementos de la lista de deseos después de una eliminación exitosa.

Función agregarAlCarrito()

Maneja la funcionalidad para agregar un producto deseado al carrito de compras. Realiza una solicitud HTTP POST al servidor con los detalles del usuario y el producto deseado, como el ID del usuario, el ID del producto y la cantidad, y maneja cualquier error que pueda surgir durante el proceso de agregar al carrito.

```

function Wishlist() {
  const auth = AuthUser();
  const [items, setItems] = useState([]);

  useEffect(() => {
    fetchWishlistItems();
    window.scrollTo(0, 0);
  }, [auth.user.id]);

  const fetchWishlistItems = () => [
    auth.http.get(`http://localhost:8000/api/wishlist/${auth.user.id}`)
      .then(response => {
        setItems(response.data);
      })
      .catch(error => {
        console.error('Error al cargar la wishlist:', error);
      });
  ];

  const removeFromWishlist = (productId) => {
    auth.http.delete(`http://localhost:8000/api/wishlist/${auth.user.id}/eliminar/${productId}`)
      .then(() => {
        fetchWishlistItems();
      })
      .catch(error => {
        console.error('Error al eliminar item de la wishlist:', error);
      });
  };

  const agregarAlCarrito = async (productId) => {
    try {
      await auth.http.post('http://localhost:8000/api/carrito/agregar', {
        user_id: auth.user.id,
        producto_id: productId,
        cantidad: 1,
        talla_id: 1,
      });
      console.log('Producto agregado al carrito correctamente');
    } catch (error) {
      console.error('Error al agregar producto al carrito:', error);
    }
  };
}

```

4.17 VerTarea.jsx

Estado Local

Utiliza el hook `useState()` para definir múltiples estados locales que gestionan los datos relacionados con las tareas, como `tasks` (`tareas`), `administrators` (`administradores`), `editingTask` (`tarea en edición`), etc.

Función `useEffect()`

Esta función se encarga de realizar la carga inicial de las tareas y los administradores. Utiliza solicitudes HTTP al servidor para obtener estos datos.

Función `deleteTask()`

La función `deleteTask()` se encarga de eliminar una tarea específica. Antes de eliminarla, verifica si el usuario actual es un director. Si lo es, realiza una solicitud HTTP `DELETE` al servidor para eliminar la tarea seleccionada. Después de la eliminación, actualiza el estado local de las tareas eliminando la tarea

eliminada de la lista.

Función handleEditClick()

Cuando el usuario hace clic en el botón de edición de una tarea, la función handleEditClick() se activa. Esta función establece el estado editingTask con la tarea seleccionada y actualiza el estado taskInput con los detalles de la tarea seleccionada, lo que permite que el formulario de edición se llene con la información de la tarea.

Función handleUpdateTask()

La función handleUpdateTask() se encarga de manejar la actualización de una tarea. Primero, verifica si el usuario actual es un director. Si es así, realiza una solicitud HTTP PUT al servidor con los detalles actualizados de la tarea. Despues de la actualización, actualiza las tareas locales para reflejar los cambios realizados.

Función filteredTasks

Esta función filtra las tareas según el término de búsqueda proporcionado por el usuario y también filtra las tareas según su estado (pendiente o completado). Esto permite al usuario encontrar fácilmente las tareas que está buscando.

Paginación

Divide las tareas en páginas para facilitar la navegación. Calcula el número total de páginas necesarias y maneja la paginación de las tareas pendientes y completadas por separado. Esto mejora la experiencia del usuario al visualizar grandes conjuntos de tareas.

4.18 ProductosAuth.jsx y Productos.jsx

Estado Local

- productos: Utiliza el hook usePedirDatos() para obtener datos de productos.
- seasonImages: Almacena las imágenes correspondientes a la temporada actual.
- imagesLoaded: Controla si todas las imágenes se han cargado correctamente.
- imageLoadCount: Lleva el recuento de las imágenes que se han cargado.

Función useEffect()

La primera useEffect() realiza la carga inicial de datos y se asegura de que la página se desplace al principio cuando se monta el componente. La segunda useEffect() se activa cuando el recuento de imágenes cargadas es igual al número total de imágenes multiplicado por 2. Esto establece imagesLoaded en true.

Función handleImageLoad()

Incrementa el contador de carga de imágenes cada vez que una imagen se carga correctamente.

Función getSeasonImages()

Determina la temporada actual según el mes actual y devuelve un conjunto específico de imágenes correspondientes a esa temporada.

Si `imagesLoaded` es `false`, muestra un mensaje de carga.

4.18 Perfil.jsx

Estado Local

- `originalUserDetail`: Almacena los detalles originales del usuario, como nombre, email, fecha de nacimiento y teléfono.
- `editedUserDetail`: Almacena los detalles del usuario que se están editando.
- `tab`: Controla la pestaña activa en la interfaz de usuario, como '`'datosPersonales'` o '`'wishlist'`'.
- `successMessage`: Almacena mensajes de éxito después de enviar el formulario de actualización.
- `errorMessages`: Almacena mensajes de error en caso de que falle la solicitud de actualización.

Función useEffect()

Cuando el componente se monta, esta función se encarga de realizar la carga inicial de los detalles del usuario.

Función logoutUser()

Maneja el cierre de sesión del usuario si hay un token definido.

Función fetchUserDetail()

Realiza una solicitud HTTP para obtener los detalles del usuario. Posteriormente, actualiza tanto `originalUserDetail` como `editedUserDetail` con los datos recuperados del servidor.

Función handleTabChange()

Cambia la pestaña activa en la interfaz de usuario y navega a la página correspondiente si la pestaña es '`'wishlist'`'.

Función handleInputChange()

Actualiza el estado de `editedUserDetail` cuando hay cambios en los campos del formulario.

Función handleFormSubmit()

Maneja el envío del formulario de actualización de detalles del usuario. Realiza una solicitud HTTP PUT para actualizar los detalles del usuario en el servidor. Si la solicitud es exitosa, actualiza `originalUserDetail`, muestra un mensaje de éxito y limpia los mensajes de error después de 3 segundos. Si la solicitud falla, maneja los errores y muestra los mensajes de error correspondientes.

Documentación – Código

```
const Perfil = () => {
  const { token, http, logout } = AuthUser();
  const navigate = useNavigate();
  const [originalUserDetail, setOriginalUserDetail] = useState({ name: '', email: '' , fecha_nacimiento: '', telefono: ''});
  const [editedUserDetail, setEditedUserDetail] = useState({ name: '', email: '' , fecha_nacimiento: '', telefono: ''});
  const [tab, setTab] = useState('datosPersonales');
  const [successMessage, setSuccessMessage] = useState('');
  const [errorMessages, setErrorMessages] = useState({});

  const logoutUser = () => {
    if(token != undefined){
      logout();
    }
  }

  useEffect(() => {
    fetchUserDetail();
  }, [ ]);

  const fetchUserDetail = async () => {
    try {
      const res = await http.post('/me');
      setOriginalUserDetail(res.data);
      setEditedUserDetail(res.data);
    } catch (error) {
      console.error('Failed to fetch user details', error);
    }
  };

  const handleTabChange = (newTab) => {
    setTab(newTab);
    if (newTab === 'wishlist') {
      navigate('/wishlist');
    }
  };

  const handleInputChange = (e) => {
    const { name, value } = e.target;
    setEditedUserDetail(prev => ({ ...prev, [name]: value }));
  };

  const handleFormSubmit = async (e) => {
    e.preventDefault();
    try {
      const res = await http.put('/user/actualizar', editedUserDetail);
      setOriginalUserDetail(editedUserDetail);
      setSuccessMessage('¡Cambios guardados exitosamente!');
      setTimeout(() => setSuccessMessage(''), 3000); // Limpiar el mensaje después de 3 segundos
    } catch (error) {
      console.error('Failed to update user details', error);
      if (error.response && error.response.data && error.response.data.errors) {
        setErrorMessages(error.response.data.errors);
      }
    }
  };
};
```

4.19 Facturas.jsx

Estado Local:

- facturas: Almacena la lista de facturas recuperadas del servidor.
- currentPage: Almacena el número de página actual para la paginación de las facturas.

Función useEffect():

Cuando el componente se monta, esta función se encarga de realizar la carga inicial de las facturas.

Función fetchFacturas():

Realiza una solicitud HTTP para obtener las facturas del usuario actual. Luego, convierte los detalles de cada factura del formato JSON a un objeto de JavaScript utilizando `JSON.parse()`. Posteriormente, actualiza el estado de facturas con las facturas recuperadas del servidor.

Paginación:

Calcula el índice del primer y último elemento de las facturas que se deben mostrar en la página actual, basado en el número de facturas por página. Luego, obtiene las facturas correspondientes a la página actual utilizando los índices calculados anteriormente. Finalmente, actualiza el número de página actual cuando el usuario navega a una página diferente.

```
const Facturas = () => {
  const [facturas, setFacturas] = useState([]);
  const auth = AuthUser();
  const userId = auth.user.id;
  const [currentPage, setCurrentPage] = useState(1);
  const [facturasPerPage] = useState(4);

  useEffect(() => {
    fetchFacturas();
  }, []);

  const fetchFacturas = async () => {
    try {
      const response = await auth.http.get(`http://localhost:8000/api/facturas/${userId}`);
      const parsedFacturas = response.data.map(factura => ({
        ...factura,
        detalle: JSON.parse(factura.detalle)
      }));
      setFacturas(parsedFacturas);
    } catch (error) {
      console.error('Error fetching invoices:', error);
    }
  };

  const indexOfLastFactura = currentPage * facturasPerPage;
  const indexOfFirstFactura = indexOfLastFactura - facturasPerPage;
  const currentFacturas = facturas.slice(indexOfFirstFactura, indexOfLastFactura);
  const paginate = pageNumber => setCurrentPage(pageNumber);
}
```

4.20 EditarProductos.jsx

Estado Local:

- formData: Almacena los datos del formulario de edición de productos.
- categorias: Almacena la lista de categorías recuperadas del servidor.
- mensajeExito: Controla la visualización del mensaje de éxito.
- mensajeError: Controla la visualización del mensaje de error.

Función useEffect():

Cuando el componente se monta, esta función se encarga de realizar la carga inicial del producto y de las categorías.

Funciones fetchProduct() y fetchCategorias():

Estas funciones están encargadas de realizar solicitudes HTTP para obtener los detalles del producto y las categorías respectivamente. Luego, actualizan los estados locales formData y categorias con los datos recuperados del servidor.

Función handleInputChange():

Esta función maneja los cambios en los campos del formulario. Cada vez que un campo se modifica, se actualiza el estado formData con los nuevos valores del campo modificado.

Función handleSubmit():

Encargada de manejar el envío del formulario de edición de productos. Crea un objeto FormData para enviar los datos del formulario, incluyendo los archivos de imagen. Luego, realiza una solicitud HTTP POST al servidor para actualizar el producto con los nuevos datos. Finalmente, muestra un mensaje de éxito si la solicitud se completa correctamente, o un mensaje de error si ocurre algún problema.

4.21 EditarAdmin.jsx

Estado Local:

- formInputs: Almacena los datos del formulario de edición del administrador.
- errors: Almacena los mensajes de error devueltos por el servidor al intentar actualizar el administrador.
- message: Almacena el mensaje de éxito cuando se actualiza el administrador correctamente.
- generoOptions: Almacena las opciones disponibles para el campo de género del administrador.

