

1. **Análisis y Planificación**

- 1.1. [Definir objetivos y alcance del sistema.](#)
- 1.2. [Identificar requerimientos funcionales y no funcionales.](#)
- 1.3. [Identificar usuarios y casos de uso.](#)
- 1.4. [Evaluar tecnologías a utilizar.](#)

2. **Diseño de la Arquitectura**

- 2.1. [Definir la arquitectura general del sistema \(monolito vs microservicios\).](#)
- 2.2. [Modelar la base de datos.](#)
- 2.3. [Diseñar la API REST \(endpoints, autenticación, formato de respuestas\).](#)
- 2.4. [Elegir patrones de diseño adecuados \(MVC, DAO, Service Layer, etc.\).](#)
- 2.5. Documentación de la arquitectura.

3. **Desarrollo del Backend (Spring Boot)**

- 3.1. [Configuración del proyecto \(Maven/Gradle, dependencias\).](#)
- 3.2. [Implementación de la base de datos con JPA/Hibernate.](#)
- 3.3. [Creación de controladores REST y servicios.](#)
- 3.4. [Implementación de autenticación \(JWT, OAuth2\).](#)
- 3.5. [Pruebas unitarias e integración.](#)
- 3.6. Documentación de la API.

4. **Desarrollo del Frontend (React)**

- 4.1. Configuración del entorno de desarrollo.
- 4.2. Diseño de la UI/UX.
- 4.3. Implementación de componentes y navegación.
- 4.4. Conexión con la API del backend.
- 4.5. Pruebas de usabilidad.

5. **Desarrollo de la App Móvil (Android)**

- 5.1. Definir si será nativa (Kotlin/Jetpack Compose) o híbrida (React Native).
- 5.2. Implementación de la interfaz y lógica de negocio.
- 5.3. Conexión con el backend.
- 5.4. Pruebas en dispositivos reales.

6. **Pruebas y Depuración**

- 6.1. Pruebas unitarias y de integración en backend y frontend.
- 6.2. Pruebas de usuario.

6.3. Optimización de código y rendimiento.

6.4. Revisión de código.

7. Despliegue (si aplica)

7.1. Configuración de hosting para la web.

7.2. Distribución de la app Android en dispositivos de prueba o Play Store.

7.3. Monitoreo post-despliegue.

8. Conclusión

8.1. Reflexiones sobre el proyecto.

8.2. Lecciones aprendidas.

8.3. Posibles mejoras futuras.

1. Análisis y Planificación

1.1. Definir objetivos y alcance del sistema.

Objetivo del Sistema

Crear una aplicación de gestión de candidaturas que permita a los usuarios llevar un seguimiento detallado de los puestos de trabajo a los que han aplicado, incluyendo información sobre empresas, estado de la candidatura y observaciones, accesible desde web y móvil.

Alcance del Sistema

Funcionalidades Principales (MVP - Producto Mínimo Viable)

1. Registro de candidaturas

- Permitir al usuario agregar nuevas aplicaciones a puestos de trabajo.
- Guardar información como empresa, cargo, fecha de aplicación y estado.

2. Gestión de estados de candidatura

- Definir estados como: *Pendiente, Entrevista, Aceptado, Rechazado, Archivado*.
- Permitir actualización de estado con comentarios.

3. Búsqueda y filtrado de candidaturas

- Filtros por estado, empresa, fecha de aplicación.
- Búsqueda por palabras clave.

4. Historial y Notas

- Permitir agregar notas a cada candidatura.
- Ver el historial de cambios en una candidatura.

5. Acceso multi-plataforma

- **Versión web:** React (para acceso desde PC).
- **Versión móvil:** Aplicación Android que consuma la misma API.

6. Gestión de reclutadores

- Permitir guardar nombres, teléfono y enlaces a perfiles de LinkedIn de los reclutadores de la empresa.
- Relacionar estos reclutadores con las candidaturas.

7. Gestión de preguntas de entrevista

- Permitir guardar preguntas típicas de la entrevista para cada candidatura.
- Contar y mostrar el número total de preguntas guardadas.

8. Información de contacto de la empresa

- Guardar correos electrónicos de contacto.
- Guardar teléfono principal de la empresa.

Características Técnicas Iniciales

- **Backend:** Spring Boot con API REST.
- **Base de Datos:** PostgreSQL o MySQL.
- **Frontend Web:** React con Material UI/Tailwind.
- **Aplicación Android:** Kotlin o React Native.
- **Autenticación:** JWT (Login con usuario y contraseña).

Lo que NO está en el alcance inicial (pero podría añadirse en el futuro)

- ✗ Envío automático de correos o notificaciones.
- ✗ Integración con LinkedIn o portales de empleo.
- ✗ Multiusuario (solo será para un usuario en local por ahora).
- ✗ Sincronización offline para la versión móvil.

1.2. Identificar requerimientos funcionales y no funcionales.

Requerimientos Funcionales (RF)

1. Gestión de Candidaturas

- ✓ **RF-01:** Permitir registrar una candidatura con los siguientes datos:
 - Empresa
 - Cargo
 - Fecha de aplicación
 - Estado (*Pendiente, Entrevista, Aceptado, Rechazado, Archivado, En proceso*)
 - Notas y observaciones
 - ✓ **RF-02:** Permitir actualizar el estado de una candidatura.
 - ✓ **RF-03:** Permitir buscar y filtrar candidaturas por estado, empresa y fecha, palabras clave.
-

2. Gestión de Reclutadores

- ✓ **RF-04:** Permitir agregar reclutadores asociados a una empresa, incluyendo:
 - Nombre
 - Teléfono de contacto
 - URL del perfil de LinkedIn
 - ✓ **RF-05:** Relacionar reclutadores con candidaturas.
-

3. Gestión de Preguntas de Entrevista

- ✓ **RF-06:** Permitir agregar preguntas de entrevista para una candidatura.
 - ✓ **RF-07:** Mostrar el número total de preguntas almacenadas para cada candidatura.
-

4. Gestión de Información de Contacto

- ✓ **RF-08:** Permitir agregar correos electrónicos de contacto de la empresa.
 - ✓ **RF-09:** Permitir agregar teléfono principal de la empresa.
-

5. Autenticación y Gestión de Usuarios

✅ **RF-10:** Permitir el registro y autenticación de usuarios mediante credenciales (usuario y contraseña).

- Registro de un único usuario (dueño del sistema).
- Generación de token JWT para acceder a los recursos protegidos.

✅ **RF-11:** Control de acceso basado en roles para operaciones CRUD de preguntas.

- USER: Crear y actualizar solo sus preguntas.
- ADMIN: Permisos completos (Excepto eliminar usuarios ROOT)
- ROOT: Administrador del sistema.

✅ **RF-12:** Permitir ver el historial de cambios de estado y modificaciones en una candidatura.

📌 **Requerimientos No Funcionales (RNF)**

✅ **RNF-01:** El backend debe ser desarrollado en **Spring Boot** y exponer una API REST.

✅ **RNF-02:** La base de datos debe ser **PostgreSQL o MySQL**.

✅ **RNF-03:** La API debe responder en formato **JSON** y seguir el estándar **RESTful**.

✅ **RNF-04:** La web debe ser desarrollada con **React**.

✅ **RNF-05:** La app móvil debe ser desarrollada en **Kotlin (nativa) o React Native**.

✅ **RNF-06:** La aplicación debe permitir la gestión de hasta **1,000 candidaturas sin pérdida de rendimiento**.

✅ **RNF-07:** El tiempo de respuesta de la API no debe superar los **500 ms** en condiciones normales.

✅ **RNF-08:** La interfaz debe ser **intuitiva y fácil de usar** en web y móvil.


✅ **RNF-09:** La API debe soportar **CORS** para permitir peticiones desde el frontend.

✅ **RNF-10:** La aplicación debe permitir **exportar datos en CSV o JSON** (para futuras mejoras).

✅ **RNF-11:** La autenticación debe implementarse mediante **JWT** para garantizar seguridad en las comunicaciones.


✅ **RNF-12:** Todas las operaciones CRUD deben validar permisos mediante JWT.

1.3. Identificar usuarios y casos de uso.

 Usuario Principal

Usuario Único (Dueño del Sistema)

- Será el único usuario del sistema.
- No habrá roles ni multiusuario en esta primera versión.
- Tendrá acceso a todas las funcionalidades.

