Trabajo Programación Entorno Servidor 2º DAW

Por Samuel Gómez, Carlos Cortés, Victor Marín y Adrián Herrero

Índice

Introducción

Objetivo del proyecto

Dawazon se presenta como una plataforma de comercio electrónico moderna y robusta, diseñada bajo el paradigma de aplicación web dinámica. Su objetivo es ofrecer una experiencia de usuario fluida y una gestión eficiente del catálogo, centralizando la lógica y la presentación en el servidor.

El núcleo del sistema ha sido desarrollado utilizando Java 25 sobre el framework Spring Boot. A diferencia de las aplicaciones de una sola página (SPA), Dawazon utiliza Server-Side Rendering (SSR) con el motor de plantillas Pebble, lo que garantiza una carga rápida inicial, mejor SEO y una arquitectura simplificada

Almacenamiento de Datos:

El sistema implementa una estrategia de persistencia híbrida para maximizar el rendimiento y la flexibilidad:

- PostgreSQL: Base de datos relacional empleada para la gestión estructurada del catálogo de productos y usuarios, garantizando la integridad referencial y un manejo robusto de la concurrencia.
- MongoDB: Base de datos NoSQL orientada a documentos, utilizada para almacenar datos con estructuras variables o flexibles, como el historial de pedidos y los logs del sistema.

Infraestructura y Despliegue:

- Docker: La aplicación sigue un enfoque "Cloud-Native", encontrándose totalmente contenerizada para agilizar el despliegue y asegurar la consistencia operativa entre entornos.

Alcance

El sistema abarca las siguientes funcionalidades clave:

- Gestión de Catálogo: Administración integral de productos (nombre, precio, categoría, descripción e imágenes).
- Interfaz de Usuario Dinámica: Generación de vistas HTML en el servidor adaptadas al estado de la sesión del usuario.
- Gestión de Sesiones: Control de acceso y carritos de compra persistentes mediante sesiones HTTP.
- Operaciones CRUD: Implementación completa para la gestión del inventario a través de formularios web.
- Validación de Datos: Reglas de negocio estrictas aplicadas tanto en el cliente (HTML5) como en el servidor (Bean Validation).
- Persistencia Políglota: Integración eficiente de SQL y NoSQL.

Tecnologías empleadas

- Lenguaje: Java 25.
- Framework Backend: Spring Boot.
- Persistencia:
    - PostgreSQL.
    - MongoDB.
- Frontend / Templating: Motor de plantillas Pebble.
- Infraestructura:
    - Docker
    - Docker Compose.
- Entorno de Desarrollo: IntelliJ IDEA.
- Herramientas Adicionales:
    - Redis
    - JUnit
    - Dokka
- Gestión de Proyecto: Trello.

Arquitectura del sistema

La arquitectura del sistema se basa en el patrón Modelo-Vista-Controlador (MVC) sobre Spring Boot. Este diseño monolítico modular permite que el servidor gestione tanto la lógica de negocio como la representación visual.

1. Estructura general del sistema

El sistema procesa las peticiones del navegador, ejecuta la lógica pertinente y devuelve páginas HTML completas.

- El Cliente (Navegador): Envía peticiones HTTP estándar (GET/POST) y renderiza el HTML/CSS recibido.
- El Servidor (Spring Boot):
    - Recibe la petición en el Controlador.
    - Procesa la lógica de negocio y accede a la base de datos.
    - Selecciona una plantilla Pebble.
    - Inyecta los datos (Modelo) en la plantilla y genera el HTML final.

2. Capas del sistema

Capa de Controladores (MVC Controllers) Es el punto de entrada. Gestionan la navegación y el flujo de la aplicación.

- Responsabilidades: Mapear URLs a vistas, procesar envíos de formularios (@ModelAttribute), gestionar objetos de sesión (como el usuario logueado o el carrito) y redirigir flujos.
- Ejemplo: ProductsController (carga la lista de productos en el modelo y devuelve "productos/lista.html").

Capa de Servicios (Services)

Alberga la lógica de negocio pura, desacoplada de la vista.

- Responsabilidades: Validaciones complejas, cálculos, transformación de datos, gestión de transacciones y comunicación con los repositorios.
- Ejemplo: ProductService.

Capa de Repositorios (Repositories)

- Implementación: Repositorios JPA/Hibernate para PostgreSQL y MongoRepository para MongoDB.
- Ejemplos: ProductsRepository.

Capa de Vistas (Templates)

- Tecnología: Pebble.
- Responsabilidades: Archivos .html con lógica de presentación (bucles, condicionales) que transforman los objetos Java en código HTML visible para el usuario.

3. Componentes principales de la arquitectura

Motor de Vistas (Pebble)

El sistema utiliza Pebble como motor de plantillas. Es el encargado de la capa de presentación.

- Herencia de Plantillas: Uso de una plantilla base (layout.html) que define la cabecera, el pie de página y los estilos comunes, permitiendo que las vistas específicas solo inyecten el contenido principal.
- Seguridad: Pebble escapa automáticamente las variables para prevenir ataques XSS (Cross-Site Scripting).

Controladores Web (Spring MVC)

Gestionan la interacción usuario-sistema mediante verbos HTTP clásicos:

- GET: Para solicitar y renderizar páginas (ej. ver el catálogo, ver formulario de login).
- POST: Para enviar datos de formularios (ej. crear un producto, realizar un pedido).
- Validación: Uso de BindingResult para capturar errores en los formularios y volver a renderizar la vista con los mensajes de error correspondientes.

Gestión de Sesiones y Seguridad

Utiliza Spring Security junto con sesiones HTTP.

- Autenticación: Formulario de login propio procesado por Spring Security.
- Autorización: Control de acceso a rutas basado en roles.
- Estado: El carrito de la compra y los datos del usuario se mantienen en la sesión del servidor (respaldada por Redis para persistencia).

Bases de Datos.

- PostgreSQL: Para datos relacionales y estructurados (Usuarios, Productos, Categorías).
- MongoDB: Para documentos de Pedidos (guardando una "foto fija" del producto y cliente en el momento de la compra).

Docker.

La infraestructura se gestiona mediante docker-compose.yml, levantando los siguientes servicios:

- Spring Boot App: La aplicación web.
- PostgreSQL: BBDD Relacional.
- MongoDB: BBDD Documental.
- Redis: Almacenamiento de sesiones HTTP y caché.
- Nginx: Proxy inverso y servidor de estáticos.
- Apache/Nginx (Auxiliares): Para servir documentación y reportes de tests.

4. Flujo general de funcionamiento

1. Petición del Usuario: El usuario hace clic en un enlace o envía un formulario (ej. "Añadir al carrito"). El navegador envía una petición HTTP al servidor.

2. DispatcherServlet: Spring recibe la petición y la enruta al Controlador correspondiente según la URL.

3. Procesamiento (Controlador/Servicio):
    - El controlador recibe los datos (ej. parámetros del formulario).
    - Llama al Servicio para ejecutar la lógica de negocio (ej. verificar stock, guardar en base de datos).
    - El Servicio interactúa con los Repositorios (Postgre/Mongo).

4. Selección de Vista: El controlador selecciona qué plantilla mostrar (ej. redirect:/carrito o return "producto/detalle").

5. Renderizado (Pebble): El motor de plantillas combina el HTML estático con los datos Java (Modelo) proporcionados por el controlador.

6. Respuesta: El servidor envía el HTML generado al navegador del usuario.

Patrones Utilizados

En el desarrollo de Dawazon se han aplicado rigurosamente patrones de diseño y arquitectura de software. Estas directrices estructurales facilitan la organización lógica del código, aseguran la mantenibilidad a largo plazo y permiten la escalabilidad horizontal y vertical de la aplicación.

A continuación, se detallan los patrones principales implementados explícitamente en el diseño:

1. Patrón MVC (Modelo-Vista-Controlador)

Es el patrón arquitectónico base del proyecto. Separa la lógica de control, la interfaz de usuario y los datos.

