Sistema de Gestión de Restaurantes

Proyecto final de FCT

John Edinson Lopez Contreras

Desarrollo de Aplicaciones Multiplataforma

I.E.S. Virgen del Carmen

2024

Contents

Introducción

El desarrollo de aplicaciones multiplataforma se ha convertido en una herramienta esencial en la era digital actual, donde la diversidad de dispositivos y sistemas operativos requiere soluciones que puedan funcionar de manera eficiente en diferentes entornos. Esta tendencia ha ganado popularidad debido a su capacidad para reducir costos y tiempos de desarrollo, así como para maximizar el alcance de las aplicaciones. Los desarrolladores pueden escribir una vez su código y desplegarlo en múltiples plataformas, incluyendo iOS, Android, y la web, asegurando así que sus aplicaciones lleguen a una audiencia más amplia sin la necesidad de duplicar esfuerzos.

El impacto del desarrollo multiplataforma en la sociedad es significativo. Gracias a frameworks como Flutter, React Native y Xamarin, los desarrolladores pueden crear aplicaciones de alta calidad con interfaces de usuario consistentes y un rendimiento robusto en todas las plataformas. Esto no solo facilita la vida de los usuarios finales al ofrecerles una experiencia unificada, sino que también empodera a las empresas y organizaciones a ser más ágiles y adaptables en un mercado competitivo.

En el mercado actual, existen diversas aplicaciones multiplataforma que han demostrado ser fundamentales en distintos sectores. Por ejemplo, aplicaciones como WhatsApp y Instagram, que se encuentran disponibles tanto en iOS como en Android, han revolucionado la forma en que nos comunicamos e interactuamos socialmente. Del mismo modo, aplicaciones como Spotify y Netflix han transformado el entretenimiento, proporcionando acceso a música y videos en múltiples dispositivos con una experiencia de usuario coherente.

En este contexto, la aplicación que estamos desarrollando, el “Sistema de Gestión de Restaurantes”, tiene el potencial de aportar significativamente a la comunidad. A diferencia de muchas aplicaciones en el mercado que se centran exclusivamente en la experiencia del cliente, nuestra aplicación se distingue por atender a tres tipos de usuarios: el administrador que gestiona todos los aspectos del sistema, el empresario dueño del restaurante que supervisa el rendimiento y las operaciones, y los usuarios finales que interactúan con la app para buscar restaurantes, hacer reservas y dejar reseñas.

En resumen, nuestra aplicación no solo busca facilitar la gestión interna de los restaurantes, sino también enriquecer la experiencia del usuario final y contribuir al crecimiento sostenible de los negocios en el sector de la restauración. Al aprovechar las ventajas del desarrollo multiplataforma, estamos creando una herramienta poderosa y accesible que puede ser utilizada en cualquier dispositivo, asegurando así que todos los usuarios, independientemente de su sistema operativo preferido, puedan beneficiarse de sus funcionalidades. Esta sinergia entre tecnología y funcionalidad es lo que posicionará al “Sistema de Gestión de Restaurantes” como una aplicación esencial en el mercado.

Descripción del Proyecto

El presente documento detalla la creación, desarrollo y despliegue del Sistema de Gestión de Restaurantes, una aplicación móvil para Android e iOS diseñada para facilitar la reserva de mesas en restaurantes. Este sistema permite a los usuarios buscar restaurantes, ver detalles, realizar reservas y dejar calificaciones y opiniones. A lo largo de este documento, se describen los requisitos funcionales, las tecnologías utilizadas, el análisis y diseño del sistema, así como las instrucciones para su despliegue e instalación.

Con este documento, buscamos proporcionar una visión integral de cómo el desarrollo multiplataforma puede transformar la industria de la restauración, ofreciendo soluciones prácticas y efectivas que beneficien a todos los involucrados.

Especificación de Requisitos Funcionales

Requisitos Funcionales

1. **Registro y Autenticación de Usuarios**
    - Los usuarios pueden registrarse proporcionando nombre, correo electrónico y contraseña.
    - Los usuarios pueden iniciar sesión con sus credenciales.
    - El sistema debe validar la autenticidad del usuario.
2. **Gestión de Restaurantes**
    - Los administradores pueden agregar, editar y eliminar restaurantes.
    - Los usuarios pueden buscar restaurantes por diferentes criterios.
    - Los usuarios pueden ver los detalles de los restaurantes, incluyendo menú y horario.
3. **Reservas**
    - Los usuarios pueden realizar reservas especificando fecha, hora y número de comensales.
    - Los usuarios pueden ver, modificar y cancelar sus reservas.
    - Los administradores pueden gestionar todas las reservas.
4. **Calificaciones y Opiniones**
    - Los usuarios pueden dejar calificaciones y opiniones sobre los restaurantes.
    - Los usuarios pueden ver las calificaciones y opiniones de otros usuarios.
5. **Favoritos**
    - Los usuarios pueden agregar y eliminar restaurantes de su lista de favoritos.

