


1. Contenido

1.	Contenido	1
2.	Análisis y Planificación.....	3
2.1.	Definir objetivos y alcance del sistema.	3
2.2.	Identificar requerimientos funcionales y no funcionales.....	5
2.3.	Identificar usuarios y casos de uso.	8
✓	 Usuario Principal.....	8
2.4.	Evaluar tecnologías a utilizar.	9
3.	Diseño de la Arquitectura	10
3.1.	Definir la arquitectura general del sistema (monolito vs microservicios).	10
3.2.	Modelar la base de datos.	11
3.3.	Diseñar la API REST (endpoints, autenticación, formato de respuestas).....	13
✓	Endpoints de Autenticación	14
✓	Endpoints de Candidaturas	15
✓	Endpoints de Empresas	16
✓	Endpoints de Reclutadores.....	17
✓	Endpoints de Preguntas de Entrevista	17
3.4.	Elegir patrones de diseño adecuados (MVC, DAO, Service Layer, etc.).....	18
4.	Desarrollo del Backend (Spring Boot).....	20
4.1.	Configuración del proyecto (Maven/Gradle, dependencias).	21
✓	Crear el proyecto en Spring Initializr	21
✓	Importar el proyecto en IntelliJ/VS Code	21
✓	Configurar el application.properties o application.yml	21
✓	Ejecutar la aplicación	21
4.2.	Implementación de la base de datos con JPA/Hibernate.	22
✓	Entidades Principales	22
✓	Enumeraciones	24
✓	Estrategias JPA/Hibernate implementadas	25
✓	Ejemplo de implementación (fragmento)	25
4.3.	Creación de controladores REST y servicios.....	27
✓	Implementación de la capa servicios.....	27
✓	Implementación de controlares REST	28
✓	Patrón de diseño implementado.....	28
✓	Ejemplo Representativo	29
4.4.	Implementación de autenticación (JWT, OAuth2).....	32
✓	Estructura de la implementación JWT.....	32
✓	Flujo de autenticación	32
✓	Componentes Clave	32
✓	Protección de endpoints con filtros de seguridad	33
4.5.	Pruebas unitarias e integración.	35
✓	Estrategia de pruebas	¡Error! Marcador no definido.
✓	Herramientas utilizadas	36
✓	Casos de prueba para componente.....	¡Error! Marcador no definido.
✓	Resultados de cobertura	¡Error! Marcador no definido.
✓	Beneficios de la estrategia de pruebas	¡Error! Marcador no definido.

✓	Configuración de pruebas	¡Error! Marcador no definido.
✓	Conclusiones	43
4.6.	Documentación de la API	44
5.	Desarrollo del Frontend (React).....	64
5.1.	Configuración del entorno de desarrollo.....	64
5.2.	Diseño de la UI/UX.	64
5.3.	Implementación de componentes y navegación.	64
5.4.	Conexión con la API del backend.....	64
5.5.	Pruebas de usabilidad.	64
6.	Desarrollo de la App Móvil (Android)	64
6.1.	Definir si será nativa (Kotlin/Jetpack Compose) o híbrida (React Native).	64
6.2.	Implementación de la interfaz y lógica de negocio.....	64
6.3.	Conexión con el backend.	64
6.4.	Pruebas en dispositivos reales.	64
7.	Pruebas y Depuración	64
7.1.	Pruebas unitarias y de integración en backend y frontend.	64
7.2.	Pruebas de usuario.	64
7.3.	Optimización de código y rendimiento.	64
7.4.	Revisión de código.	64
8.	Despliegue (si aplica).....	64
8.1.	Configuración de hosting para la web.	64
8.2.	Distribución de la app Android en dispositivos de prueba o Play Store.	64
8.3.	Monitoreo post-despliegue.	64
9.	Conclusión.....	64
9.1.	Reflexiones sobre el proyecto.....	64
9.2.	Lecciones aprendidas.....	64
9.3.	Posibles mejoras futuras.....	65

2. Análisis y Planificación

2.1. 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.

2.2. 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
 - Los usuarios pueden crear y actualizar reclutadores durante el proceso de gestión de candidaturas
 - Los usuarios solo pueden modificar reclutadores asociados a sus propias candidaturas
 - Los administradores (ADMIN y ROOT) pueden gestionar todos los reclutadores del sistema
 - ✓ **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 y gestionar información de las empresas.

- Los usuarios pueden crear nuevas empresas durante el proceso de registro de candidaturas.
- Los usuarios pueden agregar correos electrónicos de contacto de la empresa.
- Los usuarios pueden modificar información de las empresas que hayan creado.
- Los administradores (ADMIN y ROOT) pueden gestionar todas las empresas del sistema

✓ **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.

2.3. 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.
CU-12	Crear empresa durante registro de candidatura	Permite al usuario registrar una nueva empresa en el sistema durante el proceso de registro de una candidatura.
CU-13	Crear reclutador durante gestión de candidaturas	Permite al usuario registrar un nuevo reclutador en el sistema durante el proceso de gestión de candidaturas.

2.4. 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)

3. Diseño de la Arquitectura

3.1. 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.

3.2. Modelar la base de datos.

Tablas y Relaciones

1. 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

2. 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

3. 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

4. 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

5. 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

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

Consideraciones Generales

- 1.1.1. Todos los **endpoints** devolverán respuestas en formato **JSON**
- 1.1.2. Para operaciones que requieren autenticación, se usará **JWT** en el encabezado **"Authorization: Bearer {token}"**
- 1.1.3. Se implementará seguridad basada en roles (**USER, ADMIN, ROOT**)
- 1.1.4. 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"
  }
}
```

Manejo de errores en la API

La API implementa un sistema consistente de manejo de errores, desarrollando códigos HTTP apropiados según la situación:

- **400 Bad Request:** Se utiliza cuando la solicitud contiene dato inválidos o incompletos (validación de campos, conflicto de unicidad en nombre de usuario o correo electrónico).
- **401 Unauthorized:** Se devuelve cuando no hay autenticación válida o las credenciales son incorrectas.
- **403 Prohibido:** Se retorna cuando el usuario está autenticado pero no tiene permisos para acceder al recurso.
- **404 No encontrado:** Se utiliza cuando el recurso solicitado no existe.
- **500 Error interno del servidor:** Se devuelve para errores inesperados del servidor.

Validación de Datos en el Registro de Usuarios

El sistema implemente validaciones rigurosas en el proceso de registro:

1. **Verificación de unicidad:** Valida que el nombre de usuario y correo electrónico no existen previamente en el sistema.
2. **Validación de formato:**
 - a. El nombre de usuario debe tener entre 3 y 50 caracteres.
 - b. La contraseña debe tener un mínimo de 6 caracteres.
 - c. El correo electrónico debe tener un formato válido.
3. **Roles:** Por defecto se asignan roles **"USUARIO"** si no se especifica.

✓ 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 + { "token": "eyJhbGciOiJIUzI...", "id": "550e8400-e29b-41d4-a716-446655440000", "username": "usuario1", "email": "usuario@example.com", "role": "USER" } 401 Unauthorized (credenciales inválidas)
POST	/api/auth/register	Registrar un nuevo usuario	Público (versión inicial) / ADMIN (versiones posteriores)	201 Created + { "message": "Usuario registrado correctamente", "username": "usuario1", "role": "USER" } 400 Bad Request + { "Error": "El nombre de usuario ya está en uso" } 400 Bad Request + { "Error": "El email ya está en uso" }
POST	/api/auth/logout	Cerrar sesión (invalidar token)	Autenticado	200 OK + { "message": "Sesión cerrada correctamente" }
GET	/api/auth/me	Obtener información del usuario actual	Autenticado	200 OK + { "id": "550e8400-e29b-41d4-a716-446655440000", "username": "usuario1", "email": "usuario@example.com", "role": "USER" } 401 Unauthorized (usuario no autenticado)

✓ Endpoints de Candidaturas

1. 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

2. 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

1. Gestión de Empresas (RF-08, RF-09)

Método	Endpoint	Descripción	Roles Permitidos	Respuesta
GET	/api/empresas	Obtener todas las empresas (usuarios ven las propias, admins ven todas)	USER+	200 OK + Lista de empresas
GET	/api/empresas/with-users	Obtener empresas con información de usuarios asociados	ADMIN+	200 OK + Lista de empresas con usuarios asociados
GET	/api/empresas/{id}	Obtener empresa por ID	USER+	200 OK + Empresa
POST	/api/empresas	Crear nueva empresa (solo admins)	ADMIN+	201 Created + Empresa creada
POST	/api/empresas/crear-con-candidatura	Crear nueva empresa durante registro de candidatura o retornar existente	USER+	201 Created + Empresa creada o 200 OK + Empresa existente
PUT	/api/empresas/{id}	Actualizar empresa	USER+ (propias empresas) o ADMIN+	200 OK + Empresa actualizada
DELETE	/api/empresas/{id}	Eliminar empresa	ADMIN+	204 No Content

✓ Endpoints de Reclutadores

1. 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
POST	/api/reclutadores/crear-con-candidatura	Crear nuevo reclutador durante gestión de candidaturas o retornar existente	USER+	201 Created + Reclutador creado o 200 OK + Reclutador existente
PUT	/api/reclutadores/{id}/user-update	Actualizar reclutador asociado a candidaturas del usuario	USER+	200 OK + Reclutador actualizado o 403 Forbidden

✓ Endpoints de Preguntas de Entrevista

1. 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

3.4. 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.

4. 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

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

Pasos para configurar el proyecto en IntelliJ o VS Code

✓ 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)

✓ Importar el proyecto en IntelliJ/VS Code

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

✓ Configurar el application.properties o application.yml