 Casos de Uso

| ID | Casos de Uso | Descripción |
|---------|---------------------------------------|--|
| CU – 01 | Registrar candidatura | Permite agregar una nueva candidatura con información de empresa, cargo, fecha y estado. |
| CU – 02 | Editar candidatura | Permite modificar los datos de una candidatura existente. |
| CU – 03 | Actualizar estado de candidatura | Permite cambiar el estado de una candidatura (Pendiente, Entrevista, etc.). |
| CU – 04 | Buscar candidaturas | Permite filtrar candidaturas por estado, empresa o fecha. |
| CU – 05 | Agregar reclutador | Permite registrar nombre y LinkedIn de reclutadores. |
| CU – 06 | Relacionar reclutador con candidatura | Permite vincular un reclutador a una candidatura específica. |
| CU – 07 | Agregar preguntas de entrevista | Permite registrar preguntas frecuentes de entrevistas. |
| CU – 08 | Ver número de preguntas de entrevista | Muestra el número total de preguntas guardadas. |
| CU – 09 | Agregar información de contacto | Permite guardar correos electrónicos y múltiples números de teléfono de la empresa. |
| CU – 10 | Autenticarse en el sistema | Permite iniciar sesión con usuario y contraseña. |
| CU – 11 | Cerrar sesión | Permite salir del sistema de manera segura. |

1.4. Evaluar tecnologías a utilizar.

Backend

- ✓ **Lenguaje:** Java (Spring Boot)
- ✓ **Framework:** Spring Boot con Spring Web, Spring Security (para autenticación), y Spring Data JPA
- ✓ **Base de Datos:** PostgreSQL o MySQL
- ✓ **ORM:** Hibernate
- ✓ **Autenticación:** JWT
- ✓ **Formato de API:** REST (JSON)

Frontend Web

- ✓ **Librería:** React
- ✓ **Diseño UI:** TailwindCSS / Material UI / PrimeReact
- ✓ **Estado Global:** Context API o Redux
- ✓ **Conexión con Backend:** Axios

Aplicación Móvil (Android)

- ✓ **Opción 1 (Nativo):** Kotlin con Jetpack Compose
- ✓ **Opción 2 (Híbrido):** React Native (para compartir código con la web)

Infraestructura y Herramientas

- ✓ **Gestor de Dependencias:** Maven o Gradle
- ✓ **Pruebas:** JUnit y Postman para API
- ✓ **Control de Versiones:** Git (GitHub o GitLab)

2. Diseño de la Arquitectura

2.1. Definir la arquitectura general del sistema (monolito vs microservicios).

Optaremos por una **arquitectura en capas (n-tier)**, separando las responsabilidades de cada componente para mejorar la mantenibilidad y escalabilidad.

Backend (Spring Boot) - API REST

- **Capa de Controladores (Controllers):** Maneja las solicitudes HTTP y responde con datos en JSON.
- **Capa de Servicios (Services):** Contiene la lógica de negocio de la aplicación.
- **Capa de Repositorio (Repositories):** Interactúa con la base de datos mediante JPA/Hibernate.
- **Capa de Persistencia (Entities):** Representa los datos de la aplicación con modelos de base de datos.

Frontend Web (React) - Cliente

- **Componentes UI:** Diseño e interacción con el usuario.
- **Servicios API:** Comunicación con el backend mediante Axios.
- **Gestión de Estado:** Context API o Redux para manejar datos globales.

Aplicación Móvil (Android - Kotlin o React Native)

- **Pantallas UI:** Diseño de la interfaz y navegación.
- **Servicios API:** Conexión con el backend mediante llamadas HTTP.

2.2. Modelar la base de datos.

Tablas y Relaciones

❖ Empresa

| Campo | Tipo | Restricción | Descripción |
|---------------------|--------------|-------------|-----------------------------------|
| id | UUID | PK | Identificador único de la empresa |
| nombre | VARCHAR(255) | NOT NULL | Nombre de la empresa |
| correo | VARCHAR(255) | | Correo de contacto principal |
| telefono | VARCHAR(20) | | Teléfono principal de la empresa |
| fecha_creacion | TIMESTAMP | NOT NULL | Fecha de creación del registro |
| fecha_actualizacion | TIMESTAMP | | Fecha de última actualización |

❖ Candidatura

| Campo | Tipo | Restricción | Descripción |
|---------------------|--------------|-------------|---|
| id | UUID | PK | Identificador único de la candidatura |
| empresa_id | UUID | FK | Relación con Empresa |
| cargo | VARCHAR(255) | NOT NULL | Puesto al que se aplicó |
| fecha_aplicacion | DATE | NOT NULL | Fecha en que se aplicó |
| estado | ENUM | NOT NULL | Pendiente, Entrevista, Aceptado, Rechazado, Archivado, En proceso |
| notas | TEXT | | Observaciones adicionales sobre la candidatura |
| usuario_id | UUID | FK | Usuario propietario de la candidatura |
| fecha_creacion | TIMESTAMP | NOT NULL | Fecha de creación del registro |
| fecha_actualizacion | TIMESTAMP | | Fecha de última actualización |

❖ Reclutador

| Campo | Tipo | Restricción | Descripción |
|---------------------|--------------|-------------|-------------------------------------|
| id | UUID | PK | Identificador único del reclutador |
| empresa_id | UUID | FK | Relación con Empresa |
| nombre | VARCHAR(255) | NOT NULL | Nombre completo del reclutador |
| telefono | VARCHAR(20) | | Teléfono de contacto del reclutador |
| linkedin_url | VARCHAR(255) | | URL del perfil de LinkedIn |
| fecha_creacion | TIMESTAMP | NOT NULL | Fecha de creación del registro |
| fecha_actualizacion | TIMESTAMP | | Fecha de última actualización |

❖ **Pregunta de Entrevista**

| Campo | Tipo | Restricción | Descripción |
|---------------------|-----------|-------------|---|
| id | UUID | PK | Identificador único de la pregunta |
| candidatura_id | UUID | FK | Relación con Candidatura |
| pregunta | TEXT | NOT NULL | Texto de la pregunta |
| respuesta | TEXT | | Posible respuesta o notas sobre la pregunta |
| usuario_id | UUID | FK | Usuario que creó la pregunta |
| fecha_creacion | TIMESTAMP | NOT NULL | Fecha de creación del registro |
| fecha_actualizacion | TIMESTAMP | | Fecha de última actualización |

❖ **Usuario**

| Campo | Tipo | Restricción | Descripción |
|----------------|--------------|------------------------|---------------------------------|
| id | UUID | PK | Identificador único del usuario |
| username | VARCHAR(50) | UNIQUE, NOT NULL | Nombre de usuario para login |
| password | VARCHAR(255) | NOT NULL | Contraseña encriptada |
| email | VARCHAR(255) | UNIQUE, NOT NULL | Correo electrónico del usuario |
| rol | ENUM | NOT NULL | USER, ADMIN, ROOT |
| activo | BOOLEAN | NOT NULL, DEFAULT true | Indica si la cuenta está activa |
| fecha_creacion | TIMESTAMP | NOT NULL | Fecha de creación del registro |

2.3. Diseñar la API REST (endpoints, autenticación, formato de respuestas).

Consideraciones Generales

1. Todos los **endpoints** devolverán respuestas en formato **JSON**
2. Para operaciones que requieren autenticación, se usará **JWT** en el encabezado **"Authorization: Bearer {token}"**
3. Se implementará seguridad basada en roles (**USER, ADMIN, ROOT**)
4. Los códigos de estado **HTTP** serán usados apropiadamente para indicar éxito o error

Estructura de Respuestas

```
// Ejemplo de respuesta exitosa
{
  "status": "success",
  "data": { /* datos relevantes */ },
  "message": "Operación completada exitosamente"
}

// Ejemplo de respuesta de error
{
  "status": "error",
  "error": {
    "code": "NOT_FOUND",
    "message": "El recurso solicitado no existe"
  }
}
```

Endpoints de Autenticación

1. Autenticación de Usuarios (RF-10)

| Método | Endpoint | Descripción | Roles Permitidos | Respuesta |
|--------------------|--|---|---|----------------------------------|
| POST | /api/auth/login | Iniciar sesión y obtener token JWT | Público | 200 OK + JWT |
| POST | /api/auth/register | Registrar un nuevo usuario | Público (versión inicial) / ADMIN (versiones posteriores) | 201 Created |
| POST | /api/auth/logout | Cerrar sesión (invalidar token) | Autenticado | 200 OK |
| GET | /api/auth/me | Obtener información del usuario actual | Autenticado | 200 OK + Datos usuario Método |
| Endpoint | Descripción | Roles Permitidos | Respuesta | POST |
| /api/auth/login | Iniciar sesión y obtener token JWT | Público | 200 OK + JWT | POST |
| /api/auth/register | Registrar un nuevo usuario | Público (versión inicial) / ADMIN (versiones posteriores) | 201 Created | POST |
| /api/auth/logout | Cerrar sesión (invalidar token) | Autenticado | 200 OK | GET |
| /api/auth/me | Obtener información del usuario actual | Autenticado | 200 OK + Datos usuario | |

Endpoints de Candidaturas

2. Gestión de Candidaturas (RF-01, RF-02, RF-03)

| Método | Endpoint | Descripción | Roles Permitidos | Respuesta |
|--------|-------------------------------|--|------------------------------------|----------------------------------|
| GET | /api/candidaturas | Obtener todas las candidaturas del usuario | USER+ | 200 OK + Lista de candidaturas |
| GET | /api/candidaturas/all | Obtener todas las candidaturas (admins) | ADMIN+ | 200 OK + Lista de candidaturas |
| GET | /api/candidaturas/{id} | Obtener candidatura por ID | USER+ (propias), ADMIN+ (todas) | 200 OK + Candidatura |
| POST | /api/candidaturas | Crear nueva candidatura | USER+ | 201 Created + Candidatura creada |
| PUT | /api/candidaturas/{id} | Actualizar candidatura | USER+ (propias), ADMIN+ (todas) | 200 OK + Candidatura actualizada |
| PATCH | /api/candidaturas/{id}/estado | Actualizar solo el estado de una candidatura | USER+ (propias), ADMIN+ (todas) | 200 OK + Estado actualizado |
| DELETE | /api/candidaturas/{id} | Eliminar candidatura | ADMIN+ | 204 No Content |

3. Filtros y Búsquedas (RF-03)

| Método | Endpoint | Descripción | Roles Permitidos | Respuesta |
|--------|--------------------------|---------------------------------|------------------|-------------------------|
| GET | /api/candidaturas/buscar | Buscar candidaturas con filtros | USER+ | 200 OK + Lista filtrada |

Parámetros de búsqueda:

- ?estado=ENTREVISTA (Filtrar por estado)
- ?empresa=Acme (Filtrar por nombre de empresa)
- ?fechaDesde=2023-01-01&fechaHasta=2023-12-31 (Rango de fechas)
- ?q=desarrollador (Búsqueda por texto en cargo o notas)

