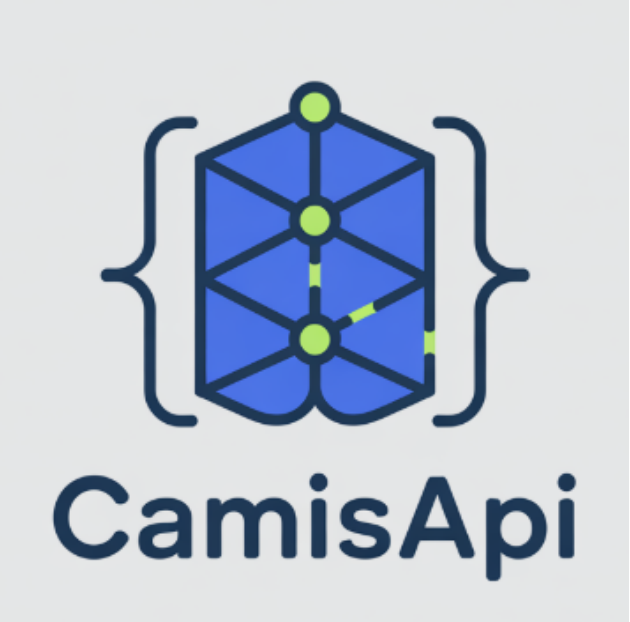
CamisAPI



	I.E.S LUIS VIVES - Grupo 2º DAW CamisApi	Curso: 2025-2026 Fecha: 10/11/2025
Creado Por	- Marcos Jodar - Angel Sanchez - Cristian Ortega - Jorge Morgado	
Desarrollo de aplicaciones Web → Proyecto Spring Boot		Profesor: Jose Luis

ÍNDICE

DESCRIPCIÓN DEL PROYECTO

CamisApi es una aplicación diseñada como una tienda online para la venta de camisetas de futbol. Muestra a los usuarios interesados en el mundo del fútbol un catálogo de camisetas únicas, donde pueden seleccionar las que más les guste y comprarlas. Primero la añadirían al carrito de la compra, para más tarde realizar la compra. Lo novedoso de esta tienda es que todas las camisetas que se venden son únicas y exclusivas, permitiendo administrar la reserva y venta de cada camiseta, no hay otra en todo el mundo. Así que solo puede existir una unidad disponible, evitando que dos personas quieran comprar el mismo producto.

Esta aplicación también cuenta con un proceso de pedidos, donde el usuario puede elegir una de las camisetas, elegir si comprarla o no y recibir el producto. Así nuestros clientes quedarán más satisfechos.

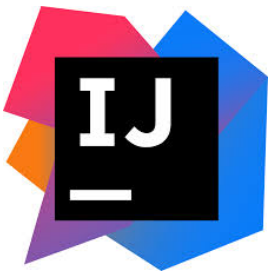
Nuestro objetivo es convertirnos en una tienda mundialmente reconocida por nuestras camisetas exclusivas y únicas, pudiendo vender en cualquier lado del planeta, incluso a sitios más complicados de acceder como partes de África y Asia. çnosotros nos dirigimos a el publico que le guste el fútbol, a casi todo el mundo le gusta, sean niños ancianos, adultos, hombre, mujeres, a prácticamente todo el mundo le gusta el futbol.

TECNOLOGÍAS UTILIZADAS

Esta sección abarca todos los componentes esenciales para la creación de la aplicación, incluyendo tanto el lado del servidor (backend) como el lado del cliente (frontend), que también se ha añadido, junto con las tecnologías utilizadas para su implementación. Además, también se detallan bibliotecas empleadas para optimizar el desarrollo y las bases de datos seleccionadas, para gestionar correctamente los datos del sistema.

TECNOLOGÍA

Nuestro proyecto ha sido creado con **IntelliJ IDEA**, un programa informático, denominado Entorno de Desarrollo Integrado (IDE).



IntelliJ fue creado por **JetBrains**, enfocado en desarrolladores que trabajan con el lenguaje de programación **Java** con el objetivo principalmente de proporcionar un ambiente completo para escribir, compilar, depurar y desplegar aplicaciones software. Aunque también es

compatible con otros lenguajes como Kotlin, Groovy, Scala, JavaScript, TypeScript y más, proporcionando herramientas y características específicas para cada uno.
IntelliJ IDEA ofrece una interfaz de usuario intuitiva y muy personalizable, facilitando la navegación y la escritura de código.

FRAMEWORK JAVA

Spring Boot version 3.5.7
Spring Boot es un proyecto que se basa en el Spring Framework y simplifica el proceso de configuración y ejecución de aplicaciones Spring. Las características clave de Spring Boot incluyen:

Autoconfiguración: Spring Boot puede configurar automáticamente una aplicación basada en las dependencias que se han agregado al proyecto.

Standalone: Spring Boot permite crear aplicaciones independientes con un servidor embebido, lo que significa que no necesitas un servidor web o de aplicaciones separado.

Opinión predefinida: Spring Boot tiene una opinión predefinida para configurar proyectos Spring, aunque permite a los desarrolladores modificar la configuración para satisfacer sus necesidades.

Dependencias de inicio: Proporciona starters que son un conjunto de dependencias convenientes que simplifican la configuración de la aplicación.

Actuator: Proporciona funcionalidades de producción listas para usar, como la monitorización y la gestión de la aplicación.

Pruebas: Spring Boot proporciona soporte para pruebas con Spring Boot Test Starter, lo que facilita la escritura de pruebas para las aplicaciones Spring Boot.

Se integra con otros proyectos Spring, como Spring Data, Spring Security y Spring Cloud. Así facilita la creación de apps con variedad de servicios de Spring.

SEGURIDAD

Spring security con JWT version 4.4.0
Spring Security es un framework de seguridad de aplicaciones para aplicaciones Java basadas en Spring Framework. Proporciona una capa de seguridad a nivel de aplicación que se integra fácilmente con otras tecnologías de Spring, como Spring MVC, Spring Boot y Spring Data.

La seguridad es un aspecto crítico en el desarrollo de aplicaciones web, y Spring Security ofrece una amplia gama de características y funcionalidades para proteger las aplicaciones contra amenazas y ataques.
Características:
Autenticación: Permite autenticar a los usuarios en la aplicación, verificando sus credenciales, como nombre de usuario y contraseña.

Autorización: Permite controlar el acceso a diferentes partes de la aplicación en función del rol y de los permisos de los usuarios autenticados.

Protección contra ataques: Proporciona protección integrada contra ataques de inyección SQL, ataques de secuencias de comandos entre sitios (XSS), ataques de falsificación de solicitudes entre sitios (CSRF) y muchos más.

Gestión de sesiones: Gestiona las sesiones de usuario.

Integración con otras tecnologías de Spring: Se integra sin problemas con Spring MVC, Spring Boot y Spring Data.

En nuestra aplicación CamisApi Spring Security y JWT han jugado un papel muy importante, protegiendo la API y controlando los accesos a los distintos endpoints. Para ser más específicos; se valida que el usuario exista y que sus credenciales son correctas. También es el encargado de autorizar los roles de los usuarios que entran a la app, ya sea cliente o administrador.