```
spring.application.name=gestion-candidaturas

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

✓ Ejecutar la aplicación

- mvn spring-boot:run o ejecutar desde IntelliJ

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

La implementación de la capa de persistencia se realizó utilizando JPA/Hibernate, siguiendo una estructura de entidades que refleja el modelo de datos diseñado en la fase de análisis.

✓ Entidades Principales

Se implementaron las siguientes entidades JPA con sus relaciones correspondientes:

1. Entidad User

Representa a los usuarios del sistema con sus credenciales y roles.

Atributos Principales:

- **`id`** : Identificador único (UUID).
- **`username`** : Nombre de usuario único (String).
- **`password`** : Contraseña encriptada (String)
- **`email`** : Correo electrónico único (String).
- **`roles`** : Rol del usuario – USER, ADMIN, ROOT (Enum).

Relaciones:

- **`OneToMany` con `Candidatura`** : Un usuario puede tener múltiples candidaturas.
- **`OneToMany` con `Pregunta`** : Un usuario puede crear múltiples preguntas.

2. ENTIDAD EMPRESA

Representa las empresas a las que se puede aplicar candidaturas:

Atributos principales:

- **`id`** : Identificador único (UUID).
- **`nombre`** : Nombre de la empresa (String).
- **`correo`** : Correo electrónico de contacto (String).
- **`telefono`** : Teléfono de contacto (String).

Relaciones:

- **`OneToMany` con `Candidatura`** : Una empresa puede tener múltiples candidaturas.
- **`OneToMany` con `Reclutador`** : Una empresa puede tener múltiples reclutadores.

3. ENTIDAD CANDIDATURA

Representa una aplicación a un puesto de trabajo.

Atributos Principales:

- ``id`` : Identificador único (UUID).
- ``cargo`` : Puesto al que se aplica (String).
- ``fecha`` : Fecha de aplicación (Date).
- ``notas`` : Observaciones adicionales (Text).

Relaciones:

- ``ManyToOne` con `User`` : Cada candidatura pertenece a un usuario
- ``ManyToOne` con `Empresa`` : Cada candidatura está asociada a una empresa
- ``ManyToMany` con `Reclutador`` : Una candidatura puede tener múltiples reclutadores
- ``OneToMany` con `Pregunta`` : Una candidatura puede tener múltiples preguntas

4. ENTIDAD RECLUTADOR

Representa a las personas encargadas del proceso de selección en las empresas.

Atributos Principales:

- ``id`` : Identificador único (UUID)
- ``nombre`` : Nombre completo (String)
- ``telefono`` : Teléfono de contacto (String)
- ``linkedinUrl`` : URL del perfil de LinkedIn (String)

Relaciones:

- ``ManyToOne` con `Empresa`` : Cada reclutador pertenece a una empresa
- ``ManyToMany` con `Candidatura`` : Un reclutador puede estar asociado a múltiples candidaturas

5. ENTIDAD PREGUNTA

Representa preguntas de entrevista asociadas a una candidatura.

Atributos Principales:

- ``id``: Identificador único (UUID)
- ``pregunta``: Texto de la pregunta (String)
- ``respuesta``: Posible respuesta o notas (String)

Relaciones:

- ``ManyToOne`` con ``Candidatura``: Cada pregunta pertenece a una candidatura
- ``ManyToOne`` con ``User``: Cada pregunta es creada por un usuario

✓ Enumeraciones

Se implementaron las siguientes enumeraciones para manejar valores constantes:

1. Enum EstadoCandidatura

Define los posibles estados de una candidatura:

- **PENDIENTE**
- **ENTREVISTA**
- **ACEPTADA**
- **RECHAZADA**
- **ARCHIVADA**
- **EN_PROCESO**

2. ENUM ROLE

Define los roles de usuario en el sistema:

- **USER**
- **ADMIN**
- **ROOT**

✓ Estrategias JPA/Hibernate implementadas

1. Generación de identificadores

Se utilizó UUID como tipo de identificador para todas las entidades, generados automáticamente mediante la estrategia `GenerationType.IDENTITY`.

2. Mapeo de relaciones

- Se implementaron relaciones bidireccionales donde fue necesario
- Se utilizó `mappedBy` para indicar el propietario de la relación
- Se configuró `cascade = CascadeType.ALL` para operaciones en cascada donde fue apropiado
- Se usó `orphanRemoval = true` para eliminar registros huérfanos

3. Auditoría de entidades

Se implementaron campos de auditoría en todas las entidades:

- `@CreationTimestamp` para registrar la fecha de creación
- `@UpdateTimestamp` para registrar la fecha de última actualización

4. Validaciones

Se utilizaron anotaciones de validación de **Jakarta Bean Validation**:

- `@NotNull` y `@NotBlank` para campos obligatorios
- `@Size` para limitar longitud de strings
- `@Email` para validar formato de correos
- `@Column(unique = true)` para garantizar unicidad

✓ Ejemplo de implementación (fragmento)

Fragmento representativo de la implementación de la entidad 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<>();

```

4.3. Creación de controladores REST y servicios.

La implementación de la capa de servicios y controladores **REST** se realizó siguiendo los principios de arquitectura en capas y buenas prácticas de diseño **RESTful**.

✓ Implementación de la capa servicios

Se implementaron servicios para cada entidad principal del sistema, siguiendo el patrón de diseño Service Layer:

1. Interfaces de servicio

Para cada entidad se creó una interfaz que define las operaciones disponibles:

- **UserService:** Gestión de usuarios y autenticación
- **EmpresaService:** Gestión de empresas
- **CandidaturaService:** Gestión de candidaturas
- **ReclutadorService:** Gestión de reclutadores
- **PreguntaService:** Gestión de preguntas de entrevista

Este enfoque permite:

- Separar la definición del comportamiento de su implementación
- Facilitar la inyección de dependencias y pruebas unitarias
- Mejorar la mantenibilidad del código

2. Implementaciones de servicio

Cada interfaz de servicio tiene su implementación correspondiente:

- **UserServiceImpl:** Implementa operaciones de usuario y autenticación
- **EmpresaServiceImpl:** Implementa operaciones de empresas
- **CandidaturaServiceImpl:** Implementa operaciones de candidaturas
- **ReclutadorServiceImpl:** Implementa operaciones de reclutadores
- **PreguntaServiceImpl:** Implementa operaciones de preguntas

Características implementadas en los servicios:

- **Transaccionalidad:** Uso de **@Transactional** para garantizar operaciones atómicas
- **Control de acceso:** Verificación de permisos antes de realizar operaciones
- **Validaciones de negocio:** Lógica de validación específica del dominio
- **Manejo de excepciones:** Control de errores y excepciones específicas del negocio

✓ Implementación de controlares REST

Se implementaron controladores **REST** para exponer las operaciones del sistema a través de **HTTP**:

1. Controladores implementados

- **AuthController**: Gestión de autenticación y registro
- **EmpresaController**: Operaciones CRUD para empresas
- **CandidaturaController**: Operaciones CRUD para candidaturas
- **ReclutadorController**: Operaciones CRUD para reclutadores
- **PreguntaController**: Operaciones CRUD para preguntas

2. Características de los controladores

- **Mapeo de rutas**: Uso de anotaciones **@RequestMapping**, **@GetMapping**, **@PostMapping**, etc.
- **Control de acceso**: Uso de **@PreAuthorize** para restricciones basadas en roles
- **Validación de datos**: Uso de **@Valid** para validar objetos de entrada
- **Respuestas HTTP apropiadas**: Uso de **ResponseEntity** para devolver códigos de estado adecuados
- **Manejo de recursos**: Implementación del patrón **REST** para **URLs** y métodos **HTTP**.

✓ Patrón de diseño implementado

La implementación sigue varios patrones de diseño:

1. Patrón MVC adaptado a REST

- **Modelo**: Entidades **JPA** y **DTOs**
- **Vista**: Representaciones **JSON**
- **Controlador**: Controladores **REST**

2. Patrón de repositorio

- Interfaces que extienden **JpaRepository**
- Consultas personalizadas mediante **@Query**
- Métodos derivados de nombres (**query methods**)

3. Patrón de inyección de dependencias

- Inyección por constructor para componentes requeridos
- Uso de **@Autowired** para inyección automática

✓ Ejemplo Representativo

Fragmento de implementación del controlador de candidaturas:

```
/**
 * Controlador REST para operaciones relacionadas con candidaturas.
 *
 * @see RF-01: Permitir registrar candidaturas
 * @see RF-02: Permitir actualizar el estado de una candidatura
 * @see RF-03: Permitir buscar y filtrar candidaturas
 */
@RestController
@RequestMapping("/api/candidaturas")
public class CandidaturaController {

    private final CandidaturaService candidaturaService;
    private final UserService userService;

    /**
     * Constructor para inyección de dependencias.
     *
     * @param candidaturaService Servicio para operaciones con
candidaturas
     * @param userService Servicio para operaciones con usuarios
     */
    public CandidaturaController(CandidaturaService
candidaturaService, UserService userService) {
        this.candidaturaService = candidaturaService;
        this.userService = userService;
    }

    /**
     * Obtiene todas las candidaturas del usuario autenticado.
     *
     * @return Lista de candidaturas del usuario
     *
     * @see RF-03: Visualización de candidaturas
     */
    @GetMapping
    @PreAuthorize("hasRole('USER')")
    public ResponseEntity<List<Candidatura>> getCandidaturas() {
        // Obtener el usuario actual
        User currentUser = userService.getCurrentUser();

        // Obtener candidaturas del usuario
        List<Candidatura> candidaturas =
candidaturaService.findByUserId(currentUser.getId());
        return ResponseEntity.ok(candidaturas);
    }

    /**
     * Obtiene todas las candidaturas del sistema (solo
administradores).
     *
     * @return Lista completa de candidaturas
     *
     * @see RF-03: Visualización de candidaturas (administradores)
     * @see RF-11: Control de acceso basado en roles
     */
    @GetMapping("/all")
    @PreAuthorize("hasAnyRole('ADMIN', 'ROOT')")
    public ResponseEntity<List<Candidatura>> getAllCandidaturas() {
        List<Candidatura> candidaturas = candidaturaService.findAll();
        return ResponseEntity.ok(candidaturas);
    }
}
```

```

/**
 * Obtiene una candidatura específica por su ID.
 * Los usuarios normales solo pueden ver sus propias candidaturas.
 * Los administradores pueden ver cualquier candidatura.
 *
 * @param id ID de la candidatura
 * @return La candidatura si existe y el usuario tiene acceso,
error 404 o 403 en caso contrario
 *
 * @see RF-03: Consulta detallada de candidaturas
 * @see RF-11: Control de acceso basado en roles
 */
@GetMapping("/{id}")
@PreAuthorize("hasAnyRole('USER', 'ADMIN', 'ROOT')")
public ResponseEntity<Candidatura>
getCandidaturaById(@PathVariable UUID id) {
    // Buscar la candidatura
    Optional<Candidatura> candidatura =
candidaturaService.findById(id);

    if (candidatura.isEmpty()) {
        return ResponseEntity.notFound().build();
    }

    // Obtener el usuario actual
    User currentUser = userService.getCurrentUser();

    // Verificar permisos de acceso
    if
(candidatura.get().getUser().getId().equals(currentUser.getId()) ||
        currentUser.hasRole("ADMIN") ||
currentUser.hasRole("ROOT")) {
        return ResponseEntity.ok(candidatura.get());
    } else {
        return
ResponseEntity.status(HttpStatus.FORBIDDEN).build();
    }
}

/**
 * Crea una nueva candidatura asociada al usuario autenticado.
 *
 * @param candidatura Datos de la candidatura a crear
 * @return La candidatura creada
 *
 * @see RF-01: Registro de candidaturas
 */
@PostMapping
@PreAuthorize("hasRole('USER')")
public ResponseEntity<Candidatura> createCandidatura(@Valid
@RequestBody Candidatura candidatura) {
    // Asignar el usuario actual como propietario
    candidatura.setUser(userService.getCurrentUser());

    // Guardar la candidatura
    Candidatura nuevaCandidatura =
candidaturaService.save(candidatura);
    return
ResponseEntity.status(HttpStatus.CREATED).body(nuevaCandidatura);
}
// Más operaciones CRUD

```

Esta implementación de controladores y servicios garantiza:

- **Separación de responsabilidades:** Cada capa tiene una función bien definida
- **Seguridad:** Control de acceso a recursos según roles de usuario
- **Validación:** Integridad de los datos entrantes
- **Coherencia:** Respuestas HTTP estandarizadas
- **Mantenibilidad:** Código bien estructurado y documentado

4.4. Implementación de autenticación (JWT, OAuth2).

✓ Estructura de la implementación JWT.

Se ha implementado un sistema de autenticación basado en JWT (JSON Web Tokens) con las siguientes características:

- **Autenticación sin estado (stateless):** No se almacenan sesiones en el servidor
- **Tokens auto-contenidos:** Contienen toda la información necesaria para la autenticación
- **Espiración configurable:** Los tokens tiene un tiempo de vida limitado (24 horas por defecto)
- **Firma segura:** Utilizan HMAC-SHA512 para garantizar la integridad
- **Control de acceso basado en roles:** USER, ADMIN, ROOT

✓ Flujo de autenticación

1. Registro de usuario:

- Cliente envía credenciales (username, email, password)
- Servidor valida datos y crea nuevo usuario
- Contraseña se guarda encriptada con BCrypt

2. Inicio de sesión:

- Cliente envía credenciales (username, password)
- Servidor valida credenciales
- Si son correctas, genera el token JWT
- Devuelve token al cliente.

3. Acceso a recursos protegidos:

- Cliente incluye token en cabecera Authorization
- Servidor valida el token
- Si es válido, autoriza según rol del usuario

4. Cierre de sesión:

- Cliente elimina el token (del lado del cliente)
- Servidor no necesita hacer nada (autenticación sin estado)

✓ Componentes Clave

JwtUtil: Utilidad para generar y validar tokens

JwtAuthenticationFilter: Intercepta solicitudes y valida tokens

SecurityConfig: Configura reglas de seguridad y filtros

AuthController: Maneja endpoints de autenticación

✓ Protección de endpoints con filtros de seguridad

5. Configuración de filtros de seguridad

Se ha implementado un sistema de protección de endpoints mediante filtros de seguridad que garantiza:

◇ VALIDACIÓN DEL TOKEN JWT EN CADA SOLICITUD HTTP

- Interceptación de todas las solicitudes mediante 'JwtAuthenticationFilter'
- Extracción del token del encabezado 'Authorization'
- Validación de firma y expiración del token
- Autenticación del usuario en el contexto de seguridad

◇ CONFIGURACIÓN CORS PARA PERMITIR SOLICITUDES DESDE EL FRONTEND

- Permite solicitudes desde los orígenes configurados (ej: <http://localhost:3000>)
- Admite métodos HTTP específicos (GET, POST, PUT, PATCH, DELETE, OPTIONS)
- Permite encabezados necesarios como 'Authorization' y 'Content-Type'

6. Protección de endpoints mediante anotaciones

Para proteger los endpoints específicos según los roles de usuario, se ha implementado un sistema de autorización basado en anotaciones:

◇ ANOTACIONES '@PREAUTHORIZE' EN LOS CONTROLADORES

- Restricción de acceso a nivel de método
- Verificación de roles antes de la ejecución del método
- Sintaxis expresiva: '@PreAuthorize("hasRole('ROLE')")'

◇ EJEMPLOS DE IMPLEMENTACIÓN:

```
```java
// Acceso para cualquier usuario autenticado
@PreAuthorize("isAuthenticated()")

// Acceso solo para usuarios con rol USER
@PreAuthorize("hasRole('USER')")

// Acceso para usuarios con rol ADMIN o ROOT
@PreAuthorize("hasAnyRole('ADMIN', 'ROOT')")

// Acceso solo para propietarios del recurso o administradores
@PreAuthorize("@securityService.isOwner(id) or hasRole('ADMIN')")
```

#### ◇ POLÍTICA DE ACCESO POR DEFECTO: TODOS LOS ENDPOINTS REQUIEREN AUTENTICACIÓN EXCEPTO /API/AUTH/LOGIN Y API/AUTH/REGISTER

#### ◇ MANEJO DE EXCEPCIONES DE SEGURIDAD:

- Respuestas 401 (Unauthorized) para solicitudes sin autenticación
- Respuestas 403 (Forbidden) para accesos no autorizados

- Mensajes de error claros para facilitar la depuración

## 4.5. Estrategia de pruebas.

En el proyecto se ha implementado una estrategia de pruebas estructurada en varios niveles para garantizar la calidad y robustez del sistema. Las pruebas unitarias e integración son una parte fundamental para asegurar el correcto funcionamiento de la aplicación, permitiendo identificar errores tempranamente y facilitar el mantenimiento continuo.

