**Documento de Arquitectura de la Solución – Prueba Técnica Banco Falabella**

**1. Introducción**

Este documento describe la arquitectura y la estructura de la solución desarrollada para la prueba técnica del Banco Falabella. La aplicación es un **servicio REST** construido con **Spring Boot**, que permite registrar usuarios junto a sus teléfonos, manejar la validación de correo electrónico duplicado y generar tokens de autenticación.

El sistema utiliza **H2 In-Memory Database** para pruebas, **Spring Data JPA** para persistencia, **Spring Security** con JWT y **OpenAPI/Swagger** para documentación de la API.

**2. Tecnologías utilizadas**

* **Java 21** – lenguaje de programación principal.
* **Spring Boot 4 (M2)** – framework para desarrollo de aplicaciones web.
* **Spring Data JPA** – ORM para acceso a base de datos relacional.
* **H2 Database** – base de datos en memoria para pruebas.
* **Spring Security** – manejo de autenticación y autorización.
* **JWT (io.jsonwebtoken)** – generación de tokens de seguridad.
* **SpringDoc OpenAPI 3** – documentación de API REST.
* **Lombok** – reducción de código repetitivo (@Getter, @Setter, @Builder).
* **JUnit 5 + Mockito** – pruebas unitarias y de integración.
* **Validation (jakarta.validation)** – validación de DTOs (@NotEmpty, @Email, @Pattern).

**3. Estructura del proyecto**

El proyecto sigue una **arquitectura en capas**, que separa las responsabilidades de la siguiente manera:

src/main/java/com/prueba/bancofalabella/prueba/tecnica

├─ config

│ ├─ JpaConfig.java # Configuración JPA/Auditing

│ ├─ SecurityConfig.java # Configuración de Spring Security

│ └─ SwaggerConfig.java # Configuración de OpenAPI

├─ dto

│ ├─ ErrorResponse.java

│ ├─ PhoneRequest.java

│ ├─ RegisterRequest.java

│ └─ RegisterResponse.java

├─ entity

│ ├─ User.java

│ └─ Phone.java

├─ exception

│ ├─ ApiException.java

│ ├─ ApiExceptionHandler.java

│ ├─ EmailAlreadyExistsException.java

│ └─ GlobalExceptionHandler.java

├─ repository

│ └─ UsuarioRepository.java

├─ security

│ ├─ JwtAuthenticationFilter.java

│ ├─ JwtProperties.java

│ └─ JwtService.java

├─ service

│ ├─ UsuarioService.java

│ └─ UsuarioServiceImpl.java

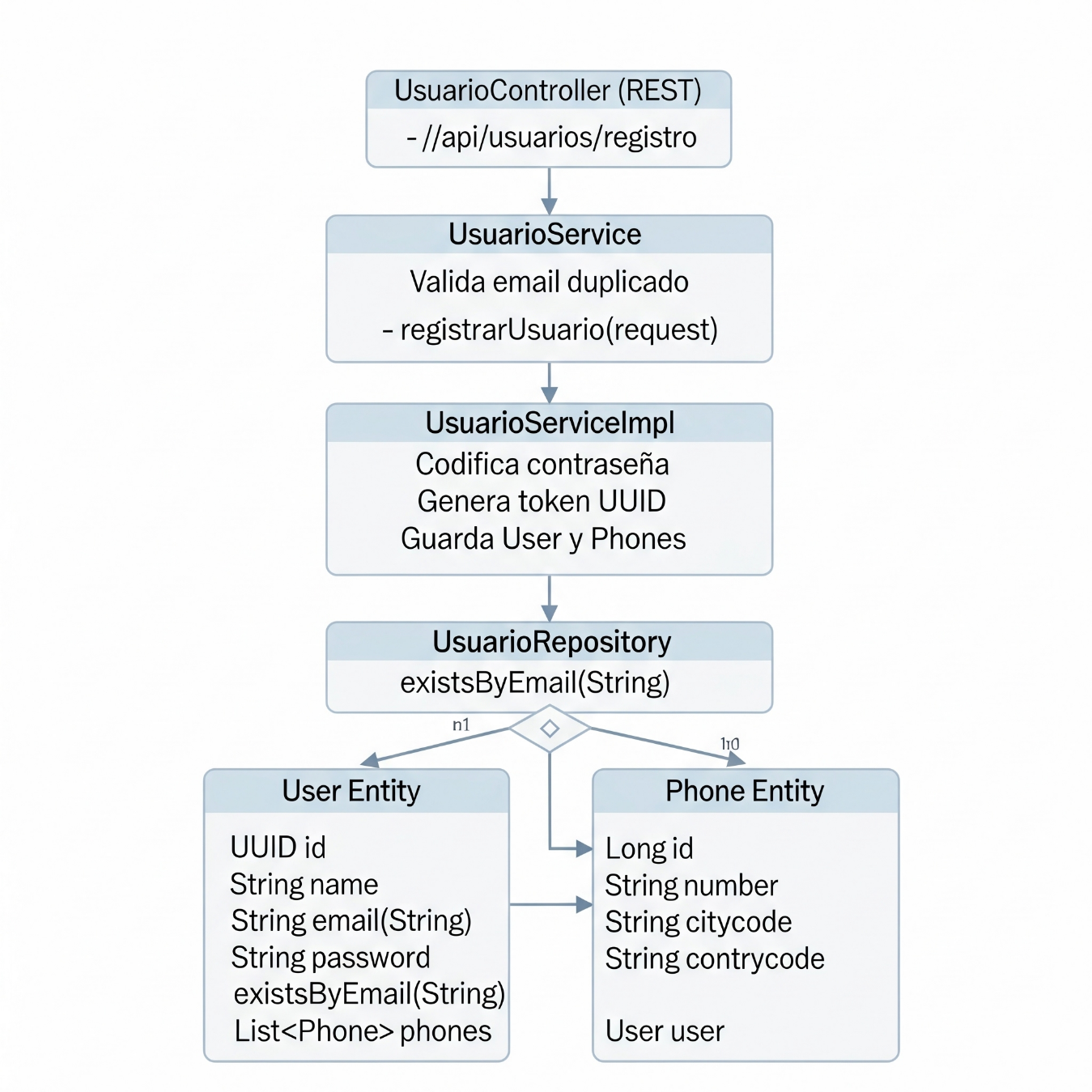
├─ UsuarioController

│ └─ UsuarioController.java

└─ PruebaBancoFalabellaApplication.java

**4. Diagrama de la solución**

**4.1 Flujo de la aplicación**

****

Leyenda de flujo:

DTOs → Servicio → Repositorio → Entidades (User/Phone)

**4.2 Seguridad y JWT**

* **JwtAuthenticationFilter** intercepta las peticiones HTTP para futuras validaciones.
* **JwtService** genera tokens JWT basados en la configuración de **JwtProperties**.
* Actualmente, la prueba no valida el JWT entrante, pero la arquitectura permite integrarlo fácilmente.

**5. Explicación de capas**

1. **Controlador (UsuarioController)**
   * Expone la API REST.
   * Convierte JSON en DTOs.
   * Devuelve respuestas con RegisterResponse o errores manejados por GlobalExceptionHandler.
2. **Servicio (UsuarioService / UsuarioServiceImpl)**
   * Contiene la lógica de negocio.
   * Valida email duplicado y genera tokens UUID.
   * Codifica contraseñas con BCrypt.
3. **Repositorio (UsuarioRepository)**
   * Interactúa con la base de datos H2.
   * Implementa métodos de búsqueda y persistencia de User.
4. **Entidades (User, Phone)**
   * Representan tablas users y phones.
   * Relación uno-a-muchos (User → Phone).
   * Contienen campos de auditoría (created, modified, lastLogin).
5. **DTOs**
   * RegisterRequest y PhoneRequest para recibir datos.
   * RegisterResponse para responder con datos de usuario.
   * ErrorResponse para reportar errores.
6. **Excepciones**
   * Manejo centralizado de errores con GlobalExceptionHandler.
   * Excepciones específicas: EmailAlreadyExistsException, ApiException.