PERSISTENCIA

JPA (Java Persistence API) proporciona una forma de gestionar la persistencia de datos en aplicaciones Java. Permite mapear objetos Java a tablas de bases de datos relacionales y gestionar los datos sin tener que escribir SQL directamente.
En nuestra aplicación se usó para gestionar la persistencia de datos en PostgreSQL, la base de datos relacional:

- Mapeo de entidades Java a tablas de la BD con anotaciones
- Escribir los métodos CRUD sin tener que escribir SQL, como save, findById, delete, etc.
- También nos ha permitido crear consultas personalizadas con `@Query`

BACK-END

El back-end es la parte del sistema que maneja la lógica de nuestra aplicación, las operaciones del servidor y la gestión de bases de datos. Es fundamental para el procesamiento de datos, la seguridad y la eficiencia de una aplicación.

Lenguajes de programación

Java version: 25
¿Por qué elegir JAVA para la realización del proyecto?

Java es uno de los lenguajes de programación más utilizados en el mundo y más demandados en el mercado laboral, debido a su simplicidad, la orientación a objetos, y su sintaxis clara, ayudando a entender conceptos de programación importantes que son aplicables a otros lenguajes como Kotlin, Scala o Groovy. Estos lenguajes son más avanzados, pero conociendo Java, se entienden mejor sus diferencias y ventajas.

Herramientas de Gestión de Dependencias: *Gradle*

Herramienta de construcción para proyectos software, donde su función principal es compilar, construir, probar y empaquetar aplicaciones, además de gestionar dependencias externas.

COMUNICACIÓN, PROTOCOLOS Y APIS

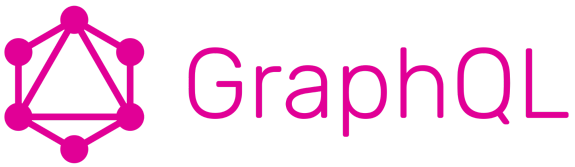
Tecnologías o arquitecturas para intercambio de información entre cliente y servidor:

Una API REST (Representational State Transfer) es un **estilo de arquitectura** de software para el desarrollo de aplicaciones web. REST se basa en principios y estándares que permiten **construir interfaces de programación de aplicaciones** (API) fácilmente.

Utiliza métodos HTTP estándar, como GET, POST, DELETE y PUT, para realizar operaciones en los recursos. Los recursos, que son cualquier tipo de objeto, dato o servicio que se quiera proporcionar a través de la API, se identifican a través de URLs.

En nuestra aplicación es muy utilizada, por ejemplo, se utiliza para obtener el catálogo de productos, añadir items al carrito y crear pedidos. Cada acción del usuario se traduce en una solicitud REST que actualiza la información en la base de datos.

Websocket es un protocolo de comunicación bidireccional en tiempo real que se establece **entre un cliente y un servidor** a través de una **conexión TCP** (Transmission Control Protocol). Permiten una comunicación persistente y bidireccional entre el cliente y el servidor.
Una vez que se establece la conexión WebSocket, tanto el cliente como el servidor pueden enviar mensajes en cualquier momento sin necesidad de esperar una solicitud explícita.
Como se hace en nuestra aplicación, …



GraphQL es un lenguaje de consulta para APIs y un entorno de ejecución perfecto para tus datos, ya que envía los justos y necesarios.

En CamisApi la hemos utilizado exclusivamente en el carrito de la compra para crear peticiones personalizadas por parte del cliente.

BASES DE DATOS

Sistemas de almacenamiento y gestión de datos:

MongoDB es un sistema de base de datos NoSQL basado en documentos. MongoDB almacena los datos en documentos BSON, que es una representación binaria de JSON.
En nuestra aplicación se utiliza para el almacenamiento del catálogo de camisetas. Gracias a su modelo orientado a documentos, permite manejar los productos exclusivos sin

necesidad de un esquema rígido, y hace posible incorporar cambios en sus atributos sin costar mucho.

Sistema de gestión de bases de datos relacional y de código abierto que combina las características de las bases de datos SQL con funciones avanzadas como soporte para objetos, JSON y extensiones

Funciona como la base de datos principal para la parte transaccional del sistema, gestionando *usuarios*, *carritos* y *pedidos*. Al ser una base de datos relacional, garantiza la coherencia e integridad en los procesos como compras, reservas y pagos.

Redis (Remote Dictionary Server) es una base de datos NoSQL en memoria de código abierto que funciona como almacén de estructuras de datos clave-valor. Es extremadamente rápida ya que mantiene todos los datos en memoria RAM.

Base de datos en memoria (o en archivo) usado en Java, sobre todo con Spring Boot, para desarrollar o probar proyectos sin instalar una base de datos real.
Ventajas:

- Ligera y rápida
- No requiere instalación
- Ideal para pruebas y aprendizaje
- Se integra fácilmente con JPA o JDBC

TESTING

El testing es una parte fundamental en el desarrollo de software. Nos permite asegurar que nuestro código funciona correctamente y que no se rompe cuando hacemos cambios en él.
Para realizar los test unitarios con **Spring Boot** hemos utilizado **JUnit 5** y **Mockito**.
Añadiendo la dependencia de Starter de Test en nuestro proyecto:

En nuestra aplicación, JUnit y Mockito se han utilizado para garantizar el funcionamiento de la lógica de negocio y los componentes de la API:

- JUnit: Hemos creado y ejecutado tests unitarios estructuradamente. Con JUnit hemos comprobado que los métodos de los servicios y controladores devuelvan los resultados esperados ante distintos escenarios. Como en métodos como: crear un pedido, actualizar su estado o eliminarlo.

- Mockito: En cambio, con Mockito, nos ha permitido simular dependencias externas (como repositorios o mappers) sin necesidad de conectarse a la base de datos o ejecutar toda la aplicación. Esto facilita probar solo la lógica del servicio o controlador de forma aislada.

TestContainers version: 1.19.3
Biblioteca de que proporciona **contenedores Docker ligeros** y desechables para **pruebas automatizadas**. Permite a los desarrolladores ejecutar pruebas de integración que dependen de servicios externos como bases de datos, colas de mensajes, navegadores web y más, en un entorno controlado y replicable.

API DOCS

Swagger es una herramienta que nos permite generar documentación de APIs REST a partir de la especificación OpenAPI. **OpenAPI version: 2.3.0**
En nuestra aplicación esta parte no funciona correctamente.

Arquitectura y Diseño

En esta sección se describe la estructura general de CamisAPI, explicando cómo los distintos componentes de la aplicación interactúan entre sí y el flujo de información entre usuarios, productos, carritos y pedidos.

CamisAPI implementa una **arquitectura híbrida de persistencia** basada en dos motores de base de datos con el objetivo de optimizar el rendimiento, escalabilidad y consistencia de los datos según su naturaleza: MongoDB y PostgreSQL

ARQUITECTURA

Modelos en MongoDB

Producto (Producto.java)
Colección: productos
Ubicación: src/main/java/srangeldev/camisapi/rest/productos/models/Producto.java
Cada camiseta es única. No existe concepto de "stock" ni unidades múltiples.