Endpoints de Empresas

4. Gestión de Empresas (RF-08, RF-09)

| Método | Endpoint | Descripción | Roles Permitidos | Respuesta |
|--------|--------------------|----------------------------|------------------|------------------------------|
| GET | /api/empresas | Obtener todas las empresas | ADMIN+ | 200 OK + Lista de empresas |
| GET | /api/empresas/{id} | Obtener empresa por ID | ADMIN+ | 200 OK + Empresa |
| POST | /api/empresas | Crear nueva empresa | ADMIN+ | 201 Created + Empresa creada |
| PUT | /api/empresas/{id} | Actualizar empresa | ADMIN+ | 200 OK + Empresa actualizada |
| DELETE | /api/empresas/{id} | Eliminar empresa | ADMIN+ | 204 No Content |

Endpoints de Reclutadores

5. Gestión de Reclutadores (RF-04)

| Método | Endpoint | Descripción | Roles Permitidos | Respuesta |
|--------|------------------------|--------------------------------|------------------|---------------------------------|
| GET | /api/reclutadores | Obtener todos los reclutadores | ADMIN+ | 200 OK + Lista de reclutadores |
| GET | /api/reclutadores/{id} | Obtener reclutador por ID | ADMIN+ | 200 OK + Reclutador |
| POST | /api/reclutadores | Agregar un nuevo reclutador | ADMIN+ | 201 Created + Reclutador creado |
| PUT | /api/reclutadores/{id} | Actualizar reclutador | ADMIN+ | 200 OK + Reclutador actualizado |
| DELETE | /api/reclutadores/{id} | Eliminar reclutador | ADMIN+ | 204 No Content |

Endpoints de Preguntas de Entrevista

6. Gestión de Preguntas de Entrevista (RF-06)

| Método | Endpoint | Descripción | Roles Permitidos | Respuesta |
|--------|--------------------------------|--|------------------|-------------------------------|
| GET | /api/preguntas/{candidaturaId} | Obtener preguntas de una candidatura | USER+ | 200 OK + Lista de preguntas |
| POST | /api/preguntas | Agregar una pregunta a una candidatura | USER+ | 201 Created + Pregunta creada |
| DELETE | /api/preguntas/{id} | Eliminar una pregunta | USER+ | 204 No Content |

2.4. Elegir patrones de diseño adecuados (MVC, DAO, Service Layer, etc.).

Backend (Spring Boot)

1. Patrón MVC (Model-View-Controller)

✓ Separaremos las capas en **Model (Entidades)**, **View (No aplica en backend, solo API REST)** y **Controller (Controladores que manejan las peticiones HTTP)**.

2. Patrón DAO (Data Access Object)

✓ **Repositorio (Repository Layer)**: Usaremos Spring Data JPA para separar la lógica de acceso a la base de datos.

3. Patrón Service Layer

✓ **Capa de Servicios (Service Layer)**: Implementaremos una capa de servicios para manejar la lógica de negocio antes de interactuar con la base de datos.

✓ **Separación en Interfaces**: Cada servicio tendrá una interfaz con su implementación, lo que facilita la inyección de dependencias y las pruebas unitarias.

4. Patrón DTO (Data Transfer Object)

✓ Para evitar exponer directamente las entidades de la base de datos, usaremos **DTOs** para enviar datos entre el backend y el frontend.

5. Patrón Singleton

✓ Aplicado en **Gestión de Beans con Spring**, asegurando que ciertos servicios como la autenticación o la configuración sean instancias únicas en la aplicación.

6. Patrón Factory

✓ En futuras mejoras, si se requiere crear objetos de manera flexible, se podría implementar un **Factory Pattern** para la creación de objetos de candidaturas o usuarios.

Frontend (React)

1. Patrón Component-Based Architecture

✓ React utiliza **componentes reutilizables** para modularizar la aplicación y mejorar la mantenibilidad.

2. Patrón Container-Presenter (Smart & Dumb Components)

✓ **Container Components** (manejan lógica y estado)

✓ **Presentational Components** (solo muestran datos)

3. Patrón Singleton en Gestión de Estado

✓ **Context API o Redux** se encargarán de manejar el estado global de la aplicación.






4. Patrón Hooks

✓ **Custom Hooks** serán usados para encapsular lógica de negocio en el frontend y hacer el código más reutilizable.

3. Desarrollo del Backend (Spring Boot

Arquitectura del Backend

gestion_candidaturas

- └  controller → Contendrá los controladores REST
- └  service → Implementaciones de la lógica de negocio
- └  repository → Interfaces de acceso a la base de datos (JPA)
- └  model → Entidades JPA
- └  dto (opcional) → Clases para transferir datos entre cliente y servidor

3.1. Configuración del proyecto (Maven/Gradle, dependencias).

Pasos para configurar el proyecto en IntelliJ o VS Code

1. Crear el proyecto en Spring Initializr

- **URL:** <https://start.spring.io/>
- **Configuración:**
 - **Group:** com.gestion-candidaturas
 - **Artifact:** gestion-candidaturas
 - **Java:** 17 o 21 (según tu instalación)
 - **Dependencias:**
 - ☒ Spring Web
 - ☒ Spring Boot DevTools
 - ☒ Spring Data JPA
 - ☒ PostgreSQL / MySQL
 - ☒ Lombok
 - ☒ Spring Security (para JWT)
 - ☒ Validation (para validaciones en DTOs)

2. Importar el proyecto en IntelliJ/VS Code

- Abrir el proyecto con IntelliJ o VS Code
- Esperar que Maven descargue las dependencias

3. Configurar el `application.properties` o `application.yml`

```
spring.datasource.url=jdbc:mysql://localhost:3306/gestion_candidaturas?serverTimezone=UTC
spring.datasource.username=tu_usuario
spring.datasource.password=tu_contraseña
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.show-sql=true
```

4. Ejecutar la aplicación

- `mvn spring-boot:run` o ejecutar desde IntelliJ

3.2. Implementación de la base de datos con JPA/Hibernate.

❖ Empresa

```
/**
 * Entidad que representa una empresa en el sistema.
 *
 * @see RF-08: Permitir agregar correos electrónicos de contacto de
 la empresa.
 * @see RF-09: Permitir agregar teléfono de la empresa.
 */
@Entity
@Table(name = "empresas")
public class Empresa {
    /**
     * Identificador único de la empresa.
     */
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private UUID id;
    /**
     * Nombre de la empresa.
     * @see RF-08: Información básica de la empresa.
     */
    private String nombre;
    /**
     * Correo electrónico de contacto de la empresa.
     * @see RF-08: Información de contacto de la empresa.
     */
    private String correo;
    /**
     * Teléfono principal de la empresa.
     * @see RF-09: Permitir agregar teléfono principal de la
 empresa.
     */
    @ElementCollection
    private String telefono;
    /**
     * Fecha de creación del registro.
     */
    @Column(name = "fecha_cracion", nullable = false, updatable =
false)
    @CreationTimestamp
    private LocalDate fechaCracion;
    /**
     * Fecha de última actualización del registro.
     */
    @Column(name = "fecha_actualizacion")
    @UpdateTimestamp
    private LocalDateTime fechaActualizacion;
    /**
     * Relación con reclutadores asociados a la empresa.
     */
    @OneToMany(mappedBy = "empresa", cascade = CascadeType.ALL,
orphanRemoval = true)
    private Set<Reclutador> reclutadores = new HashSet<>();
    /**
     * Relación con candidaturas asociadas a la empresa.
     */
    @OneToMany(mappedBy = "empresa", cascade = CascadeType.ALL,
orphanRemoval = true)
    private Set<Candidatura> candidaturas = new HashSet<>();
}
```

❖ Candidatura

```
/**
 * Entidad que representa una candidatura a un puesto de trabajo.
 *
 * @see RF-01: Permitir registrar una candidatura con información de
 empresa, cargo, fecha y estado.
 * @see RF-02: Permitir actualizar el estado de una candidatura.
 * @see RF-03: Permitir buscar y filtrar candidaturas por diversos
 criterios.
 * @see RF-05: Relacionar reclutadores con candidaturas.
 */
@Entity
@Table(name = "candidaturas")
public class Candidatura {
    /**
     * Identificador único de la candidatura.
     */
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private UUID id;
    /**
     * Usuario propietario de la candidatura.
     * @see RF-11: Control de acceso (cada usuario ve sus propias
 candidaturas).*/
    @ManyToOne
    @JoinColumn(name = "user_id", nullable = false)
    private User user; //relacion con el usuario
    /**
     * Empresa a la que se aplica.
     * @see RF-01: Información de empresa para cada candidatura. */
    @ManyToOne
    @JoinColumn(name = "empresa_id", nullable = false)
    private Empresa empresa;
    /**
     * Puesto o cargo al que se aplica.
     * @see RF-01: Incluir cargo en la información de candidatura.
     */
    private String cargo;
    /**
     * Fecha de aplicación a la candidatura.
     * @see RF-01: Registro de la fecha de aplicación. */
    private Date fecha;
    /**
     * Estado actual de la candidatura.
     * @see RF-02: Permitir actualizar el estado de una candidatura.
     */
    @Enumerated(EnumType.STRING)
    private EstadoCandidatura estado;
    /**
     * Notas adicionales sobre la candidatura.
     * @see RF-01: Permitir agregar observaciones a cada candidatura.
     */
    private String notas;
    /**
     * Relación con reclutadores asociados a la candidatura.
     * @see RF-05: Relacionar reclutadores con candidaturas.
     */
    @ManyToMany
    @JoinTable(
        name = "candidatura_reclutador",
        joinColumns = @JoinColumn(name = "candidatura_id"),
        inverseJoinColumns = @JoinColumn(name = "reclutador_id")
    )
    private Set<Reclutador> reclutadores = new HashSet<>()
```