Función useEffect():

Esta función se utiliza para inicializar las opciones de género disponibles. Se ejecuta solo una vez, cuando el componente se monta.

Función fetchAdmin():

Esta función realiza una solicitud HTTP para obtener los detalles del administrador con el ID proporcionado. Luego, actualiza el estado formInputs con los datos recuperados del servidor.

Función datosNuevos():

Esta función maneja los cambios en los campos del formulario. Cada vez que un campo se modifica, actualiza el estado formInputs con los nuevos valores.

Función actualizarAdmin():

Esta función se encarga de manejar el envío del formulario de edición del administrador. Primero, realiza una solicitud HTTP PUT al servidor para actualizar los detalles del administrador con los datos del estado formInputs. Si la solicitud se completa correctamente, muestra un mensaje de éxito y redirige al usuario a la página de visualización. En caso de error, maneja los errores devueltos por el servidor y los muestra al usuario.

```

const EditarAdmin = () => [
  const { id } = useParams();
  const navigate = useNavigate();
  const auth = AuthUser();
  const [formInputs, setFormInputs] = useState({ name: '', email: '', fecha_nacimiento: '', telefono: '', password: '' });
  const [errors, setErrors] = useState({});
  const [message, setMessage] = useState('');
  const [generoOptions, setGeneroOptions] = useState([]);

  useEffect(() => {
    setGeneroOptions(['Selecciona tu género', 'Hombre', 'Mujer']);
  }, []);

  useEffect(() => {
    const fetchAdmin = async () => {
      try {
        const { data } = await auth.http.get(`http://localhost:8000/api/admin/users/${id}`);
        setFormInputs(data);
      } catch (error) {
        console.error('Error fetching admin data:', error);
      }
    };
    fetchAdmin();
  }, [id]);

  const datosNuevos = (e) => {
    const { name, value } = e.target;
    setFormInputs(inputs => ({ ...inputs, [name]: value }));
  };

  const actualizarAdmin = async (e) => {
    e.preventDefault();
    setErrors({});
    setMessage('');
    try {
      await auth.http.put(`http://localhost:8000/api/admin/users/${id}`, formInputs);
      setMessage('Admin actualizado correctamente!');
      navigate('/ver');
    } catch (error) {
      if (error.response && error.response.status === 422) {
        setErrors(error.response.data.errors);
      } else {
        console.error('Error updating admin:', error);
      }
    }
  };
];

```

4.22 DetallesGuest.jsx y Detalles.jsx

Estado Local:

- product: Almacena los detalles del producto recuperados del servidor.
- sizes: Almacena las tallas disponibles para el producto, incluido el stock disponible para cada talla.
- selectedSize: Almacena la talla seleccionada por el usuario para el producto.
- productosRecomendados: Almacena una lista de productos recomendados relacionados con el producto actual.
- loading: Controla la visualización de un indicador de carga mientras se recuperan los datos del servidor.
- error: Almacena un mensaje de error en caso de que ocurra un error al cargar los detalles del producto.
- currentSlide: Almacena el índice del slider de imágenes actual.
- hideMiniatures: Controla la visibilidad de las miniaturas de las imágenes si hay demasiadas para mostrar en el contenedor.
- addToCartSuccess: Controla la visualización del mensaje de éxito después de agregar un producto al carrito.
- showMessage: Controla la visualización del mensaje en la interfaz de

usuario.

- wishlistError: Almacena un mensaje de error si ocurre un error al intentar agregar un producto a la lista de deseos.
- sizeError: Almacena un mensaje de error si ocurre un error relacionado con la selección de la talla.
- authError: Almacena un mensaje de error si un usuario no autenticado intenta realizar una acción que requiere inicio de sesión.

Función useEffect():

La primera llamada a useEffect() se encarga de actualizar el estado showMessage a false cuando se cambia la talla seleccionada.

La segunda llamada a useEffect() se utiliza para realizar la carga inicial de los detalles del producto y los productos recomendados cuando el componente se monta.

Funciones fetchProductDetails() y fetchProductosRecomendados():

Estas funciones realizan solicitudes HTTP para obtener los detalles del producto y los productos recomendados respectivamente. Actualizan los estados locales product, sizes y productosRecomendados con los datos recuperados del servidor.

Funciones addToCart() y addToWishlist():

Estas funciones manejan la adición del producto al carrito y a la lista de deseos respectivamente. Verifican si el usuario está autenticado y si es un administrador antes de realizar las operaciones correspondientes. Luego, realizan solicitudes HTTP para agregar el producto al carrito o a la lista de deseos.

Función handleClose():

Esta función se utiliza para cerrar el mensaje de éxito o de error mostrado en la interfaz de usuario.

4.23 CrearTarea.jsx

Estado Local:

- taskInput: Almacena los datos del formulario para crear una nueva tarea, incluyendo el título, descripción, estado y persona asignada.
- administrators: Almacena la lista de administradores recuperada del servidor.
- mensajeExito: Controla la visualización del mensaje de éxito después de agregar una nueva tarea.
- mensajeError: Almacena un mensaje de error en caso de que ocurra un error al agregar una tarea.

Función useEffect():

Esta función se utiliza para realizar la carga inicial de la lista de administradores cuando el componente se monta.

Función fetchAdministrators():

Esta función realiza una solicitud HTTP para obtener la lista de administradores del servidor. Luego, actualiza el estado administrators con los datos recuperados.

Función handleChange():

Esta función maneja los cambios en los campos del formulario. Cada vez que un campo se modifica, actualiza el estado taskInput con los nuevos valores del campo modificado.

Función handleSubmit():

Encargada de manejar el envío del formulario para crear una nueva tarea. Primero verifica si el usuario actual es un director utilizando la función isDirector() proporcionada por AuthUser(). Si no es un director, muestra un mensaje de error. En caso contrario, realiza una solicitud HTTP POST al servidor con los datos del formulario. Si la solicitud se completa correctamente, muestra un mensaje de éxito y restablece el estado taskInput para preparar el formulario para una nueva tarea. Si ocurre algún error durante el proceso, muestra un mensaje de error y registra el error en la consola.

4.24 ConfirmacionPago.jsx

Función useEffect():

Esta función se utiliza para redirigir al usuario a la página de inicio ('/') cuando el componente se monta o cuando la ubicación cambia. Utiliza la función navigate() proporcionada por useNavigate() para realizar la redirección. El efecto se activa cuando el componente se monta por primera vez y cuando la ubicación del componente cambia, es decir, cuando se produce una navegación dentro de la aplicación.

4.25 Categoria.jsx

Estado Local:

- categorias: Almacena la lista de categorías recuperadas del servidor.
- productos: Almacena la lista de productos correspondientes a la categoría activa.
- categoriaActiva: Almacena el ID de la categoría seleccionada por el usuario.
- wishlisted: Almacena un objeto que indica si un producto está en la lista de deseos del usuario.

Funciones useEffect():

fetchCategorias(): Esta función se utiliza para realizar una solicitud HTTP para obtener la lista de categorías desde el servidor cuando el componente se monta por primera vez. Luego, actualiza el estado categorias con los datos recuperados. Además, si la lista de categorías no está vacía, establece la primera categoría como la categoría activa.

fetchProductosPorCategoria(categoríaId): Esta función se ejecuta cuando categoriaActiva cambia. Realiza una solicitud HTTP para obtener los productos asociados a la categoría seleccionada. Si el usuario está autenticado, también verifica si los productos están en la lista de deseos del usuario y actualiza el estado productos con los datos recuperados.

Función agregarAWishlist(productId):

Esta función se utiliza para agregar un producto a la lista de deseos del

Documentación – Código

usuario. Realiza una verificación para asegurarse de que el producto no esté ya en la lista de deseos antes de enviar la solicitud HTTP POST al servidor para agregar el producto a la lista de deseos del usuario. Luego, actualiza el estado productos para reflejar el cambio en el estado de la lista de deseos del producto agregado.

```
function Categoria() {
  const [categorias, setCategorias] = useState([]);
  const [productos, setProductos] = useState([]);
  const [categoriaActiva, setCategoriaActiva] = useState('');
  const auth = AuthUser();
  const [wishlist, setWishlist] = useState({});

  useEffect(() => {
    fetchCategorias();
  }, []);

  useEffect(() => {
    if (categoriaActiva) {
      fetchProductosPorCategoria(categoriaActiva);
    }
  }, [categoriaActiva]);

  const fetchCategorias = async () => {
    try {
      const response = await axios.get('http://localhost:8000/api/categorias');
      setCategorias(response.data);
      if (response.data.length > 0) {
        setCategoriaActiva(response.data[0].id);
      }
    } catch (error) {
      console.error('Error al obtener categorías:', error);
    }
  };
  const fetchProductosPorCategoria = async (categoriaId) => {
    try {
      const response = await axios.get(`http://localhost:8000/api/productos/categoría/${categoriaId}`);
      let productos = response.data;

      if (auth.user && auth.user.id) {
        try {
          const wishlistResponse = await auth.http.get(`http://localhost:8000/api/wishlist/${auth.user.id}`);
          const wishlistIds = new Set(wishlistResponse.data.map(item => item.producto_id));

          productos = productos.map(producto => {
            ...producto,
            enWishlist: wishlistIds.has(producto.id)
          });
        } catch (wishlistError) {
          console.error('Error fetching wishlist:', wishlistError);
        }
      }

      setProductos(productos);
    } catch (error) {
      console.error('Error al obtener productos:', error);
    }
  };
}
```

4.26 Carrito.jsx

Estado Local:

- `productosCarrito`: Almacena los productos actualmente en el carrito de compras.
- `sizeError`: Almacena un mensaje de error relacionado con problemas de tamaño de stock.
- `paymentError`: Almacena un mensaje de error relacionado con problemas de pago.
- `paymentCompleted`: Controla si el pago se ha completado con éxito.
- `hayProductos`: Controla si hay productos en el carrito.
- `actualizandoCantidad`: Controla si se está actualizando la cantidad de un producto en el carrito.

Funciones useEffect():

La primera `useEffect` realiza la carga inicial de los productos en el carrito y establece un intervalo para actualizar periódicamente los productos cada 5 minutos.

La segunda `useEffect` carga el script de PayPal cuando hay productos en el carrito.

Función calcularSubtotal():

Calcula el subtotal de los productos en el carrito sumando el precio de cada producto multiplicado por la cantidad de dicho producto.

Función calcularIVA():

Calcula el impuesto al valor agregado (IVA) del subtotal proporcionado.

Función calcularTotal():

Calcula el total a pagar sumando el subtotal y el IVA.

Función loadPaypalScript():

Carga el script de PayPal si aún no está cargado y luego crea el botón de PayPal utilizando la API de PayPal.

Función handlePostPayment():

Maneja la lógica posterior al pago con PayPal, incluida la verificación de stock, la creación de la factura, la actualización del stock, la limpieza del carrito y la navegación a la página de confirmación de pago.

Función actualizarStock():

Actualiza el stock de los productos después de que se ha completado un pago exitoso.

Función obtenerProductosCarrito():

Obtiene los productos en el carrito del servidor, incluidas las tallas disponibles para cada producto.

Función cambiarTalla():

Cambia la talla de un producto en el carrito y actualiza la lista de productos en el carrito.

Función actualizarCantidad():

Actualiza la cantidad de un producto en el carrito y maneja la lógica relacionada con el stock disponible.