Campo	Tipo	Descripción
id	String	ObjectId de MongoDB
nombre	String	Nombre del producto
equipo	String	Nombre del club/selección
talla	String	Talla (S, M, L, XL, XXL)
descripcion	String	Descripción detallada
precio	Double	Precio en euros
imageUrl	String	URL de la imagen
estado	EstadoProducto	DISPONIBLE, RESERVADO, VENDIDO
fechaCreacion	LocalDateTime	Fecha de alta en catálogo

Estados del Producto (EstadoProducto.java)

```
public enum EstadoProducto {
    DISPONIBLE,  // Puede ser añadido al carrito
    RESERVADO,   // Está en un carrito (temporal)
    VENDIDO      // Comprado, no disponible
}
```

Flujo de estados:

1. DISPONIBLE → Usuario añade al carrito → RESERVADO
2. RESERVADO → Usuario elimina o expira → DISPONIBLE
3. RESERVADO → Usuario completa compra → VENDIDO

Modelos en PostgreSQL

User (User.java)
Tabla: users
Ubicación: src/main/java/srangeldev/camisapi/rest/users/models/User.java
Representa a cada usuario/cliente del sistema.

Campo	Tipo	Descripción
id	Long	Primary Key autogenerada
nombre	String	Nombre completo del usuario
username	String	Nombre de usuario único
password	String	Contraseña cifrada
roles	Set<Rol>	Roles del usuario (USER, ADMIN)
createdAt	LocalDateTime	Fecha de creación
updatedAt	LocalDateTime	Fecha de última actualización
isDeleted	Boolean	Marcado de borrado lógico

Carrito (Carrito.java)
Tabla: carritos
Ubicación: src/main/java/srangeldev/camisapi/rest/carrito/models/Carrito.java
Almacena productos temporales del usuario.

Campo	Tipo	Descripción
id	Long	Primary Key (autoincremental)
userId	Long	Referencia al id del User
items	List<String>	Lista de IDs (String) de Productos
modificadoEn	LocalDateTime	Última modificación
creadoEn	LocalDateTime	Fecha de creación

Comportamiento:

- Al añadir un producto → cambiar estado a RESERVADO en MongoDB
- Al eliminar o expirar → cambiar estado a DISPONIBLE en MongoDB

Pedido (Pedido.java)
Tabla: pedidos
Ubicación: src/main/java/srangeldev/camisapi/rest/pedidos/models/Pedido.java
Representa una compra confirmada con snapshot de productos.

Campo	Tipo	Descripción
id	Long	Primary Key (autoincremental)
userId	Long	Referencia al id del User
estado	EstadoPedido	Estado del pedido
fechaCreacion	LocalDateTime	Fecha de creación
total	Double	Total del pedido en euros
detalles	List<DetallePedido>	Snapshot de productos vendidos
direccionEnvio	String	Dirección de entrega (opcional)
fechaPago	LocalDateTime	Fecha del pago (opcional)
fechaEnvio	LocalDateTime	Fecha de envío (opcional)
numeroSeguimiento	String	Número de tracking (opcional)

DetallePedido (DetallePedido.java)
Embebido en: Pedido
Tabla auxiliar: pedido_detalles
Snapshot inmutable del producto en el momento de la venta.

Campo	Tipo	Descripción
productoId	String	Referencia al ObjectId del Producto
nombre	String	Nombre del producto (snapshot)
talla	String	Talla (snapshot)
equipo	String	Equipo (snapshot)
precioPagado	Double	Precio final pagado
imageUrl	String	URL de la imagen (snapshot)

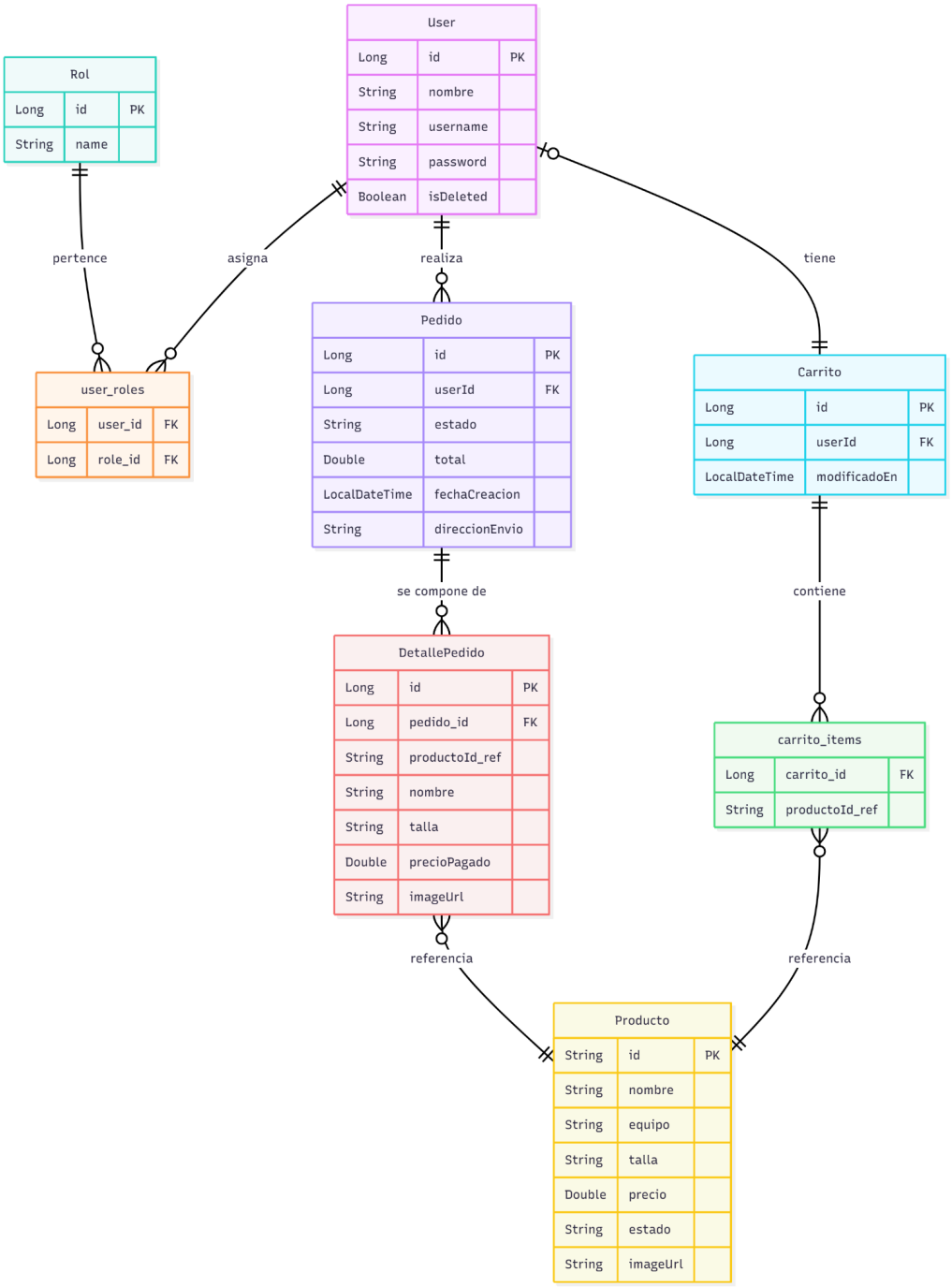
Estados del Pedido (EstadoPedido.java)

```
public enum EstadoPedido {
    PENDIENTE_PAGO,  // Esperando confirmación de pago
```

```
    PAGADO,         // Pago confirmado
    ENVIADO,        // En tránsito
    ENTREGADO,      // Entregado al cliente
    CANCELADO       // Pedido cancelado
}
```