Requisitos No Funcionales

1. **Seguridad**
    - La información de los usuarios debe ser protegida mediante cifrado.
    - Las transacciones deben ser seguras y protegidas contra ataques.
2. **Rendimiento**
    - El sistema debe ser capaz de manejar múltiples solicitudes simultáneamente.
    - El tiempo de respuesta de las operaciones comunes debe ser mínimo.
3. **Escalabilidad**
    - El sistema debe ser escalable para soportar un aumento en el número de usuarios y transacciones.

Tecnología

Backend

El backend de la aplicación se desarrolló utilizando diversas tecnologías de la pila de Spring para Java. A continuación, se presenta una explicación detallada de cada componente, su instalación y configuración.

Tecnologías Utilizadas

1. **Spring Boot**: Un framework para desarrollar aplicaciones web y servicios RESTful en Java de manera rápida y sencilla.
2. **Spring MVC**: Un módulo de Spring para el desarrollo de aplicaciones web basado en el patrón Modelo-Vista-Controlador.
3. **Spring Data JPA**: Una abstracción sobre JPA (Java Persistence API) que simplifica el acceso a la base de datos.
4. **Spring Security**: Un módulo de Spring que proporciona autenticación y autorización robustas.
5. **PostgreSQL**: Un sistema de gestión de bases de datos relacional potente y de código abierto.
6. **Fire Base**: Firebase es una plataforma desarrollada por Google que proporciona una serie de herramientas y servicios.

Instalación y Configuración

1. Instalación de Spring Boot Para comenzar con Spring Boot, debes tener instalado Java Development Kit (JDK) y Apache Maven. Una vez que estos requisitos previos estén en su lugar, puedes crear un nuevo proyecto de Spring Boot.

1. **Crear un Proyecto de Spring Boot**: Puedes crear un proyecto desde el sitio web Spring Initializr seleccionando las dependencias necesarias, como Spring Web, Spring Data JPA, y Spring Security.

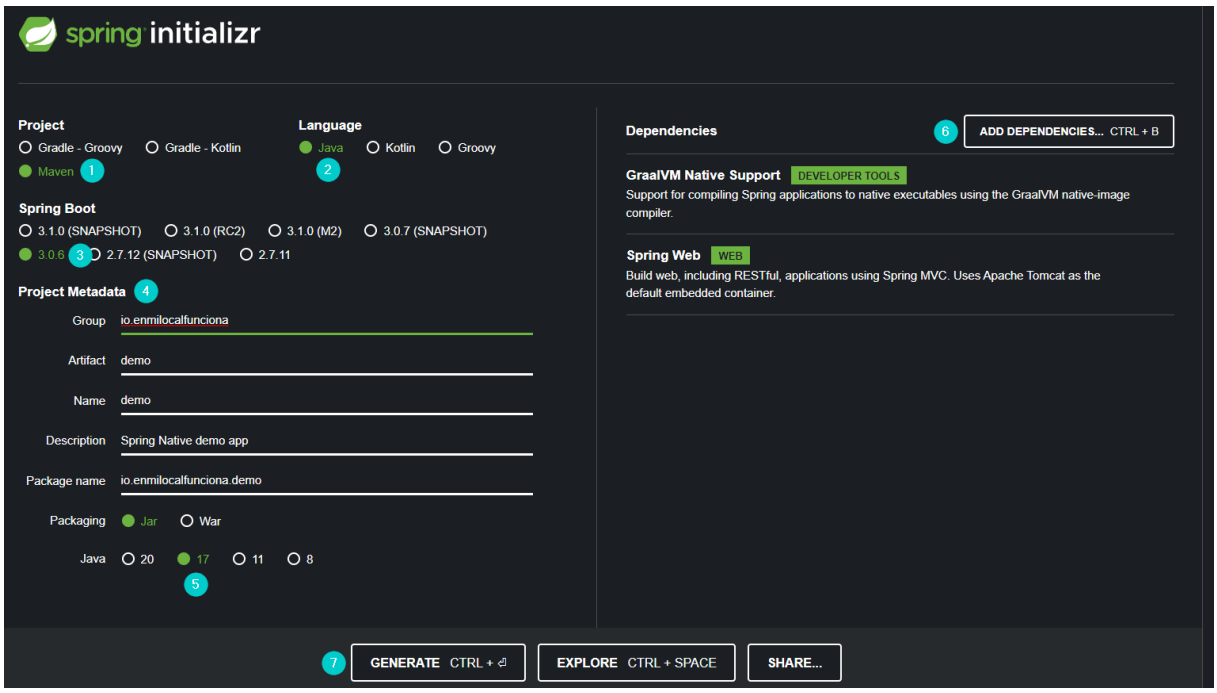

Figure 1: Spring Initializr

Alternativamente, puedes crear el proyecto utilizando Maven en la línea de comandos:

```
mvn archetype:generate -DgroupId=com.example -DartifactId=
    restaurant-reservations
-DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode
    =false
```

2. **Agregar Dependencias**: Edita el archivo `pom.xml` para incluir las dependencias de Spring Boot y otros módulos necesarios:

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
        <groupId>org.postgresql</groupId>
```