- Modelo: DTOs y Entidades que transportan información.
- Vista: Plantillas Pebble (.html).
- Controlador: Clases Java con anotaciones @Controller.

2. Patrón DTO (Data Transfer Object)

El patrón DTO es fundamental para el mapeo de formularios.

- Uso: Se utilizan clases FormDTO (como ProductRequestDTO) para capturar los inputs del usuario, validarlos y luego transferir esa información a las entidades de dominio, evitando exponer la estructura de la base de datos directamente en la vista HTML.

Tecnologías

Para el desarrollo del proyecto se ha seleccionado un conjunto de herramientas modernas y estándares de la industria que garantizan la calidad del código, la eficiencia en el flujo de trabajo y la robustez de la infraestructura.

- Lenguaje: Java 25.
- Framework: Spring Boot (Spring MVC, Spring Security, Spring Data).
- Motor de Plantillas: Pebble.
- Bases de Datos: PostgreSQL y MongoDB.
- Caché/Sesiones: Redis.
- Infraestructura: Docker, Docker Compose.
- IDE: IntelliJ IDEA.
- Gestión: Git, GitHub, Trello.
- Documentación: Dokka (HTML), Microsoft Word.

Planificación Trello

Para el reparto de tareas y poder planificar el proyecto eficientemente se ha hecho uso de Trello, un software de administración muy intuitivo que ha sido de gran ayuda.

A continuación, el Trello del proyecto:

https://trello.com/b/FToRIHAm/tienda-de-objetos-bonicos

Requisitos funcionales y no funcionales

La definición de requisitos establece las bases funcionales y operativas del sistema Dawazon, asegurando que el software cumpla con las expectativas de negocio y los estándares de calidad técnica.

1. Requisitos Funcionales (RF)

Los requisitos funcionales describen los comportamientos específicos y las funciones que el sistema debe soportar para permitir la operativa diaria de la tienda.

- RF1 Operaciones CRUD (productos, categorías y pedidos).
- RF2 Carrito de compra.
- RF3 Id único a cada entidad.
- RF4 API REST.
- RF5 Interfaz gráfica dinámica (Pebble).
- RF6 Pago con tarjeta (Stripe).
- RF7 Usuarios, seguridad (Spring Security) y HTTP Session.
- RF8 Tarea programada.

2. Requisitos No Funcionales (RNF)

Los requisitos no funcionales definen los atributos de calidad del sistema, como el rendimiento, la seguridad y la mantenibilidad.

- RNF1 Caché Redis.
- RNF2 Testeo del código (unitarios + integración con test container + E2E Playwrigt).
- RNF3 Docker.
- RNF4 GitHub.
- RNF5 Proxy inverso con Nginx.

3. Requisitos de Información (RI)

- RI1 Modelo + DTOs de usuario.
- RI2 Modelo + DTOs de cliente.
- RI3 Modelo + DTOs de producto.
- RI4 Modelo de categoría.
- RI5 Modelo + DTOs de pedido.
- RI6 Modelo + DTOs de comentario.

- RI7 Modelo + DTOs de carrito.

Diagrama de clases

Principios SOLID aplicados

Principio de responsabilidad única (SRP)

- Separación estricta entre ProductoController (gestión de vistas/HTTP) y ProductoService (lógica de negocio).

Principio de abierto/cerrado (OCP)

- Uso de interfaces para, por ejemplo el servicio de almacenamiento (StorageService), separando el “contrato” de la implementación específica, representada por la clase.

Principio de sustitución de Liskov (LSP)

- Implementación correcta de interfaces en los servicios.

Principio de segregación de interfaces (ISP)

- Interfaces de servicio específicas por dominio en lugar de una interfaz gigante.

Principio de inversión de dependencias (DIP)

- Inyección de dependencias en los controladores a través de interfaces, gestionada por SpringBoot.

Diseño de la base de datos

El sistema utiliza un diseño híbrido de bases de datos, combinando una base de datos relacional (Postgre) y una NoSQL orientada a documentos (MongoDB). Esta arquitectura permite aprovechar las ventajas de ambos modelos según el tipo de información que maneja la aplicación.

1. Base de Datos Relacional: PostgreSQL

La base de datos Postgre se utiliza para almacenar información estructurada y altamente relacionada. En este proyecto, se emplea principalmente para gestionar la información principal del catálogo de productos, categorías y usuarios.

La entidad sigue una estructura estándar JPA:

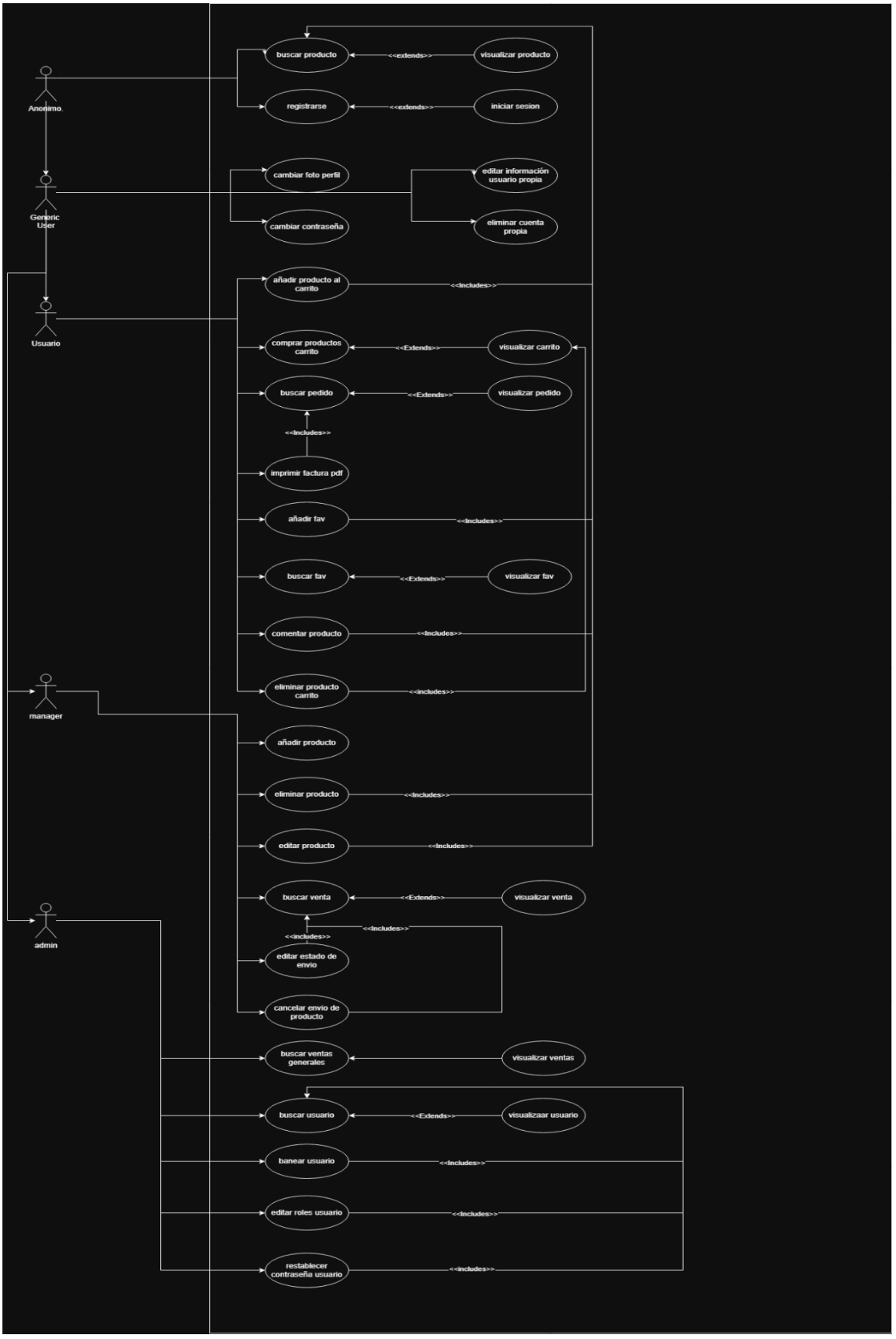
- Usuarios: Credenciales, roles, datos de perfil.
- Productos y Categorías: Definición del catálogo con relaciones fuertes (Integridad Referencial).