MongoDB (Productos)	PostgreSQL (Users, Carritos y Pedidos)
VENTAJAS	**VENTAJAS**
❖ Flexibilidad para cambios en el catálogo ❖ Búsquedas rápidas por equipo, talla, etc. ❖ Escalabilidad horizontal ❖ No requiere migraciones para nuevos campos	❖ Transacciones ACID para compras ❖ Consistencia en operaciones críticas ❖ Integridad referencial entre usuarios, carritos y pedidos ❖ Histórico inmutable de pedidos ❖ Gestión segura de autenticación y roles

DIAGRAMA DE CLASES



FLUJO DE FUNCIONAMIENTO

1 ========== Usuario añade producto al carrito ==========

1. Usuario selecciona producto (estado: DISPONIBLE)
2. Sistema verifica disponibilidad en MongoDB
3. Crear/Actualizar Carrito en PostgreSQL con productoId
4. Cambiar estado del producto a RESERVADO en MongoDB

2 ========== Usuario completa la compra ==========

1. Usuario confirma carrito
2. Sistema crea Pedido en PostgreSQL
3. Para cada producto en carrito:
    a. Leer datos completos del producto en MongoDB
    b. Crear DetallePedido con snapshot de datos
    c. Cambiar estado del producto a VENDIDO en MongoDB
4. Eliminar Carrito en PostgreSQL
5. Establecer estado del pedido como PENDIENTE_PAGO

3 ========== Usuario abandona el carrito ==========

1. Job programado busca carritos antiguos
2. Para cada producto en carrito expirado:
    a. Cambiar estado a DISPONIBLE en MongoDB
3. Eliminar Carrito en PostgreSQL

CONFIGURACIÓN

VARIABLES DE ENTORNO

Variables necesarias para configurar la aplicación, como credenciales de bases de datos, puertos, claves de seguridad y parámetros de conexión a servicios externos. Es decir, **valores que necesita la app.**

CamisApi utiliza **Spring Profiles** para manejar configuraciones según el entorno. Hay cuatro archivos de configuración principales:

1. **`application.properties`** → configuración general de la aplicación: Nombre de la app, versión de API, JPA y cache
   Ejemplos:

```
spring.application.name=CamisApi
api.version=/api/v1
spring.jpa.hibernate.ddl-auto=update
spring.cache.type=redis
```

2. **`application-dev.properties`** → configuración para desarrollo local, es decir, PostgreSQL, MongoDB y Redis locales
   Ejemplos:

```
spring.datasource.url=jdbc:postgresql://localhost:5432/camisapi
spring.datasource.username=postgres
spring.datasource.driver-class-name=org.postgresql.Driver
```

3. **`application-docker.properties`** → configuración para entornos en contenedor Docker: igual que el anterior pero con contenedor Docker.

4. **`application-test.properties`** → configuración para tests unitarios y automaticos con H2 y deshabilitando servicios externos como MongoDB y Redis.
   Ejemplos:

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
# Redis - Deshabilitado para tests
spring.cache.type=none
```

CONFIGURACIÓN DEL SERVIDOR

Esta sección describe cómo preparar el entorno necesario para ejecutar la app CamisApi tanto en desarrollo como en producción. Es decir, **cómo preparar la máquina para correr la app.**

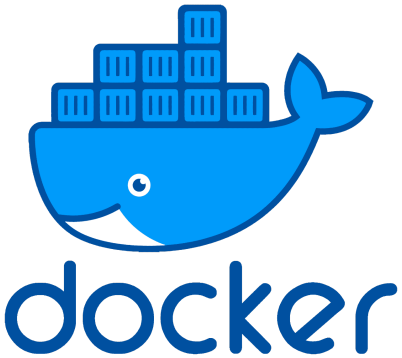
REQUISITOS PREVIOS PARA EJECUTAR LA APP EN PRODUCCIÓN:
JDK → Preferible Windows
Motor SQL → PostgreQSL 14
Motor NoSQL → MongoDB 5
Cache → Redis 6

INICIAR APLICACIÓN en producción:

OPCIÓN 1: Con IntelliJ
1. Abrir el proyecto
2. Seleccionar perfil dev
3. Ejecutar con el botón *Run*

OPCIÓN 2: Con Docker
1. Asegurar que docker esté instalado
2. Ejecutar con Docker compose
3. Añadir donde estará disponible: `http://localhost:8080/api/v1`

DESPLIEGUE



Uso de la API

COMO ES UNA API

USO DE LA API REST
Ahora documentaremos el uso de la API REST en nuestro proyecto.
La API se encuentra dividida en distintos módulos: Carrito, Pedidos, Productos y Usuarios.
Todas las rutas utilizan **formato JSON** tanto en el cuerpo de la petición como en la respuesta.
Autenticación y Seguridad
Dependiendo del módulo, algunos endpoints requieren estar autenticado y otros pertenecer al rol ADMIN, como ocurre en UserController donde se usa @PreAuthorize("hasRole('ADMIN')").

CARRITO

Método	Endpoint	Descripción	Respuesta
GET	/carritos	Obtiene todos los carritos	200 OK
GET	/carritos/{id}	Obtiene un carrito por ID	200 OK / 404
GET	/carritos/usuario/{userId}	Obtiene el carrito asociado a un usuario	200 OK / 404
POST	/carritos	Crea un carrito nuevo	201 Created
PUT	/carritos/{id}	Modifica un carrito	200 OK / 404
DELETE	/carritos/{id}	Elimina un carrito	200 OK / 404

PEDIDOS

Método	Endpoint	Descripción	Respuesta

POST	/api/pedidos	Genera un nuevo pedido a partir del carrito	201 Created
GET	/api/pedidos	Lista todos los pedidos	200 OK
GET	/api/pedidos/{id}	Obtiene un pedido específico	200 OK / 404
GET	/api/pedidos/usuario/{userId}	Obtiene pedidos por ID de usuario	200 OK
GET	/api/pedidos/estado?estado={ENUM}	Filtra pedidos por estado	200 OK
PATCH	/api/pedidos/{id}/estado	Actualiza el estado del pedido	200 OK
DELETE	/api/pedidos/{id}	Elimina un pedido existente	204 No Content

PRODUCTOS

Método	Endpoint	Descripción	Respuesta
POST	/api/productos	Crea un producto	201 Created / 400
GET	/api/productos	Lista productos	200 OK
GET	/api/productos/{id}	Obtiene un producto por ID	200 OK / 404
PUT	/api/productos/{id}	Actualiza producto	200 OK / 400 / 404
DELETE	/api/productos/{id}	Elimina producto	204 No Content / 404
GET	/api/productos/buscar/nombre?nombre=x	Filtra por nombre	200 OK
GET	/api/productos/buscar/equipo?equipo=x	Filtra por equipo	200 OK
GET	/api/productos/buscar/estado?estado=x	Filtra por estado	200 OK
GET	/api/productos/buscar/talla?talla=x	Filtra por talla	200 OK

USUARIOS

Método	Endpoint	Descripción	Respuesta

GET	/api/v1/users	Obtiene todos los usuarios	200 OK
GET	/api/v1/users/{id}	Obtiene usuario por ID	200 OK / 400 / 404
GET	/api/v1/users/nombre/{nombre}	Filtra por nombre	200 OK
POST	/api/v1/users	Crea usuario	201 Created / 400
PUT	/api/v1/users/{id}	Actualiza usuario	200 OK / 400 / 404
DELETE	/api/v1/users/{id}	Elimina usuario	204 No Content / 400 / 404

Códigos de error comunes:
400 → Parámetros inválidos o violación de validaciones
401 → No autorizado
403 → Permiso denegado (falta rol ADMIN)
404 → Recurso no encontrado
409 → Conflicto de estado o duplicados
500 → Error interno no controlado

Nuestra API WebSocket permite comunicación bidireccional en tiempo real entre clientes y el servidor. Se utiliza para notificar cambios relacionados con productos, como por ejemplo nuevos productos, actualizaciones de estado, reservas o ventas.

URL de conexión WebSocket
`ws://<host>:<puerto>/ws/<api.version>/productos`

Con la configuración actual y versión por defecto:
`ws://localhost:8080/ws/v1/productos`

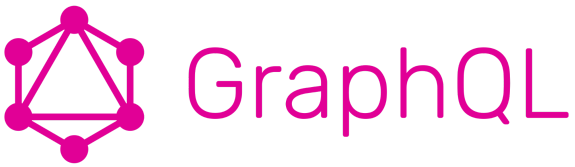
FUNCIONAMIENTO

1. El cliente se conecta mediante WebSocket a la ruta.
2. El servidor agrega la sesión al pool (sessions).
3. El cliente puede enviar mensajes en formato texto.
4. El servidor recibe el mensaje y lo reenvía a todos los clientes conectados (broadcast).
5. El servidor también puede emitir mensajes desde la lógica de negocio llamando a broadcastObject(obj) para enviarlo como JSON.

Política de CORS
Actualmente permite todos los orígenes (`.setAllowedOriginPatterns("*")`), por lo que no hay restricciones entre frontends o dominios mientras usen el mismo endpoint.

CASOS DE USO

Caso	Ejemplo	Beneficio
Notificación de producto nuevo	Nueva camiseta añadida	Actualización instantánea en interfaces
Cambio de estado	DISPONIBLE → RESERVADO	Evitar compras dobles
Chat o mensajes entre clientes	"¿Tienes talla L?"	Canal de comunicación
Actualizaciones del carrito	Productos añadidos o retirados	Vista sincronizada



Se a integrado GraphQL en el Carrito
Además de la API REST, se ha implementado GraphQL para consumir datos del carrito de mejor manera, permitiendo obtener únicamente los campos necesarios en una única operación y facilitando futuras integraciones.

URL Base del Endpoint GrapgQL: `POST http://localhost:8080/graphql`

La implementación se realizó con Spring Boot GraphQL y se diseñaron los siguientes componentes:

MODELOS UTILIZADOS

Tipo	Nombre	Descripción
Input	CarritoCreateInput	Datos necesarios para crear un carrito
Input	CarritoAddItemInput	Agregar item al carrito
Input	CarritoRemoveItemInput	Remover item del carrito
Input	CarritoUpdateInput	Modificar contenido del carrito
Output	CarritoGraphQLResponse	Respuesta unificada para clientes GraphQL

Testing

POSTMAN

Autenticación JWT

La API usa JWT (JSON Web Tokens) para autenticación. Debes obtener un token y usarlo en todas las peticiones protegidas.

1.1 Login (Obtener Token)

Endpoint: `POST /auth/login`
Headers: `Content-Type: application/json`

Body (raw JSON):