```
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
    </dependency>
    <!-- Otros módulos necesarios -->
</dependencies>
```

Configuración de Spring Boot

1. **Configurar el archivo `application.properties`**: Define las propiedades de configuración para la conexión con PostgreSQL:

```
spring.datasource.url=jdbc:postgresql://localhost:5432/
    restaurant_db
spring.datasource.username=tu_usuario
spring.datasource.password=tu_contraseña
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.
    PostgreSQLDialect
```

2. **Configurar Spring Security**: Define la configuración de seguridad para tu aplicación en una clase `SecurityConfig`:

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .csrf().disable()
            .authorizeRequests()
            .antMatchers("/api/public/**").permitAll()
            .anyRequest().authenticated()
            .and()
            .sessionManagement().sessionCreationPolicy(
                SessionCreationPolicy.STATELESS);

        // Configuración de filtros JWT si es necesario
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth)
        throws Exception {
        // Configuración de autenticación
    }
}
```

Configuración de Spring Data JPA

1. **Definir Entidades**: Crea clases de entidad que representan las tablas de la base de datos. Por ejemplo, para un restaurante:

```java
@Entity
@Data
@NoArgsConstructor
@Buildel
public class Restaurante {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nombre;
    private String direccion;
    // Otros campos
}
```

2. **Crear Repositorios**: Define interfaces de repositorio para realizar operaciones CRUD sobre las entidades:

```java
public interface RestauranteRepository extends JpaRepository<
    Restaurante, Long> {
}
```

Configuración de PostgreSQL **PostgreSQL** es un sistema de gestión de bases de datos relacional y objeto-relacional, conocido por su robustez, extensibilidad y soporte para estándares SQL. Es una de las bases de datos más utilizadas en aplicaciones de misión crítica y de alto rendimiento.

1. **Instalar PostgreSQL**: Descarga e instala PostgreSQL desde su sitio oficial.

