

# Manual de instalación

## 1. Introducción

Este manual describe los pasos necesarios para:

- Instalar y configurar el **backend** (API REST en Spring Boot + PostgreSQL).
  - Instalar y ejecutar el **frontend** (React + Vite).
  - Preparar el entorno para desarrollo local.
  - Dejar listo el proyecto para luego poder desplegarlo (Render + Vercel).
- 

## 2. Requisitos previos

### 2.1. Generales

- Sistema operativo: Windows, Linux o macOS.
- RAM recomendada: mínimo 8 GB.
- Acceso a internet (para descargar dependencias y librerías).
- **Git** instalado.

### 2.2. Backend (Java / Spring Boot)

- **Java JDK 17** instalado.
- **Maven** instalado (o usar el wrapper mvnw del proyecto).
- **Docker** instalado (para la base de datos PostgreSQL).

### 2.3. Frontend (React / Vite)

- **Node.js** versión 18 o superior.
  - **npm** o **yarn** (npm viene incluido con Node).
-

### **3. Clonar el repositorio**

1. Abrir una terminal o consola.
2. Clonar el proyecto:

```
git clone <URL_DEL_REPO>
cd <CARPETA_DEL_PROYECTO>
```

(En esta carpeta tendrás algo como backend/ y frontend/ o nombres similares, según cómo lo tengas organizado.)

---

### **4. Instalación y configuración del backend**

#### **4.1. Levantar PostgreSQL con Docker**

En la terminal, ejecutar:

```
docker run --name unistay-db \
-e POSTGRES_USER=postgres \
-e POSTGRES_PASSWORD=0101 \
-e POSTGRES_DB=unistay \
-p 5432:5432 \
-d postgres:17
```

Esto crea una base de datos:

- Host: localhost
- Puerto: 5432
- Base de datos: unistay
- Usuario: postgres
- Password: 0101

Puedes verificar que el contenedor esté corriendo con:

```
docker ps
```

#### **4.2. Configuración de application.properties (modo local)**

Dentro del módulo backend, ubica el archivo:

```
src/main/resources/application.properties
```

Y asegúrate que en modo local tenga algo equivalente a:

```
server.port=8080
```

```
spring.servlet.multipart.enabled=true
spring.servlet.multipart.max-file-size=10MB
spring.servlet.multipart.max-request-size=10MB

# ===== DATABASE =====
spring.datasource.url=jdbc:postgresql://localhost:5432/unistay
spring.datasource.username=postgres
spring.datasource.password=0101
spring.datasource.driver-class-name=org.postgresql.Driver

# ===== JPA / HIBERNATE =====
spring.jpa.hibernate.ddl-auto=validate
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.jdbc.batch_size=20
spring.jpa.properties.hibernate.order_inserts=true
spring.jpa.properties.hibernate.order_updates=true

logging.level.org.hibernate.SQL=DEBUG
logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE
logging.level.org.springframework.security=DEBUG

# ===== FILE UPLOADS =====
file.upload-dir=./uploads_unistay
file.base-url=/uploads_unistay

# ===== JWT =====
app.jwt.secret=TU_SECRET_JWT_LOCAL
app.jwt.expiration-ms=86400000

# ===== GOOGLE KEYS =====
google.api.key=TU_GOOGLE_API_KEY_LOCAL
google.oauth.client-id=TU_CLIENT_ID_GOOGLE_OAUTH

spring.security.filter.dispatcher-types=REQUEST,ASYNC,ERROR
spring.security.oauth2.resourceserver.jwt.jwk-set-uri=
spring.mvc.hiddenmethod.filter.enabled=true

server.servlet.session.timeout=30m
server.servlet.session.tracking-modes=cookie

# ===== EMAIL (modo desarollo con Gmail) =====
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=TU_CORREO_GMAIL
spring.mail.password=TU_PASSWORD_O_APP_PASSWORD
spring.mail.protocol=smtp
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
spring.mail.default-encoding=UTF-8
spring.mail.properties.mail.smtp.starttls.required=true
spring.mail.properties.mail.smtp.ssl.trust=smtp.gmail.com
spring.mail.properties.mail.debug=true
```

```
spring.mail.properties.mail.smtp.connectiontimeout=5000
spring.mail.properties.mail.smtp.timeout=5000
spring.mail.properties.mail.smtp.writetimeout=5000

spring.flyway.enabled=false

# ===== MAILERSEND (para producción, opcional en local) =====
mailersend.api.token=TU_API_TOKEN_MAILERSEND
mailersend.from.email=remitente@tudominio.com
```

En producción tú ya lo manejas con variables de entorno ( `${DB_URL}` ,  `${DB_USER}` , etc.), pero en local es más cómodo usar valores directos.