```
{
  "username": "admin",
  "password": "admin123"
}
```

Respuesta exitosa (200 OK):

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "username": "admin",
  "message": "Login exitoso"
}
```

Respuesta fallida (401 Unauthorized):

```
{
  "token": null,
  "username": null,
  "message": "Credenciales incorrectas"
}
```

1.2 Configurar Token en Postman

OPCIÓN A: Por petición individual

1. En la pestaña Headers añade:
    - Key: Authorization
    - Value: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

OPCIÓN B: Variable de colección (recomendado)

1. Copia el token de la respuesta del login
2. En Postman → Pestaña Authorization
3. Type: Bearer Token
4. Token: Pega el token copiado

5. Usa una variable: `{{jwt_token}}`

Script Post-Response para automatizar (en el endpoint de login):

```
// Test tab del endpoint POST /auth/login
if (pm.response.code === 200) {
    var jsonData = pm.response.json();
    pm.collectionVariables.set("jwt_token", jsonData.token);
    console.log("Token guardado:", jsonData.token);
}
```

1.3 Verificar Autenticación

Endpoint: `GET /auth/test`
Headers: `Authorization: Bearer {{jwt_token}}`

Respuesta exitosa (200 OK):
Acceso autorizado - JWT funciona correctamente

Endpoints de Usuarios

Base URL: `/api/v1/users`
Requiere: Autenticación JWT + Rol ADMIN

2.1 Listar Todos los Usuarios

Endpoint: `GET /api/v1/users`
Headers: `Authorization: Bearer {{jwt_token}}`

Respuesta (200 OK):

```
[
  {
    "id": 1,
    "nombre": "Administrador",
    "username": "admin",
    "roles": ["ROLE_ADMIN", "ROLE_USER"],
    "createdAt": "2025-11-10T10:30:00",
    "updatedAt": "2025-11-10T10:30:00",
    "isDeleted": false
  },
  {
    "id": 2,
    "nombre": "Juan Pérez",
    "username": "juanp",
    "roles": ["ROLE_USER"],
    "createdAt": "2025-11-11T14:20:00",
    "updatedAt": "2025-11-11T14:20:00",
    "isDeleted": false
  }
]
```

2.2 Obtener Usuario por ID

Endpoint: `GET /api/v1/users/{id}`
Ejemplo: GET /api/v1/users/1
Respuesta (200 OK):

```
{
  "id": 1,
  "nombre": "Administrador",
  "username": "admin",
  "roles": ["ROLE_ADMIN", "ROLE_USER"],
  "createdAt": "2025-11-10T10:30:00",
  "updatedAt": "2025-11-10T10:30:00",
  "isDeleted": false
}
```

Respuesta error (404 Not Found):

```
{
  "statusCode": 404,
  "message": "Usuario no encontrado con id: 999"
}
```

2.3 Buscar Usuarios por Nombre

Endpoint: `GET /api/v1/users/nombre/{nombre}`
Ejemplo: `GET /api/v1/users/nombre/Juan`
Respuesta (200 OK):

```
[
  {
    "id": 2,
    "nombre": "Juan Pérez",
    "username": "juanp",
    "roles": ["ROLE_USER"],
    "createdAt": "2025-11-11T14:20:00",
    "updatedAt": "2025-11-11T14:20:00",
    "isDeleted": false
  }
]
```

2.4 Crear Usuario

Endpoint: `POST /api/v1/users`
Headers: `Content-Type: application/json`
Authorization: `Bearer {{jwt_token}}`

Body (raw JSON):

```
{
  "nombre": "María López",
  "username": "marial",
  "password": "password123",
  "roles": ["USER"]
}
```

Respuesta (201 Created):

```
{
```

```
  "id": 3,
  "nombre": "María López",
  "username": "marial",
  "roles": ["ROLE_USER"],
  "createdAt": "2025-11-18T09:15:00",
  "updatedAt": "2025-11-18T09:15:00",
  "isDeleted": false
}
```