Función limpiarCarrito():

Elimina todos los productos del carrito después de que se ha completado un pago exitoso.

Función eliminarProducto():

Elimina un producto específico del carrito.

```
const createPaypalButton = () => {
    const subtotal = calcularSubtotal();
    const iva = calcularIVA(subtotal);

    const total = (subtotal + iva).toFixed(2);
    const paypalButtonContainer = document.getElementById('paypal-button-container');
    paypalButtonContainer.innerHTML = '';

    console.log('Subtotal:', subtotal);
    console.log('IVA:', iva);
    console.log('Total:', total);

    window.paypal.Buttons({
        style: {
            color: 'black',
            label: 'pay',
        },
        createOrder: (data, actions) => {
            return actions.order.create({
                purchase_units: [
                    {
                        amount: { value: total }
                    }
                ]
            });
        },
        onApprove: async (data, actions) => {
            const order = await actions.order.capture();
            console.log(order);
            handlePostPayment();
        },
        onError: (err) => {
            console.error('Error en el pago con PayPal:', err);
            setPaymentError('Error en el pago con PayPal. Por favor, inténtalo de nuevo.');
        }
    }).render('#paypal-button-container');
};
```

Documentación – Código

```
const loadPaypalScript = () => {
  if (window.paypal) {
    console.log("PayPal SDK already loaded.");
    createPaypalButton();
    return;
  }

  const script = document.createElement("script");
  script.src = 'https://www.paypal.com/sdk/js?client-id=${process.env.REACT_APP_PAYPAL_CLIENT_ID}&currency=EUR';
  script.onload = () => {
    console.log("PayPal SDK loaded successfully.");
    createPaypalButton();
  };
  document.body.appendChild(script);
};

const createPaypalButton = () => {
  const subtotal = calcularSubtotal();
  const iva = calcularIVA(subtotal);

  const total = (subtotal + iva).toFixed(2);
  const paypalButtonContainer = document.getElementById('paypal-button-container');
  paypalButtonContainer.innerHTML = '';

  console.log('Subtotal:', subtotal);
  console.log('IVA:', iva);
  console.log('Total:', total);

  window.paypal.Buttons({
    style:{
      color:'black',
      label:'pay',
    },
    createOrder: (data, actions) => {
      return actions.order.create([
        purchase_units: [
          {
            amount: { value: total }
          }
        ]
      });
    },
    onApprove: async (data, actions) => {
      const order = await actions.order.capture();
      console.log(order);
      handlePostPayment();
    },
    onError: (err) => {
      console.error('Error en el pago con PayPal:', err);
      setPaymentError('Error en el pago con PayPal. Por favor, inténtalo de nuevo.');
    }
  }).render('#paypal-button-container');
};
```

4.27 AñadirForm.jsx

Estado Local:

- formInputs: Almacena los datos del formulario para añadir un nuevo administrador, incluyendo el nombre, correo electrónico, contraseña, fecha de nacimiento, teléfono y género.
- generoOptions: Almacena las opciones de género para el campo de selección del formulario.
- mensajeExito: Controla la visualización del mensaje de éxito después de añadir un nuevo administrador.

Función useEffect():

Esta función se utiliza para inicializar las opciones de género cuando el componente se monta.

Función datosNuevos():

Esta función maneja los cambios en los campos del formulario. Cada vez que un campo se modifica, actualiza el estado formInputs con los nuevos valores del campo modificado.

Función nuevoAdmin():

Encargada de manejar el envío del formulario para añadir un nuevo administrador. Realiza una solicitud HTTP POST al servidor con los datos del formulario. Si la solicitud se completa correctamente, muestra un mensaje de éxito y restablece el estado formInputs para preparar el formulario para un nuevo administrador. Si ocurre algún error durante el proceso, registra el error en la consola.

4.28 AdminStock.jsx

Estado Local:

- stocks: Almacena la lista de stocks recuperada del servidor.
- currentPage: Almacena el número de página actual para la paginación de stocks.
- incrementAmount: Almacena la cantidad de incremento para cada stock.
- decrementAmount: Almacena la cantidad de decremento para cada stock.
- visiblePageCount: Almacena la cantidad visible de páginas en la paginación.
- searchTerm: Almacena el término de búsqueda para filtrar los stocks.
- searchBy: Almacena el criterio de búsqueda seleccionado ('producto', 'talla' o 'cantidad').
- errorStock: Controla la visualización de un mensaje de error al obtener el stock.
- errorIncrem: Controla la visualización de un mensaje de error al incrementar el stock.
- stockActu: Controla la visualización de un mensaje de éxito después de actualizar el stock.
- stockNo: Controla la visualización de un mensaje de error al no encontrar el stock.
- errorDecrem: Controla la visualización de un mensaje de error al decrementar el stock.

Función useEffect():

Realiza la carga inicial de los stocks cuando el componente se monta o cuando el término de búsqueda o el criterio de búsqueda cambian.

Función fetchStock():

Realiza una solicitud HTTP para obtener los stocks del servidor. Luego, filtra los stocks según el término de búsqueda y el criterio de búsqueda seleccionado y actualiza el estado stocks con los datos recuperados.

Función incrementStock():

Maneja el incremento de stock para un stock específico. Realiza una solicitud HTTP POST al servidor con la cantidad de incremento y actualiza los stocks después de la operación.

Función handleStockAmountChange():

Actualiza el estado incrementAmount y decrementAmount con la cantidad de incremento y decremento para un stock específico.

Función decrementStock():

Maneja el decremento de stock para un stock específico. Realiza una solicitud HTTP POST al servidor con la cantidad de decremento y actualiza los stocks

después de la operación.

Cálculos de paginación:

Calcula el rango de páginas visibles en la paginación de acuerdo con el número total de stocks y el número de stocks por página.

4.29 AdminPanel.jsx

Estado Local:

- admins: Almacena la lista de administradores recuperada del servidor.
- searchTerm: Almacena el término de búsqueda introducido por el usuario para filtrar la lista de administradores.
- itemsPerPage: Almacena el número de elementos por página para la paginación.
- showPageNumbers: Almacena el número máximo de números de página que se mostrarán en la paginación.
- currentPage: Almacena el número de página actual que se está mostrando.
- showDeleteConfirmation: Controla la visualización del cuadro de confirmación de eliminación de administrador.
- adminToDelete: Almacena el administrador que se va a eliminar.

Función useEffect():

Se utiliza para realizar la carga inicial de la lista de administradores cuando el componente se monta.

Función fetchAdmins():

Realiza una solicitud HTTP para obtener la lista de administradores del servidor y actualiza el estado admins con los datos recuperados.

Función eliminarAdmin(id):

Maneja la eliminación de un administrador específico. Realiza una solicitud HTTP DELETE al servidor para eliminar el administrador con el ID proporcionado y luego vuelve a cargar la lista de administradores.

Funciones handleDeleteConfirmation(), handleConfirmDelete() y handleCancelDelete():

Estas funciones manejan el proceso de confirmación y cancelación de la eliminación de un administrador. handleDeleteConfirmation() muestra el cuadro de confirmación y establece el estado adminToDelete con el administrador seleccionado para eliminar. handleConfirmDelete() confirma la eliminación del administrador seleccionado y actualiza la lista de administradores. handleCancelDelete() cancela la eliminación y oculta el cuadro de confirmación.

Funciones de paginación:

Estas funciones calculan los índices de los elementos que se deben mostrar en la página actual y manejan la navegación entre páginas. paginate() establece el número de página actual. renderPageNumbers() genera los números de página y los muestra en la interfaz de usuario, permitiendo al usuario navegar entre las diferentes páginas.

4.30 HomeAdmin.jsx

Estado Local:

- stockData: Almacena los datos de inventario recuperados del servidor.
- loading: Controla el estado de carga de los datos del inventario.
- userCount: Almacena el número total de usuarios registrados.
- showRegister: Controla la visualización del formulario de registro.
- totalSales: Almacena el total de ventas.
- ordersToday: Almacena el número de pedidos realizados hoy.
- activeProductsCount: Almacena el número total de productos activos.
- facturas: Almacena la lista de facturas recuperadas del servidor.
- searchTerm: Almacena el término de búsqueda introducido por el usuario.

Función useEffect():

Se utiliza para realizar la carga inicial de datos relacionados con el inventario, el recuento de usuarios, el total de ventas, los pedidos de hoy, el recuento de productos activos y las facturas.

Funciones fetchStockData(), fetchUserCount(), fetchTotalSales(), fetchOrdersToday(), fetchActiveProductsCount() y fetchFacturas():

Estas funciones realizan solicitudes HTTP para obtener los datos necesarios del servidor. Luego, actualizan los estados locales correspondientes con los datos recuperados.

Función handleEstadoEnvioChange():

Esta función se encarga de manejar el cambio de estado de envío de una factura específica. Realiza una solicitud HTTP PUT al servidor para actualizar el estado de envío de la factura seleccionada y actualiza el estado local de las facturas después de la actualización.

Filtrado de Facturas:

Filtre las facturas en función del término de búsqueda introducido por el usuario.

Gestión de Visualización:

Si los datos todavía se están cargando, muestra un mensaje indicando que se están cargando los datos del inventario. Una vez que se completa la carga, calcula los productos principales, los productos con el stock más bajo y el producto con el mayor stock.

4.31 HomeDirector.jsx

Estado Local:

- totalSales: Almacena el total de ventas.
- ordersToday: Almacena el número de pedidos realizados hoy.
- activeProductsCount: Almacena el número de productos activos.
- userCount: Almacena el número total de usuarios.
- topCustomers: Almacena la lista de los clientes principales.
- searchTerm: Almacena el término de búsqueda utilizado para filtrar las facturas.
- facturas: Almacena la lista de facturas.

- loading: Controla el estado de carga.

Funciones fetchTotalSales(), fetchOrdersToday(), fetchActiveProductsCount(), fetchUserCount() y fetchTopCustomers():

Estas funciones realizan solicitudes HTTP al servidor para obtener diferentes datos estadísticos, como el total de ventas, el número de pedidos realizados hoy, el número de productos activos, el número total de usuarios y la lista de clientes principales. Luego, actualizan los estados locales correspondientes con los datos recuperados del servidor.

Función handleEstadoEnvioChange():

Esta función se utiliza para cambiar el estado de envío de una factura específica. Realiza una solicitud HTTP PUT al servidor para actualizar el estado de envío de la factura con el nuevo estado proporcionado. Luego, actualiza el estado local facturas con la factura actualizada.

Función fetchFacturas():

Esta función realiza una solicitud HTTP al servidor para obtener la lista de facturas. Luego, actualiza el estado local facturas con los datos recuperados del servidor y establece el estado de carga en false.

Función useEffect():

Esta función se utiliza para realizar la carga inicial de los diferentes datos estadísticos y la lista de facturas cuando el componente se monta. Se llama a todas las funciones de carga de datos dentro de este efecto de manera que se ejecuten solo una vez al inicio.

4.32 AsignarOfertas.jsx

Estado Local:

- productos: Almacena la lista de productos recuperados del servidor.
- ofertas: Almacena la lista de ofertas recuperadas del servidor.
- loading: Controla el estado de carga mientras se recuperan los datos del servidor.
- currentPage: Almacena el número de página actual para la paginación de productos.
- productsPerPage: Almacena el número de productos que se mostrarán por página.
- showPageNumbers: Almacena el número de números de página que se mostrarán en el paginador.
- searchTerm: Almacena el término de búsqueda introducido por el usuario.
- mensajeExito: Controla la visualización del mensaje de éxito después de asignar una oferta.
- mensajeError: Controla la visualización del mensaje de error en caso de que ocurra un error durante la asignación de una oferta.