### 4.3. Variables de entorno (para producción / Render)

En Render deberás configurar al menos:

- `DB_URL`
- `DB_USER`
- `DB_PASS`
- `JWT_SECRET`
- `GOOGLE_API_KEY`
- `GOOGLE_OAUTH_CLIENT_ID`
- `EMAIL_USER`
- `EMAIL_PASS (si usas SMTP)`
- `MAILERSEND_API_TOKEN`
- `MAILERSEND_FROM_EMAIL`

El `application.properties` ya está preparado para leer esos valores cuando vayas a producción.

### 4.4. Construir y ejecutar el backend

Desde la carpeta del backend:

```
mvn clean install
mvn spring-boot:run
```

Si todo está correcto, la API quedará escuchando en:

- `http://localhost:8080`

Puedes probar que está levantado consultando algún endpoint público (por ejemplo, alguno de `/api/auth/...` o un endpoint de salud si lo tienes configurado).

---

## **5. Instalación y configuración del frontend**

### **5.1. Configurar la URL del backend**

En el frontend tienes algo como:

```
let baseURL = import.meta.env.VITE_BACKEND_URL;
```

Crea un archivo .env en la raíz del frontend con:

```
VITE_BACKEND_URL=http://localhost:8080
```

Para producción (Vercel) cambiarás este valor por la URL pública del backend en Render, por ejemplo:

```
VITE_BACKEND_URL=https://unistay-software.onrender.com
```

### **5.2. Instalar dependencias**

Desde la carpeta del frontend:

```
npm install
```

### **5.3. Ejecutar en modo desarrollo**

```
npm run dev
```

Normalmente Vite levanta el proyecto en:

- http://localhost:5173

Entra en esa URL y deberías ver el frontend de UniStay funcionando y consumiendo el backend local.

---

## 6. Pruebas básicas después de la instalación

Una vez que backend y frontend estén corriendo:

1. **Registro y login**
    - Registra un nuevo usuario desde la interfaz.
    - Inicia sesión y revisa que se genere el token y se guarde en el almacenamiento del navegador (localStorage o similar).
  2. **Publicaciones y solicitudes de interés**
    - Crear una publicación (si tu rol lo permite).
    - Desde otra cuenta, enviar una “solicitud de interés”.
    - Confirmar que la solicitud aparece en las secciones correspondientes.
  3. **Recuperación de contraseña**
    - Usar la opción “¿Olvidaste tu contraseña?”.
    - Verificar que se genera el token y se envía el correo (en local, usando Gmail o el proveedor que tengas configurado).
    - Probar el enlace de reset-password en el frontend.
  4. **Notificaciones por correo en solicitudes**
    - Crear una solicitud y proponer disponibilidad.
    - Confirmar que, aunque el correo falle, la lógica de negocio sí actualiza el estado de la solicitud (ya lo dejamos preparado para que el correo no rompa el flujo).
- 

## 7. Despliegue (resumen muy breve)

### 7.1. Backend en Render

1. Crear una base de datos PostgreSQL en Render.
2. Crear un servicio web para el backend:
  - Build command: mvn clean install.
  - Start command: java -jar target/UniStay-0.0.1-SNAPSHOT.jar (ajusta el nombre del .jar si es distinto).
3. Configurar las variables de entorno (DB\_URL, DB\_USER, DB\_PASS, JWT\_SECRET, etc.).
4. Hacer deploy y verificar que la API responda en la URL de Render.

### 7.2. Frontend en Vercel

1. Importar el repositorio en Vercel.
2. Configurar la variable VITE\_BACKEND\_URL con la URL del backend en Render.
3. Build command: npm run build
4. Output directory: dist
5. Deployar y probar la aplicación desde la URL de Vercel.

---

## 8. Problemas frecuentes

- **Error 403 Forbidden al consumir la API**
  - Verificar que el frontend esté enviando el token JWT en el header Authorization: Bearer <token>.
  - Revisar reglas en SecurityConfiguration y que la ruta esté incluida o autenticada correctamente.
- **Error CORS desde el frontend**
  - Confirmar que en CorsConfiguration estén incluidos:
    - http://localhost:5173 (desarrollo)
    - https://uni-stay-software.vercel.app (producción)
  - Verificar que .cors(Customizer.withDefaults()) esté configurado en HttpSecurity.
- **Error al recargar páginas en Vercel (rutas protegidas)**
  - Asegurarse de tener un vercel.json con rewrites que redirijan todas las rutas a index.html para que React maneje el routing.
- **Errores con envío de correos en Render (timeouts / 403)**
  - Verificar el proveedor de correo (SMTP o MailerSend).
  - Revisar MAILERSEND\_API\_TOKEN, dominio/autenticación y que el backend no dependa del éxito del correo para completar la lógica de negocio (ya lo ajustaste en los servicios).