### ✓ Niveles de prueba implementados

#### 1. Pruebas unitarias:

Verifican el correcto funcionamiento de componentes individuales

- Clases de utilidad
  - **JwtUtil:** Generación y validación de tokens JWT
- Filtros de seguridad
  - **JwtAuthenticationFilter:** Interceptación y procesamiento de solicitudes HTTP con tokens JWT

#### 2. Pruebas de integración:

Verifican la interacción correcta entre componentes

##### Controladores REST:

- **AuthController:** Autenticación y registro de usuarios
- **CandidaturaController:** Gestión de candidaturas
- **EmpresaController:** Gestión de empresas
- **PreguntaController:** Gestión de preguntas de entrevista
- **ReclutadorController:** Gestión de reclutadores

#### 3. Pruebas de sistema (End-to-End)

Verifican el comportamiento completo del sistema desde la perspectiva del usuario:

##### Flujos completos de usuario;

- Registro y autenticación
- Gestión de empresas
- Ciclo de vida de candidaturas
- Asociación de reclutadores
- Gestión de preguntas de entrevista

## ✓ Herramientas utilizadas

- **JUnit 5:** Framework principal de pruebas unitarias y de integración
- **Mockito:** Framework para crear mocks en pruebas unitarias
- **MockMvc:** Para pruebas de integración de endpoints REST
- **Spring Security Test:** Utilidades para pruebas de seguridad
- **JaCoCo:** Para análisis de cobertura de código
- **Postman:** Para pruebas end-to-end de API
- **Newman:** Herramienta de línea de comandos para automatizar pruebas Postman
- **Reporter HTML Extra:** Generación de reportes detallados para pruebas Postman

## ✓ Pruebas unitarias y de integración

### 1. Casos de prueba para JwtUtil

ID Caso	Descripción	Estado Esperado
JU-01	Generación de token JWT	Éxito: token válido generado
JU-02	Extracción de username de token	Éxito: username correcto extraído
JU-03	Validación de token con usuario correcto	Éxito: token validado
JU-04	Validación de token con usuario incorrecto	Error: token inválido
JU-05	Validación de token expirado	Error: token inválido

### 2. Pruebas para JwtAuthenticationFilter

ID Caso	Descripción	Entrada	Estado Esperado
JF-01	Solicitud con token válido	Authorization con token válido	Autenticación establecida
JF-02	Solicitud sin token	Sin encabezado Authorization	Sin autenticación
JF-03	Solicitud con token inválido	Token con firma incorrecta	Sin autenticación
JF-04	Solicitud con token expirado	Token expirado	Sin autenticación
JF-05	Manejo de excepciones en filtro	Token malformado	Sin autenticación, sin excepción propagada

### 3. Pruebas para AuthController

ID Caso	Descripción	Entrada	Estado Esperado
AC-01	Registro con datos válidos	Username, email y password válidos	201 Created + datos usuario
AC-02	Registro con username existente	Username duplicado	400 Bad Request + error
AC-03	Registro con email existente	Email duplicado	400 Bad Request + error
AC-04	Login con credenciales válidas	Username y password correctos	200 OK + token JWT
AC-05	Login con credenciales inválidas	Password incorrecto	401 Unauthorized + error
AC-06	Obtener información de usuario autenticado	Token válido	200 OK + datos usuario
AC-07	Cierre de sesión	Token válido	200 OK + confirmación

### 4. Pruebas para CandidaturaController

ID	Descripción	Entrada	Estado Esperado
CC-01	Obtener candidaturas del usuario	Usuario autenticado	200 OK + lista de candidaturas
CC-02	Obtener todas las candidaturas (admin)	Admin autenticado	200 OK + lista completa
CC-03	Obtener candidatura por ID existente	ID de candidatura existente	200 OK + detalles
CC-04	Obtener candidatura por ID no existente	ID de candidatura inexistente	404 Not Found
CC-05	Crear candidatura	Datos de candidatura válidos	201 Created + candidatura
CC-06	Actualizar candidatura como propietario	ID y datos válidos	200 OK + candidatura actualizada
CC-07	Actualizar estado de candidatura	ID y nuevo estado	200 OK + candidatura actualizada
CC-08	Buscar candidaturas con filtros	Parámetros de búsqueda	200 OK + candidaturas filtradas

## 5. Pruebas para EmpresaController

ID	Descripción	Entrada	Estado Esperado
EC-01	Obtener todas las empresas	Usuario autenticado	200 OK + lista de empresas
EC-02	Obtener empresa por ID existente	ID de empresa existente	200 OK + detalles
EC-03	Obtener empresa por ID no existente	ID de empresa inexistente	404 Not Found
EC-04	Crear empresa como administrador	Datos de empresa válidos	201 Created + empresa
EC-05	Crear empresa durante registro de candidatura	Datos de empresa válidos	201 Created + empresa
EC-06	Crear empresa con nombre existente durante candidatura	Datos con nombre duplicado	200 OK + empresa existente
EC-07	Obtener empresas con usuarios asociados (admin)	Admin autenticado	200 OK + lista detallada

## 6. Pruebas para PreguntaController

ID	Descripción	Entrada	Estado Esperado
PC-01	Obtener preguntas por candidatura	ID de candidatura	200 OK + lista de preguntas
PC-02	Obtener conteo de preguntas	ID de candidatura	200 OK + número de preguntas
PC-03	Crear pregunta	Datos de pregunta válidos	201 Created + pregunta
PC-04	Actualizar pregunta como propietario	ID y datos válidos	200 OK + pregunta actualizada
PC-05	Actualizar pregunta sin ser propietario	ID y datos válidos	403 Forbidden
PC-06	Actualizar pregunta como administrador	ID y datos válidos	200 OK + pregunta actualizada
PC-07	Actualizar pregunta inexistente	ID inexistente	404 Not Found
PC-08	Eliminar pregunta como propietario	ID de pregunta	204 No Content
PC-09	Eliminar pregunta sin ser propietario	ID de pregunta	403 Forbidden
PC-10	Eliminar pregunta como administrador	ID de pregunta	204 No Content
PC-11	Eliminar pregunta inexistente	ID inexistente	404 Not Found

## 7. Pruebas para ReclutadorController

ID	Descripción	Entrada	Estado Esperado
RC-01	Obtener todos los reclutadores	Usuario autenticado	200 OK + lista de reclutadores
RC-02	Obtener reclutador por ID existente	ID de reclutador existente	200 OK + detalles
RC-03	Obtener reclutador por ID no existente	ID de reclutador inexistente	404 Not Found
RC-04	Obtener reclutadores por empresa	ID de empresa	200 OK + lista de reclutadores
RC-05	Crear reclutador como administrador	Datos de reclutador válidos	201 Created + reclutador
RC-06	Crear reclutador durante gestión de candidatura	Datos de reclutador válidos	201 Created + reclutador
RC-07	Crear reclutador con nombre existente	Datos con nombre duplicado	200 OK + reclutador existente
RC-08	Actualizar reclutador asociado a candidaturas del usuario	ID y datos válidos	200 OK + reclutador actualizado
RC-09	Actualizar reclutador no asociado a candidaturas del usuario	ID y datos válidos	403 Forbidden
RC-10	Actualizar reclutador como administrador	ID y datos válidos	200 OK + reclutador actualizado
RC-11	Actualizar reclutador inexistente	ID inexistente	404 Not Found
RC-12	Eliminar reclutador como administrador	ID de reclutador	204 No Content
RC-13	Eliminar reclutador inexistente	ID inexistente	404 Not Found
RC-14	Asociar reclutador a candidatura	IDs de reclutador y candidatura	200 OK
RC-15	Asociar reclutador o candidatura inexistente	IDs inválidos	404 Not Found
RC-16	Desasociar reclutador de candidatura	IDs de reclutador y candidatura	200 OK
RC-17	Desasociar reclutador o candidatura inexistente	IDs inválidos	404 Not Found

## ✓ Pruebas End-to-End con Postman

Además de las pruebas unitarias y de integración, se han implementado pruebas end-to-end utilizando Postman. Estas pruebas verifican el comportamiento completo del sistema en un entorno integrado y cubren los principales flujos de usuario. **Detección temprana de errores:** Las pruebas unitarias permiten identificar problemas antes de que afecten a otras partes del sistema.