2. Base de Datos NoSQL: MongoDB

La base de datos MongoDB se utiliza para almacenar información que puede requerir una estructura más flexible o variable. En nuestro proyecto, lo hemos usado para definir la estructura de Pedidos ya que lleva dos clases embebidas tales como Cliente y LineaPedido lo que puede requerir cambios constantes en el modelo.

Este enfoque es especialmente útil ya que, al confirmar una compra, se genera un documento en MongoDB que contiene una copia de los datos del cliente y de los productos en ese instante (snapshot). Esto asegura que, si un producto cambia de precio o nombre en el futuro (en PostgreSQL), el historial del pedido (en MongoDB) permanezca inalterado y fiel a la realidad del momento de la compra.

Casos de uso


Anonimo
Generic User
Usuario
manager
admin
buscar producto
<<extends>>
visualizar producto
registrarse
<<extends>>
iniciar sesion
cambiar foto perfil
editar información usuario propia
cambiar contraseña
eliminar cuenta propia
añadir producto al carrito
<<Includes>>
comprar productos carrito
<<Extends>>
visualizar carrito
buscar pedido
<<Extends>>
visualizar pedido
<<Includes>>
imprimir factura pdf
añadir fav
<<Includes>>
buscar fav
<<Extends>>
visualizar fav
comentar producto
<<Includes>>
eliminar producto carrito
<<includes>>
añadir producto
eliminar producto
<<Includes>>
editar producto
<<Includes>>
buscar venta
<<Extends>>
visualizar venta
<<Includes>>
<<includes>>
editar estado de envio
cancelar envio de producto
buscar ventas generales
visualizar ventas
buscar usuario
<<Extends>>
visualizaar usuario
banear usuario
<<Includes>>
editar roles usuario
<<includes>>
restablecer contraseña usuario
<<includes>>

Nombre	Usuario no registrado busca / filtra
Id	CU - 1
Descripción	Un usuario no registrado busca o filtra productos en la página principal.
Actores implicados	Anónimo.
Precondiciones	Ninguna.
Pasos / curso normal	1.- El usuario elige los criterios de búsqueda o la categoría filtrada. 2.- El usuario aplica los filtros y consulta los resultados.
Postcondiciones	Ninguna.
Alternativa	No hay.

Nombre	Crear Usuario
Id	CU - 2
Descripción	Un usuario no registrado se registra en la página principal
Actores implicados	Anónimo.
Precondiciones	Ninguna.
Pasos / curso normal	1.- El usuario entra en la página principal. 2.- El usuario pulsa el botón de registrarse. 3.- El usuario rellena el formulario de registro. 4.- El usuario completa el formulario y vuelve a la página de inicio registrado
Postcondiciones	Ninguna.
Alternativa	No hay.

Nombre	Inicio de sesión de usuario
Id	CU - 3
Descripción	Un usuario que ya tiene cuenta creada se identifica para acceder al sistema.
Actores implicados	Anónimo.
Precondiciones	Tener una cuenta creada.
Pasos / curso normal	1.- El usuario hace click en el botón de la imagen de usuario. 2.- El usuario introduce su correo y la contraseña. 3.- El usuario accede al sistema.
Postcondiciones	Ninguna.
Alternativa	Ninguna.

Nombre	Cerrar sesión
Id	CU - 4
Descripción	Un usuario que ha iniciado sesión se quiere ir y cierra la sesión.
Actores implicados	Usuario.
Precondiciones	Usuario logueado.
Pasos / curso normal	1.- El usuario debe de estar logueado. 2.- El usuario entra en la página principal. 3.- El usuario pulsa el botón de su perfil. 4.- El usuario pulsa el botón de cerrar sesión.
Postcondiciones	Ninguna.
Alternativa	No hay.

Nombre	Crear Manager
Id	CU - 5

Descripción	Un administrador crea un manager de un usuario registrado previamente.
Actores implicados	Admin y usuario (futuro manager).
Precondiciones	Admin logueado y Usuario previamente creado.
Pasos / curso normal	1.- El usuario debe de estar registrado. 2.- El administrador entra en la página principal. 3.- El administrador pulsa el botón de loguearse. 4.- El administrador registra al usuario como manager.
Postcondiciones	Ninguna.
Alternativa	No hay.

Nombre	**Añadir al carrito de compra**
Id	CU - 6
Descripción	Un usuario logueado añade un producto a su carrito de compra.
Actores implicados	Usuario.
Precondiciones	Tener una cuenta creada y estar logueado.
Pasos / curso normal	1.- El usuario navega por los productos disponibles en la tienda. 2.- El usuario accede al detalle de un producto que le interesa. 3.- El usuario añade dicho producto a su carrito de compra mediante el botón correspondiente.
Postcondiciones	Ninguna.
Alternativa	No hay.

Nombre	**Editar cantidad de producto**
Id	CU - 7

Descripción	Un usuario logueado modifica la cantidad de uno de los productos de su carrito de compra.
Actores implicados	Usuario.
Precondiciones	Tener una cuenta creada y estar logueado.
Pasos / curso normal	1.- El usuario hace click en el botón del carrito de compra. 2.- El usuario consulta los productos que tiene añadidos al carrito. 3.- El usuario edita la cantidad de uno de los productos añadidos.
Postcondiciones	Ninguna.
Alternativa	No hay.

Nombre	**Eliminar del carrito de compra**
Id	CU - 8
Descripción	Un usuario logueado consulta su carrito de compra y elimina uno de los productos agregados.
Actores implicados	Usuario.
Precondiciones	Tener una cuenta creada y estar logueado.
Pasos / curso normal	1.- El usuario hace click en el botón del carrito de compra. 2.- El usuario consulta los productos que tiene añadidos al carrito. 3.- El usuario elimina uno de los productos mediante el botón correspondiente.
Postcondiciones	Ninguna.
Alternativa	No hay.

Nombre	**Añadir a favoritos**

Id	CU - 9
Descripción	Un usuario registrado añade productos a favoritos que le interesan.
Actores implicados	Usuario.
Precondiciones	Usuario registrado.
Pasos / curso normal	1.- El usuario entra en la página principal. 2.- El usuario inicia sesión en la página. 3.- El usuario busca un producto que le interese. 4.- El usuario entra en el producto interesado. 5.- El usuario pulsa el botón de añadir a favorito.
Postcondiciones	Ninguna.
Alternativa	No hay.

Nombre	**Ver favoritos**
Id	CU - 10
Descripción	Un usuario registrado ve los productos añadidos a favoritos.
Actores implicados	Usuario.
Precondiciones	Usuario registrado.
Pasos / curso normal	1.- El usuario entra en la página principal. 2.- El usuario inicia sesión en la página. 3.- El usuario entra a su perfil. 4.- El usuario pulsa el botón favorito.
Postcondiciones	Ninguna.
Alternativa	No hay.

Nombre	**Eliminar de favorito**

Id	CU - 11
Descripción	Un usuario logueado consulta sus productos favoritos y elimina uno de los productos agregados.
Actores implicados	Usuario.
Precondiciones	Tener una cuenta creada y estar logueado.
Pasos / curso normal	1.- El usuario hace click en el botón del perfil. 2.- El usuario pulsa el botón favorito. 3.- El usuario elimina uno de los productos mediante el botón correspondiente.
Postcondiciones	Ninguna.
Alternativa	No hay.