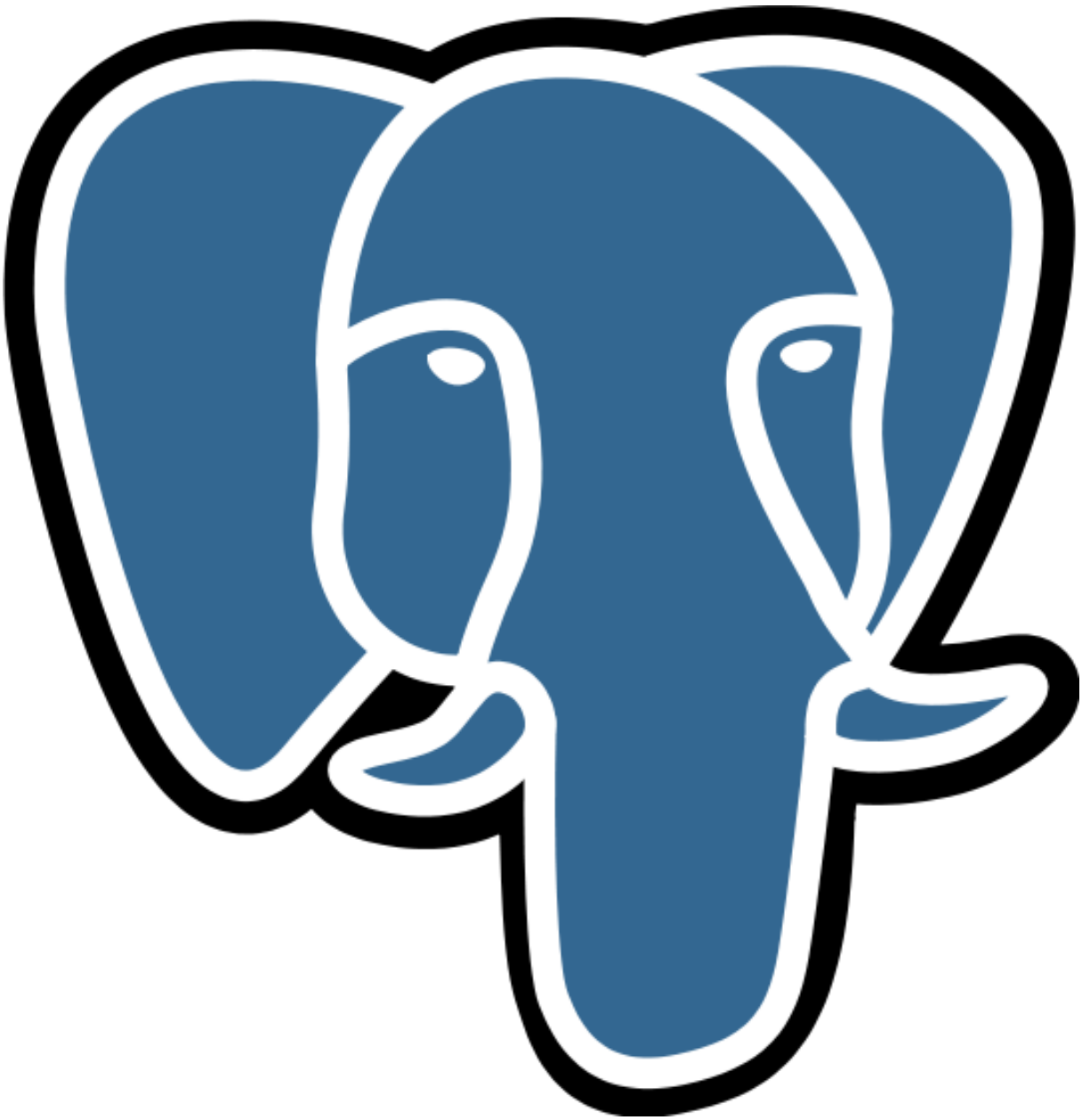

Figure 2: PostgreSQL Download

2. **Configurar la Base de Datos**: Una vez instalado, puedes configurar la base de datos para tu aplicación siguiendo estos pasos:

    - **Accede a la Consola de PostgreSQL**: Abre la terminal y accede a la consola de PostgreSQL utilizando el comando `psql`. Es posible que necesites especificar el usuario de PostgreSQL si no estás utilizando el usuario predeterminado.

```
sudo -u postgres psql
```

- **Crea una Base de Datos**: Utiliza el comando `CREATE DATABASE` para crear una nueva base de datos. En este caso, crearemos una base de datos llamada `restaurant_db`.

```
CREATE DATABASE restaurant_db;
```

- **Crea un Usuario**: Crea un nuevo usuario que tendrá permisos sobre la base de datos. Sustituye `tu_usuario` y `tu_contraseña` por el nombre de usuario y la contraseña que prefieras.

```
CREATE USER tu_usuario WITH PASSWORD 'tu_contraseña';
```

- **Asigna Privilegios al Usuario**: Concede todos los privilegios sobre la base de datos recién creada al usuario. Esto incluye permisos para leer, escribir, y modificar datos en la base de datos.

```
GRANT ALL PRIVILEGES ON DATABASE restaurant_db TO tu_usuario;
```

Características Clave de PostgreSQL

- **Fiabilidad y Seguridad**: PostgreSQL es conocido por su durabilidad y consistencia de datos, gracias a características como transacciones ACID, puntos de recuperación y copia de seguridad en caliente.
- **Extensibilidad**: Permite a los usuarios definir sus propios tipos de datos, operadores y funciones. Además, admite extensiones que pueden añadir nuevas funcionalidades al sistema de base de datos.
- **Soporte para JSON**: PostgreSQL soporta tipos de datos JSON y JSONB, lo que facilita el trabajo con datos no estructurados o semi-estructurados.
- **Compatibilidad con SQL Estándar**: PostgreSQL sigue de cerca los estándares SQL, lo que garantiza que el código SQL sea portable y conforme a las especificaciones ANSI SQL.
- **Alta Disponibilidad y Recuperación ante Desastres**: Ofrece soluciones robustas para replicación y failover, asegurando que los datos estén disponibles incluso en caso de fallos del sistema.

Configuración Avanzada

- **Ajuste de Parámetros de Configuración**: El archivo `postgresql.conf` permite ajustar una variedad de parámetros que afectan el rendimiento y el comportamiento del servidor PostgreSQL, tales como la memoria compartida, el tamaño del buffer de escritura, y la configuración de los logs.
- **Roles y Seguridad**: PostgreSQL permite una gestión detallada de roles y permisos, lo que es crucial para aplicaciones con múltiples usuarios y niveles de acceso.

Integración con Aplicaciones

- **Drivers y Bibliotecas**: Existen numerosos drivers y bibliotecas para integrar PostgreSQL con aplicaciones escritas en diversos lenguajes de programación, como Java (JDBC), Python (psycopg2), Node.js (pg), y muchos más.
- **ORMs (Object-Relational Mappers)**: PostgreSQL es compatible con la mayoría de los ORMs populares, facilitando la interacción entre la base de datos y los modelos de datos en el código de la aplicación. Ejemplos incluyen Hibernate (Java), SQLAlchemy (Python), y Sequelize (Node.js).

Al comprender y aprovechar las capacidades de PostgreSQL, puedes desarrollar aplicaciones robustas, seguras y eficientes, maximizando tanto el rendimiento como la mantenibilidad de tus sistemas de bases de datos.

Integración de Firebase en el Backend Firebase es una plataforma desarrollada por Google que proporciona una serie de herramientas y servicios para el desarrollo de aplicaciones, entre ellos Firebase Storage, que se utiliza para almacenar y servir archivos, como imágenes. En este proyecto, se utilizará Firebase Storage para almacenar las imágenes de los restaurantes.

Pasos para Configurar Firebase

1. Crear un Proyecto en Firebase

1. **Accede a la Consola de Firebase**: Ve a la Consola de Firebase e inicia sesión con tu cuenta de Google.
2. **Crear un Nuevo Proyecto**: Haz clic en “Agregar proyecto” y sigue las instrucciones para crear un nuevo proyecto.

Figure 3: Firebase Create Project

3. **Configurar Firebase Storage**: Una vez que tu proyecto esté creado, navega a la sección “Storage” en el menú de la izquierda y haz clic en “Comenzar” para habilitar Firebase Storage.