Función useEffect():

Esta función se utiliza para realizar la carga inicial de la lista de productos y de ofertas cuando el componente se monta.

Funciones fetchProductos() y fetchOfertas():

Estas funciones están encargadas de realizar solicitudes HTTP para obtener la lista de productos y de ofertas respectivamente. Luego, actualizan los estados locales productos y ofertas con los datos recuperados del servidor.

Función handleOfferChange():

Esta función maneja el cambio de oferta para un producto específico. Realiza una solicitud HTTP POST al servidor para asignar una oferta a un producto. Si la asignación es exitosa, muestra un mensaje de éxito y actualiza la lista de productos. Si ocurre algún error durante el proceso, muestra un mensaje de error y registra el error en la consola.

Función paginate():

Esta función se encarga de manejar la paginación de la lista de productos. Recibe el número de página como argumento y actualiza el estado currentPage con ese valor, lo que cambia los productos mostrados en la página actual.

Cálculos de paginación:

Estos cálculos determinan qué números de página se deben mostrar en el paginador, dependiendo de la página actual y del número total de páginas. Esto permite al usuario navegar fácilmente entre las páginas de productos.

4.33 AsignarOfertasPorCategoria.jsx

Estado Local:

- categorias: Almacena la lista completa de categorías recuperada del servidor.
- categoriasFiltradas: Almacena las categorías filtradas según el término de búsqueda proporcionado por el usuario.

- ofertas: Almacena la lista de ofertas recuperada del servidor.
- loading: Controla la visualización del indicador de carga.
- currentPage: Almacena el número de la página actual de categorías.
- mensajeExito: Controla la visualización del mensaje de éxito.
- mensajeError: Controla la visualización del mensaje de error.
- searchTerm: Almacena el término de búsqueda proporcionado por el usuario.
- productsPerPage: Define la cantidad de categorías que se muestran por página.
- showPageNumbers: Define la cantidad de números de página que se muestran en la paginación.

Función useEffect():

Realiza la carga inicial de las categorías y las ofertas desde el servidor.
Funciones fetchCategorias() y fetchOfertas():

Realizan solicitudes HTTP para obtener la lista de categorías y ofertas respectivamente. Luego, actualizan los estados locales categorias y ofertas con los datos recuperados.

Función handleOfferChange():

Maneja la asignación de una oferta a una categoría específica. Realiza una solicitud HTTP POST al servidor con los detalles de la categoría y la oferta. Si la solicitud se completa correctamente, muestra un mensaje de éxito y vuelve a cargar las categorías. En caso de error, muestra un mensaje de error y registra el error en la consola.

Función handleRemoveOffer():

Maneja la eliminación de una oferta de una categoría específica. Realiza una solicitud HTTP POST al servidor con los detalles de la categoría. Si la solicitud se completa correctamente, muestra un mensaje de éxito y vuelve a cargar las categorías. En caso de error, muestra un mensaje de error y registra el error en la consola.

Función paginate():

Maneja la paginación de las categorías. Actualiza el estado currentPage con el número de página seleccionado por el usuario.

4.34 CrearOferta.jsx

Estado Local:

- formData: Almacena los datos del formulario para crear una nueva oferta, incluyendo el nombre, descuento, fecha de inicio, fecha de finalización, ID del producto asociado y el ID de la oferta.
- mensajeExito: Controla la visualización del mensaje de éxito después de crear una nueva oferta.
- mensajeError: Controla la visualización del mensaje de error en caso de que ocurra un error al crear la oferta.

Función handleChange():

Esta función se encarga de manejar los cambios en los campos del formulario. Cada vez que un campo se modifica, actualiza el estado formData con los nuevos valores del campo modificado.

Función crearOferta():

Encargada de manejar el proceso de creación de una nueva oferta. Realiza una solicitud HTTP POST al servidor con los datos del formulario para crear la oferta. Si la solicitud se completa correctamente, muestra un mensaje de éxito y actualiza el estado formData con el ID de la oferta creada. Si ocurre algún error durante el proceso, muestra un mensaje de error y registra el error en la consola.

4.35 CrearProducto.jsx

Estado Local:

- formData: Almacena los datos del formulario para crear un nuevo producto, incluyendo el nombre, descripción, precio, imágenes y categoría.
- categorias: Almacena la lista de categorías recuperada del servidor.
- mensajeExito: Controla la visualización del mensaje de éxito después de crear un nuevo producto.
- mensajeError: Controla la visualización del mensaje de error en caso de

que ocurra un error al crear un producto.

Función useEffect():

Realiza la carga inicial de la lista de categorías cuando el componente se monta.

Función fetchCategorias():

Realiza una solicitud HTTP para obtener la lista de categorías del servidor. Luego, actualiza el estado categorias con los datos recuperados.

Función handleInputChange():

Esta función maneja los cambios en los campos del formulario. Cada vez que un campo se modifica, actualiza el estado formData con los nuevos valores del campo modificado.

Función handleSubmit():

Encargada de manejar el envío del formulario para crear un nuevo producto. Primero, crea un objeto FormData para enviar los datos del formulario, incluyendo las imágenes. Luego, realiza una solicitud HTTP POST al servidor con los datos del formulario. Si la solicitud se completa correctamente, muestra un mensaje de éxito y restablece los estados de mensaje de éxito y mensaje de error. Si ocurre algún error durante el proceso, muestra un mensaje de error y registra el error en la consola.

4.36 EditarOfertas.jsx

Estado Local:

- formData: Almacena los datos del formulario de edición de la oferta.
- successMessage: Controla la visualización del mensaje de éxito después de actualizar la oferta.
- errorMessage: Almacena un mensaje de error en caso de que ocurra un error al actualizar la oferta.

Función handleChange():

Esta función maneja los cambios en los campos del formulario. Cada vez que un campo se modifica, actualiza el estado formData con los nuevos valores del campo modificado.

Funciones handleCloseSuccessMessage() y handleCloseErrorMessage():

Estas funciones se utilizan para cerrar los mensajes de éxito y de error respectivamente. Al llamarlas, se establece el estado correspondiente (successMessage o errorMessage) en una cadena vacía, lo que oculta el mensaje.

Función handleSubmit():

Encargada de manejar el envío del formulario para actualizar la oferta. Realiza una solicitud HTTP PUT al servidor con los datos del formulario. Si la solicitud se completa correctamente, muestra un mensaje de éxito y actualiza la lista de ofertas llamando a la función setOfertas(). Si ocurre algún error durante el proceso, muestra un mensaje de error y registra el error en la consola.

4.37 EliminarProducto.jsx

Estado Local:

- `productos`: Almacena la lista de productos recuperada del servidor.
- `currentPage`: Almacena el número de página actual para la paginación de productos.
- `mensajeExito`: Controla la visualización del mensaje de éxito después de eliminar un producto.
- `mensajeError`: Controla la visualización del mensaje de error en caso de que ocurra un error al eliminar un producto.
- `confirmarEliminar`: Almacena el ID del producto que se va a eliminar o null si no hay ningún producto seleccionado para eliminar.

Función useEffect():

Esta función se utiliza para realizar la carga inicial de la lista de productos cuando el componente se monta.

Función fetchProductos():

Realiza una solicitud HTTP para obtener la lista de productos del servidor. Luego, actualiza el estado `productos` con los datos recuperados.

Función deleteProduct():

Maneja la eliminación de un producto específico. Realiza una solicitud HTTP `DELETE` al servidor para eliminar el producto seleccionado. Si la solicitud se completa correctamente, actualiza el estado `productos` eliminando el producto eliminado de la lista. Muestra un mensaje de éxito y oculta el mensaje de error. En caso de que ocurra algún error durante el proceso, muestra un mensaje de error y oculta el mensaje de éxito. Finalmente, establece `confirmarEliminar` a null para deseleccionar el producto que se va a eliminar.

Función paginate():

Maneja la paginación de la lista de productos. Recibe el número de página como parámetro y actualiza el estado `currentPage` con ese valor. Además, la función realiza la lógica para calcular los números de página disponibles y los almacena en el array `pageNumbers`. Si el número total de páginas es menor o igual a 8, muestra todas las páginas. De lo contrario, muestra las páginas desde 1 hasta 8 y ajusta el rango para centrar la página actual en la paginación.

4.38 ListaDeOfertas.jsx

Estado Local:

- `ofertas`: Almacena la lista de ofertas recuperadas del servidor.
- `ofertaSeleccionada`: Almacena la oferta seleccionada por el usuario.
- `currentPage`: Almacena el número de página actual en la paginación.
- `itemsPerPage`: Almacena el número de ofertas que se mostrarán por página.
- `searchTerm`: Almacena el término de búsqueda proporcionado por el usuario.

Función useEffect():

Esta función se utiliza para realizar la carga inicial de las ofertas cuando el componente se monta.

Función fetchOfertas():

Realiza una solicitud HTTP para obtener la lista de ofertas del servidor. Luego, actualiza el estado ofertas con los datos recuperados.

Función handleSelectOferta():

Maneja la selección de una oferta. Si la oferta seleccionada ya está seleccionada, la función la deselecciona. De lo contrario, establece la oferta seleccionada en el estado.

Función paginate():

Maneja la paginación de las ofertas. Recibe el número de página como argumento y actualiza el estado currentPage con ese valor.

Cálculo de Páginas:

Calcula el número total de páginas necesarias para la paginación y determina los números de página a mostrar en el paginador. Si la cantidad de páginas a mostrar es mayor que el número total de páginas, ajusta el rango de inicio y fin de las páginas para que no se muestren páginas inexistentes.

4.39 ListaTarea.jsx

Estado Local:

- tasks: Almacena la lista de tareas recuperadas del servidor.
- loading: Controla la visualización del mensaje de carga mientras se obtienen las tareas del servidor.
- currentPage: Almacena el número de la página actual en la paginación.
- mensajeExito: Controla la visualización del mensaje de éxito después de completar una tarea.

Función useEffect():

Esta función se utiliza para realizar la carga inicial de las tareas cuando el componente se monta.

Función fetchTasks():

Esta función realiza una solicitud HTTP para obtener la lista de tareas del servidor. Luego, actualiza el estado tasks con los datos recuperados y cambia el estado loading a false para indicar que la carga ha finalizado.

Función completeTask():

Encargada de marcar una tarea como completada. Realiza una solicitud HTTP PUT al servidor para actualizar el estado de la tarea. Después de completar la solicitud, vuelve a obtener la lista de tareas actualizada y muestra un mensaje de éxito.

Cálculo de la Paginación:

Calcula el índice de la primera y última tarea visible en la página actual, así como el número total de páginas necesarias para mostrar todas las tareas.

Función paginate():

Esta función se utiliza para cambiar la página actual cuando el usuario hace clic en los enlaces de paginación.

Mensaje de Carga:

Si loading es true, muestra un mensaje indicando que las tareas se están cargando.