Nombre	**Realizar un pedido**
Id	CU - 12
Descripción	Un usuario realiza el proceso de compra completo.
Actores implicados	Usuario.
Precondiciones	El usuario debe estar logueado con su cuenta.
Pasos / curso normal	1.- El usuario entra en la página principal. 2.- El usuario inicia sesión en la página. 3.- El usuario encuentra uno o más productos que le interesan. 4.- El usuario añade los productos al carrito. 5.- El usuario accede a su carrito de compra y comprueba que está todo en orden. 6.- El usuario pulsa el botón para iniciar el proceso de pago. 7.- El usuario introduce los datos de la tarjeta de crédito.

	8.- El usuario confirma / edita los datos de la dirección de entrega. 9.- El usuario confirma el pedido.
Postcondiciones	Ninguna.
Alternativa	1.- Cancelación del pedido por voluntad del usuario. 2.- Cancelación del pedido por datos de la tarjeta de crédito incorrectos. 3.- Cancelación del pedido por datos de la dirección de entrega con errores de validación.

Nombre	**Vista productos Manager**
Id	CU - 13
Descripción	Un manager logueado consulta sus productos creados.
Actores implicados	Manager.
Precondiciones	Tener una cuenta creada, estar logueado y tener productos creados.
Pasos / curso normal	1.- El manager hace click en el botón del perfil. 2.- El manager pulsa el botón mis productos. 3.- El manager entra en la vista de productos que haya creado en cualquier estado.
Postcondiciones	Ninguna.
Alternativa	No hay.

Nombre	**Vista de usuarios**

Id	CU - 14
Descripción	Un administrador logueado consulta los usuarios/manager creados.
Actores implicados	Admin.
Precondiciones	Tener una cuenta creada y estar logueado como administrador.
Pasos / curso normal	1.- El administrador hace click en el botón de perfil. 2.- El administrador pulsa el botón usuarios. 3.- El administrador entra en la vista de usuarios que están creados.
Postcondiciones	Ninguna.
Alternativa	No hay.

Nombre	**Editar usuarios**
Id	CU - 15
Descripción	Un administrador logueado consulta los usuarios/manager creados para eliminar uno.
Actores implicados	Admin.
Precondiciones	Tener una cuenta creada y estar logueado como administrador.
Pasos / curso normal	1.- El administrador hace click en el botón de perfil. 2.- El administrador pulsa el botón usuarios. 3.- El administrador entra en la vista de usuarios que están creados. 4.- El administrador pulsa el botón de editar usuario. 5.- El administrador entra en los detalles del usuario. 6.- El administrador modifica los cambios necesarios. 7.- El administrador pulsa el botón de guardar.

Postcondiciones	Ninguna.

Nombre	**Eliminacion de usuarios**
Id	CU - 16
Descripción	Un administrador logueado consulta los usuarios/manager creados para eliminar uno.
Actores implicados	Admin.
Precondiciones	Tener una cuenta creada y estar logueado como administrador.
Pasos / curso normal	1.- El administrador hace click en el botón de perfil. 2.- El administrador pulsa el botón usuarios. 3.- El administrador entra en la vista de usuarios que están creados. 4.- El administrador pulsa el botón de eliminar y elimina el usuario.
Postcondiciones	Ninguna.
Alternativa	No hay.

Nombre	**Crear Producto**
Id	CU - 17
Descripción	Un Manager da de alta un nuevo producto en el sistema para ponerlo a la venta.
Actores implicados	Manager.
Precondiciones	Estar logueado y tener rol de Manager
Pasos / curso normal	1.- El Manager accede a su perfil y pulsa en Mis productos. 2.- El sistema muestra la vista mis productos manager. 3.- El Manager pulsa el botón Añadir producto.

	4.- El sistema muestra el formulario de creación. 5.- El Manager rellena los datos. 6.- El Manager pulsa Guardar.
Postcondiciones	El producto aparece en el listado y es visible para los usuarios.
Alternativa	No hay.

Nombre	**Crear Producto**
Id	CU - 18
Descripción	Un Manager modifica la información de un producto existente.
Actores implicados	Manager.
Precondiciones	Estar logueado, ser Manager y tener productos creados.
Pasos / curso normal	1.- El Manager accede a su perfil y pulsa en Mis productos. 2.- El Manager selecciona un producto específico. 3.- El Manager muestra la vista del producto. 4.- El Manager pulsa el botón Editar. 5.- El sistema permite modificar los campos editables. 6.- El Manager pulsa Guardar.
Postcondiciones	La información del producto se actualiza en la base de datos.
Alternativa	No hay.

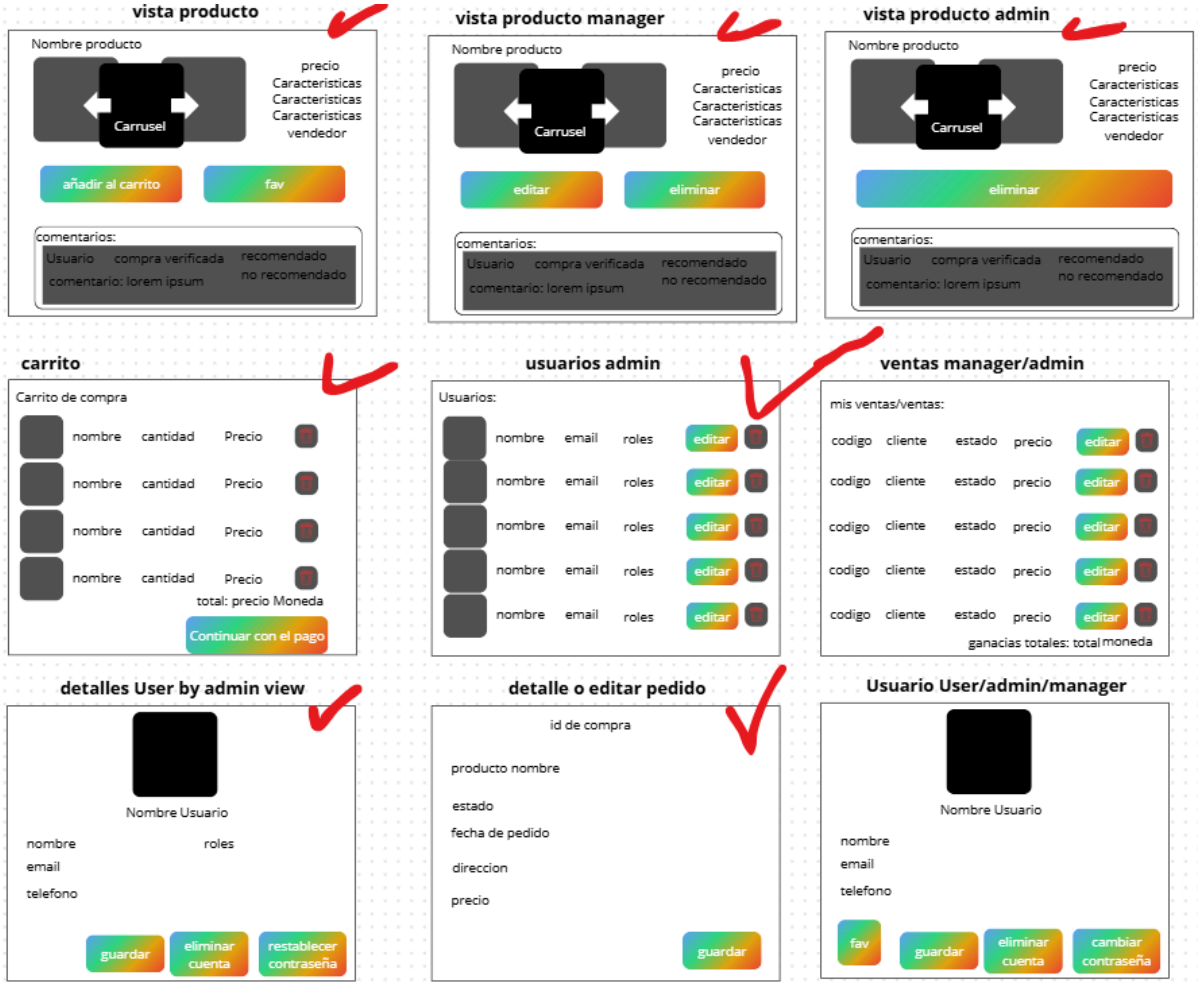
Nombre	**Eliminar Producto**
Id	CU - 19

Descripción	Un Manager retira un producto de la venta.
Actores implicados	Manager.
Precondiciones	Estar logueado y tener permisos sobre el producto.
Pasos / curso normal	1.- El Manager accede a su perfil y pulsa en Mis productos. 2.- El Manager pulsa el botón eliminar. 3.- El sistema solicita confirmación 4.- El producto se elimina.
Postcondiciones	La información del producto se actualiza en la base de datos.
Alternativa	No hay.