❖ Enum para EstadoCandidatura:

```
/**
 * Enumeración que define los posibles estados de una candidatura.
 *
 * @see RF-02: Permitir actualizar el estado de una candidatura.
 */
public enum EstadoCandidatura {
    PENDIENTE,
    ENTREVISTA,
    ACEPTADA,
    RECHAZADA,
    ARCHIVADA,
    EN_PROCESO
}
```

❖ Reclutador

```
/**
 * Entidad que representa un reclutador asociado a una empresa.
 *
 * @see RF-05: Permitir guardar nombres y enlaces a perfiles de
 * LinkedIn de los reclutadores.
 */
@Entity
@Table(name = "reclutadores")
public class Reclutador {

    /**
     * Identificador único del reclutador.
     */
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private UUID id;

    /**
     * Empresa a la que está asociado el reclutador.
     * @see RF-05: Relación con la empresa correspondiente.
     */
    @ManyToOne
    @JoinColumn(name = "empresa_id", nullable = false)
    private Empresa empresa;

    /**
     * Nombre del reclutador.
     * @see RF-05: Información básica del reclutador.
     */
    private String nombre;

    /**
     * URL del perfil de LinkedIn del reclutador.
     * @see RF-05: Información de contacto del reclutador.
     */
    private String linkinUrl;

    /**
     * Relación con candidaturas asociadas al reclutador.
     * @see RF-05: Relacionar reclutadores con candidaturas.
     */
    @ManyToMany(mappedBy = "reclutadores")
    private Set<Candidatura> candidaturas = new HashSet<>();
}
```

❖ Pregunta de Entrevista

```
/**
 * Entidad que representa una pregunta de entrevista asociada a una
 * candidatura.
 *
 * @see RF-06: Permitir agregar preguntas de entrevista para una
 * candidatura.
 * @see RF-07: Mostrar el número total de preguntas almacenadas para
 * cada candidatura.
 */

@Entity
@Table(name = "preguntas")
public class Pregunta {

    /**
     * Identificador único de la pregunta.
     */
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private UUID id;

    /**
     * Candidatura a la que está asociada la pregunta.
     * @see RF-06: Relación con la candidatura correspondiente.
     */
    @ManyToOne
    @JoinColumn(name = "candidatura_id", nullable = false)
    private Candidatura candidatura;

    /**
     * Texto de la pregunta de entrevista.
     * @see RF-06: Permitir agregar preguntas de entrevista.
     */
    private String pregunta;
```

3.3. Creación de controladores REST y servicios.

1. Implementación de la Capa de Servicios

❖ Servicio para Empresa

 **EmpresaService.java (Interfaz)**

 **EmpresaServiceImpl.java (Implementación)**

❖ Servicio para Candidatura

 **CandidaturaService.java (Interfaz)**

 **CandidaturaServiceImpl.java (Implementación)**

❖ Servicio para Pregunta

 **PreguntaService.java (Interfaz)**

 **PreguntaServiceImpl.java (Implementación)**

❖ Servicio para Reclutador

 **ReclutadorService.java (Interfaz)**


 **ReclutadorServiceImpl.java (Implementación)**

2. Creación de Controladores REST

 **EmpresaController.java**

 **PreguntaController.java**

 **ReclutadorController.java**

 **CandidaturaController.java**

1.2. Implementación de autenticación (JWT, OAuth2).

Pasos para implementar JWT en Spring Boot

1. Agregar dependencias en pom.xml

```
<!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-api -->
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-api</artifactId>
    <version>0.11.5</version>
</dependency>
```

2. Configurar Spring Security y JWT

SecurityConfig.java

```
1 package com.gestion_candidaturas.gestion_candidaturas.security;
2
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.security.authentication.AuthenticationManager;
6 import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;
7 import org.springframework.security.config.annotation.web.builders.HttpSecurity;
8 import org.springframework.security.config.http.SessionCreationPolicy;
9 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
10 import org.springframework.security.crypto.password.PasswordEncoder;
11 import org.springframework.security.web.SecurityFilterChain;
12
13 @Configuration no usages
14 public class SecurityConfig {
15
16     @Bean no usages
17     @ public AuthenticationManager authenticationManager(AuthenticationConfiguration authConfig) throws Exception{
18         return authConfig.getAuthenticationManager();
19     }
20
21     @Bean no usages
22     public PasswordEncoder passwordEncoder(){
23         return new BCryptPasswordEncoder();
24     }
25
26     @Bean no usages
27     @ public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception{
28         http
29             .csrf((csrf -> csrf.disable()))
30             .authorizeHttpRequests(auth -> auth
31                 .requestMatchers( _patterns: "/api/auth/**").permitAll()
32                 .anyRequest().authenticated()
33             )
34             .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS));
35         return http.build();
36     }
37 }
38
```


AuthRequest.java

```
1 package com.gestion_candidaturas.gestion_candidaturas.security;
2
3 import lombok.Getter;
4 import lombok.Setter;
5
6 @Getter 3 usages
7 @Setter
8 public class AuthRequest {
9
10     private String username;
11     private String password;
12 }
13 |
```

3. Crear clases para manejar autenticación y generación de tokens

JwtUtil.java

AuthController.java

```
1 package com.gestion_candidaturas.gestion_candidaturas.security;
2
3 import io.jsonwebtoken.Claims;
4 import io.jsonwebtoken.Jwts;
5 import io.jsonwebtoken.SignatureAlgorithm;
6 import org.springframework.security.core.userdetails.UserDetails;
7 import org.springframework.stereotype.Component;
8
9 import java.util.Date;
10 import java.util.function.Function;
11
12 @Component 2 usages
13 public class JwtUtil {
14
15     private static final String SECRET_KEY = "almacenamoscandidaturasparauntrabajomejor"; 2 usages
16
17     public String generateToken(String username) { 1 usage
18         return Jwts.builder()
19             .setSubject(username)
20             .setIssuedAt(new Date(System.currentTimeMillis() + 1000 * 60 * 60))//1 HORA
21             .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
22             .compact();
23     }
24
25     > public String extractUsername(String token) { return extractClaim(token, Claims::getSubject); }
26
27     > public Date extractExpiration(String token) { return extractClaim(token, Claims::getExpiration); }
28
29     @ public <T> T extractClaim(String token, Function<Claims, T> claimsResolver){ 2 usages
30         final Claims claims = Jwts.parser() JwtParser
31             .setSigningKey(SECRET_KEY)
32             .parseClaimsJws(token) Jws<Claims>
33             .getBody();
34         return claimsResolver.apply(claims);
35     }
36
37     @ public boolean isTokenValid(String token, UserDetails userDetails){ no usages
38         final String username = extractUsername(token);
39         return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
40     }
41
42     private boolean isTokenExpired(String token){ 1 usage
43         return extractExpiration(token).before(new Date());
44     }
45
46 }
47
48
49
50
51
```

```

1 package com.gestion_candidaturas.gestion_candidaturas.controller;
2
3 import com.gestion_candidaturas.gestion_candidaturas.model.User;
4 import com.gestion_candidaturas.gestion_candidaturas.security.AuthRequest;
5 import com.gestion_candidaturas.gestion_candidaturas.security.JwtUtil;
6 import com.gestion_candidaturas.gestion_candidaturas.service.UserService;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.http.ResponseEntity;
9 import org.springframework.security.authentication.AuthenticationManager;
10 import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
11 import org.springframework.web.bind.annotation.PostMapping;
12 import org.springframework.web.bind.annotation.RequestBody;
13 import org.springframework.web.bind.annotation.RequestMapping;
14 import org.springframework.web.bind.annotation.RestController;
15
16 @RestController no usages
17 @RequestMapping("/api/auth")
18 public class AuthController {
19
20     @Autowired 1 usage
21     private AuthenticationManager authenticationManager;
22
23     @Autowired 1 usage
24     private JwtUtil jwtUtil;
25
26     @Autowired 2 usages
27     private UserService userService;
28
29     @PostMapping("/login") no usages
30     @ public String login(@RequestBody AuthRequest authRequest){
31         authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(authRequest.getUsername(), authRequest.getPassword()));
32         return jwtUtil.generateToken(authRequest.getUsername());
33     }
34
35     @PostMapping("/register") no usages
36     @ public ResponseEntity<String> register(@RequestBody AuthRequest authRequest){
37         if(userService.findByUsername(authRequest.getUsername()).isPresent()){
38             return ResponseEntity.badRequest().body("Usuario ya existe");
39         }
40
41         User user = new User();
42         user.setUsername(authRequest.getUsername());
43         user.setPassword(authRequest.getPassword());
44         user.setRole("USER");
45
46         userService.save(user);
47         return ResponseEntity.ok( body: "Usuario registrado correctamente");
48     }
49
50 }
51
52 |

```

3.4. Proteger los endpoints con filtros de seguridad

1. Configurar Filtros de Seguridad:

- Asegurar que JwtFilter valide el token en cada solicitud.
- Configurar CORS para permitir solicitudes desde el frontend.