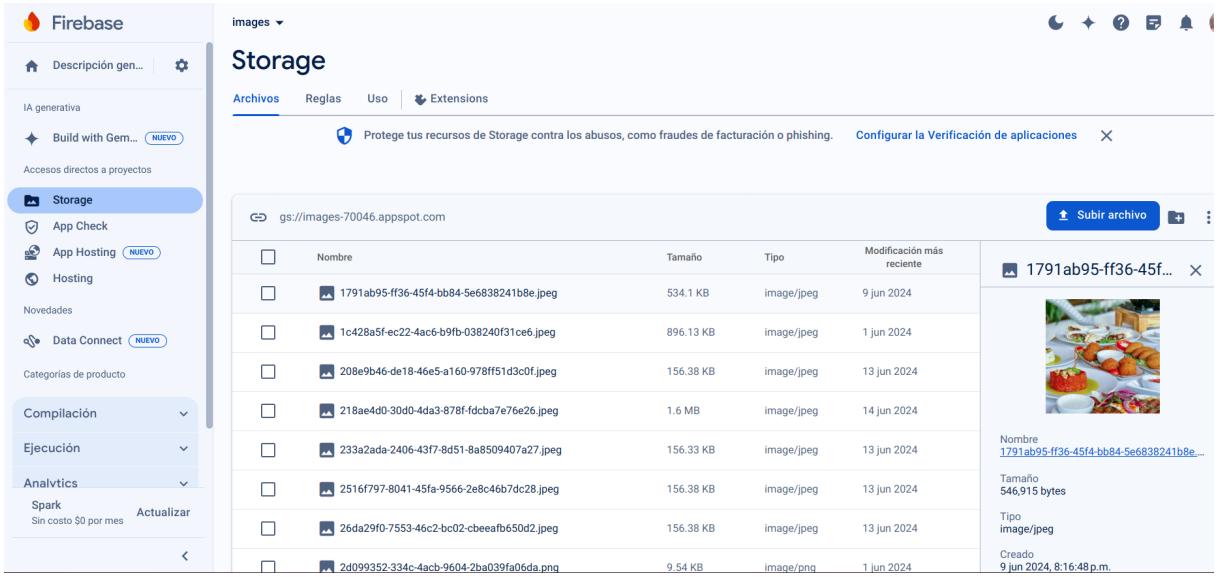

Figure 4: Firebase Storage

2. Obtener la Clave de Servicio

1. **Generar Clave de Servicio**: Ve a la configuración del proyecto (icono de engranaje) y selecciona “Cuentas de servicio”. Haz clic en “Generar nueva clave privada” para descargar un archivo JSON con las credenciales de tu proyecto.

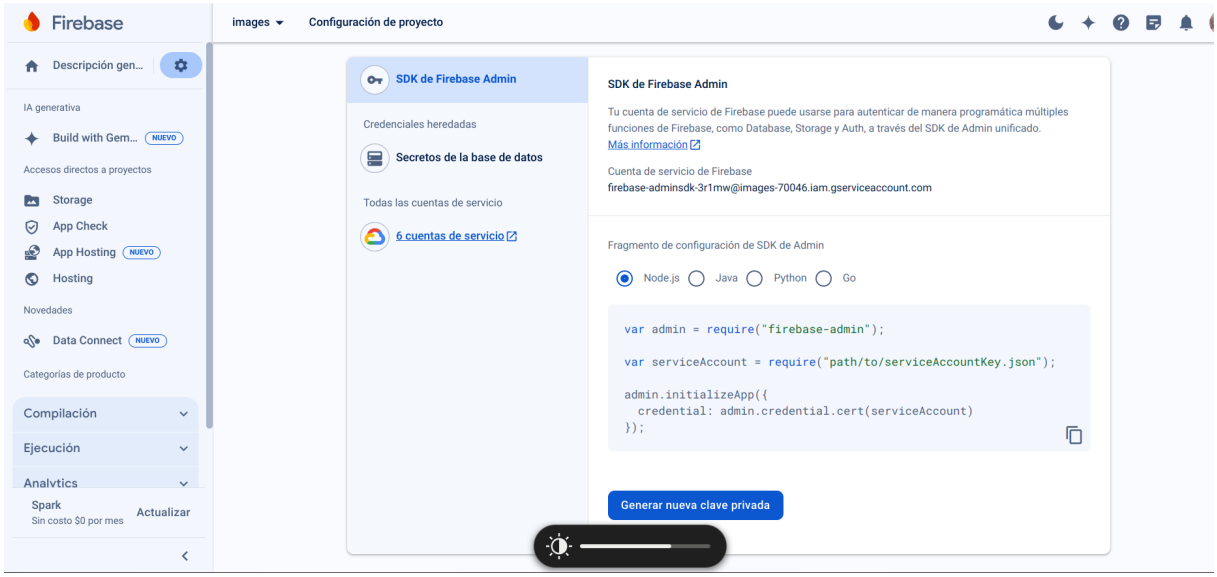

Figure 5: Firebase Service Account

Configurar Firebase en Spring Boot

1. **Agregar Dependencia de Firebase**: Agrega la dependencia de Firebase al archivo `pom.xml`:

```xml
<dependencies>
   <dependency>
        <groupId>com.google.firebase</groupId>
        <artifactId>firebase-admin</artifactId>
        <version>6.16.0</version>
    </dependency>
    <dependency>
        <groupId>com.google.cloud</groupId>
        <artifactId>google-cloud-firestore</artifactId>
        <version>3.21.3</version>
    </dependency>
</dependencies>
```