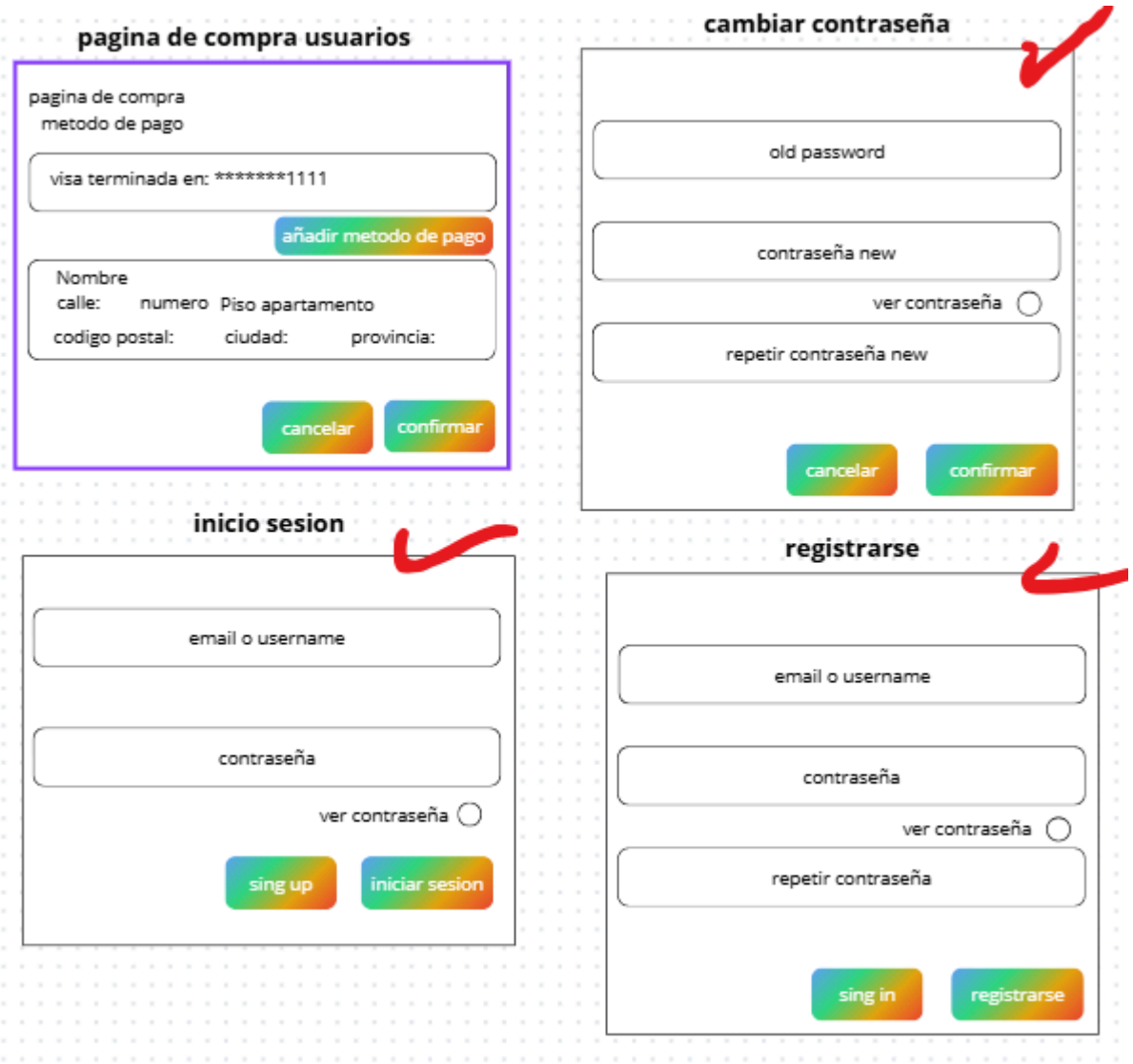
Nombre	**Consultar Historial de Compras**
Id	CU - 20
Descripción	Un usuario consulta el listado de pedidos que ha realizado anteriormente.
Actores implicados	Usuario.
Precondiciones	Usuario logueado y haber realizado al menos una compra.
Pasos / curso normal	2.- El usuario selecciona la opción "Mis pedidos". 3.- El sistema muestra la vista compras user con una lista de códigos, precios y botones de detalles.
Postcondiciones	Ninguna.
Alternativa	No hay.

Boceto de las vistas

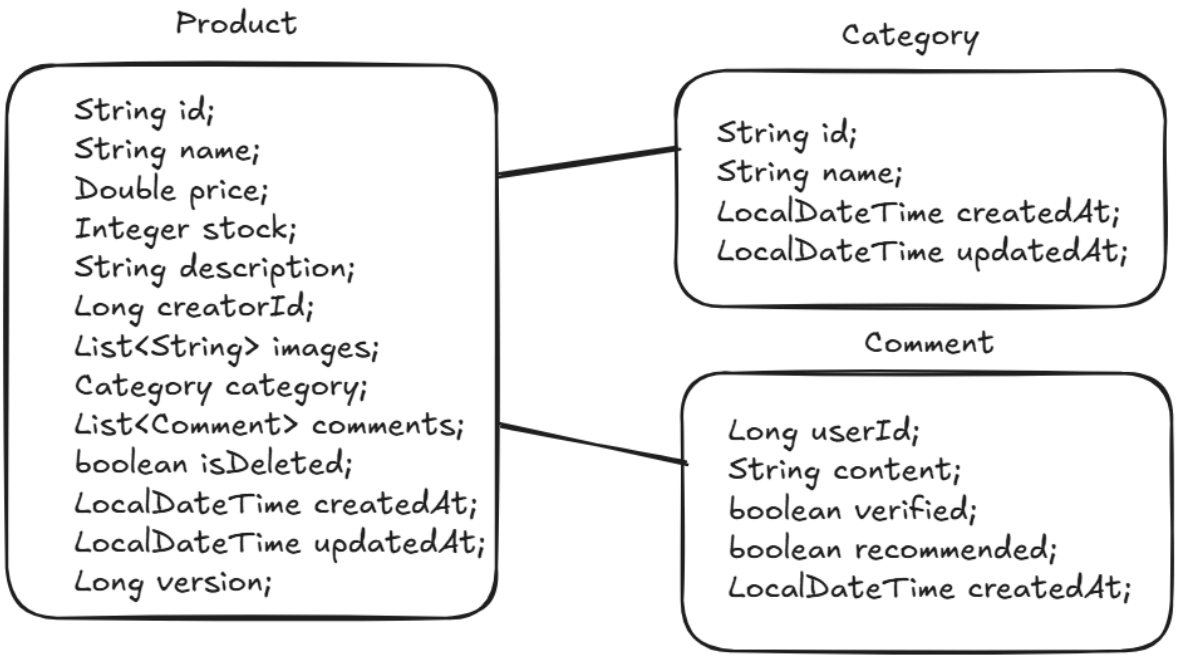

Navbar general
logo
Dawazon
buscador
user
sin extender
Categorías
Muebles
Videojuegos
Libros
extendido
otras más
User navbar
Mis pedidos:
Navbar admin
Usuarios:
Ventas y envios:
Navbar Manager
Mis productos:
Footer
Contacto
Quiénes somos
Legal
copirrie
vista productos general/
favoritos User/
manager productos generales
admin vista
Recomendaciones/favoritos.
mis productos manager
Tus productos
añadir producto