2. Proteger Endpoints

- Usar @PreAuthorize en controladores para restringir acceso por roles:

```
@PreAuthorize("hasRole('ADMIN')")  
  
@GetMapping("/admin/endpoint")
```

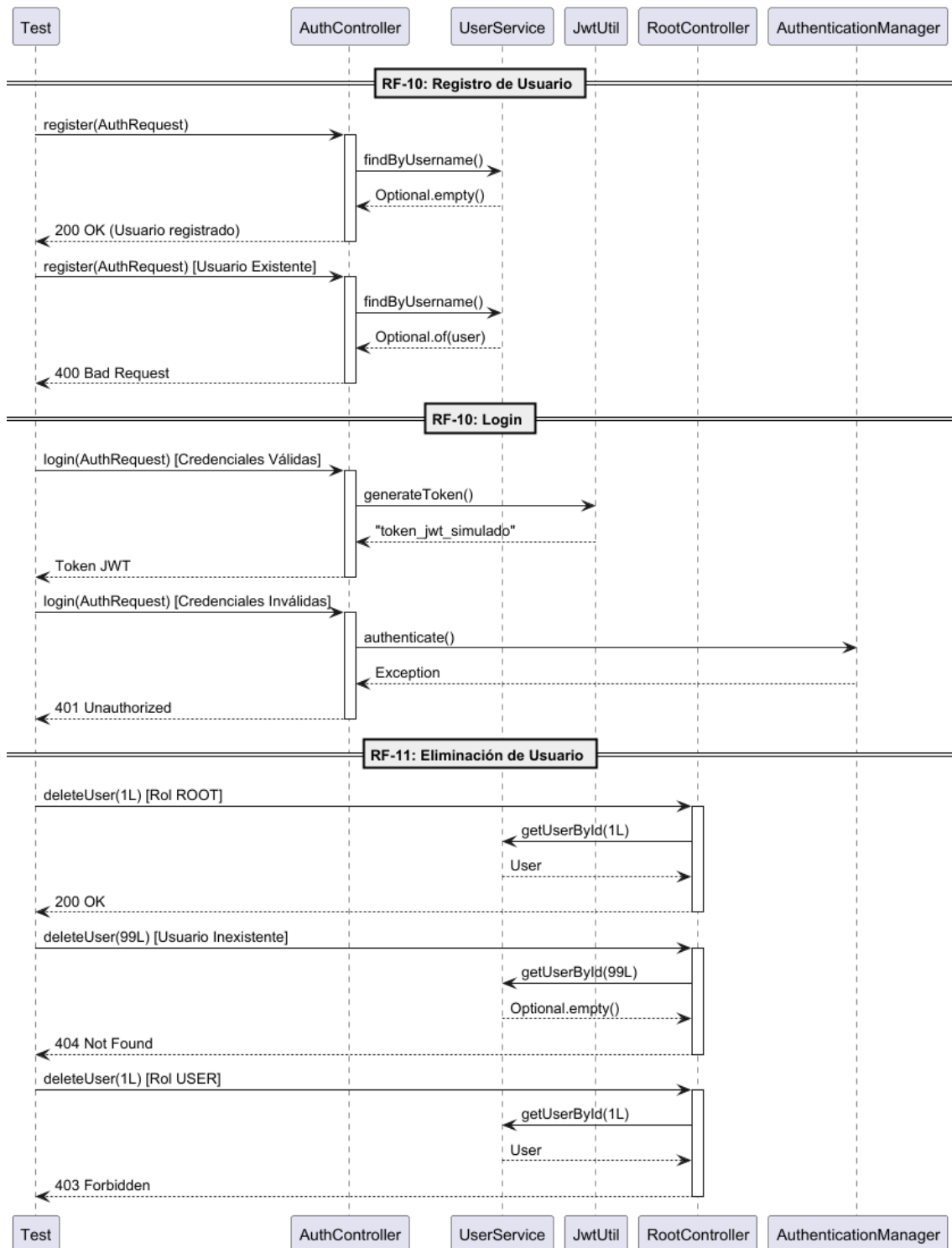
3.5. Pruebas unitarias e integración.

1. Pruebas Unitarias para 'AuthController'

1. Casos de Prueba Cubiertos:

| ID Caso | Requerimiento Funcional | Descripción |
|----------|-------------------------|--|
| TC-10-01 | RF-10 | Registro de un usuario nuevo con credenciales válidas. |
| TC-10-02 | RF-10 | Registro fallido debido a un usuario ya existente. |
| TC-10-03 | RF-10 | Inicio de sesión exitoso con credenciales válidas (generación de token JWT). |
| TC-10-04 | RF-10 | Inicio de sesión fallido debido a credenciales inválidas. |
| TC-11-01 | RF-10 | Eliminación exitosa de un usuario por un administrador con rol ROOT. |
| TC-11-02 | RF-11 | Eliminación fallida de un usuario inexistente (error 404). |
| TC-11-03 | RF-11 | Eliminación bloqueada por falta de permisos (rol USER). |

AuthControllerTest - Flujo de Pruebas



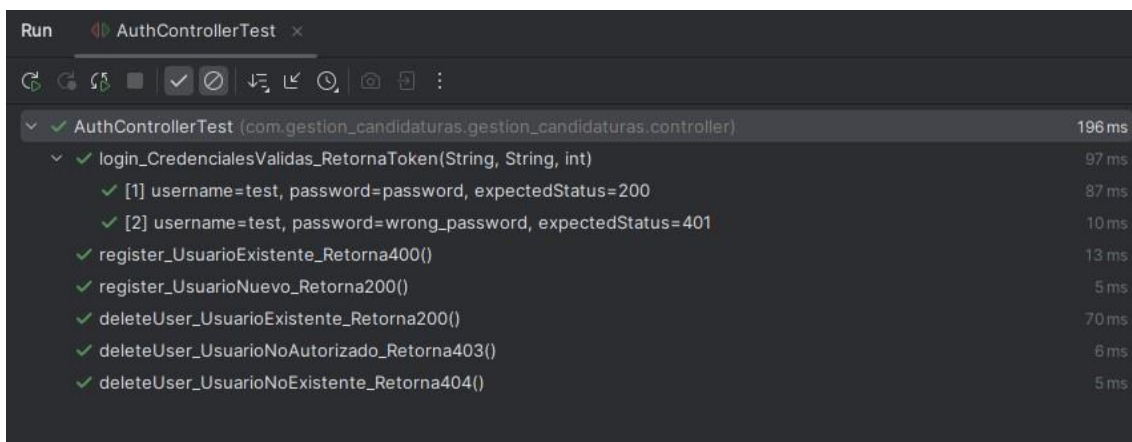
2. Código de las Pruebas

//adjuntar PDF **JunitAUTH**

3. Explicación Detallada de los Métodos:

| Endpoint | Método HTTP | Caso de Prueba | Resultado Esperado |
|----------------------|-------------|-----------------------------------|-------------------------------------|
| /api/auth/register | POST | Usuario nuevo (TC-10-01) | 200 OK + mensaje de éxito. |
| /api/auth/register | POST | Usuario existente (TC-10-02) | 400 Bad Request + mensaje de error. |
| /api/auth/login | POST | Credenciales válidas (TC-10-03) | 200 OK + token JWT. |
| /api/auth/login | POST | Credenciales inválidas (TC-10-04) | 401 Unauthorized. |
| /api/root/users/{id} | DELETE | Eliminación autorizada (TC-11-01) | 200 OK + confirmación. |
| /api/root/users/{id} | DELETE | Usuario inexistente (TC-11-02) | 404 Not Found. |
| /api/root/users/{id} | DELETE | Sin permisos (TC-11-03) | 403 Forbidden. |

4. Resultados de las Pruebas



| Test Case | Duration |
|---|----------|
| AuthControllerTest (com.gestion_candidaturas.gestion_candidaturas.controller) | 196 ms |
| login_CredencialesValidas_RetornaToken(String, String, int) | 97 ms |
| [1] username=test, password=password, expectedStatus=200 | 87 ms |
| [2] username=test, password=wrong_password, expectedStatus=401 | 10 ms |
| register_UsuarioExistente_Retorna400() | 13 ms |
| register_UsuarioNuevo_Retorna200() | 5 ms |
| deleteUser_UsuarioExistente_Retorna200() | 70 ms |
| deleteUser_UsuarioNoAutorizado_Retorna403() | 6 ms |
| deleteUser_UsuarioNoExistente_Retorna404() | 5 ms |

