# Anexo A – Manual de instalación

Este manual describe los pasos necesarios para instalar y ejecutar el sistema de gestión de inventario online en un entorno local de desarrollo. El sistema está dividido en dos módulos principales: **backend** y **frontend**, ambos disponibles en los subdirectorios code/back/ y code/front/ del repositorio de GitHub. Para detalles técnicos más concretos, se puede consultar el contenido de los archivos README.md de cada módulo.

---

## Requisitos generales

- **Java 17**
- **Node.js 20+**
- **Angular CLI** (npm install -g @angular/cli)
- **PostgreSQL 15 o superior**
- **Maven**
- (Opcional) **Docker**

---

## Backend – Spring Boot

El backend implementa toda la lógica de negocio, seguridad, persistencia de datos y documentación de la API.

**Ruta:** code/back/

**Tecnologías principales**

- Java 17 + Spring Boot 3
- PostgreSQL
- JWT para autenticación
- Swagger UI (OpenAPI)
- Maven

**Estructura destacada**

code/back/

├── controller/ # Endpoints REST

├── service/ # Lógica de negocio

├── repository/ # Acceso a datos

├── entity/ # Modelo JPA

├── dto/ # Objetos de transferencia

├── security/ # JWT y configuración

└── config/ # Swagger y Seeder

**Configuración inicial**

1. Crear la base de datos PostgreSQL local (gestion_inventario) en el puerto 5433.
2. Copiar la plantilla application.properties (proporcionada) a src/main/resources/.
3. Completar con tus credenciales y configuración personalizada:

properties

spring.datasource.url=jdbc:postgresql://localhost:5433/gestion_inventario

spring.datasource.username=TU_USUARIO

spring.datasource.password=TU_PASSWORD

...

app.jwtSecret=CLAVE_SECRETA

app.jwtExpirationMs=86400000

OPENAI_API_KEY=TU_API_KEY

**Ejecución**

Desde el directorio raíz del backend:

./mvnw spring-boot:run

Para empaquetar como .jar:

./mvnw clean package

java -jar target/inventario-back-0.0.1-SNAPSHOT.jar

**Acceso a Swagger**

Una vez levantado:

http://localhost:8080/swagger-ui/index.html

**Notas**

- Seeder automático: si la base de datos está vacía, se poblará con datos realistas generados (categorías, productos, almacenes, usuarios…).
- Estructura desacoplada con uso de DTOs.
- Control de acceso por roles mediante JWT.

---

## Frontend – Angular 19 + PrimeNG

Interfaz SPA para la gestión y visualización del inventario, distribuido por roles de usuario.

**Ruta:** code/front/

**Tecnologías principales**

- Angular 19
- PrimeNG

- SCSS, RxJS, TypeScript
- JWT con Bearer Token

**Estructura destacada**

code/front/app/

├── admin/ # Gestión de usuarios

├── auth/ # Login y guards

├── inventario/ # Productos y almacenes

├── movimientos/ # Transacciones

├── notificaciones/

├── stats/ # Dashboard con KPIs

├── shared/ # Navbar, componentes comunes

**Instalación y ejecución**

cd code/front

npm install

npm run start

Acceso:

http://localhost:4200/

**Seguridad**

- Guards por rol implementados en el frontend.
- Token JWT incluido automáticamente en cada petición.
- Acceso restringido a vistas y rutas según permisos del usuario.

---

## Verificación y testing

- El backend incluye pruebas unitarias con JUnit y Mokito.
- El frontend usa *.spec.ts con Karma/Jest para tests unitarios.
- Para ejecutar los tests frontend: ng test

---

## Autor

**Eduardo García Romera**
egarcia3266@alumno.uned.es

Este sistema ha sido desarrollado como parte del Trabajo de Fin de Grado en Ingeniería Informática (UNED). Está pensado para simular el funcionamiento real de una empresa tipo Amazon que requiere gestionar múltiples almacenes, productos, transacciones y perfiles de usuario.