4.40 CategoriaChica/CategoríaChica/Hombre/Mujer/Adolescentes y duplicado para la vista guest

Estado Local:

- productosOriginales: Almacena la lista original de productos recuperada del servidor.
- productosMostrados: Almacena la lista de productos filtrada y mostrada en la interfaz de usuario.
- mostrarFiltros: Controla la visualización de los filtros en la interfaz de usuario.
- filtroOferta: Controla si se aplica el filtro de productos en oferta.
- filtroPrecio: Almacena el tipo de filtro de precio seleccionado ('menor', 'mayor' o vacío).
- filtroPrecioMin: Almacena el valor mínimo del rango de precios.
- filtroPrecioMax: Almacena el valor máximo del rango de precios.
- wishlist: Almacena la lista de productos guardados en la lista de deseos del usuario.
- vistaDoble: Controla si se muestra la vista de productos en formato de lista o en formato de cuadricula.
- busqueda: Almacena el término de búsqueda introducido por el usuario.

Función useEffect():

Realiza la carga inicial de los productos de la categoría 'PRENDA DE ROPA' para hombres desde el servidor y la lista de deseos del usuario. También aplica los filtros de búsqueda cuando cambian los valores de filtroOferta, filtroPrecio, filtroPrecioMin, filtroPrecioMax y busqueda.

Funciones fetchProductDetails() y fetchWishlist():

Realizan solicitudes HTTP para obtener los detalles de los productos y la lista de deseos respectivamente. Luego, actualizan los estados locales productosOriginales y wishlist con los datos recuperados del servidor.

Función aplicarFiltros():

Se encarga de aplicar los filtros seleccionados a la lista de productos originales. Los filtros incluyen la oferta, el precio mínimo y máximo, y el término de búsqueda.

Funciones addToWishlist() e isInWishlist():

Permiten agregar o eliminar productos de la lista de deseos del usuario. La función addToWishlist realiza las solicitudes HTTP correspondientes y actualiza la lista de deseos después de la operación. La función isInWishlist verifica si un producto específico ya está en la lista de deseos.

Función handleBusquedaChange():

Maneja los cambios en el campo de búsqueda. Cada vez que se modifica, actualiza el estado busqueda con el nuevo término de búsqueda.

4.41 APIRestGuest.jsx y APIRest.jsx

Estado Local:

- productos: Almacena la lista de productos recuperada del servidor.
- productoDetalles: Almacena los detalles de un producto específico.
- talla: Almacena la lista de tallas recuperada del servidor.
- cat: Almacena la lista de categorías recuperada del servidor.
- ofertas: Almacena la lista de ofertas recuperada del servidor.
- ofertas1: Almacena los detalles de una oferta específica.
- tallaId: Almacena los detalles de una talla específica identificada por su ID.
- tallaNombre: Almacena los detalles de una talla específica identificada por su nombre.
- ProductoNombre: Almacena los detalles de un producto específico identificado por su nombre.
- productoPrecio: Almacena los detalles de un producto específico identificado por su precio.

Función useEffect():

Esta función se utiliza para realizar la carga inicial de los productos, tallas, categorías y ofertas cuando el componente se monta.

Funciones fetchProductos, fetchProductoDetalle, fetchTalla, fetchCategoria, fetchOfertas, fetchOferta, fetchTallaNombre, fetchTallaId, fetchProductoNombre, fetchProductoPrecio:

Estas funciones están encargadas de realizar solicitudes HTTP para obtener los detalles de los productos, tallas, categorías y ofertas respectivamente. Luego, actualizan los estados locales correspondientes con los datos recuperados del servidor. Cada función maneja los posibles errores que puedan ocurrir durante la solicitud y actualiza el estado error en caso de error. Finalmente, establecen el estado loading en false para indicar que la carga de datos ha finalizado.

5. GITHUB (Control de versiones)

He subido el TFG a mi github. No he realizado una constante subida de versiones, debido a un error que me lo impedía, por ello, subo el proyecto entero, puedes acceder al siguiente enlace:

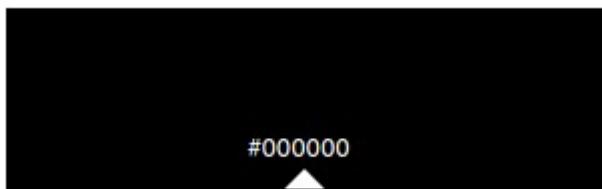
<https://github.com/crodros2601/Tienda-de-ropa>

The screenshot shows the GitHub repository page for 'Tienda-de-ropa'. At the top, there's a yellow banner indicating that the 'main' branch had recent pushes 55 minutes ago. Below the banner, the repository name 'Tienda-de-ropa' is shown with a 'Public' badge. To the right are buttons for 'Pin', 'Unwatch 1', 'Fork 0', and 'Star 0'. A green 'Compare & pull request' button is also present. The main content area shows a list of files in the 'master' branch, all updated 5 minutes ago. The files listed are: laravelTFG (version 1.0), reactjs (version 1.0), .gitattributes (version 1.0), README.md (version 1.0), package-lock.json (version 1.0), and package.json (version 1.0). On the right side, there are sections for 'About', 'Activity', 'Releases', and 'Packages'. The 'About' section contains a brief description: 'TFG realizado con laravel + react sobre una tienda de ropa.' Below it are links for 'Readme', 'Activity', 'Stars', 'Watching', and 'Forks'. The 'Releases' section indicates 'No releases published' and has a link to 'Create a new release'. The 'Packages' section indicates 'No packages published' and has a link to 'Publish your first package'. At the bottom of the page, there's a large image of a clothing store interior with a sign that says '100'.

6. MANUAL DE ESTILOS

Color y uso tipografía del color

Los colores principales de la web son:



Negro

#FFFFFF

Blanco

Los dos únicos colores utilizados en la web son el blanco y el negro. Esta elección intencional elimina el uso de colores adicionales para crear contraste y, en su lugar, permite que las imágenes añadan los tonos necesarios para destacar ciertos elementos. Esto asegura que los colores de la página se mantengan minimalistas y enfaticen la información más relevante de manera clara y directa. Las imágenes están cuidadosamente seleccionadas para proporcionar el contraste y la vitalidad visual necesaria, haciendo que los elementos importantes se noten sin necesidad de colores adicionales.

El objetivo de esta paleta de colores es crear un diseño limpio y minimalista que resalte la información de manera visualmente atractiva y sencilla. Al usar solo blanco y negro, se garantiza que los usuarios puedan enfocarse en el contenido sin distracciones innecesarias. Esto facilita una experiencia de usuario intuitiva, donde la navegación y la comprensión del contenido sean fluidas y sin esfuerzo.

La tipografía en la página web también sigue el color negro con un tamaño de 16px para las descripciones y de 30px para los títulos. Entre estos dos tamaño se va alternando en la página.

El uso de una única fuente de color para toda la tipografía asegura que el texto sea legible y coherente en toda la página. El tamaño del texto está diseñado para mantener una jerarquía visual clara, con descripciones detalladas en un tamaño de 16px y títulos prominentes en 30px. Esta estructura ayuda a los usuarios a identificar rápidamente la información más relevante y navegar por el contenido de manera efectiva.

En conjunto, la combinación de una paleta de colores minimalista y una tipografía clara y coherente crea una experiencia de usuario que es tanto visualmente atractiva como funcionalmente eficiente. El diseño asegura que los usuarios puedan interactuar con la página web de manera intuitiva, encontrando la información que necesitan sin esfuerzo.

Manual de estilos

Cabecera

Identidad visual

Tipos de cabeceras

En el diseño de la interfaz de usuario, se han implementado tres tipos distintos de cabeceras de navegación.

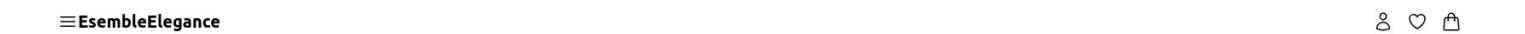
El primer tipo de cabecera destaca con un fondo blanco y texto en color blanco para un aspecto limpio y elegante. Esta elección de colores crea un contraste suave que permite que el nombre del sitio y los íconos de usuario, favoritos y carrito de compras se destaquen sobre el fondo. Ya que es una cabecera pensada únicamente para colocarlos sobre una imagen de fondo.



El segundo tipo de cabecera ofrece un enfoque más contrastante al utilizar un fondo blanco con texto en color negro. Esta combinación de colores garantiza una legibilidad óptima y un contraste visual fuerte, lo que facilita la navegación y la identificación de los elementos importantes.



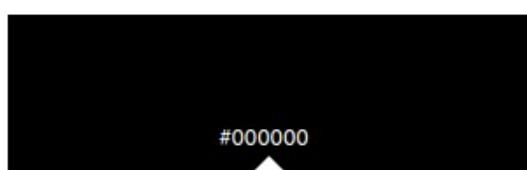
El tercer tipo de cabecera presenta una combinación única al tener un fondo blanco con texto negro para el nombre del sitio, pero con íconos de usuario, favoritos y carrito de compras en texto negro sobre un fondo blanco. Esta elección de diseño crea un punto focal interesante al destacar el nombre del sitio en contraste con los otros elementos.



Estos tres tipos de cabeceras ofrecen opciones variadas para adaptarse a diferentes preferencias de diseño mientras mantienen la funcionalidad y la estética general del sitio web.

Colores de la cabecera

Los 3 colores claves de la página webs son: blanco (#FFFFFF), negro (#000000) y los colores de las imágenes.



Manual de estilos

Fotos

Uso y proporción de imagen

En la selección de las imágenes de carácter del site han sido optimizadas en formato .webp.

Encontramos 7 tipos de imágenes:

Imagen de carrousel: 1787 x 977 px

Imagen del categorías: 893.5 x 977 px

Imágenes de la fila de categoría: 450.5 x 643 px

Imágenes de secciones 1: 1191.33 x 977 px

Imágenes de secciones 2: 595.67 x 977 px

Imágenes de productos: 412.75 x 641 px

Imágenes de carrousel detalle productos: 613 x 854.22 px

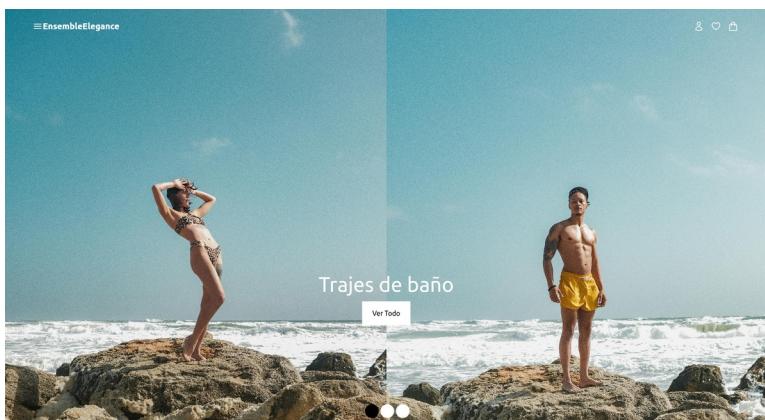


Imagen de carrousel

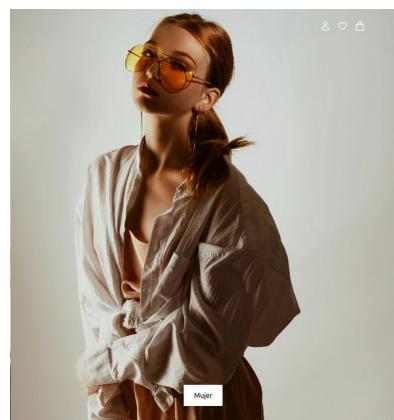
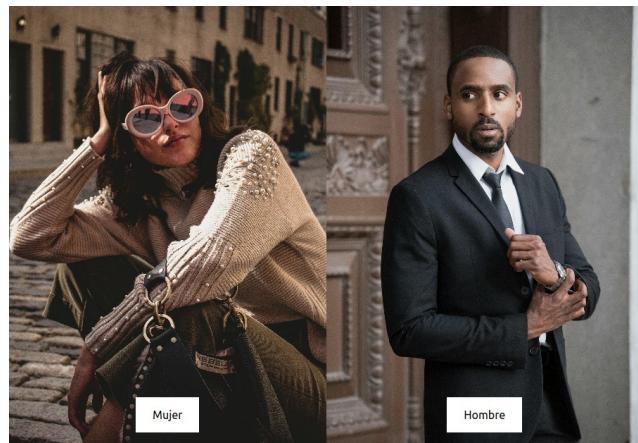


Imagen del categorías



Imágenes de la fila de categoría



Imágenes de secciones 1 y 2



Imágenes de productos



Imágenes de carrousel detalle productos

Manual de estilos

Iconos

Uso de iconos y descripción

El uso de iconos se utilizan principalmente en la interfaz como identificador de secciones. En mi página, se identifican tanto en el menú de navegación, como en los productos para identificar la wishlist.