2.5 Actualizar Usuario

Endpoint: `PUT /api/v1/users/{id}`
Ejemplo: `PUT /api/v1/users/3`
Body (raw JSON):

```
{
  "nombre": "María López García",
  "username": "marial",
  "password": "newpassword456",
  "roles": ["USER", "ADMIN"]
}
```

Respuesta (200 OK):

```
{
  "id": 3,
  "nombre": "María López García",
  "username": "marial",
  "roles": ["ROLE_USER", "ROLE_ADMIN"],
  "createdAt": "2025-11-18T09:15:00",
  "updatedAt": "2025-11-18T10:30:00",
  "isDeleted": false
}
```

2.6 Eliminar Usuario

Endpoint: `DELETE /api/v1/users/{id}`
Ejemplo: `DELETE /api/v1/users/3`
Respuesta (204 No Content):
(Sin contenido)

Endpoints de Productos

Base URL: `/api/productos`
Los productos se almacenan en MongoDB con ObjectIds

3.1 Crear Producto

Endpoint: `POST /api/productos`
Headers: `Content-Type: application/json`
Authorization: Bearer {{jwt_token}}

Body (raw JSON):

```
{
  "nombre": "Camiseta Real Madrid 23/24 Local",
  "equipo": "Real Madrid",
```

```
  "talla": "L",
  "descripcion": "Camiseta oficial del Real Madrid temporada 2023/2024",
  "precio": 89.99,
  "imageUrl": "https://example.com/real-madrid-home.jpg",
  "estado": "DISPONIBLE"
}
```

Valores válidos:

- talla: "S", "M", "L", "XL", "XXL"
- estado: "DISPONIBLE", "RESERVADO", "VENDIDO"

Respuesta (201 Created):

```
{
  "id": "673b1f2e8d4c9a5b3c1d2e3f",
  "nombre": "Camiseta Real Madrid 23/24 Local",
  "equipo": "Real Madrid",
  "talla": "L",
  "descripcion": "Camiseta oficial del Real Madrid temporada 2023/2024",
  "precio": 89.99,
  "imageUrl": "https://example.com/real-madrid-home.jpg",
  "estado": "DISPONIBLE",
  "fechaCreacion": "2025-11-18T10:45:00"
}
```

3.2 Listar Todos los Productos

Endpoint: GET /api/productos

Respuesta (200 OK):

```
[
  {
    "id": "673b1f2e8d4c9a5b3c1d2e3f",
    "nombre": "Camiseta Real Madrid 23/24 Local",
    "equipo": "Real Madrid",
    "talla": "L",
    "descripcion": "Camiseta oficial del Real Madrid temporada 2023/2024",
    "precio": 89.99,
    "imageUrl": "https://example.com/real-madrid-home.jpg",
    "estado": "DISPONIBLE",
    "fechaCreacion": "2025-11-18T10:45:00"
  },
  {
    "id": "673b201a9e5d7b6c4d3e4f5g",
    "nombre": "Camiseta FC Barcelona 23/24 Local",
    "equipo": "Barcelona",
    "talla": "M",
    "descripcion": "Camiseta oficial del FC Barcelona temporada
2023/2024",
    "precio": 85.99,
    "imageUrl": "https://example.com/barcelona-home.jpg",
    "estado": "DISPONIBLE",
    "fechaCreacion": "2025-11-18T11:00:00"
```

```
  }
]
```

3.3 Obtener Producto por ID

Endpoint: GET /api/productos/{id}
Ejemplo: GET /api/productos/673b1f2e8d4c9a5b3c1d2e3f
Respuesta (200 OK):

```
{
  "id": "673b1f2e8d4c9a5b3c1d2e3f",
  "nombre": "Camiseta Real Madrid 23/24 Local",
  "equipo": "Real Madrid",
  "talla": "L",
  "descripcion": "Camiseta oficial del Real Madrid temporada 2023/2024",
  "precio": 89.99,
  "imageUrl": "https://example.com/real-madrid-home.jpg",
  "estado": "DISPONIBLE",
  "fechaCreacion": "2025-11-18T10:45:00"
}
```

3.4 Actualizar Producto

Endpoint: PUT /api/productos/{id}
Ejemplo: PUT /api/productos/673b1f2e8d4c9a5b3c1d2e3f
Body (raw JSON):

```
{
  "nombre": "Camiseta Real Madrid 23/24 Local - OFERTA",
  "equipo": "Real Madrid",
  "talla": "L",
  "descripcion": "Camiseta oficial con descuento",
  "precio": 69.99,
  "imageUrl": "https://example.com/real-madrid-home.jpg",
  "estado": "DISPONIBLE"
}
```

Respuesta (200 OK):

```
{
  "id": "673b1f2e8d4c9a5b3c1d2e3f",
  "nombre": "Camiseta Real Madrid 23/24 Local - OFERTA",
  "equipo": "Real Madrid",
  "talla": "L",
  "descripcion": "Camiseta oficial con descuento",
  "precio": 69.99,
  "imageUrl": "https://example.com/real-madrid-home.jpg",
  "estado": "DISPONIBLE",
  "fechaCreacion": "2025-11-18T10:45:00"
}
```

3.5 Eliminar Producto

Endpoint: DELETE /api/productos/{id}
Ejemplo: DELETE /api/productos/673b1f2e8d4c9a5b3c1d2e3f
Respuesta (204 No Content):
(Sin contenido)

3.6 Buscar por Nombre

Endpoint: GET /api/productos/buscar/nombre?nombre={texto}
Ejemplo: GET /api/productos/buscar/nombre?nombre=Madrid
Respuesta (200 OK):


```
[
  {
    "id": "673b1f2e8d4c9a5b3c1d2e3f",
    "nombre": "Camiseta Real Madrid 23/24 Local",
    "equipo": "Real Madrid",
    "talla": "L",
    "precio": 89.99,
    "estado": "DISPONIBLE",
    "fechaCreacion": "2025-11-18T10:45:00"
  }
]
```


3.7 Buscar por Equipo

Endpoint: GET /api/productos/buscar/equipo?equipo={nombre}
Ejemplo: GET /api/productos/buscar/equipo?equipo=Barcelona
Respuesta (200 OK):


```
[
  {
    "id": "673b201a9e5d7b6c4d3e4f5g",
    "nombre": "Camiseta FC Barcelona 23/24 Local",
    "equipo": "Barcelona",
    "talla": "M",
    "precio": 85.99,
    "estado": "DISPONIBLE",
    "fechaCreacion": "2025-11-18T11:00:00"
  }
]
```


3.8 Buscar por Estado

Endpoint: GET /api/productos/buscar/estado?estado={estado}
Ejemplo: GET /api/productos/buscar/estado?estado=DISPONIBLE
Valores válidos: DISPONIBLE, RESERVADO, VENDIDO
Respuesta (200 OK):


```
[
 {
   "id": "673b1f2e8d4c9a5b3c1d2e3f",
   "nombre": "Camiseta Real Madrid 23/24 Local",
   "estado": "DISPONIBLE",
   "precio": 89.99
 }
]
```


3.9 Buscar por Talla

Endpoint: GET /api/productos/buscar/talla?talla={talla}
Ejemplo: GET /api/productos/buscar/talla?talla=L
Valores válidos: S, M, L, XL, XXL

Respuesta (200 OK):

```
[
  {
    "id": "673b1f2e8d4c9a5b3c1d2e3f",
    "nombre": "Camiseta Real Madrid 23/24 Local",
    "talla": "L",
    "precio": 89.99,
    "estado": "DISPONIBLE"
  }
]
```

Endpoints de Carritos

Base URL: `/api/carritos`

Al añadir productos al carrito, su estado cambia a RESERVADO automáticamente.