### 1. Colecciones implementadas

Se han desarrollado cinco colecciones de pruebas que cubren todos los aspectos del sistema:

#### ◇ AUTH API TESTS

Esta colección verifica la funcionalidad de autenticación y gestión de usuarios:

ID	Endpoint	Descripción	Resultados
AUTH-01	POST /api/auth/register	Registro de usuario normal	✓ 201 Created + confirmación
AUTH-02	POST /api/auth/register	Registro de usuario administrador	✓ 201 Created + confirmación
AUTH-03	POST /api/auth/login	Inicio de sesión con usuario normal	✓ 200 OK + token JWT
AUTH-04	POST /api/auth/login	Inicio de sesión con administrador	✓ 200 OK + token JWT
AUTH-05	GET /api/auth/me	Obtener información del usuario actual	✓ 200 OK + datos usuario
AUTH-06	GET /api/auth/me	Verificar información del administrador	✓ 200 OK + datos admin
AUTH-07	POST /api/auth/logout	Cerrar sesión	✓ 200 OK + confirmación



## ◇ EMPRESAS API TESTS

Esta colección prueba la gestión de empresas:

ID	Endpoint	Descripción	Resultados
EMP-01	GET /api/empresas	Obtener empresas (usuario normal)	✓ 200 OK + lista
EMP-02	GET /api/empresas	Obtener empresas (administrador)	✓ 200 OK + lista completa
EMP-03	POST /api/empresas	Crear empresa como administrador	✓ 201 Created + empresa
EMP-04	POST /api/empresas/crear-con-candidatura	Crear empresa durante candidatura	✓ 201 Created + empresa
EMP-05	GET /api/empresas/{id}	Obtener empresa por ID	✓ 200 OK + detalles
EMP-06	PUT /api/empresas/{id}	Actualizar empresa como usuario	✓ 200 OK + empresa actualizada
EMP-07	GET /api/empresas/with-users	Obtener empresas con usuarios (solo admin)	✓ 200 OK + lista detallada
EMP-08	GET /api/empresas/buscar	Buscar empresas por nombre	✓ 200 OK + resultados
EMP-09	DELETE /api/empresas/{id}	Eliminar empresa (solo admin)	✓ 204 No Content

## ◇ CANDIDATURAS API TESTS

Esta colección verifica la gestión de candidaturas:

ID	Endpoint	Descripción	Resultados
CAND-01	POST /api/candidaturas	Crear nueva candidatura	✓ 201 Created + candidatura
CAND-02	GET /api/candidaturas	Obtener candidaturas del usuario	✓ 200 OK + lista
CAND-03	GET /api/candidaturas/all	Obtener todas (admin)	✓ 200 OK + lista completa
CAND-04	GET /api/candidaturas/{id}	Obtener candidatura por ID	✓ 200 OK + detalles
CAND-05	PUT /api/candidaturas/{id}	Actualizar candidatura	✓ 200 OK + actualizada
CAND-06	PATCH /api/candidaturas/{id}/estado	Actualizar estado	✓ 200 OK + estado actualizado
CAND-07	GET /api/candidaturas/buscar	Buscar con filtros	✓ 200 OK + resultados filtrados
CAND-08	DELETE /api/candidaturas/{id}	Eliminar candidatura (admin)	✓ 204 No Content

## ◇ PREGUNTAS API TESTS

Esta colección prueba la gestión de preguntas de entrevista:

ID	Endpoint	Descripción	Resultados
<b>PREG-01</b>	POST /api/preguntas	Crear pregunta	✓ 201 Created + pregunta
<b>PREG-02</b>	POST /api/preguntas	Crear pregunta adicional	✓ 201 Created + pregunta
<b>PREG-03</b>	GET /api/preguntas	Obtener preguntas por candidatura	✓ 200 OK + lista
<b>PREG-04</b>	GET /api/preguntas/count	Obtener conteo de preguntas	✓ 200 OK + número
<b>PREG-05</b>	PUT /api/preguntas/{id}	Actualizar pregunta	✓ 200 OK + actualizada
<b>PREG-06</b>	DELETE /api/preguntas/{id}	Eliminar pregunta	✓ 204 No Content

## ◇ RECLUTADORES API TESTS

Esta colección verifica la gestión de reclutadores:

ID	Endpoint	Descripción	Resultados
<b>REC-01</b>	POST /api/reclutador	Crear como admin	✓ 201 Created + reclutador
<b>REC-02</b>	POST /api/reclutador/crear-concandidatura	Crear durante candidatura	✓ 201 Created + reclutador
<b>REC-03</b>	GET /api/reclutador	Obtener todos	✓ 200 OK + lista
<b>REC-04</b>	GET /api/reclutador/{id}	Obtener por ID	✓ 200 OK + detalles
<b>REC-05</b>	GET /api/reclutador/empresa/{id}	Obtener por empresa	✓ 200 OK + lista
<b>REC-06</b>	PUT /api/reclutador/{id}/user-update	Actualizar como usuario	✓ 200 OK + actualizado
<b>REC-07</b>	PUT /api/reclutador/{id}	Actualizar como admin	✓ 200 OK + actualizado
<b>REC-08</b>	POST /api/reclutador/{id}/candidaturas/{id}	Asociar a candidatura	✓ 200 OK
<b>REC-09</b>	DELETE /api/reclutador/{id}/candidaturas/{id}	Desasociar	✓ 200 OK
<b>REC-10</b>	DELETE /api/reclutador/{id}	Eliminar reclutador	✓ 204 No Content

## 2. Flujos completos verificados

Estas colecciones, ejecutadas en secuencia, verifican los siguientes flujos completos de usuario:

### ◇ REGISTRO Y AUTENTICACIÓN DE USUARIOS:

- Registro de nuevos usuarios (normal y administrador)
- Inicio de sesión y obtención de token JWT
- Acceso a recursos protegidos con token válido
- Verificación de permisos según roles

### ◇ GESTIÓN COMPLETA DE CANDIDATURAS:

- Creación de empresa si no existe
- Registro de una nueva candidatura
- Asociación con reclutadores
- Adición de preguntas de entrevista
- Actualización del estado de la candidatura
- Búsqueda de candidaturas con filtros

### ◇ GESTIÓN DE EMPRESAS Y RECLUTADORES:

- Creación y actualización de empresas
- Gestión de reclutadores asociados
- Verificación de permisos según el rol

## ✓ Conclusiones

**La estrategia de pruebas integral implementada demuestra que el sistema funciona correctamente en todos los niveles:**

1. **A nivel unitario:** Los componentes individuales como JwtUtil, filtros de seguridad y servicios funcionan correctamente de forma aislada.
2. **A nivel de integración:** Los controladores REST procesan correctamente las peticiones, aplican la lógica de negocio adecuada e interactúan correctamente con otros componentes.
3. **A nivel de sistema:** Los flujos completos de usuario, desde la autenticación hasta la gestión de todas las entidades, funcionan correctamente en un entorno integrado.

La combinación de pruebas en todos los niveles proporciona una alta confianza en la robustez del sistema y facilita el mantenimiento y evolución futuros. Además, la automatización de las pruebas permitirá detectar rápidamente cualquier regresión introducida durante el desarrollo.

## 4.6. Documentación de la API.

### ✓ Introducción

La API REST de Gestión de Candidaturas proporciona un conjunto completo de endpoints para gestionar el seguimiento de candidaturas a puestos de trabajo, incluyendo información sobre empresas, reclutadores, estados de candidatura y preguntas de entrevista. Esta API está diseñada para ser utilizada tanto por aplicaciones web como móviles.

#### Propósito

El propósito de esta API es proporcionar una interfaz completa y segura para:

- Gestionar usuarios y autenticación
- Crear y gestionar candidaturas a puestos de trabajo
- Administrar información de empresas y reclutadores
- Registrar y consultar preguntas de entrevista
- Realizar búsquedas y filtrados avanzados de candidaturas

### ✓ Información general

#### 1. Base URL

Todas las URLs referenciadas en la documentación tienen la siguiente base:

- **http://[servidor]:[puerto]/api**

Ejemplo para entorno de desarrollo local:

- **http://localhost:8080/api**

#### 2. Formatos de solicitud y respuesta

- **Solicitudes:** Todas las solicitudes POST, PUT y PATCH deben enviarse con el encabezado **Content-Type: application/json** y el cuerpo en formato JSON.
- **Respuestas:** Todas las respuestas se devuelven en formato JSON.