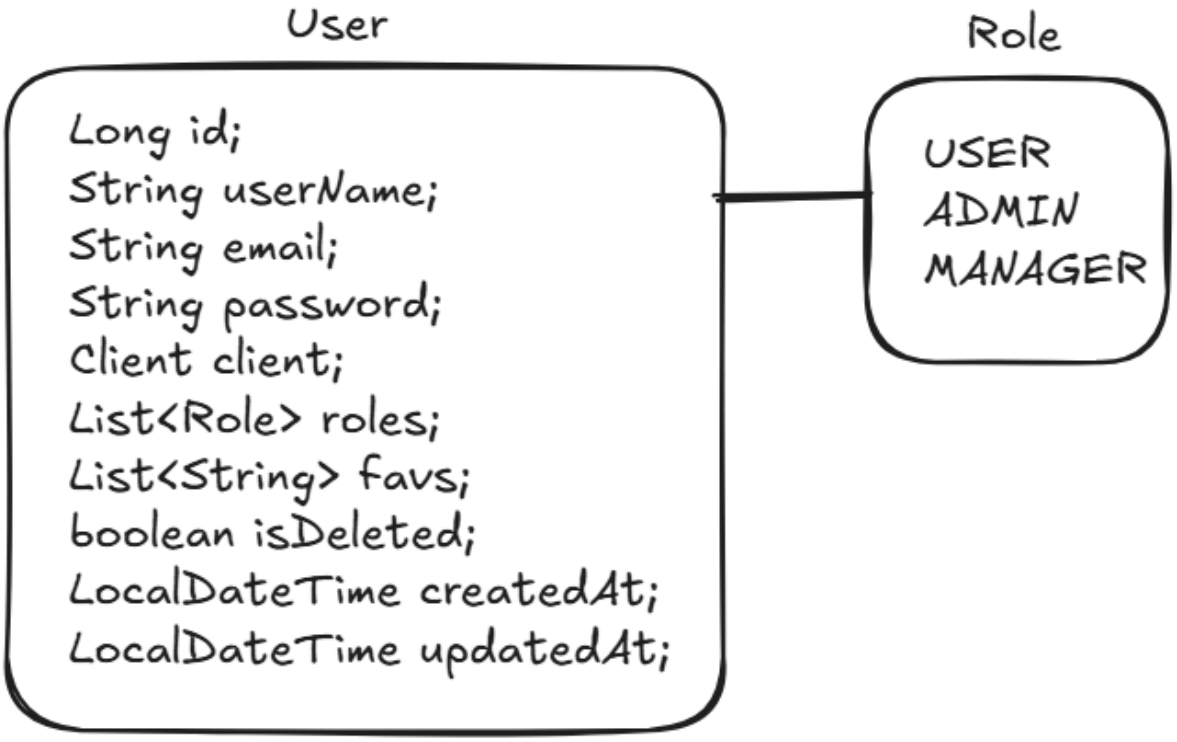

vista producto
Nombre producto
Carrusel
precio
Caracteristicas
Caracteristicas
Caracteristicas
vendedor
añadir al carrito
fav
comentarios:
Usuario
compra verificada
recomendado
no recomendado
comentario: lorem ipsum
vista producto manager
editar
eliminar
vista producto admin
carrito
Carrito de compra
nombre
cantidad
Precio
total: precio Moneda
Continuar con el pago
usuarios admin
Usuarios:
email
roles
ventas manager/admin
mis ventas/ventas:
codigo
cliente
estado
precio
ganancias totales: total moneda
detalles User by admin view
Nombre Usuario
guardar
eliminar cuenta
restablecer contraseña
detalle o editar pedido
id de compra
producto nombre
estado
fecha de pedido
direccion
precio
Usuario User/admin/manager
cambiar contraseña

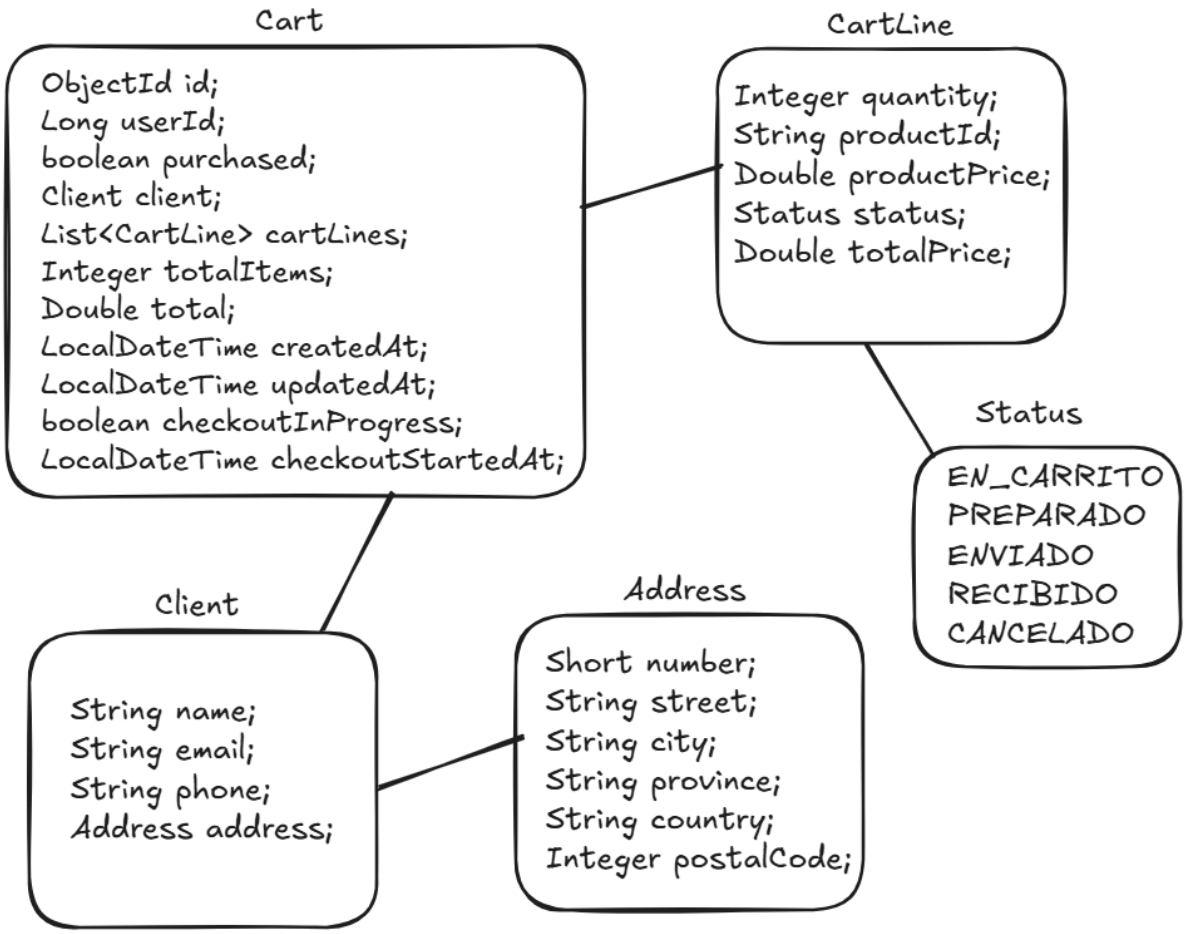

pagina de compra usuarios
añadir tarjeta
pagina de compra
metodo de pago
Nombre en la tarjeta
Número de tarjeta
Fecha de caducidad
ccv
añadir
Nombre
numero
Piso apartamento
codigo postal:
ciudad:
provincia:
cancelar
confirmar
detalles pedido
codigo
nombre
cantidad
estado
Precio
nombre
cantidad
estado
Precio
nombre
cantidad
estado
Precio
nombre
cantidad
estado
Precio
total: precio Moneda
Producto by Manager create
Nombre producto
precio
categoria
descripcion
Imagenes
resto de datos
cancelar
guardar
compras user
mis compras
codigo
precio
detalles
codigo
precio
codigo
precio
detalles
codigo
precio
detalles
codigo
precio
detalles


pagina de compra usuarios
pagina de compra
metodo de pago
visa terminada en: *******1111
añadir metodo de pago
Nombre
calle: numero Piso apartamento
codigo postal: ciudad: provincia:
cancelar
confirmar
cambiar contraseña
old password
contraseña new
ver contraseña
repetir contraseña new
cancelar
confirmar
inicio sesion
email o username
contraseña
ver contraseña
sing up
iniciar sesion
registrarse
email o username
contraseña
ver contraseña
repetir contraseña
sing in
registrarse