**6. Documentación de la API (Swagger/OpenAPI)**

* **SwaggerConfig.java** configura OpenAPI 3 para la documentación de la API.
* La documentación se expone automáticamente en las siguientes rutas:
  + /v3/api-docs → JSON de especificación OpenAPI.
  + /swagger-ui.html o /swagger-ui/index.html → Interfaz web interactiva para probar los endpoints.
* Permite visualizar y probar los endpoints REST como /api/usuarios/registro.
* Ejemplo de request mostrado en Swagger:

{

"name": "Juan Rodriguez",

"email": "juan@rodriguez.org",

"password": "hunter2",

"phones": [

{

"number": "1234567",

"citycode": "1",

"contrycode": "57"

}

]

}

* Esta herramienta facilita la interacción con la API para desarrolladores y testers, proporcionando ejemplos, esquemas de datos y descripción de cada endpoint.

**7. Carga de Datos Inicial (data.sql)**

* El archivo **data.sql** se encuentra en la ruta **src/main/resources** del proyecto.
* Contiene los scripts de inserción de datos iniciales para la base de datos H2 en memoria.
* Se ejecuta automáticamente al iniciar la aplicación gracias a la configuración de Spring Boot (spring.datasource.initialization-mode=always).
* Contiene registros de ejemplo para los usuarios y sus teléfonos, por ejemplo:

-- Usuario 1

INSERT INTO users (id, name, email, password, created, modified, last\_login, token, is\_active)

VALUES ('11111111-1111-1111-1111-111111111111', 'Juan Rodriguez', 'juan@rodriguez.org', 'hunter2',

CURRENT\_TIMESTAMP, CURRENT\_TIMESTAMP, CURRENT\_TIMESTAMP, 'token-juan', true);

INSERT INTO phones (id, number, citycode, contrycode, user\_id)

VALUES (1, '1234567', '1', '57', '11111111-1111-1111-1111-111111111111');

* Esto permite que al levantar la aplicación, la base de datos ya cuente con datos de prueba, facilitando tests y desarrollo sin necesidad de carga manual.

**8. Pruebas Unitarias**

* El proyecto incluye **pruebas unitarias** desarrolladas con **JUnit 5** y **Mockito**, ubicadas en la ruta **src/test/java**.
* Las pruebas verifican tanto la lógica de los controladores, servicios y DTOs, como la carga del contexto de Spring Boot.
* Los principales archivos de pruebas son:

1. **Controlador de usuarios:**
   * **UsuarioControllerTest** (src/test/java/com/prueba/bancofalabella/prueba/tecnica/controller/UsuarioControllerTest)
   * Verifica endpoints de registro de usuario, casos exitosos y manejo de excepciones como EmailAlreadyExistsException.
2. **Pruebas de DTOs:**
   * Ubicadas en **src/test/java/com/prueba/bancofalabella/prueba/tecnica/dto**
   * Validan el correcto funcionamiento de los builders y getters de las clases PhoneRequest, RegisterRequest y RegisterResponse.
3. **Carga del contexto de Spring Boot:**
   * **PruebaTecnicaApplicationTests** (src/test/java/com/prueba/bancofalabella/prueba/tecnica/PruebaTecnicaApplicationTests)
   * Comprueba que la aplicación se levanta correctamente con el perfil de pruebas (@ActiveProfiles("test")).

* Estas pruebas aseguran la **integridad del proyecto**, la **validación de entradas** y el **comportamiento esperado de los endpoints y servicios** antes de ejecutar la aplicación en entornos reales.