Icono para iniciar sesión / para acceder al perfil si estás logueado.



Icono para acceder al carrito / para iniciar sesión si no estas logueado.



Icono para acceder a la lista de acceder o agregar productos a la lista de deseo.

Iconos de redes sociales, con un diseño de fondo negro y el ícono blanco, al contrario que los iconos anteriores.

Descripción de la estructura

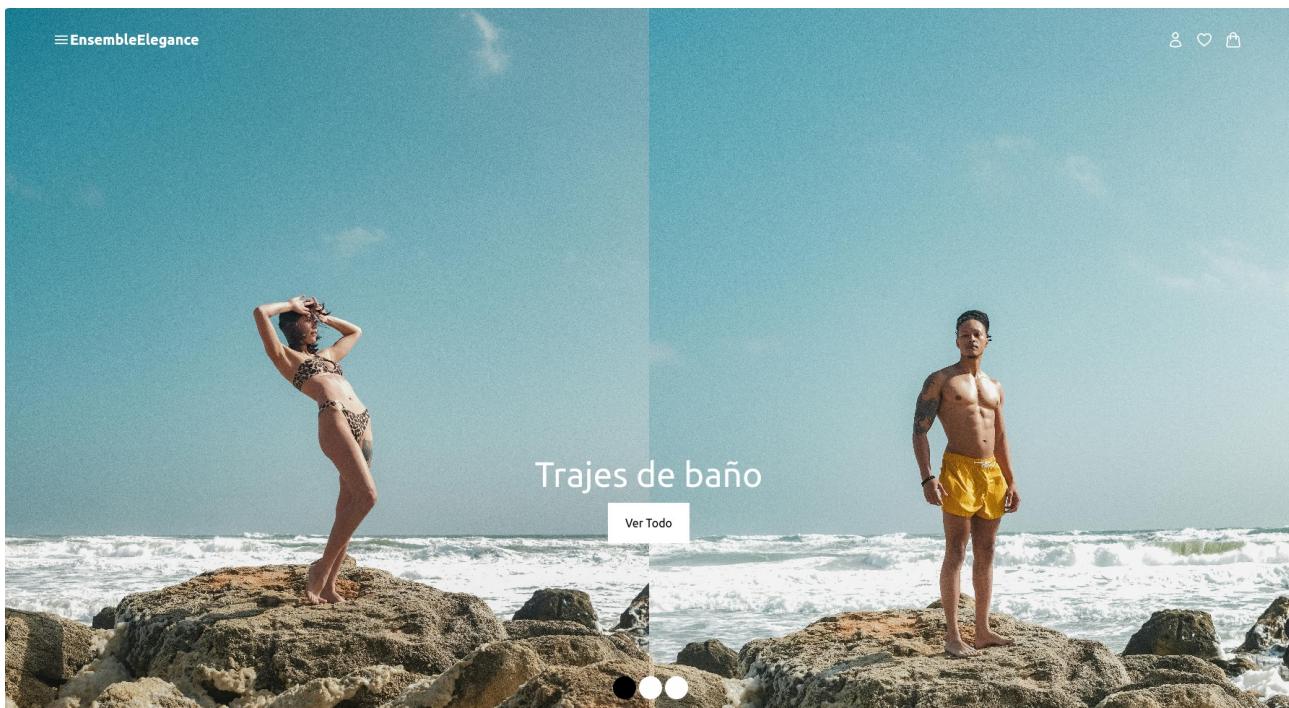
La página de inicio está diseñada para proporcionar una experiencia visualmente atractiva y fácil de navegar.

En la parte superior de la página se encuentra la barra de navegación principal, que permanece fija y visible mientras el usuario se desplaza. Esta barra incluye enlaces esenciales y opciones de usuario que facilitan la navegación por el sitio.

Inmediatamente debajo de la barra de navegación se encuentra el carrusel de imágenes. Es deslizante y se puede arrastrar, lo que permite a los usuarios interactuar fácilmente con las imágenes.

Incluye puntos que permiten a los usuarios ver su posición actual en el carrusel y navegar entre las imágenes. Las imágenes se reproducen automáticamente con una velocidad de 3500 ms. Utiliza una transición personalizada "all .5" con una duración de 500 ms para cambios suaves entre imágenes.

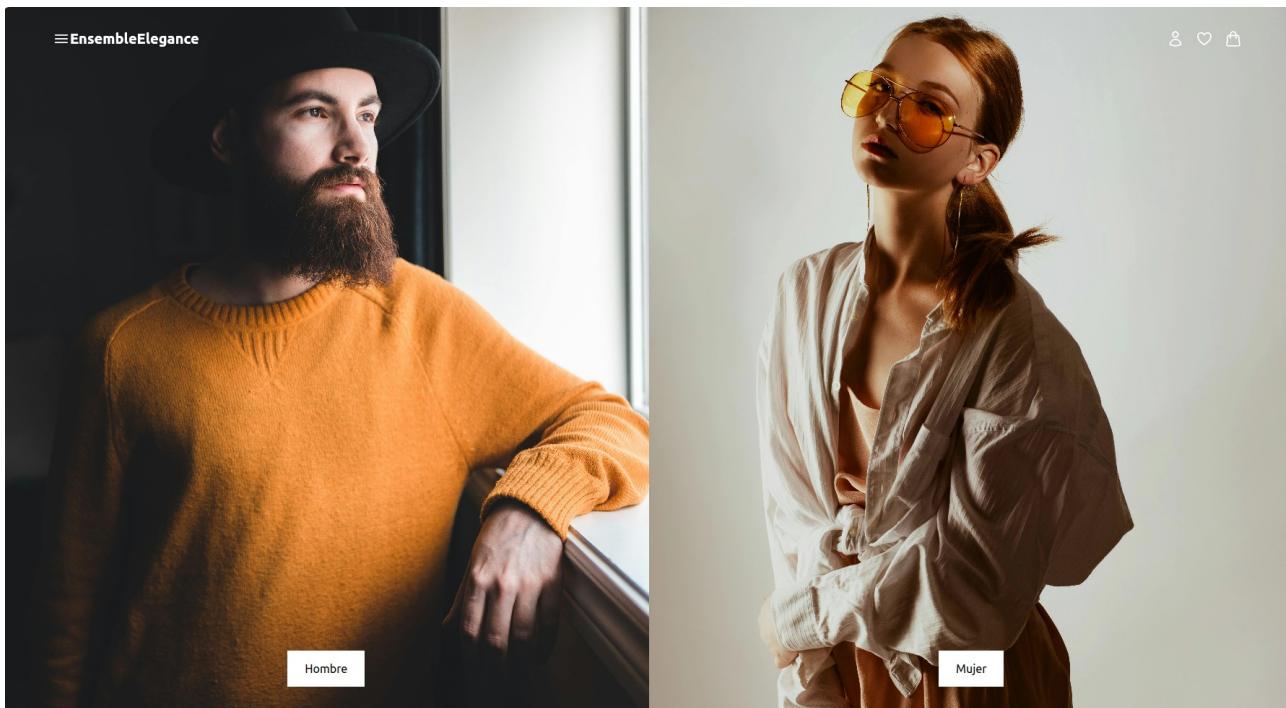
Cada imagen del carrusel está acompañada por un título y un botón "Ver Todo" que lleva al usuario a una página específica del sitio, proporcionando una llamada a la acción clara y accesible. El contenido textual se presenta utilizando un estilo de texto en blanco para garantizar una legibilidad con texto en letra en color negro.



Debajo del carrusel, la página se organiza en una cuadrícula de dos columnas, cada una dedicada a una categoría de productos. Estas secciones contiene una imagen a pantalla completa con un enlace que lleva a la página de productos de la categoría correspondiente. Un texto superpuesto, centrado y en color blanco y se destaca sobre un botón blanco con texto negro.

Manual de estilos

Estructura web



Estructura de las páginas de cada género

La página se organiza en una cuadícula de tres columnas que presenta imágenes a pantalla completa. Cada imagen está vinculada a una categoría específica de productos y sirve como un punto de entrada visualmente atractivo para diferentes secciones del sitio. La estructura se divide en tres filas principales, en algunas categorías se divide en dos filas principales:



Manual de estilos

Estructura web



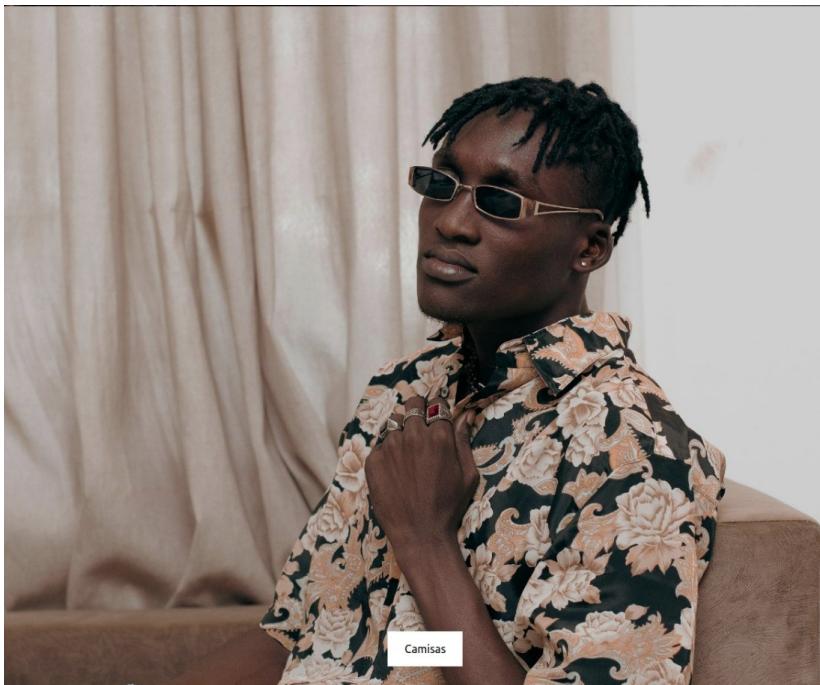
Polos



Pantalones



Sudaderas



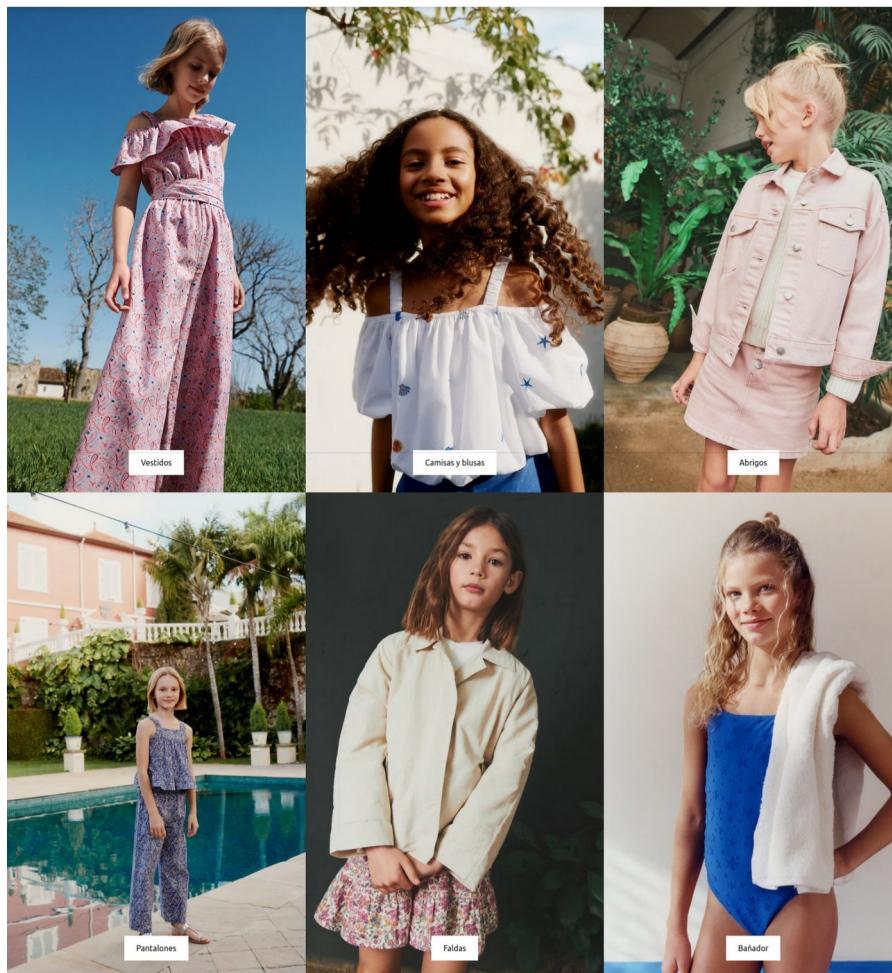
Camisas



Bañadores

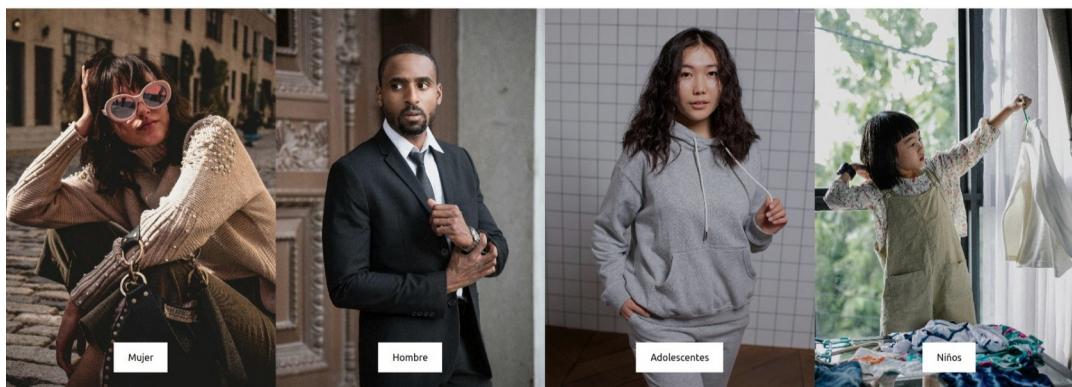
Manual de estilos

Estructura web



Debajo de las filas principales de la cuadrícula, se presenta una sección dedicada a las categorías principales del sitio, titulada "CATEGORÍAS". Esta sección está organizada en una fila con cuatro columnas, cada una representando una categoría específica. Esta estructura proporciona una interfaz intuitiva y visualmente coherente, asegurando que los usuarios puedan navegar y explorar las diferentes secciones del sitio con facilidad.

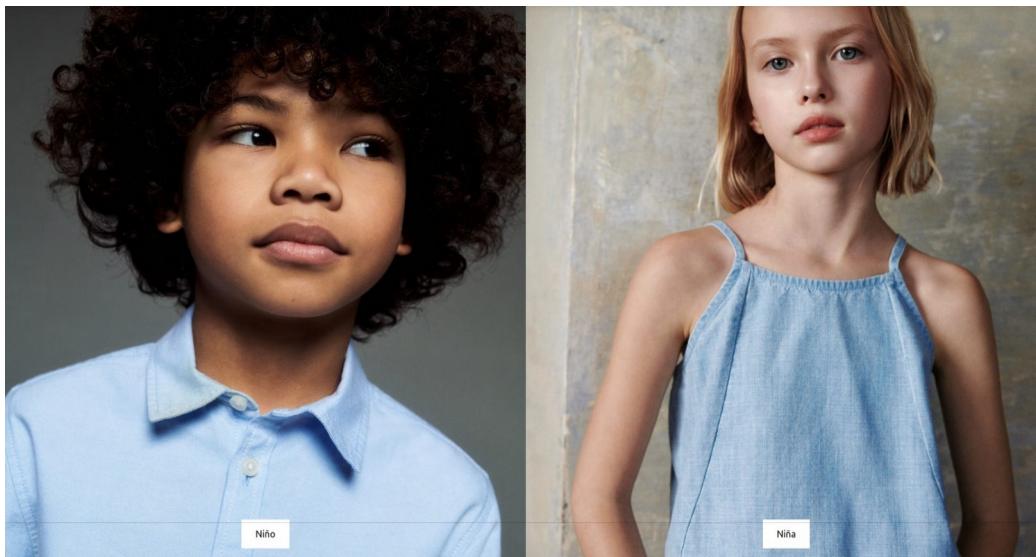
CATEGORÍAS



Manual de estilos

Estructura web

En la sección de adolescentes y niñas, hay dos categoría para definir el género masculino y femenino. Esta están dividida en una fila y dos columnas, cada columna con su correspondiente imagen.



Sección productos

Justo debajo de la barra de navegación, se encuentran los controles de búsqueda y vista. Estos elementos permiten a los usuarios buscar productos específicos y cambiar entre diferentes vistas de la página.

La sección principal de la página muestra los productos disponibles en una cuadricular de dos columnas. Cada producto se presenta con una imagen principal y detalles relevantes, como el nombre del producto, el color y el precio. Los usuarios pueden hacer clic en cada producto para ver más detalles. Los usuarios pueden agregar productos a su lista de deseos haciendo clic en el ícono de corazón.

Vista Filtro ...

Blusa plumeti bordado
19.99€

Blusa lino botones
22.99€

Blusa estampada fruncido
18.99€

Blusa volantes lino
25.99€

Manual de estilos

Estructura

Detalle Producto

La sección principal de la página muestra el producto en detalle. Aquí se incluye un carrusel de imágenes que permite a los usuarios ver diferentes ángulos y detalles del producto.

En la parte derecha del carrusel, se muestra el nombre del producto y su precio. Si hay una oferta disponible, se presenta el precio original junto con el precio de oferta resaltado en rojo. Esto ayuda a los usuarios a identificar rápidamente cualquier descuento disponible.

Se proporciona una descripción detallada del producto para informar a los usuarios sobre sus características. Esto ayuda a los usuarios a tomar decisiones informadas sobre su compra.

Los usuarios pueden seleccionar diferentes tallas utilizando botones de opción que muestran la disponibilidad de cada talla y justo debajo están los botones "Añadir a la cesta" y "Añadir a la lista de deseos" permiten a los usuarios agregar el producto a su carrito de compras o lista de deseos con un solo clic.



Blusa plumeti bordado

19.99€

100% algodón. Tejido plumeti. Detalle bordado. Diseño recto. Manga corta. Cierre de botones.

XXS	XS	S	M
L	XL	XXL	3XL
Añadir a la cesta			

Al final de la página, se muestran productos recomendados relacionados con el artículo actual. Estos productos ayudan a los usuarios a descubrir otros productos que puedan interesarles, ampliando así su experiencia de compra.

Manual de estilos

Estructura

Recomendados



Blusa lino botones



Blusa estampada fruncido



Blusa volantes lino

Manual de estilos

Inputs y filtros

Inputs y filtros

En el input se utiliza un diseño limpio y minimalista, con un fondo blanco que resalta sobre el fondo de la página. Se ha aplicado un borde y sombreado sutil para enfatizar visualmente el input sin distraer la atención del usuario.

Se utiliza un tamaño de texto adecuado para una fácil legibilidad, y se ha aplicado un color de texto oscuro sobre el fondo blanco para una mejor visibilidad.



Buscar productos...

Los filtros se presentan en una lista desplegable que se activa al hacer clic en el botón correspondiente. Cada filtro viene acompañado de una etiqueta descriptiva y una opción de selección, ya sea mediante casillas de verificación o botones de radio, según el tipo de filtro. Este proporciona claridad y facilidad de uso al usuario, con una organización limpia que simplifica la identificación y selección de los filtros deseados. Además, se ha empleado un esquema de color coherente con el resto del diseño de la página para asegurar una apariencia visualmente integrada.

Finalmente, se han incluido íconos visuales, como las flechas indicativas para el rango de precios, que mejoran la comprensión y usabilidad de los filtros, facilitando así la navegación y la búsqueda de productos.

Filtro ...



Por último, el filtro de vista permite a los usuarios alternar entre diferentes modos de visualización de los productos, como la vista de lista o la vista de cuadrícula. Al igual que los demás filtros, se presenta con una etiqueta descriptiva y una opción de selección, asegurando así una experiencia de usuario completa y coherente en la exploración de productos en la plataforma.

Vista 

Mensaje error

El mensaje de error está diseñado para ser claro y conciso, destacando sobre un fondo semitransparente para captar la atención del usuario sin distraer demasiado de la experiencia principal de navegación. Presentado dentro de un cuadro blanco con bordes redondeados y sombra, el mensaje incluye un encabezado en negrita para indicar la naturaleza del error, junto con un botón de cierre discreto en la esquina superior derecha. El cuerpo del mensaje ofrece una descripción clara y organizada del error específico, utilizando un tamaño de fuente legible y un diseño que se integra armoniosamente con el resto del diseño del sitio web. En conjunto, este estilo asegura que los usuarios puedan identificar y comprender rápidamente el problema.



Manual de estilos

Botones

Botones

El botón presenta un borde sólido de color negro para delinear claramente su contorno y destacarlo del fondo. El texto del botón es negro, asegurando un contraste alto con el fondo blanco, lo que facilita su lectura. El fondo blanco del botón proporciona un contraste nítido con el texto negro, otorgándole un aspecto limpio y contemporáneo. Al pasar el cursor sobre el botón, tanto el fondo como el color del texto cambian: el fondo se vuelve negro y el texto se vuelve blanco. Esta transformación visual indica que el botón es interactivo y responde a la acción del usuario, creando así una experiencia dinámica y atractiva durante la interacción.