#### 3. Estructura de las respuestas

Las respuestas exitosas generalmente siguen esta estructura:

```
{
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "atributo1": "valor1",
 "atributo2": "valor2",
 ...
}
```

Para respuestas que contienen listas:

```
[
 {
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "atributo1": "valor1",
 ...
 },
 {
 "id": "098f6bcd-4621-3373-8ade-4e832627b4f6",
 "atributo1": "valor2",
 ...
 }
]
```

#### 4. Respuestas de error

En caso de error, la API devolverá un código de estado HTTP adecuado junto con un objeto JSON explicando el error:

```
{
 "status": "error",
 "error": {
 "code": "ERROR_CODE",
 "message": "Descripción detallada del error"
 }
}
```

### ✓ Autenticación y Autorización

La API utiliza autenticación basada en tokens JWT (JSON Web Tokens).

#### 1. Obtención de token

Para obtener un token JWT, se debe realizar una solicitud POST a **/auth/login** con credenciales válidas.

◇ EJEMPLO DE SOLICITUD:

```
POST /api/auth/login
Content-Type: application/json
```

```
{
 "username": "usuario",
 "password": "contraseña"
}
```

◇ EJEMPLO DE RESPUESTA:

```
{
 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6Ikpz...",
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "username": "usuario",
 "email": "usuario@example.com",
 "role": "USER"
}
```

## 2. Uso del token

El token debe incluirse en todas las solicitudes a endpoints protegidos, en el encabezado de autorización:

**Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI...**

## 3. Roles y permisos

La API implementa tres niveles de acceso:

- **USER:** Acceso básico para gestionar sus propias candidaturas
- **ADMIN:** Acceso completo a todas las entidades del sistema
- **ROOT:** Administrador del sistema con permisos especiales

### ✓ Endpoints de Autenticación

#### 1. Registro de usuarios

##### POST /api/auth/register

Crea un nuevo usuario en el sistema.

###### ◇ CUERPO DE LA SOLICITUD:

```
{
 "username": "nuevo_usuario",
 "email": "usuario@example.com",
 "password": "contraseña",
 "rol": "USER" // Opcional, valor por defecto: "USER"
}
```

###### ◇ RESPUESTA EXITOSA (201 CREATED):

```
{
 "message": "Usuario registrado correctamente",
 "username": "nuevo_usuario",
 "role": "USER"
}
```

###### ◇ ERRORES POSIBLES:

- **400 Bad Request:** Nombre de usuario o email ya en uso
- **400 Bad Request:** Validación fallida (formato incorrecto)

## 2. Inicio de sesión

### POST /api/auth/login

Autentica a un usuario y devuelve un token JWT.

#### ◇ CUERPO DE LA SOLICITUD:

```
{
 "username": "usuario",
 "password": "contraseña"
}
```

#### ◇ RESPUESTA EXITOSA (200 OK):

```
{
 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IWR5bmludC0iLCJ1aWQiOiJ550e8400-e29b-41d4-a716-446655440000",
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "username": "usuario",
 "email": "usuario@example.com",
 "role": "USER"
}
```

#### ◇ ERRORES POSIBLES:

- **401 Unauthorized:** Credenciales inválidas

## 3. Información del usuario actual

### GET /api/auth/me

Devuelve información del usuario autenticado.

#### ◇ RESPUESTA EXITOSA (200 OK):

```
{
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "username": "usuario",
 "email": "usuario@example.com",
 "rol": "USER"
}
```

#### ◇ ERRORES POSIBLES:

- **401 Unauthorized:** Usuario no autenticado

#### ◇ CIERRE DE SESIÓN

### POST /api/auth/logout

Cierra la sesión del usuario (invalida el token).

#### ◇ RESPUESTA EXITOSA (200 OK):

```
{
 "message": "Sesión cerrada correctamente"
}
```

## ✓ Endpoints de Empresas

### 1. Obtener todas las empresas

#### GET /api/empresas

Retorna todas las empresas disponibles para el usuario actual. Los usuarios normales ven solo empresas de sus candidaturas, los administradores ven todas.

◇ RESPUESTA EXITOSA (200 OK):

```
[
 {
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "nombre": "Empresa Ejemplo",
 "correo": "contacto@empresa.com",
 "telefono": "912345678"
 },
 ...
]
```

### 2. Obtener empresas con usuarios asociados (sólo admin)

#### GET /api/empresas/with-users

Retorna información detallada de empresas incluyendo usuarios que tienen candidaturas en ellas.

◇ RESPUESTA EXITOSA (200 OK):

```
[
 {
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "nombre": "Empresa Ejemplo",
 "correo": "contacto@empresa.com",
 "telefono": "912345678",
 "usuariosAsociados": [
 {
 "id": "550e8400-e29b-41d4-a716-446655440001",
 "username": "usuario1",
 "numeroCandidaturas": 2
 },
 ...
]
 },
 ...
]
```

### 3. Obtener empresa por ID

#### GET /api/empresas/{id}

Retorna una empresa específica por su ID.

◇ RESPUESTA EXITOSA (200 OK):

```
{
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "nombre": "Empresa Ejemplo",
 "correo": "contacto@empresa.com",
 "telefono": "912345678"
}
```



◇ ERRORES POSIBLES:

- **404 Not Found:** Empresa no encontrada

#### 4. Crear empresa (sólo admin)

##### POST /api/empresas

Crea una nueva empresa en el sistema.

◇ CUERPO DE LA SOLICITUD:

```
{
 "nombre": "Nueva Empresa",
 "correo": "contacto@nuevaempresa.com",
 "telefono": "912345678"
}
```

◇ RESPUESTA EXITOSA (201 CREATED):

```
{
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "nombre": "Nueva Empresa",
 "correo": "contacto@nuevaempresa.com",
 "telefono": "912345678"
}
```

#### 5. Crear empresa durante registro de candidatura

##### POST /api/empresas/crear-con-candidatura

Permite a usuarios normales crear empresas durante el proceso de registro de candidaturas.

◇ CUERPO DE LA SOLICITUD:

```
{
 "nombre": "Nueva Empresa Candidatura",
 "correo": "contacto@empresa.com",
 "telefono": "912345678"
}
```

◇ RESPUESTA EXITOSA (201 CREATED O 200 OK SI YA EXISTE):

```
{
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "nombre": "Nueva Empresa Candidatura",
 "correo": "contacto@empresa.com",
 "telefono": "912345678"
}
```

## 6. Actualizar empresa

### PUT /api/empresas/{id}

Actualiza una empresa existente.

#### ◇ CUERPO DE LA SOLICITUD:

```
{
 "nombre": "Empresa Actualizada",
 "correo": "nuevo@empresa.com",
 "telefono": "987654321"
}
```

#### ◇ RESPUESTA EXITOSA (200 OK):

```
{
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "nombre": "Empresa Actualizada",
 "correo": "nuevo@empresa.com",
 "telefono": "987654321"
}
```

#### ◇ ERRORES POSIBLES:

- **404 Not Found:** Empresa no encontrada
- **403 Forbidden:** Sin permisos para actualizar

## 7. Eliminar empresa (sólo admin)

### DELETE /api/empresas/{id}

Elimina una empresa del sistema.

#### ◇ RESPUESTA EXITOSA (204 NO CONTENT)

#### ◇ ERRORES POSIBLES:

- **404 Not Found:** Empresa no encontrada

## 8. Buscar empresas por nombre

### GET /api/empresas/buscar?nombre={texto}

Busca empresas que contengan el texto especificado en su nombre.

#### ◇ RESPUESTA EXITOSA (200 OK):

```
[
 {
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "nombre": "Empresa con Texto",
 "correo": "contacto@empresa.com",
 "telefono": "912345678"
 },
 ...
]
```

## ✓ Endpoints de Candidaturas

### 1. Obtener candidaturas del usuario

#### GET /api/candidaturas

Retorna todas las candidaturas del usuario autenticado.

◇ RESPUESTA EXITOSA (200 OK):

```
[
 {
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "empresa": {
 "id": "550e8400-e29b-41d4-a716-446655440001",
 "nombre": "Empresa Ejemplo"
 },
 "cargo": "Desarrollador Java",
 "fecha": "2023-12-01",
 "estado": "PENDIENTE",
 "notas": "Candidatura de prueba"
 },
 ...
]
```

### 2. Obtener todas las candidaturas (sólo admin)

#### GET /api/candidaturas/all

Retorna todas las candidaturas del sistema.