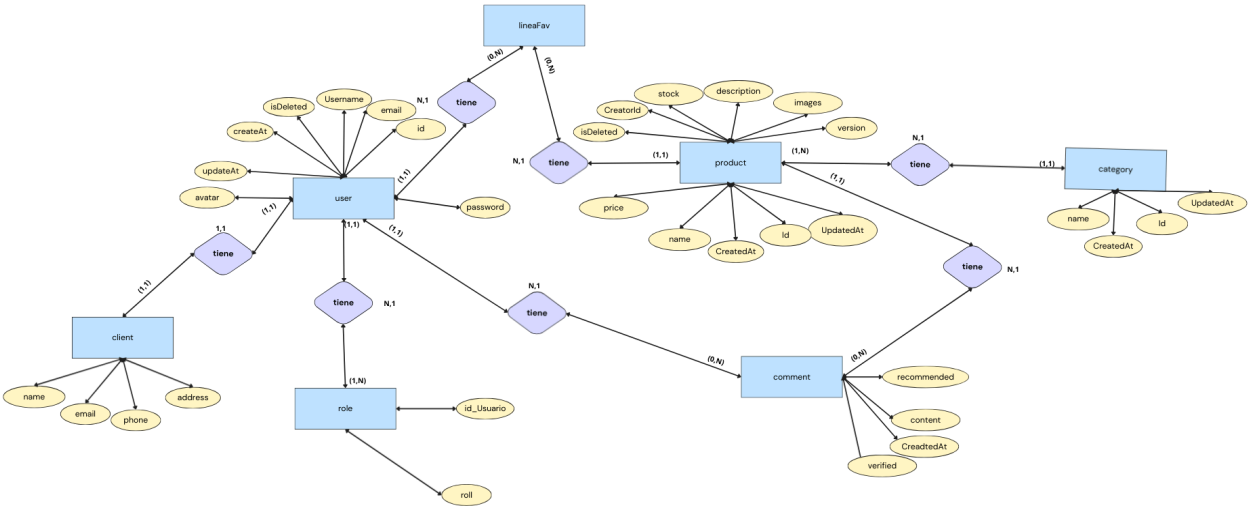
Modelos


Product
String id;
String name;
Double price;
Integer stock;
String description;
Long creatorId;
List<String> images;
Category category;
List<Comment> comments;
boolean isDeleted;
LocalDateTime createdAt;
LocalDateTime updatedAt;
Long version;
Category
String id;
String name;
LocalDateTime createdAt;
LocalDateTime updatedAt;
Comment
Long userId;
String content;
boolean verified;
boolean recommended;
LocalDateTime createdAt;


User
Long id;
String userName;
String email;
String password;
Client client;
List<Role> roles;
List<String> favs;
boolean isDeleted;
LocalDateTime createdAt;
LocalDateTime updatedAt;
Role
USER
ADMIN
MANAGER



Diagramas entidad-relación


lineaFav
(0,N)
(0,N)
tiene
Username
isDeleted
email
N,1
id
createAt
updateAt
avatar
user
(1,1)
password
(1,1)
1,1
tiene
(1,1)
client
name
email
phone
address
tiene
N,1
(1,1)
role
id_Usuario
roll
tiene
N,1
(0,N)
comment
recommended
content
CreadtedAt
verified
tiene
N,1
stock
description
images
CreatorId
isDeleted
version
N,1
tiene
(1,1)
product
price
name
CreatedAt
Id
UpdatedAt
(1,N)
(1,1)
tiene
N,1
(1,1)
category
name
Id
CreatedAt
UpdatedAt