Crear Carrito

Endpoint: POST /api/carritos

Headers:

Content-Type: application/json

Authorization: Bearer {{jwt_token}}

Body (raw JSON):

```
{
  "userId": 2,
  "items": [
    "673b1f2e8d4c9a5b3c1d2e3f",
    "673b201a9e5d7b6c4d3e4f5g"
  ]
}
```

Los items son ObjectIds de MongoDB (String)

Respuesta (201 Created):

```
{
  "id": 1,
  "userId": 2,
  "items": [
    "673b1f2e8d4c9a5b3c1d2e3f",
    "673b201a9e5d7b6c4d3e4f5g"
  ],
  "creadoEn": "2025-11-18T12:00:00",
  "modificadoEn": "2025-11-18T12:00:00"
}
```

Efecto secundario: Los productos con esos IDs cambiarán su estado a RESERVADO en MongoDB.

4.2 Listar Todos los Carritos

Endpoint: GET /api/carritos

Respuesta (200 OK):

```
[
```

```
  {
    "id": 1,
    "userId": 2,
    "items": [
      "673b1f2e8d4c9a5b3c1d2e3f",
      "673b201a9e5d7b6c4d3e4f5g"
    ],
    "creadoEn": "2025-11-18T12:00:00",
    "modificadoEn": "2025-11-18T12:00:00"
  }
]
```

4.3 Obtener Carrito por ID

Endpoint: `GET /api/carritos/{id}`

Ejemplo: `GET /api/carritos/1`

Respuesta (200 OK):

```
{
  "id": 1,
  "userId": 2,
  "items": [
    "673b1f2e8d4c9a5b3c1d2e3f",
    "673b201a9e5d7b6c4d3e4f5g"
  ],
  "creadoEn": "2025-11-18T12:00:00",
  "modificadoEn": "2025-11-18T12:00:00"
}
```

4.4 Obtener Carrito por Usuario

Endpoint: `GET /api/carritos/usuario/{userId}`

Ejemplo: `GET /api/carritos/usuario/2`

Respuesta (200 OK):

```
{
  "id": 1,
  "userId": 2,
  "items": [
    "673b1f2e8d4c9a5b3c1d2e3f",
    "673b201a9e5d7b6c4d3e4f5g"
  ],
  "creadoEn": "2025-11-18T12:00:00",
  "modificadoEn": "2025-11-18T12:00:00"
}
```

4.5 Actualizar Carrito

Endpoint: `PUT /api/carritos/{id}`

Ejemplo: `PUT /api/carritos/1`

Body (raw JSON):

```
{
  "userId": 2,
  "items": [
    "673b1f2e8d4c9a5b3c1d2e3f",
```

```
    "673b201a9e5d7b6c4d3e4f5g",
    "673b2105af6e8c7d5e4f5g6h"
  ]
}
```

Respuesta (200 OK):

```
{
  "id": 1,
  "userId": 2,
  "items": [
    "673b1f2e8d4c9a5b3c1d2e3f",
    "673b201a9e5d7b6c4d3e4f5g",
    "673b2105af6e8c7d5e4f5g6h"
  ],
  "creadoEn": "2025-11-18T12:00:00",
  "modificadoEn": "2025-11-18T12:15:00"
}
```

4.6 Eliminar Carrito

Endpoint: DELETE /api/carritos/{id}

Ejemplo: DELETE /api/carritos/1

Respuesta (200 OK):

```
{
  "id": 1,
  "userId": 2,
  "items": [
    "673b1f2e8d4c9a5b3c1d2e3f",
    "673b201a9e5d7b6c4d3e4f5g"
  ],
  "creadoEn": "2025-11-18T12:00:00",
  "modificadoEn": "2025-11-18T12:15:00"
}
```

Efecto secundario: Los productos del carrito volverán al estado DISPONIBLE en MongoDB.

Endpoints de Pedidos

Base URL: `/api/pedidos`

Los pedidos almacenan un snapshot de los productos en el momento de la compra.

5.1 Crear Pedido

Endpoint: POST /api/pedidos

Headers: Content-Type: application/json

Authorization: Bearer {{jwt_token}}

Body (raw JSON):

```
{
  "userId": 2,
  "direccionEnvio": "Calle Falsa 123, Madrid, 28001",
  "productosIds": [
```

```
    "673b1f2e8d4c9a5b3c1d2e3f",
    "673b201a9e5d7b6c4d3e4f5g"
  ]
}
```

Respuesta (201 Created):

```
{
  "id": 1,
  "userId": 2,
  "estado": "PENDIENTE_PAGO",
  "fechaCreacion": "2025-11-18T13:00:00",
  "total": 175.98,
  "direccionEnvio": "Calle Falsa 123, Madrid, 28001",
  "fechaPago": null,
  "fechaEnvio": null,
  "numeroSeguimiento": null,
  "detalles": [
    {
      "productoId": "673b1f2e8d4c9a5b3c1d2e3f",
      "nombre": "Camiseta Real Madrid 23/24 Local",
      "talla": "L",
      "equipo": "Real Madrid",
      "precioPagado": 89.99,
      "imageUrl": "https://example.com/real-madrid-home.jpg"
    },
    {
      "productoId": "673b201a9e5d7b6c4d3e4f5g",
      "nombre": "Camiseta FC Barcelona 23/24 Local",
      "talla": "M",
      "equipo": "Barcelona",
      "precioPagado": 85.99,
      "imageUrl": "https://example.com/barcelona-home.jpg"
    }
  ]
}
```

Efecto secundario: Los productos cambian a estado VENDIDO en MongoDB.

5.2 Listar Todos los Pedidos

Endpoint: GET /api/pedidos

Respuesta (200 OK):

```
[
  {
    "id": 1,
    "userId": 2,
    "estado": "PENDIENTE_PAGO",
    "fechaCreacion": "2025-11-18T13:00:00",
    "total": 175.98,
    "direccionEnvio": "Calle Falsa 123, Madrid, 28001",
    "detalles": [ /* ... */ ]
```

```
  }
]
```

5.3 Obtener Pedido por ID

Endpoint: GET /api/pedidos/{id}
Ejemplo: GET /api/pedidos/1
Respuesta (200 OK):

```
{
  "id": 1,
  "userId": 2,
  "estado": "PENDIENTE_PAGO",
  "fechaCreacion": "2025-11-18T13:00:00",
  "total": 175.98,
  "direccionEnvio": "Calle Falsa 123, Madrid, 28001",
  "fechaPago": null,
  "fechaEnvio": null,
  "numeroSeguimiento": null,
  "detalles": [
    {
      "productoId": "673b1f2e8d4c9a5b3c1d2e3f",
      "nombre": "Camiseta Real Madrid 23/24 Local",
      "talla": "L",
      "equipo": "Real Madrid",
      "precioPagado": 89.99,
      "imageUrl": "https://example.com/real-madrid-home.jpg"
    }
  ]
}
```