◇ RESPUESTA EXITOSA (200 OK):

```
[
 {
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "user": {
 "id": "550e8400-e29b-41d4-a716-446655440002",
 "username": "usuariol1"
 },
 "empresa": {
 "id": "550e8400-e29b-41d4-a716-446655440001",
 "nombre": "Empresa Ejemplo"
 },
 "cargo": "Desarrollador Java",
 "fecha": "2023-12-01",
 "estado": "PENDIENTE",
 "notas": "Candidatura de prueba"
 },
 ...
]
```

### 3. Obtener candidatura por ID

#### GET /api/candidaturas/{id}

Retorna una candidatura específica por su ID.

◇ RESPUESTA EXITOSA (200 OK):

```
{
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "empresa": {
 "id": "550e8400-e29b-41d4-a716-446655440001",
 "nombre": "Empresa Ejemplo"
 },
 "cargo": "Desarrollador Java",
 "fecha": "2023-12-01",
 "estado": "PENDIENTE",
 "notas": "Candidatura de prueba"
}
```

◇ ERRORES POSIBLES:

- **404 Not Found:** Candidatura no encontrada
- **403 Forbidden:** Sin permisos para ver esta candidatura

#### 4. Crear candidatura

##### POST /api/candidaturas

Crea una nueva candidatura para el usuario autenticado.

◇ CUERPO DE LA SOLICITUD:

```
{
 "empresa": {
 "id": "550e8400-e29b-41d4-a716-446655440001"
 },
 "cargo": "Desarrollador Java",
 "fecha": "2023-12-01",
 "estado": "PENDIENTE",
 "notas": "Candidatura de prueba"
}
```

◇ RESPUESTA EXITOSA (201 CREATED):

```
{
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "empresa": {
 "id": "550e8400-e29b-41d4-a716-446655440001",
 "nombre": "Empresa Ejemplo"
 },
 "cargo": "Desarrollador Java",
 "fecha": "2023-12-01",
 "estado": "PENDIENTE",
 "notas": "Candidatura de prueba"
}
```

#### 5. Actualizar candidatura

##### PUT /api/candidaturas/{id}

Actualiza una candidatura existente.

#### ◇ CUERPO DE LA SOLICITUD:

```
{
 "empresa": {
 "id": "550e8400-e29b-41d4-a716-446655440001"
 },
 "cargo": "Senior Java Developer",
 "fecha": "2023-12-01",
 "estado": "ENTREVISTA",
 "notas": "Candidatura actualizada"
}
```

#### ◇ RESPUESTA EXITOSA (200 OK):

```
{
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "empresa": {
 "id": "550e8400-e29b-41d4-a716-446655440001",
 "nombre": "Empresa Ejemplo"
 },
 "cargo": "Senior Java Developer",
 "fecha": "2023-12-01",
 "estado": "ENTREVISTA",
 "notas": "Candidatura actualizada"
}
```

#### ◇ ERRORES POSIBLES:

- **404 Not Found:** Candidatura no encontrada
- **403 Forbidden:** Sin permisos para actualizar esta candidatura

## 6. Actualizar estado de candidatura

### PATCH /api/candidaturas/{id}/estado?estado={ESTADO}

Actualiza solo el estado de una candidatura existente.

#### ◇ VALORES VÁLIDOS PARA ESTADO:

- PENDIENTE
- ENTREVISTA
- ACEPTADA
- RECHAZADA
- ARCHIVADA
- EN\_PROCESO

#### ◇ RESPUESTA EXITOSA (200 OK):

```
{
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "empresa": {
 "id": "550e8400-e29b-41d4-a716-446655440001",
 "nombre": "Empresa Ejemplo"
 },
 "cargo": "Desarrollador Java",
 "fecha": "2023-12-01",
 "estado": "ENTREVISTA",
 "notas": "Candidatura de prueba"
}
```

◇ ERRORES POSIBLES:

- **404 Not Found:** Candidatura no encontrada
- **403 Forbidden:** Sin permisos para actualizar esta candidatura
- **400 Bad Request:** Estado no válido

## 7. Eliminar candidatura (sólo admin)

### DELETE /api/candidaturas/{id}

Elimina una candidatura del sistema.

◇ RESPUESTA EXITOSA (204 NO CONTENT)

◇ ERRORES POSIBLES:

- **404 Not Found:** Candidatura no encontrada

## 8. Buscar candidaturas con filtros

### GET /api/candidaturas/buscar

Busca candidaturas según múltiples criterios de filtrado.

◇ PARÁMETROS:

- **estado:** Estado de la candidatura (PENDIENTE, ENTREVISTA, etc.)
- **empresa:** Nombre de la empresa
- **fechaDesde:** Fecha mínima de aplicación (formato: yyyy-MM-dd)
- **fechaHasta:** Fecha máxima de aplicación (formato: yyyy-MM-dd)
- **q:** Texto para buscar en cargo o notas

◇ EJEMPLO:

**GET api/candidaturas/buscar?estado=ENTREVISTA&empresa=Acme&fechaDesde=2023-01-01**

◇ RESPUESTA EXITOSA (200 OK):

```
[
 {
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "empresa": {
 "id": "550e8400-e29b-41d4-a716-446655440001",
 "nombre": "Acme Inc."
 },
 "cargo": "Desarrollador Java",
 "fecha": "2023-05-15",
 "estado": "ENTREVISTA",
 "notas": "Entrevista programada para la próxima semana"
 },
 ...
]
```

## ✓ Endpoints de Preguntas

### 9. Obtener preguntas de una candidatura

#### GET /api/preguntas?candidaturaId={id}

Retorna todas las preguntas asociadas a una candidatura específica.

◇ RESPUESTA EXITOSA (200 OK):

```
jsonCopy[
 {
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "candidatura": {
 "id": "550e8400-e29b-41d4-a716-446655440001"
 },
 "pregunta": "¿Cuál es tu experiencia con Spring Boot?",
 "usuario": {
 "id": "550e8400-e29b-41d4-a716-446655440002",
 "username": "usuario1"
 }
 },
 ...
]
```

### 10. Obtener conteo de preguntas

#### GET /api/preguntas/count?candidaturaId={id}

Retorna el número de preguntas asociadas a una candidatura específica.

◇ RESPUESTA EXITOSA (200 OK):

5

### 11. Crear pregunta

#### POST /api/preguntas

Crea una nueva pregunta asociada a una candidatura.

◇ CUERPO DE LA SOLICITUD:

```
{
 "candidaturaId": "550e8400-e29b-41d4-a716-446655440001",
 "pregunta": "¿Cuántos años de experiencia tienes con Spring Boot?"
}
```

◇ RESPUESTA EXITOSA (201 CREATED):

```
{
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "candidatura": {
 "id": "550e8400-e29b-41d4-a716-446655440001"
 },
 "pregunta": "¿Cuántos años de experiencia tienes con Spring Boot?",
 "usuario": {
 "id": "550e8400-e29b-41d4-a716-446655440002",
 "username": "usuario1"
 }
}
```

## 12. Actualizar pregunta

### PUT /api/preguntas/{id}

Actualiza una pregunta existente.

#### ◇ CUERPO DE LA SOLICITUD:

```
{
 "candidaturaId": "550e8400-e29b-41d4-a716-446655440001",
 "pregunta": "¿Cuántos años de experiencia tienes con Java y Spring Boot?"
}
```

#### ◇ RESPUESTA EXITOSA (200 OK):

```
{
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "candidatura": {
 "id": "550e8400-e29b-41d4-a716-446655440001"
 },
 "pregunta": "¿Cuántos años de experiencia tienes con Java y Spring Boot?",
 "usuario": {
 "id": "550e8400-e29b-41d4-a716-446655440002",
 "username": "usuario1"
 }
}
```

#### ◇ ERRORES POSIBLES:

- **404 Not Found:** Pregunta no encontrada
- **403 Forbidden:** Sin permisos para actualizar esta pregunta

## 13. Eliminar pregunta

### DELETE /api/preguntas/{id}

Elimina una pregunta.

#### ◇ RESPUESTA EXITOSA (204 NO CONTENT)

#### ◇ ERRORES POSIBLES:

- **404 Not Found:** Pregunta no encontrada
- **403 Forbidden:** Sin permisos para eliminar esta pregunta
-



## ✓ Endpoints de Reclutadores

### 14. Obtener todos los reclutadores

#### GET /api/reclutador

Retorna todos los reclutadores disponibles.

◇ RESPUESTA EXITOSA (200 OK):

```
[
 {
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "nombre": "Reclutador Ejemplo",
 "linkinUrl": "https://linkedin.com/in/reclutador",
 "empresa": {
 "id": "550e8400-e29b-41d4-a716-446655440001",
 "nombre": "Empresa Ejemplo"
 }
 },
 ...
]
```

### 15. Obtener reclutador por ID

#### GET /api/reclutador/{id}

Retorna un reclutador específico por su ID.