Gitflow


cosas que se me olvidaron meter en el comit anterior Víctor Marín Escribano
feature/integrarProyectoCsharpEnProyectoDeJavaParaTestE2E Víctor Marín Escribano
Merge pull request #34 from charlieecy/dev Víctor Marín Escribano
Añadido el botón de cambiar contraseña y paginación en las listas de usuarios Carlos Cortés
se corrigen todos los comentarios y variables Víctor Marín Escribano
Merge pull request #33 from Sggz221/dev Víctor Marín Escribano
Tests de usuarios completados. Sggz221
Merge pull request #32 from charlieecy/dev Víctor Marín Escribano
Arreglados los tests del repositorio de productos, se hacen los del servicio. Bendisiones. Carlos Cortés
Merge pull request #31 from AdrianHerSac/dev Víctor Marín Escribano
Test hechos del carrito, common y productoRepository, categoryRepository y Mappers AdrianHerSac
Merge pull request #30 from charlieecy/dev Víctor Marín Escribano
Varios bugs de pedidos corregidos, más info en el Pull Request Carlos Cortés
quito imagenes del html de registrarse, cambio contraseñas de usuarios a user y cambio la url del success para que redirija bien, falta corregir error de success Víctor Marín Escribano
arreglado para redirigir a la pagina de pago Víctor Marín Escribano
Merge pull request #29 from charlieecy/dev Víctor Marín Escribano
Implementación del checkout del carrito Carlos Cortés
Merge pull request #28 from charlieecy/dev Víctor Marín Escribano
Solucionados varios bugs del carrito (más info en el pull request) Carlos Cortés
Merge pull request #27 from Sggz221/dev Víctor Marín Escribano
Arreglos menores en diversas funcionalidades Sggz221
Merge pull request #26 from Sggz221/dev Víctor Marín Escribano
Soluciones y bugs corregidos en la parte de manager Sggz221
Merge pull request #25 from charlieecy/dev Carlos Cortés
Solucionado el bug al crear un producto como Manager Carlos Cortés
Merge pull request #24 from Sggz221/dev Samuel Gómez Gutiérrez
Correcciones de admin y edicion de usuarios Sggz221
Merge pull request #23 from charlieecy/dev Víctor Marín Escribano
Merge remote-tracking branch 'origin/dev' into dev Carlos Cortés
corregido la pagina de error y ahora los manager pueden ver sus productos Víctor Marín Escribano
Solucionados varios bugs de Admin (más info en el Pull Request) Carlos Cortés
Merge pull request #22 from charlieecy/dev Víctor Marín Escribano
Documentación de todas las funciones Carlos Cortés
cambios en el readme Víctor Marín Escribano
Merge pull request #21 from Sggz221/dev Samuel Gómez Gutiérrez
hasta aquí hemos llegado Sggz221
Merge pull request #20 from charlieecy/dev Víctor Marín Escribano
Se arreglan bugs de usuarios y admins Carlos Cortés
Merge pull request #19 from Sggz221/dev Samuel Gómez Gutiérrez
lol Sggz221
arreglo carrito Víctor Marín Escribano
no se bro mucho cambio Víctor Marín Escribano
Merge pull request #18 from Sggz221/dev Samuel Gómez Gutiérrez
Algun bug arreglado Sggz221
añado el echedule que limpia los carritos expirados y ajusto algunas funiones para que tiren bien Víctor Marín Escribano
inicio de creacion de controller compra carrito Víctor Marín Escribano
Merge pull request #17 from charlieecy/dev Víctor Marín Escribano
Implementadas casi todas las funciones del controller de usuarios Carlos Cortés
añado el servicio completo de favoritos, realizo correcciones en authController Víctor Marín Escribano
despliegue funcional Víctor Marín Escribano
despliegue terminado Víctor Marín Escribano
añado el proxy y configuraciones del proyecto Víctor Marín Escribano
Merge pull request #16 from AdrianHerSac/dev Víctor Marín Escribano
añado el controller de productos usuario AdrianHerSac
avances en UserController AdrianHerSac
Merge pull request #15 from AdrianHerSac/dev Víctor Marín Escribano
Error 404 AdrianHerSac
Controller de autenticación creado, se arregla plantilla Pebble Víctor Marín Escribano
termino el cartController Víctor Marín Escribano
Merge pull request #14 from charlieecy/dev Víctor Marín Escribano
Implementado el controller de productos Carlos Cortés
Merge pull request #13 from Sggz221/dev Samuel Gómez Gutiérrez
Seguridad implementada Sggz221
Merge pull request #12 from Sggz221/dev Víctor Marín Escribano
Servicio de carrito terminado Sggz221
continuo el servicio Víctor Marín Escribano
Merge pull request #11 from Sggz221/dev Víctor Marín Escribano
Avance en el carrito Sggz221
Merge pull request #10 from charlieecy/dev Víctor Marín Escribano
Implementado todo lo relativo a productos y categorías Carlos Cortés
añadir includes para el correcto funcionamiento, cambios en la pagina de error Víctor Marín Escribano
añado repositorios sin test Víctor Marín Escribano
Merge branch 'templates' of https://github.com/Aragon7372/dawazon into templates Víctor Marín Escribano
Merge pull request #9 from Sggz221/templates Víctor Marín Escribano
Vista de detalles de pedido y creación/edición de producto Samuel Gómez
Merge pull request #8 from AdrianHerSac/templates Víctor Marín Escribano
Vista editar pedidos AdrianHerSac
Merge pull request #7 from charlieecy/templates Víctor Marín Escribano
Modificada la vista editUserAdmin para que sirva para usuarios, managers y admins Carlos Cortés
añado la pagina de ventas Víctor Marín Escribano
añado vista de usuarios por administrador, modifico listas y hago funcional el search bar del navbar Víctor Marín Escribano
Merge pull request #6 from AdrianHerSac/templates Víctor Marín Escribano
Vista editar usuario por admin y renombrar cart AdrianHerSac
Merge pull request #5 from Sggz221/templates Samuel Gómez Gutiérrez
Vista de pedido en detalles. Samuel Gómez
Merge pull request #4 from charlieecy/templates Víctor Marín Escribano
Vista de la lista de pedidos de un usuario creada Carlos Cortés
añado la vista de los productos propios del manager Víctor Marín Escribano
Merge remote-tracking branch 'origin/templates' into templates Víctor Marín Escribano
Merge pull request #3 from Sggz221/templates Víctor Marín Escribano
Plantilla carrito Sggz221
Merge pull request #2 from charlieecy/templates Víctor Marín Escribano
Creadas plantillas de login, registro y cambio de contraseña Carlos Cortés
termino vista producto y vista lista de productos Víctor Marín Escribano
Merge pull request #1 from Sggz221/templates Samuel Gómez Gutiérrez
Modelos preparados Sggz221
updates templates Víctor Marín Escribano
añado el footer generico, header generico y navbar generico. Al igual que varios iconos y imagenes default y la pagina default para visualizar productos (incompleta por ahora) Víctor Marín Escribano
error page Víctor Marín Escribano
readme con sus cosas añadido, mas pagina error dinamica Víctor Marín Escribano
añado, todos el proyecto inicial Víctor Marín Escribano
Initial commit Víctor Marín Escribano

Estimación de costes

HORAS DE TRABAJO ESTIMADAS		
Requisitos funcionales		
RF	Nombre	Horas estimadas
RF1	Operaciones CRUD (productos, categorías y pedidos).	40
RF2	Carrito de compra.	40
RF3	Id único a cada entidad.	1
RF4	API REST.	10
RF5	Interfaz gráfica dinámica (Pebble).	120
RF6	Pago con tarjeta (Stripe).	5
RF7	Usuarios, seguridad (Spring Security) y HTTP Session.	40
RF8	Tarea programada.	3
		259

Requisitos no funcionales		
RNF	Nombre	Horas estimadas
RNF1	Caché Redis.	5
RNF2	Testeo del código (unitarios + integración con test container + E2E Playwrigt)	40
RNF3	Docker.	20
RNF4	GitHub.	5
RNF5	Proxy inverso con Nginx.	2
		72

Requisitos de información		
RI	Nombre	Horas estimadas
RI1	Modelo + DTOs de usuario	1
RI2	Modelo + DTOs de cliente	1
RI3	Modelo + DTOs de producto	1
RI4	Modelo de categoría	1
RI5	Modelo + DTOs de pedido	1
RI6	Modelo + DTOs de comentario	1
RI7	Modelo + DTOs de carrito	1
		7

HORAS DE TRABAJO TOTALES
338

PRECIO POR HORA DE TRABAJO
135,00 €

PRECIO POR HORA EXTRA DE TRABAJO
160,00 €

COSTE TOTAL DE LAS HORAS DE TRABAJO
45.630,00 €

ESTIMACIÓN DE COSTES				
Nº	Concepto	Desglose		Coste
		Precio por hora	Horas	
1	Horas de trabajo estimadas por requisitos	135,00 €	338	45.630,00 €
		Precio por hora extra	15% de las horas totales	
2	Horas extra para cubrir imprevistos	160,00 €	50,70	8.112,00 €
		Costes antes de margen	17% sobre los costes	
3	Margen de beneficios	53.742,00 €	0,17	9.136,14 €
				62.878,14 €

Conclusiones

El desarrollo de Dawazon ha permitido consolidar los conocimientos avanzados de programación en el lado del servidor, resultando en una plataforma de comercio electrónico funcional, robusta y preparada para un entorno de producción.

Como cierre del proyecto, se destacan las siguientes conclusiones técnicas y operativas:

- Arquitectura Sólida y Mantenible: La implementación del patrón MVC (Modelo-Vista-Controlador) con Spring Boot y el motor de plantillas Pebble para aplicaciones web dinámicas.

- Persistencia Híbrida Eficiente: La estrategia de almacenamiento ha sido un éxito. La combinación de PostgreSQL para la integridad referencial del catálogo y usuarios, junto con MongoDB para la inmutabilidad histórica de los pedidos, ofrece lo mejor de ambos mundos: seguridad transaccional y flexibilidad documental.

- Despliegue y Portabilidad: La contenerización integral mediante Dockerfile y Docker Compose ha eliminado las discrepancias entre entornos. La infraestructura diseñada (Base de datos, Caché Redis, Servidor Web y Proxy Inverso) permite desplegar el sistema completo con un solo comando, cumpliendo con los estándares modernos.

- Rendimiento y Experiencia de Usuario: La gestión de sesiones y el uso de Redis como caché han optimizado significativamente los tiempos de respuesta. El renderizado en servidor (SSR) asegura que el contenido llegue rápido al cliente, mejorando tanto la experiencia de navegación como el posicionamiento SEO.

- Calidad del Software: La integración de herramientas de calidad como JUnit para pruebas automatizadas y JaCoCo para el análisis de cobertura, documentación técnica exhaustiva con Dokka, asegura que el proyecto es fiable y auditable.

En definitiva, Dawazon cumple con todos los objetivos planteados, demostrando cómo la integración de tecnologías punteras (Java 25, Spring Boot, bases de datos NoSQL) puede dar lugar a soluciones de software escalables y de nivel profesional.

Enlaces

Imagenes, trello, entidad-relacion y diseño-interfaces