5.4 Obtener Pedidos por Usuario

Endpoint: GET /api/pedidos/usuario/{userId}
Ejemplo: GET /api/pedidos/usuario/2
Respuesta (200 OK):

```
[
  {
    "id": 1,
    "userId": 2,
    "estado": "PENDIENTE_PAGO",
    "total": 175.98,
    "fechaCreacion": "2025-11-18T13:00:00"
  },
  {
    "id": 3,
    "userId": 2,
    "estado": "PAGADO",
    "total": 89.99,
    "fechaCreacion": "2025-11-17T10:30:00"
  }
]
```

5.5 Obtener Pedidos por Estado

Endpoint: GET /api/pedidos/estado?estado={estado}
Ejemplo: GET /api/pedidos/estado?estado=PAGADO
Valores válidos:

- PENDIENTE_PAGO
- PAGADO
- ENVIADO
- ENTREGADO
- CANCELADO

Respuesta (200 OK):

```
[
  {
    "id": 2,
    "userId": 3,
    "estado": "PAGADO",
    "total": 89.99,
    "fechaCreacion": "2025-11-17T15:00:00"
  }
]
```

5.6 Actualizar Estado del Pedido

Endpoint: PATCH /api/pedidos/{id}/estado?estado={nuevoEstado}
Ejemplo: PATCH /api/pedidos/1/estado?estado=PAGADO
Respuesta (200 OK):

```
{
  "id": 1,
  "userId": 2,
  "estado": "PAGADO",
  "fechaCreacion": "2025-11-18T13:00:00",
  "fechaPago": "2025-11-18T13:30:00",
  "total": 175.98,
  "detalles": [ /* ... */ ]
}
```

Transiciones válidas:

PENDIENTE_PAGO → PAGADO → ENVIADO → ENTREGADO
↓
CANCELADO

5.7 Eliminar Pedido

Endpoint: DELETE /api/pedidos/{id}
Ejemplo: DELETE /api/pedidos/1
Respuesta (204 No Content):
(Sin contenido)

Flujo Completo de Compra

Aquí está el proceso completo de una compra, paso a paso:
PASO 1: Autenticación == POST /auth/login

Body: `{ "username": "admin", "password": "admin123" }`
→ Guardar token JWT

PASO 2: Crear Productos (ADMIN) == POST /api/productos
Headers: Authorization: Bearer {token}

productoId1:
Body: {

```
  "nombre": "Camiseta Real Madrid Local",
  "equipo": "Real Madrid",
  "talla": "L",
  "precio": 89.99,
  "estado": "DISPONIBLE"
}
```

Guardar productoId1

productoId2:
Body: {

```
  "nombre": "Camiseta Barcelona Local",
  "equipo": "Barcelona",
  "talla": "M",
  "precio": 85.99,
  "estado": "DISPONIBLE"
}
```

Guardar productoId2

PASO 3: Crear Usuario == POST /api/v1/users
Headers: Authorization: Bearer {token}
Body: {

```
  "nombre": "Juan Pérez",
  "username": "juanp",
  "password": "juan123",
  "roles": ["USER"]
}
```

Guardar userId

PASO 4: Crear Carrito == POST /api/carritos
Headers: Authorization: Bearer {token}
Body: {

```
  "userId": {userId},
  "items": ["{productoId1}", "{productoId2}"]
}
```

Productos cambian a estado RESERVADO

PASO 5: Verificar Productos Reservados == GET /api/productos/buscar/estado?estado=RESERVADO
Debe mostrar los 2 productos del carrito

PASO 6: Crear Pedido == POST /api/pedidos
Headers: Authorization: Bearer {token}
Body: {

```
  "userId": {userId},
  "direccionEnvio": "Calle Ejemplo 123",
  "productosIds": ["{productoId1}", "{productoId2}"]
}
```

Productos cambian a estado VENDIDO

Se crea snapshot en detalles del pedido
Estado inicial: PENDIENTE_PAGO

PASO 7: Confirmar Pago == PATCH /api/pedidos/{pedidoId}/estado?estado=PAGADO
- Estado cambia a PAGADO
- Se registra fechaPago

PASO 8: Procesar Envío == PATCH /api/pedidos/{pedidoId}/estado?estado=ENVIADO
- Estado cambia a ENVIADO
- Se registra fechaEnvio

PASO 9: Confirmar Entrega == PATCH
/api/pedidos/{pedidoId}/estado?estado=ENTREGADO
Estado cambia a ENTREGADO

PASO 10: Consultar Historial == GET /api/pedidos/usuario/{userId}
Ver todos los pedidos del usuario

Colección de Postman

Crear Variables de Colección

1. En Postman, crea una nueva colección: "CamisAPI"
2. Click derecho → Edit
3. Pestaña Variables
4. Añade estas variables:

Variable	Initial Value	Current Value
base_url	http://localhost:8081	http://localhost:8081
jwt_token	(dejar vacío)	(dejar vacío)
user_id	2	2
producto_id_1	(dejar vacío)	(dejar vacío)
producto_id_2	(dejar vacío)	(dejar vacío)
carrito_id	(dejar vacío)	(dejar vacío)
pedido_id	(dejar vacío)	(dejar vacío)

Scripts Automáticos
En POST /auth/login (Tests tab)

```
if (pm.response.code === 200) {
    var jsonData = pm.response.json();
    pm.collectionVariables.set("jwt_token", jsonData.token);
    console.log("Token guardado");
}
```

En POST /api/productos (Tests tab)

```
if (pm.response.code === 201) {
    var jsonData = pm.response.json();
    if (!pm.collectionVariables.get("producto_id_1")) {
        pm.collectionVariables.set("producto_id_1", jsonData.id);
        console.log(" Producto 1 guardado:", jsonData.id);
    } else if (!pm.collectionVariables.get("producto_id_2")) {
        pm.collectionVariables.set("producto_id_2", jsonData.id);
        console.log("Producto 2 guardado:", jsonData.id);
    }
}
```

En POST /api/carritos (Tests tab)

```
if (pm.response.code === 201) {
    var jsonData = pm.response.json();
    pm.collectionVariables.set("carrito_id", jsonData.id);
    console.log(" Carrito guardado:", jsonData.id);
}
```

En POST /api/pedidos (Tests tab)

```
if (pm.response.code === 201) {
    var jsonData = pm.response.json();
    pm.collectionVariables.set("pedido_id", jsonData.id);
    console.log(" Pedido guardado:", jsonData.id);
}
```

Casos de Prueba Importantes

CASO 1: Producto No Disponible
1. Crear producto con estado VENDIDO
2. Intentar añadirlo al carrito:
Debe fallar (producto no disponible)

CASO 2: Carrito Vacío
1. Crear carrito con items vacíos: [ ]
2. Intentar crear pedido:
Debe fallar (carrito vacío)

CASO 3: Doble Reserva
1. Usuario A añade producto al carrito
2. Usuario B intenta añadir el mismo producto:
Debe fallar (producto RESERVADO)

CASO 4: Autorización
1. Hacer petición sin token JWT → Debe devolver 401 Unauthorized

2. Usuario sin rol ADMIN intenta GET /api/v1/users → Debe devolver 403 Forbidden

CASO 5: Validaciones
1. Crear usuario sin username → Debe devolver 400 Bad Request

2. Crear producto con precio negativo → Debe devolver 400 Bad Request

Conclusiones

LOGRADO

Se ha logrado conseguir una aplicación funcional en cuanto al funcionamiento de una tienda de camisetas de fútbol online. Con productos exclusivos y únicos. El problema ha sido la seguridad con los usuarios y los roles User y Admin.