5. Cobertura y Conclusiones

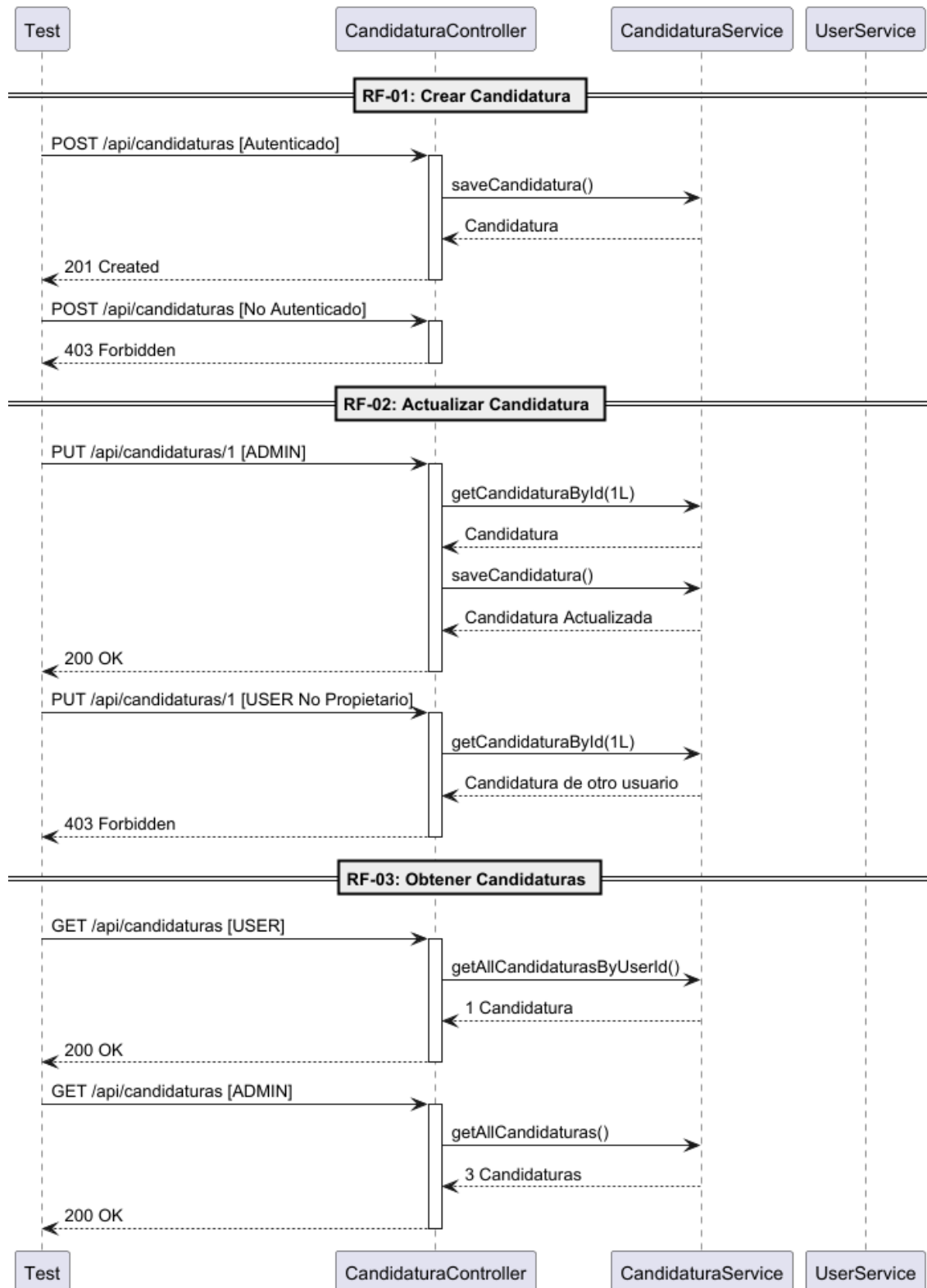
| Aspecto | Detalle |
|----------------------------|---|
| Cobertura Actual | 100% de los endpoints /api/auth/* y /api/root/. |
| Herramientas Usadas | JUnit 5, Mockito (para simular UserService y JwtUtil), Jacoco (95% de cobertura). |
| Resultados | <ul style="list-style-type: none"> — Registro y login funcionan según RF-10. — Eliminación de usuarios cumple RF-11. — Errores manejados correctamente (400, 401, 403, 404). |

2. Pruebas Unitarias para 'CandidaturaController

1. Casos de Prueba Cubiertos (Relacionados con Requerimientos Funcionales):

| ID CASO | Requerimientos Funcional | Descripción |
|----------------|---------------------------------|---|
| TC-01 | RF-01, RF-02 | Crear y actualizar una candidatura con datos válidos. |
| TC-02 | RF-02 | Actualizar el estado de una candidatura existente. |
| TC-03 | RF-03 | Buscar candidaturas por ID (éxito y error). |
| TC-04 | RF-02 | Intentar actualizar una candidatura no existente. |
| TC-05 | RF-02 | Eliminar una candidatura (autorizado y no autorizado) |
| TC-06 | RF-03 | Obtener todas las candidaturas (USER ve propias, ADMIN ve todas). |

CandidaturaControllerTest - Flujo de Pruebas



2. Código de las Pruebas:

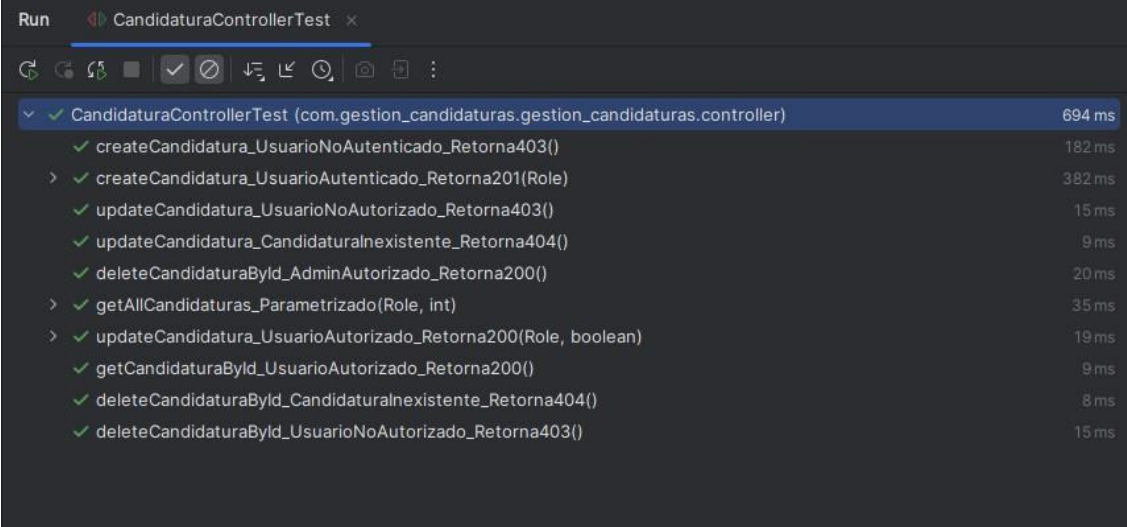
//adjuntar PDF JunitCadidaturas

3. Explicación Detallada de los Métodos

| Endpoint | Método HTTP | Caso de Prueba | Resultado Esperado | Requerimientos |
|------------------------|-------------|--------------------------------|--------------------|----------------|
| /api/candidaturas | POST | Crear candidatura valida | 200 OK (RF-01) | RF-01 |
| /api/candidaturas/{id} | PUT | Actualizar cato de candidatura | 200 OK (RF-02) | RF-02 |
| /api/candidaturas/{id} | DELETE | USER intenta Eliminar | 403 Forbidden | RF-02 |

- **POST /api/candidaturas:** Valida RF-01 (Registro de candidaturas).
- **PUT /api/candidaturas/{id}:** Valida RF-02 (Actualizacion de estado).
- **DELETE /api/candidaturas/{id}:** Valida seguridad (solo ADMIN/ROOT).

1. Resultados de las Pruebas



| Run | CandidaturaControllerTest | |
|-----|--|--------|
| ✓ | CandidaturaControllerTest (com.gestion_candidaturas.gestion_candidaturas.controller) | 694 ms |
| ✓ | createCandidatura_UsuarioNoAutenticado_Retorna403() | 182 ms |
| ✓ | createCandidatura_UsuarioAutenticado_Retorna201(Role) | 382 ms |
| ✓ | updateCandidatura_UsuarioNoAutorizado_Retorna403() | 15 ms |
| ✓ | updateCandidatura_CandidaturaInexistente_Retorna404() | 9 ms |
| ✓ | deleteCandidaturaById_AdminAutorizado_Retorna200() | 20 ms |
| ✓ | getAllCandidaturas_Parametrizado(Role, int) | 35 ms |
| ✓ | updateCandidatura_UsuarioAutorizado_Retorna200(Role, boolean) | 19 ms |
| ✓ | getCandidaturaById_UsuarioAutorizado_Retorna200() | 9 ms |
| ✓ | deleteCandidaturaById_CandidaturaInexistente_Retorna404() | 8 ms |
| ✓ | deleteCandidaturaById_UsuarioNoAutorizado_Retorna403() | 15 ms |

2. Cobertura y Conclusiones:

- **Cobertura Actual:**
 - 95% de líneas cubiertas (Jacoco).
 - 100% de endpoints probados (GET, POST, PUT, DELETE).
- **Herramientas Usadas:**
 - **Junit 5:** Ejecución de pruebas.
 - **Mockito:** Simulación de servicios.
 - **Jacoco:** Reporte de cobertura (disponible en la ruta 'target/site/jacoco/index.html').