◇ RESPUESTA EXITOSA (200 OK):

```
{
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "nombre": "Reclutador Ejemplo",
 "linkinUrl": "https://linkedin.com/in/reclutador",
 "empresa": {
 "id": "550e8400-e29b-41d4-a716-446655440001",
 "nombre": "Empresa Ejemplo"
 }
}
```

◇ ERRORES POSIBLES:

- **404 Not Found:** Reclutador no encontrado

## 16. Obtener reclutadores por empresa

### GET /api/reclutador/empresa/{empresaid}

Retorna todos los reclutadores asociados a una empresa específica.

◇ RESPUESTA EXITOSA (200 OK):

```
[
 {
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "nombre": "Reclutador 1",
 "linkinUrl": "https://linkedin.com/in/reclutador1",
 "empresa": {
 "id": "550e8400-e29b-41d4-a716-446655440001",
 "nombre": "Empresa Ejemplo"
 }
 },
 ...
]
```

## 17. Crear reclutador (sólo admin)

### POST /api/reclutador

Crea un nuevo reclutador en el sistema.

◇ CUERPO DE LA SOLICITUD:

```
{
 "nombre": "Nuevo Reclutador",
 "linkinUrl": "https://linkedin.com/in/nuevoreclutador",
 "empresa": {
 "id": "550e8400-e29b-41d4-a716-446655440001"
 }
}
```

◇ RESPUESTA EXITOSA (201 CREATED):

```
{
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "nombre": "Nuevo Reclutador",
 "linkinUrl": "https://linkedin.com/in/nuevoreclutador",
 "empresa": {
 "id": "550e8400-e29b-41d4-a716-446655440001",
 "nombre": "Empresa Ejemplo"
 }
}
```

## 18. Crear reclutador durante gestión de candidaturas

### POST /api/reclutador/crear-con-candidatura

Permite a usuarios normales crear reclutadores durante el proceso de gestión de candidaturas.

◇ CUERPO DE LA SOLICITUD:

```
{
 "nombre": "Reclutador Candidatura",
 "linkinUrl": "https://linkedin.com/in/reclutadorcandidatura",
 "empresa": {
 "id": "550e8400-e29b-41d4-a716-446655440001"
 }
}
```

◇ RESPUESTA EXITOSA (201 CREATED O 200 OK SI YA EXISTE):

```
{
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "nombre": "Reclutador Candidatura",
 "linkinUrl": "https://linkedin.com/in/reclutadorcandidatura",
 "empresa": {
 "id": "550e8400-e29b-41d4-a716-446655440001",
 "nombre": "Empresa Ejemplo"
 }
}
```

## 19. Actualizar reclutador por usuario

### PUT /api/reclutador/{id}/user-update

Permite a usuarios actualizar reclutadores asociados a sus candidaturas.

◇ CUERPO DE LA SOLICITUD:

```
{
 "nombre": "Reclutador Actualizado",
 "linkinUrl": "https://linkedin.com/in/reclutadoractualizado"
}
```

◇ RESPUESTA EXITOSA (200 OK):

```
{
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "nombre": "Reclutador Actualizado",
 "linkinUrl": "https://linkedin.com/in/reclutadoractualizado",
 "empresa": {
 "id": "550e8400-e29b-41d4-a716-446655440001",
 "nombre": "Empresa Ejemplo"
 }
}
```

◇ ERRORES POSIBLES:

- **404 Not Found:** Reclutador no encontrado
- **403 Forbidden:** Sin permisos para actualizar este reclutador

## 20. Actualizar reclutador (sólo admin)

### PUT /api/reclutador/{id}

Actualiza un reclutador existente.

#### ◇ CUERPO DE LA SOLICITUD:

```
{
 "nombre": "Reclutador Admin Actualizado",
 "linkinUrl": "https://linkedin.com/in/reclutadoradmin",
 "empresa": {
 "id": "550e8400-e29b-41d4-a716-446655440001"
 }
}
```

#### ◇ RESPUESTA EXITOSA (200 OK):

```
{
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "nombre": "Reclutador Admin Actualizado",
 "linkinUrl": "https://linkedin.com/in/reclutadoradmin",
 "empresa": {
 "id": "550e8400-e29b-41d4-a716-446655440001",
 "nombre": "Empresa Ejemplo"
 }
}
```

#### ◇ ERRORES POSIBLES:

- **404 Not Found:** Reclutador no encontrado

## 21. Eliminar reclutador (sólo admin)

### DELETE /api/reclutador/{id}

Elimina un reclutador del sistema.

#### ◇ RESPUESTA EXITOSA (204 NO CONTENT)

#### ◇ ERRORES POSIBLES:

- **404 Not Found:** Reclutador no encontrado

## 22. Asociar reclutador a candidatura

### POST /api/reclutador/{reclutadorId}/candidaturas/{candidaturaId}

Asocia un reclutador a una candidatura específica.

#### ◇ RESPUESTA EXITOSA (200 OK)

#### ◇ ERRORES POSIBLES:

- **404 Not Found:** Reclutador o candidatura no encontrados

## 23. Desasociar reclutador de candidatura

### DELETE /api/reclutador/{reclutadorId}/candidaturas/{candidaturaId}

Desasocia un reclutador de una candidatura específica.

◇ RESPUESTA EXITOSA (200 OK)

◇ ERRORES POSIBLES:

- **404 Not Found:** Reclutador o candidatura no encontrados

### ✓ Códigos de estado HTTP

La API utiliza los siguientes códigos de estado HTTP para indicar el resultado de las operaciones:

Código	Descripción	Casos de uso
200	OK	Solicitud procesada correctamente, devuelve contenido
201	Created	Recurso creado exitosamente
204	No Content	Solicitud procesada correctamente, sin contenido de retorno
400	Bad Request	Solicitud malformada, validación fallida o datos incorrectos
401	Unauthorized	No autenticado o token inválido
403	Forbidden	Sin permisos suficientes para acceder al recurso
404	Not Found	Recurso no encontrado
409	Conflict	Conflicto (por ejemplo, entidad duplicada)
500	Internal Server Error	Error interno del servidor

## ✓ Manejo de errores

Los errores en la API se devuelven con un código de estado HTTP apropiado y un cuerpo que describe el error. La estructura del cuerpo de error es consistente:

```
{
 "status": "error",
 "error": {
 "code": "ERROR_CODE",
 "message": "Descripción detallada del error"
 }
}
```

### 24. Códigos de error comunes

Código	Descripción
VALIDATION_ERROR	Error de validación en datos de entrada
RESOURCE_NOT_FOUND	Recurso no encontrado
DUPLICATE_RESOURCE	Recurso duplicado
UNAUTHORIZED	No autenticado
FORBIDDEN	Operación no permitida
INTERNAL_ERROR	Error interno del sistema

### 25. Ejemplos de errores

#### ◇ RECURSO NO ENCONTRADO (404):

```
{
 "status": "error",
 "error": {
 "code": "RESOURCE_NOT_FOUND",
 "message": "La candidatura con id 550e8400-e29b-41d4-a716-446655440000 no existe"
 }
}
```

#### ◇ VALIDACIÓN FALLIDA (400):

```
{
 "status": "error",
 "error": {
 "code": "VALIDATION_ERROR",
 "message": "El nombre de usuario ya está en uso"
 }
}
```

#### ◇ ACCESO DENEGADO (403):

```
{
 "status": "error",
 "error": {
 "code": "FORBIDDEN",
 "message": "No tienes permisos para acceder a este recurso"
 }
}
```

## ✓ **Paginación y ordenamiento**

## 5. Desarrollo del Frontend (React)

- 5.1. Configuración del entorno de desarrollo.
- 5.2. Diseño de la UI/UX.
- 5.3. Implementación de componentes y navegación.
- 5.4. Conexión con la API del backend.
- 5.5. Pruebas de usabilidad.

## 6. Desarrollo de la App Móvil (Android)

- 6.1. Definir si será nativa (Kotlin/Jetpack Compose) o híbrida (React Native).
- 6.2. Implementación de la interfaz y lógica de negocio.
- 6.3. Conexión con el backend.
- 6.4. Pruebas en dispositivos reales.

## 7. Pruebas y Depuración

- 7.1. Pruebas unitarias y de integración en backend y frontend.
- 7.2. Pruebas de usuario.
- 7.3. Optimización de código y rendimiento.
- 7.4. Revisión de código.

## 8. Despliegue (si aplica)

- 8.1. Configuración de hosting para la web.
- 8.2. Distribución de la app Android en dispositivos de prueba o Play Store.
- 8.3. Monitoreo post-despliegue.

## 9. Conclusión

- 9.1. Reflexiones sobre el proyecto.
- 9.2. Lecciones aprendidas.



### 9.3. Posibles mejoras futuras.