Iniciar Sesión



Iniciar Sesión

Manual de estilos

Estructura de sesión

Estructura de sesión

El diseño de la página de registro e inicio de sesión se centra en la claridad y accesibilidad para el usuario. Ambas páginas dividen la pantalla en dos secciones: una para el formulario en sí y otra para una imagen de fondo que complementa visualmente la experiencia. Los formularios están diseñados con campos de entrada claramente etiquetados y estilizados para una fácil identificación, con mensajes de error visibles para guiar al usuario en caso de problemas. Además, se incluyen enlaces para alternar entre registro e inicio de sesión, proporcionando una navegación fluida.

Por último, si el usuario desea cerrar la pestaña, arriba a la izquierda hay un ícono en forma de "x" de color negro para facilitar la usabilidad.

Introduce tu email
Contraseña
Iniciar Sesión



Nombre
Correo electrónico
Contraseña
dd/mm/aaaa
Teléfono
Selecciona tu género
Registrarse



Manual de estilos

Menú hamburguesa

Menú hamburguesa

El menú de navegación se despliega al hacer clic en el ícono correspondiente y ocupa la mitad izquierda de la pantalla, presentando el logotipo del sitio y una lista vertical de opciones de navegación. Cada opción puede desplegar un submenu con más opciones relacionadas, cuando se despliega dicho submenu, el nodo padre, se subraya para indicar que estás posicionado en esa opción. Este diseño se adapta a diferentes dispositivos, ofreciendo una experiencia de usuario consistente. Además, se superpone a una imagen de fondo para mejorar la estética visual. Un botón de cierre en la esquina superior izquierda permite a los usuarios cerrar el menú y volver a la vista normal de la página.

x

EnsembleElegance

Mujer

Hombre

Adolescentes

Chico	Pantalones
Chica	Tops
Vestidos	Abrigos
Blusas y camisas	Bikinis
Faldas	
Chalecos	
Niños	



x

EnsembleElegance

Mujer

Hombre

Adolescentes

Niños



Manual de estilos

Pie de página

Píe de página

El pie de página presenta tres secciones divididas en columnas responsivas. Cada sección contiene una lista de enlaces que proporcionan información relevante, como contacto, políticas de la empresa y aspectos legales. Además, se incluyen íconos de redes sociales en la parte inferior derecha para facilitar el acceso a las plataformas externas. En conjunto, el diseño del pie de página ofrece una navegación intuitiva y una presentación clara de la información importante para los usuarios.

INFORMACIÓN

- [Contactanos](#)
- [Entrega](#)
- [Devoluciones](#)
- [Preguntas frecuentes](#)
- [Declaración frecuente](#)

NUESTRA COMPAÑÍA

- [Localizador de tiendas](#)
- [Somos EsembleElegance](#)
- [Somos responsable](#)
- [Mapa del sitio](#)
- [Prensa](#)
- [Api Rest](#)

LEGAL Y COOKIES

- [Términos y condiciones](#)
- [Términos de uso del sitio](#)
- [Política de privacidad](#)
- [Cookies](#)
- [Política Fiscal](#)
- [Administrar cookies](#)

© 2024 EsembleElegance. Todos los derechos reservados.



Manual de estilos

Encabezado administrador/director

Encabezado de administrador/director

El menú del Panel de Administrador/Director se destaca por su claridad y accesibilidad. En la parte superior, se encuentra el título "Panel de Administrador" en negrita y con el fondo en blanco para aumentar la legibilidad de un texto mediante un contraste, proporcionando una identificación clara del contexto de la página. A continuación, se presenta una única opción de menú: "Cerrar Sesión". Este botón, con un diseño minimalista, permite al usuario finalizar su sesión con un simple clic, asegurando una experiencia de navegación sin complicaciones. El contraste entre el fondo blanco y el texto oscuro garantiza una buena legibilidad.

Panel de Administrador

Cerrar sesión

Panel de Director

Cerrar Sesión

Manual de estilos

Sidebar administrador/director

Sidebar

El menú de navegación de Ensemble Elegance presenta una estructura clara y fácil de usar para acceder a diferentes secciones del sitio. Cada elemento del menú, desde "Inicio" hasta "Tareas" y "Panel Oferta", se destaca con un fondo blanco y texto negro cuando está activo, lo que facilita la identificación de la sección actual. Además, algunos elementos del menú despliegan submenús al hacer clic, indicados por íconos de flecha que giran para señalar si están abiertos o cerrados. Estos submenús ofrecen enlaces adicionales relacionados con la sección principal. Todos los elementos del menú mantienen un estilo coherente, con íconos de flecha y texto en blanco para mantener la estética general del diseño.



Manual de estilos

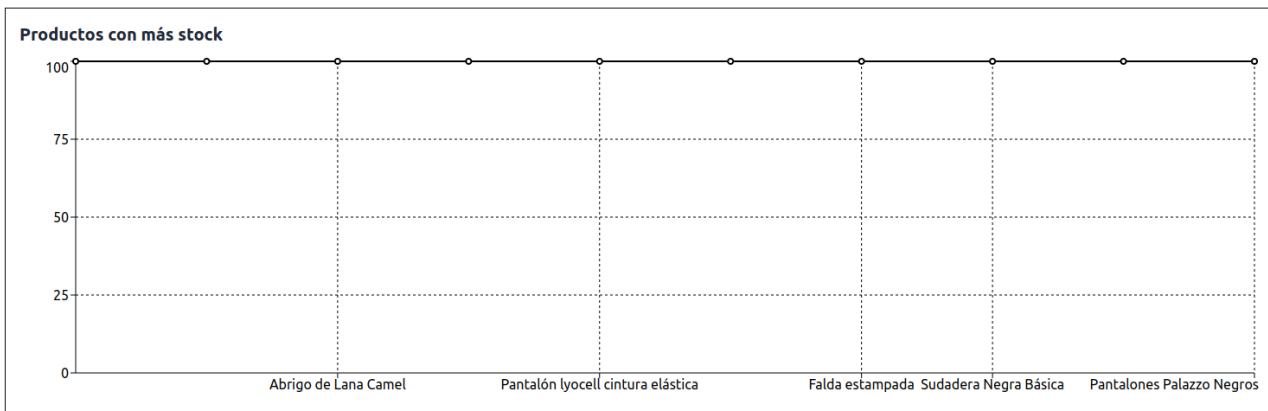
Gráficas y tablas

Gráficas y tablas

Estos valores muestran un título, un valor y un detalle en un contenedor con fondo blanco y borde negro. El texto del título está en mayúsculas y en color negro, mientras que el valor es grande y en negrita. El detalle utiliza un tamaño de texto más pequeño y también está en negro.



El gráfico tiene ejes X e Y con etiquetas y líneas punteadas de fondo. Cada línea de datos se representa con una línea sólida de color negro y un ancho de trazo de 2 píxeles. El estilo tiene un fondo blanco y un borde gris.



La lista de tareas asignadas es una tabla dentro de un contenedor con fondo blanco y borde negro. El contenedor tiene una altura máxima de 400 píxeles y un desplazamiento vertical si el contenido excede este límite. La tabla tiene celdas de título y datos con borde negro, y las tareas se muestran en filas. Si no hay tareas pendientes, se muestra un mensaje indicando esto. Además, el scroll es de color blanco y con un fondo negro.

La tabla de envío sigue el mismo estilo.

Tareas Asignadas	
Título	Estado
DESPEDIDO PARGUELA	pendiente

Envíos			
ID	Cliente	Total	Estado
1	Test User	96.79	Enviado

Manual de estilos

Tablas

Tablas

Las tablas de la página web presentan un estilo limpio y minimalista, caracterizado por la ausencia de líneas divisorias entre filas y columnas, lo que contribuye a una apariencia ordenada y moderna. Las celdas de la tabla están diseñadas con un espaciado equitativo y un texto de tamaño pequeño, lo que facilita la lectura y la identificación de la información. El uso de colores neutros y un fondo blanco resalta el contenido de la tabla.

Producto	Talla	Cantidad	Acciones		
Americana Azul Marino Slim Fit	XXS	0	<button>cantidad</button>	<button>Incrementar</button>	<button>Disminuir</button>
Americana Azul Marino Slim Fit	XS	22	<button>cantidad</button>	<button>Incrementar</button>	<button>Disminuir</button>
Americana Azul Marino Slim Fit	S	51	<button>cantidad</button>	<button>Incrementar</button>	<button>Disminuir</button>
Americana Azul Marino Slim Fit	M	82	<button>cantidad</button>	<button>Incrementar</button>	<button>Disminuir</button>
Americana Azul Marino Slim Fit	L	17	<button>cantidad</button>	<button>Incrementar</button>	<button>Disminuir</button>
Americana Azul Marino Slim Fit	XL	20	<button>cantidad</button>	<button>Incrementar</button>	<button>Disminuir</button>
Americana Azul Marino Slim Fit	XXL	63	<button>cantidad</button>	<button>Incrementar</button>	<button>Disminuir</button>
Americana Azul Marino Slim Fit	3XL	42	<button>cantidad</button>	<button>Incrementar</button>	<button>Disminuir</button>
Americana Gris Elegante	XXS	33	<button>cantidad</button>	<button>Incrementar</button>	<button>Disminuir</button>
Americana Gris Elegante	XS	98	<button>cantidad</button>	<button>Incrementar</button>	<button>Disminuir</button>

Manual de estilos

Paginación

Paginación

La paginación tiene una serie de botones que representan cada página disponible. Estos botones están centrados y se destacan mediante un fondo blanco y un borde negro. Cuando un botón corresponde a la página actual, se resalta con un fondo negro y texto blanco, lo que proporciona una clara indicación visual de la página actual. Los botones de navegación "<<", que llevan a la primera página, y ">>", que llevan a la última página, están presentes en los extremos de la lista, ofreciendo una manera conveniente de saltar rápidamente al inicio o al final de los resultados. Este diseño simple y efectivo mejora la usabilidad y la accesibilidad al facilitar la navegación entre las páginas de resultados de manera clara y ordenada.



Manual de estilos

Formulario

Formulario

El diseño del formulario se centra en proporcionar una experiencia de usuario intuitiva y eficiente al ingresar datos. Cada campo del formulario está claramente etiquetado con un título descriptivo en negrita para guiar al usuario sobre qué información debe ingresar. Los campos de entrada de texto están diseñados para centrar verticalmente el texto y presentan un borde negro que resalta el área de entrada. Para los campos de carga de imágenes, se utiliza los botones instinctivo de la página. En el select también se usa el mismo estilo que los botones.

Añadir un nuevo administrador

Este formulario es para agregar un nuevo administrador. Contiene los siguientes campos: Nombre completo (input), Correo (input), Selecciona tu género (select), Contraseña (input), dd/mm/aaaa (input con placeholder y icono de calendario), Teléfono (input) y un botón Añadir administrador (button).

En el formulario de usuario, los inputs no tienen un borde en todo el input, sino que, solo se posiciona el borde inferior del input.

ESCRIBE TUS DATOS DE REGISTRO

Este formulario es para registrar datos. Contiene los siguientes campos: Nombre (input), Correo electrónico (input), Contraseña (input), dd/mm/aaaa (input con placeholder y icono de calendario), Teléfono (input), Selecciona tu género (select) y un botón Registrarse (button).

¿Ya tienes cuenta? **INICIA SESIÓN**