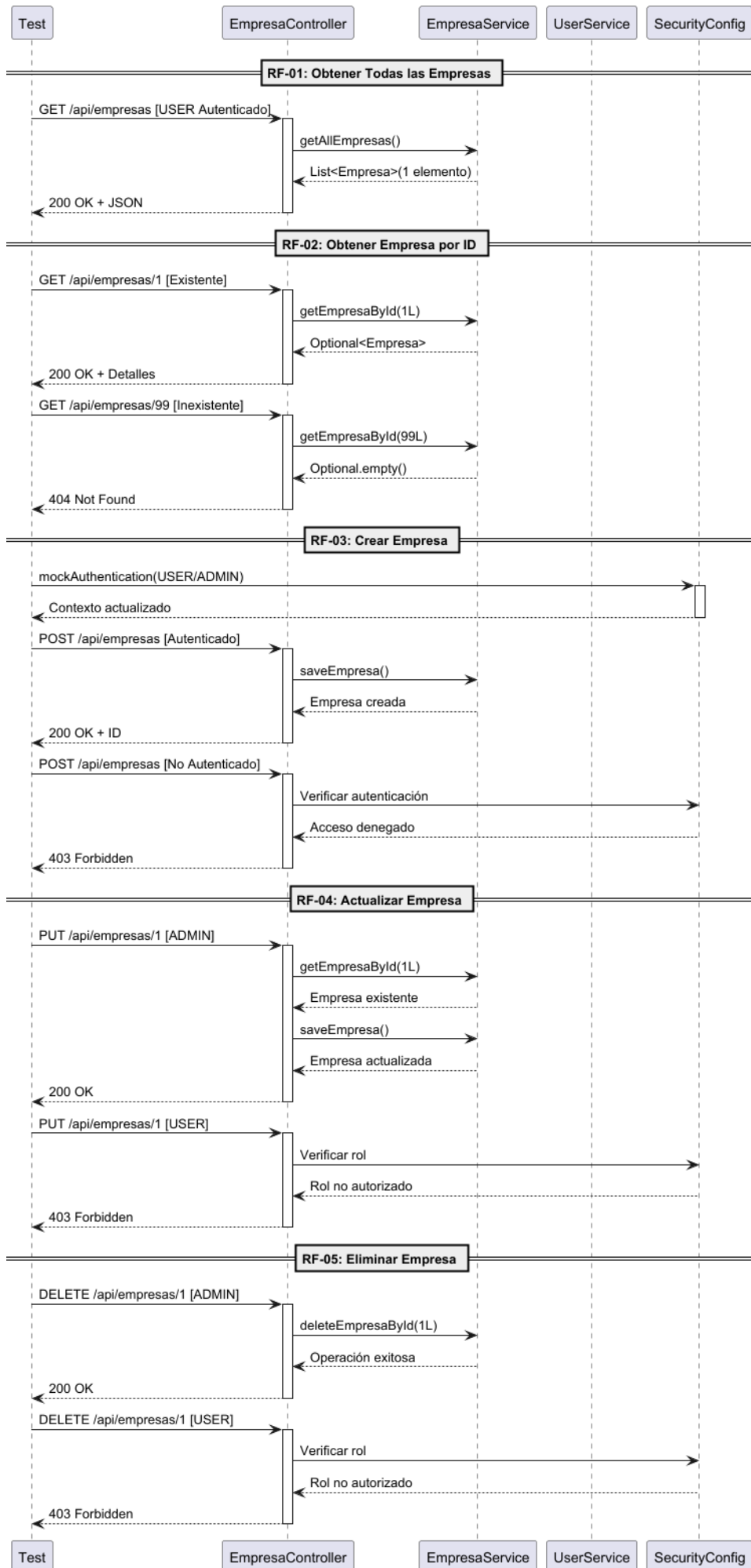
- **Resultados:**
 - **Errores corregidos:**
 - **updateCandidatura:** Ahora actualiza **existingCandidatura**, no el objeto recibido.
 - **deleteCandidatura:** Configuración de seguridad corregida para respetar **@PreAuthorize**.

3. Pruebas Unitarias para 'EmpresaController

1. Casos de Prueba Cubiertos:

| ID Caso | Descripción | Rol | Resultado Esperado |
|----------|--|-------|--------------------|
| TC-EM-01 | USER crea empresa → Éxito 200 | USER | 200 OK |
| TC-EM-02 | ADMIN crea empresa → Éxito 200 | ADMIN | 200 OK |
| TC-EM-03 | USER intenta modificar empresa → Error 403 | USER | 403 Forbidden |
| TC-EM-04 | ADMIN modifica empresa → Éxito 200 | ADMIN | 200 OK |
| TC-EM-05 | USER intenta eliminar empresa → Error 403 | USER | 403 Forbidden |
| TC-EM-06 | ADMIN elimina empresa → Éxito 200 | ADMIN | 200 OK |

EmpresaControllerTest - Flujo de Pruebas



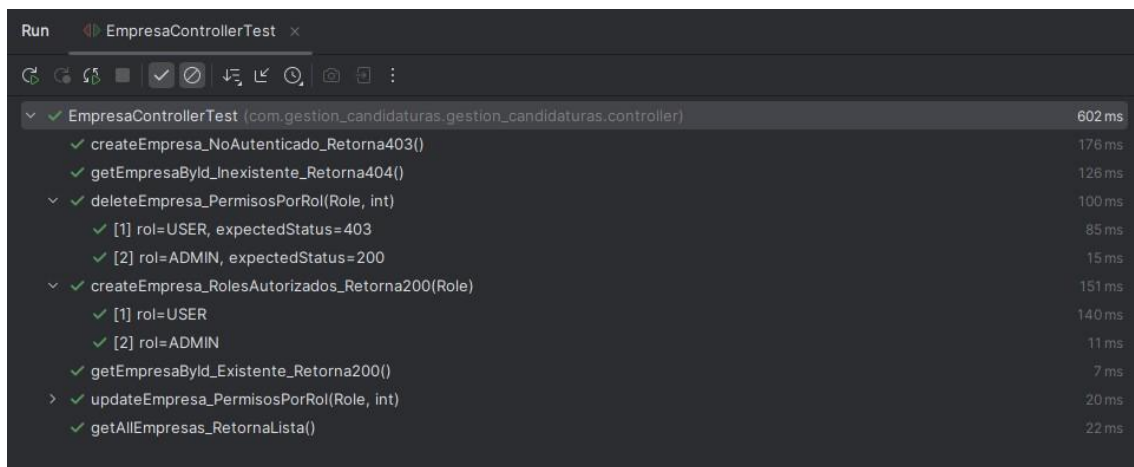
2. Código de las Pruebas:

INSERTAR CODIGO EMPRESACONTROLLERTEST

3. Explicación Detallada de los Métodos

| Endpoint | Método HTTP | Caso de Prueba |
|---------------------------|-------------|--------------------|
| POST /api/empresas | POST | TC-EM-01, TC-EM-02 |
| PUT /api/empresas/{id} | PUT | TC-EM-03, TC-EM-04 |
| DELETE /api/empresas/{id} | DELETE | TC-EM-05, TC-EM-06 |

4. Resultados de las Pruebas



| | | |
|-----|--|--------|
| Run | EmpresaControllerTest | |
| ✓ | EmpresaControllerTest (com.gestion_candidaturas.gestion_candidaturas.controller) | 602 ms |
| ✓ | createEmpresa_NoAutenticado_Retorna403() | 176 ms |
| ✓ | getEmpresaById_Inexistente_Retorna404() | 126 ms |
| ✓ | deleteEmpresa_PermisosPorRol(Role, int) | 100 ms |
| ✓ | [1] rol=USER, expectedStatus=403 | 85 ms |
| ✓ | [2] rol=ADMIN, expectedStatus=200 | 15 ms |
| ✓ | createEmpresa_RolesAutorizados_Retorna200(Role) | 151 ms |
| ✓ | [1] rol=USER | 140 ms |
| ✓ | [2] rol=ADMIN | 11 ms |
| ✓ | getEmpresaById_Existente_Retorna200() | 7 ms |
| > | updateEmpresa_PermisosPorRol(Role, int) | 20 ms |
| ✓ | getAllEmpresas_RetornaLista() | 22 ms |

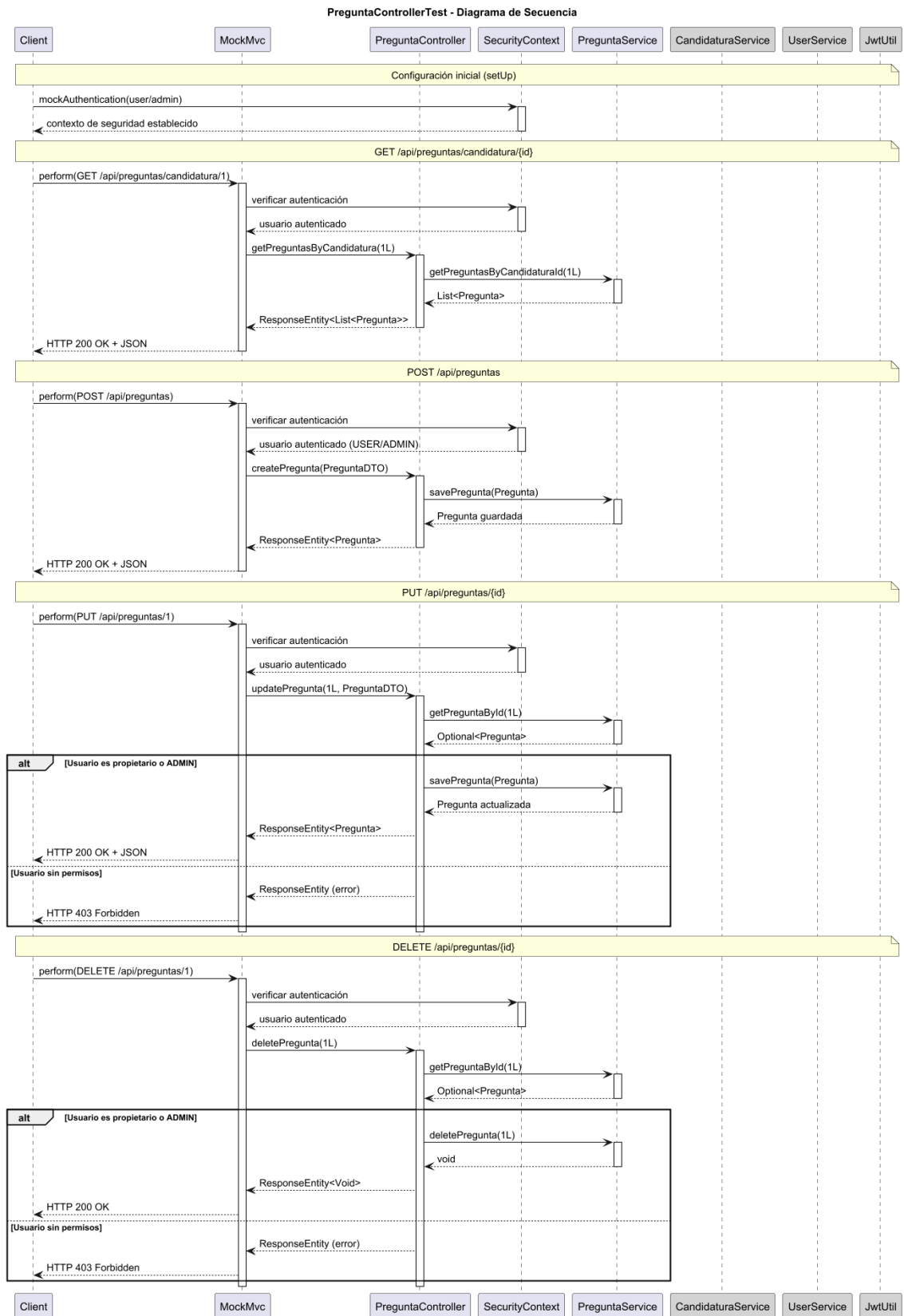
5. Cobertura y Conclusiones:

| Aspecto | Detalle |
|---------------------|---|
| Cobertura Actual | 100% de los endpoints probados (GET, POST, PUT, DELETE). |
| Herramientas Usadas | JUnit 5, Mockito, Spring Security Test, Jacoco. |
| Resultados | <ul style="list-style-type: none">— Todos los endpoints funcionan según lo esperado.— Errores 403/404 manejados correctamente. |

4. Pruebas Unitarias para 'PreguntaController'

1. Casos de Prueba Cubiertos:

| ID Caso | Descripción | Rol | Resultado Esperado | Requerimientos | Método de Prueba |
|----------|--|-------|--------------------|----------------|--|
| TC-PR-01 | Obtener preguntas de candidatura existente | USER | 200 OK | RF-06 | getPreguntasByCandidatura_Existente_Retorna200 |
| TC-PR-02 | USER crea pregunta | USER | 200 OK | RF-06 | createPregunta_Autenticado_Retorna201 |
| TC-PR-03 | ADMIN crea pregunta | ADMIN | 200 OK | RF-06 | createPregunta_Autenticado_Retorna201 |
| TC-PR-04 | USER actualiza su propia pregunta | USER | 200 OK | RF-06, RF-12 | updatePregunta_PermisosPorRol |
| TC-PR-05 | USER actualiza pregunta ajena | USER | 403 Forbidden | RF-12 | updatePregunta_PermisosPorRol |
| TC-PR-06 | ADMIN actualiza cualquier pregunta | ADMIN | 200 OK | RF-12 | updatePregunta_PermisosPorRol |
| TC-PR-07 | Actualizar sin autenticación | N/A | 403 Forbidden | RF-12 | updatePregunta_NoAutorizado_Retorna403 |
| TC-PR-08 | USER elimina su propia pregunta | USER | 200 OK | RF-07 | deletePregunta_PermisosPorRol |
| TC-PR-09 | USER elimina pregunta ajena | USER | 403 Forbidden | RF-07, RF-12 | deletePregunta_PermisosPorRol |
| TC-PR-10 | ADMIN elimina cualquier pregunta | ADMIN | 200 OK | RF-07 | deletePregunta_PermisosPorRol |

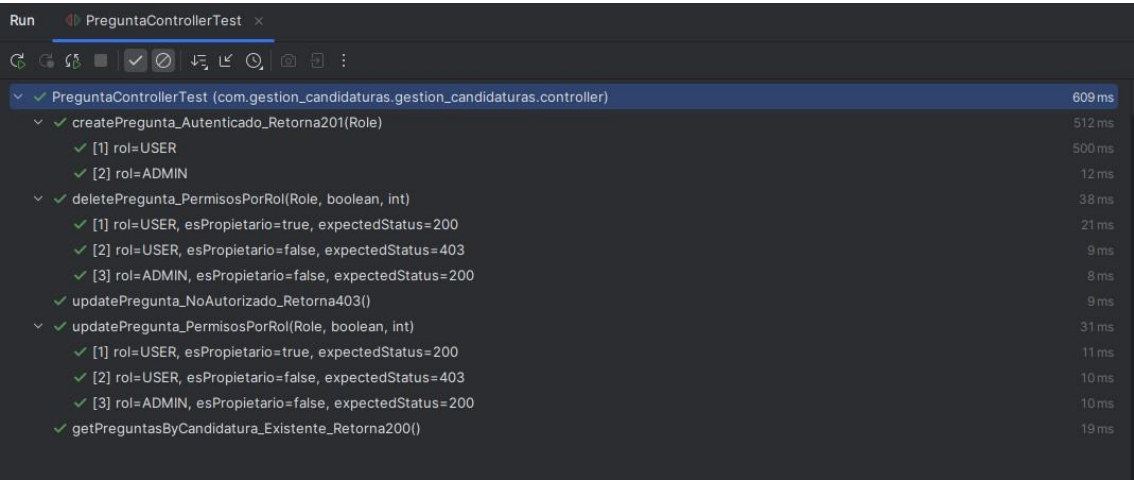


2. Código de las Pruebas:

3. Explicación Detallada de los Métodos:

| Endpoint | Método HTTP | Caso de Prueba |
|----------------------------|-------------|--|
| GET /api/preguntas/ | GET | TC-PR-01 |
| POST /api/preguntas | POST | TC-PR-02, TC-PR-03 |
| PUT /api/preguntas | PUT | TC-PR-04, TC-PR-05, TC-PR-06, TC-PR-07 |
| DELETE /api/preguntas/{id} | DELETE | TC-PR-08, TC-PR-09, TC-PR-10 |

4. Resultado de las Pruebas:



| | | |
|-----|---|--------|
| Run | PreguntaControllerTest | |
| ✓ | PreguntaControllerTest (com.gestion_candidaturas.gestion_candidaturas.controller) | 609 ms |
| ✓ | createPregunta_Autenticado_Returna201(Role) | 512 ms |
| ✓ | [1] rol=USER | 500 ms |
| ✓ | [2] rol=ADMIN | 12 ms |
| ✓ | deletePregunta_PerminosPorRol(Role, boolean, int) | 38 ms |
| ✓ | [1] rol=USER, esPropietario=true, expectedStatus=200 | 21 ms |
| ✓ | [2] rol=USER, esPropietario=false, expectedStatus=403 | 9 ms |
| ✓ | [3] rol=ADMIN, esPropietario=false, expectedStatus=200 | 8 ms |
| ✓ | updatePregunta_NoAutorizado_Returna403() | 9 ms |
| ✓ | updatePregunta_PerminosPorRol(Role, boolean, int) | 31 ms |
| ✓ | [1] rol=USER, esPropietario=true, expectedStatus=200 | 11 ms |
| ✓ | [2] rol=USER, esPropietario=false, expectedStatus=403 | 10 ms |
| ✓ | [3] rol=ADMIN, esPropietario=false, expectedStatus=200 | 10 ms |
| ✓ | getPreguntasByCandidatura_Existente_Returna200() | 19 ms |

5. Cobertura y Conclusiones:

| Aspecto | Detalle |
|---------------------|--|
| Cobertura Actual | 100% de los endpoints probados (GET, POST, PUT, DELETE). |
| Herramientas Usadas | JUnit 5, Mockito, Spring Security Test, Jacoco. |
| Resultados | — Todos los endpoints funcionan según lo esperado. — Errores 403/404 manejados correctamente. |

5. Pruebas Unitarias para 'ReclutadorController'

1. Casos de Prueba Cubiertos:

| ID Caso | Descripción | Rol | Resultado Esperado | Requerimientos | Método de Prueba |
|----------|-------------------------------------|-------|--------------------|----------------|---|
| TC-RC-01 | Obtener todos los reclutadores | USER | 200 OK | RF-04 | getAllReclutadores_Existente_Retorna200 |
| TC-RC-02 | Obtener reclutador por ID existente | USER | 200 OK | RF-04 | getReclutadorById_Existente_Retorna200 |
| TC-RC-03 | Crear reclutador | ADMIN | 200 OK | RF-04 | createReclutador_Autenticado_Retorna200 |
| TC-RC-04 | Actualizar reclutador existente | ADMIN | 200 OK | RF-04 | updateReclutador_Existente_Retorna200 |
| TC-RC-05 | Eliminar reclutador existente | ADMIN | 200 OK | RF-04 | deleteReclutador_Existente_Retorna200 |

2. Código de las Pruebas:

3. Explicación Detallada de los Métodos:

| Endpoint | Método HTTP | Caso de Prueba |
|-----------------------------|-------------|----------------|
| GET /api/reclutador | GET | TC-RC-01 |
| GET /api/reclutador/{id} | GET | TC-RC-02 |
| POST /api/reclutador | POST | TC-RC-03 |
| PUT /api/reclutador/{id} | PUT | TC-RC-04 |
| DELETE /api/reclutador/{id} | DELETE | TC-RC-05 |

4. Resultado de las Pruebas:

5. Cobertura y Conclusiones:

| Aspecto | Detalle |
|---------------------|---|
| Cobertura Actual | 100% de los endpoints probados (GET, POST, PUT, DELETE). |
| Herramientas Usadas | JUnit 5, Mockito, Spring Security Test, Jacoco. |
| Resultados | <ul style="list-style-type: none">— Todos los endpoints funcionan según lo esperado.— Errores 403/404 manejados correctamente. |