2. **Configurar Firebase**: Inicializa Firebase en tu aplicación utilizando una clave de servicio:

```java
@Configuration
public class FirebaseConfig {

    @PostConstruct
    public void initialize() {
        try {
            FirebaseOptions options = new FirebaseOptions.Builder()
                .setCredentials(GoogleCredentials.fromStream(new FileInputStream("path/to/serviceAccountKey.json")))
                .setStorageBucket("your-app.appspot.com")
                .build();

            FirebaseApp.initializeApp(options);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

3. **Subir y Obtener Imágenes**: Implementa métodos para subir imágenes a Firebase Storage y obtener sus URLs:

```java
public String uploadImage(MultipartFile file) {
    try {
        Bucket bucket = StorageClient.getInstance().bucket();
        Blob blob = bucket.create(file.getOriginalFilename(), file.getBytes(), file.getContentType());
        return blob.getMediaLink();
    } catch (IOException e) {
        e.printStackTrace();
```

```
            return null;
        }
    }
```

Ejecución del Backend en Spring Boot Spring Boot es un marco de trabajo basado en Spring que simplifica la creación y ejecución de aplicaciones autónomas basadas en Java. A continuación, se describen los pasos para configurar, compilar y ejecutar una aplicación backend en Spring Boot.

1. Ejecutar la Aplicación **Desde la Línea de Comandos**: - Ejecuta lo siguiente:

```
```bash
mvn spring-boot:run
```
```

2. Verificar la Ejecución

- Una vez iniciada la aplicación, deberías ver los logs de Spring Boot en la consola, indicando que el servidor embebido (por defecto, Tomcat) ha iniciado y está escuchando en el puerto 8080 (a menos que hayas configurado un puerto diferente en `application.properties`).

```
Tomcat started on port(s): 8080 (http)
Started Application in 3.45 seconds (JVM running for 5.234)
```

- **Acceder a los Endpoints**:
  - Puedes probar los endpoints de tu aplicación utilizando herramientas como Postman o curl. Por ejemplo, para
  - acceder al endpoint que lista todos los usuarios:

```
curl -X GET http://localhost:8080/api/usuarios \
 -H "Authorization: Bearer tu_token_de_acceso_aqui" \
 -H "Refresh-Token: tu_token_de_refresco_aqui" \
 -H "Content-Type: application/json"
```

Deberías recibir una respuesta en formato JSON con la lista de usuarios o el mensaje de error correspondiente.

Guía para hacer Peticiones a la API usando Postman Esta guía describe cómo hacer peticiones a la API utilizando Postman, incluyendo cómo configurar los encabezados necesarios para autenticación con `token` y `refreshToken`.

Prerrequisitos

1. Tener Postman instalado.
2. Contar con el `token` de acceso y `refreshToken` proporcionados por tu sistema de autenticación luego de hacer login.

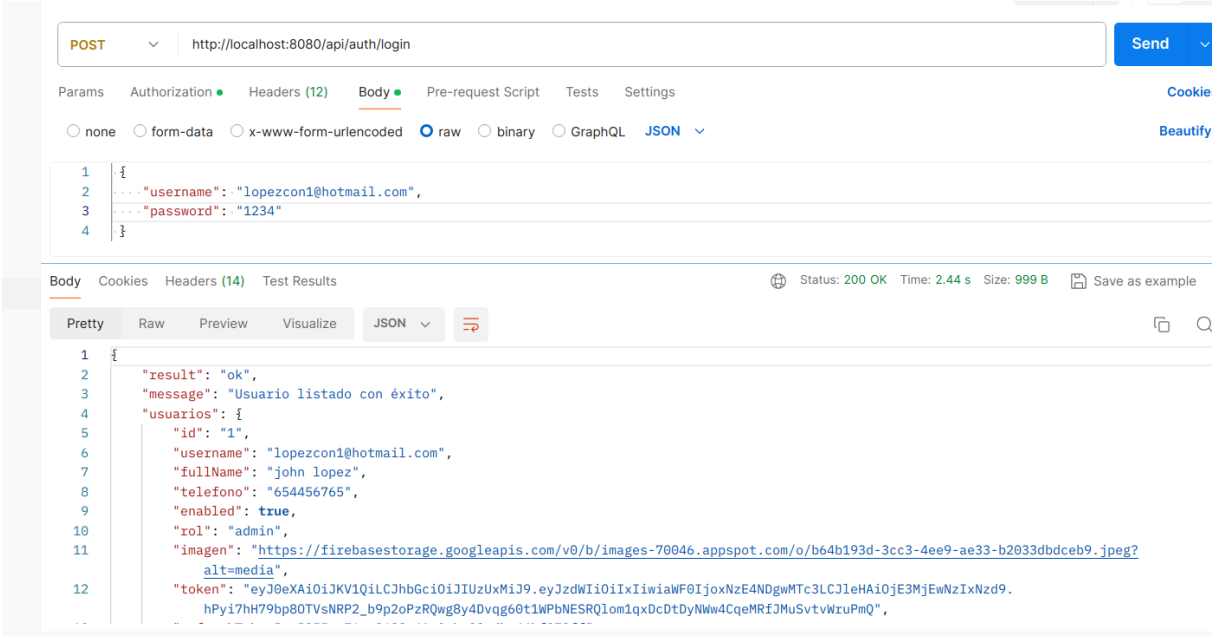

Figure 6: POST http://localhost:8080/api/auth/login

Configuración de Postman

1. Crear una nueva colección

1. Abre Postman.
2. Haz clic en el botón **New** en la esquina superior izquierda.
3. Selecciona **Collection**.
4. Asigna un nombre a tu colección y guarda.

2. Añadir una nueva petición

1. Selecciona la colección que acabas de crear.
2. Haz clic en **Add Request**.
3. Asigna un nombre a tu petición (por ejemplo, `Obtener Usuarios`).

3. Configurar la petición

1. Selecciona el método HTTP `GET`.
2. En el campo **Enter request URL**, introduce `http://localhost:8080/api/usuarios`.

4. Añadir encabezados de autenticación

1. Ve a la pestaña **Headers**.
2. Añade un nuevo encabezado:
    - **Key:** `Authorization`
    - **Value:** `Bearer tu_token_de_acceso_aqui`

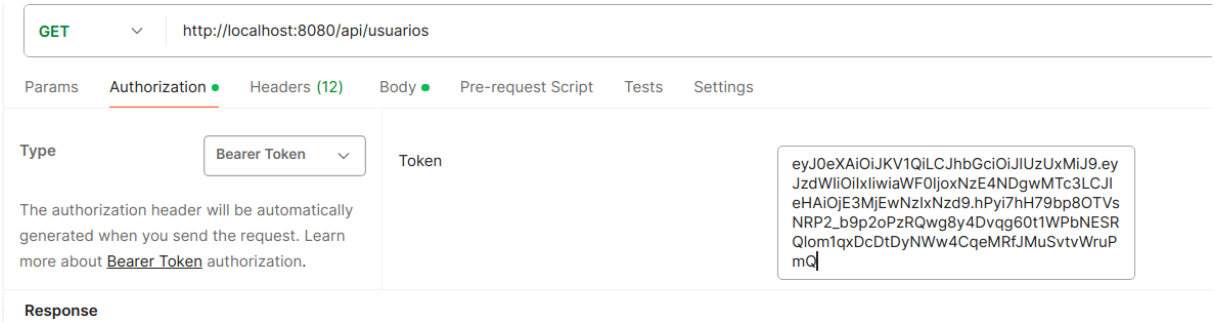

Figure 7: Token

3. Añade otro encabezado con el `refreshToken`:
    - **Key:** `Refresh-Token`
    - **Value:** `tu_token_de_refresco_aqui`

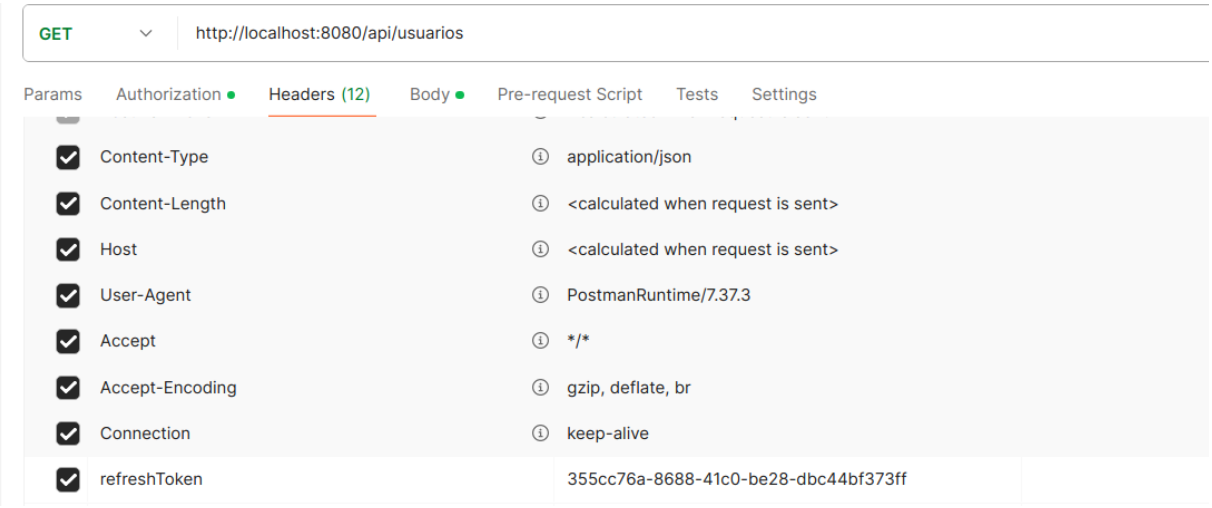

Figure 8: RefreshToken

4. Añade un encabezado de tipo de contenido:
    - **Key:** `Content-Type`
    - **Value:** `application/json`

5. Enviar la petición

1. Haz clic en el botón **Send**.
2. Revisa la respuesta de la API en la pestaña **Body** de Postman.

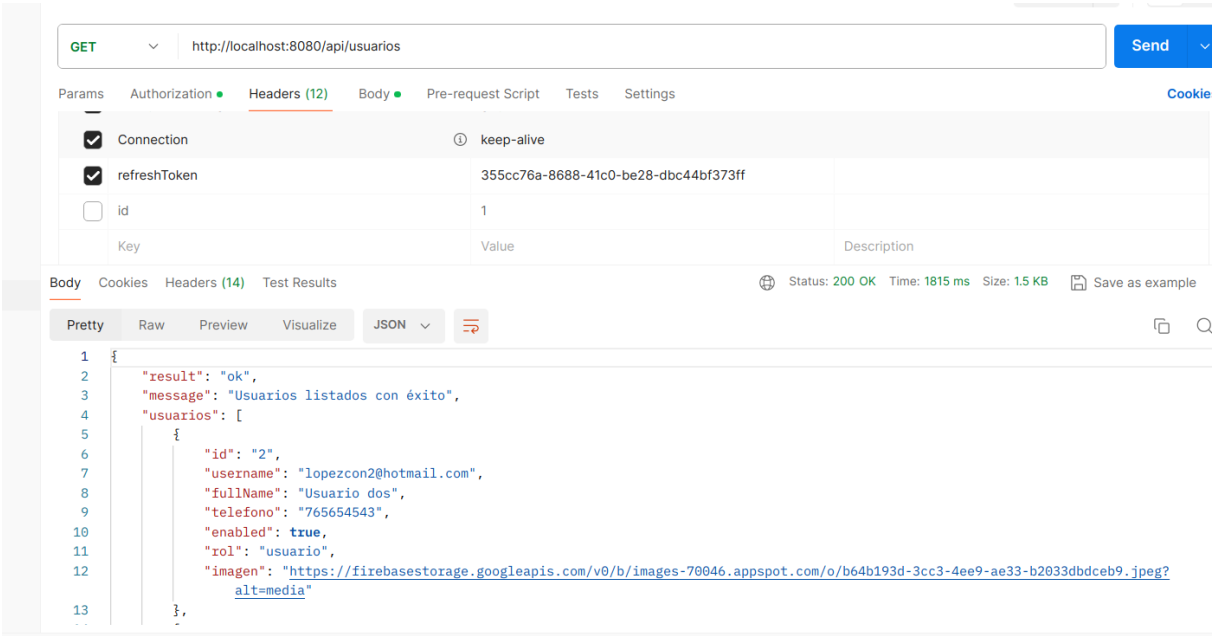

Figure 9: GET http://localhost:8080/api/usuarios

Ejemplo de Configuración

Headers

Key	Value
Authorization	Bearer tu_token_de_acceso_aqui
Refresh-Token	tu_token_de_refresco_aqui
Content-Type	application/json

Frontend

Guía de React Native con Expo

1. Instalación de Node.js Para empezar, necesitas tener Node.js instalado en tu máquina. Puedes descargarlo desde la página oficial de Node.js.

2. Verificar la Instalación de Node.js Para verificar que Node.js se ha instalado correctamente, abre tu terminal y ejecuta los siguientes comandos:

```
node -v
```

3. Instalación de Expo CLI Expo CLI es una herramienta de línea de comandos que te permite crear y gestionar proyectos de Expo. Para instalar Expo CLI, abre tu terminal y ejecuta el siguiente comando:

```
npm install -g expo-cli
```

Creación de un Proyecto Para crear un nuevo proyecto con Expo, utiliza el siguiente comando:

```
expo init my-new-project
```

Este comando te pedirá que elijas una plantilla para tu proyecto. Selecciona la plantilla que más te convenga (por ejemplo, “blank” para un proyecto vacío).

Navegar al Directorio del Proyecto Una vez creado el proyecto, navega al directorio del proyecto:

```
cd my-new-project
```

Ejecución del Proyecto Para iniciar tu proyecto, ejecuta el siguiente comando:

```
expo start
```

Este comando iniciará el servidor de desarrollo y te proporcionará un código QR. Puedes escanear este código QR con la aplicación Expo Go en tu dispositivo móvil para ver tu aplicación en tiempo real.

Figure 10: Codigo QR expo

Uso de la Aplicación Expo Go Descarga la aplicación Expo Go desde la App Store o Google Play Store en tu dispositivo móvil. Escanea el código QR proporcionado por el servidor de desarrollo para ver tu aplicación.

Comandos Útiles Ejecutar en un Emulador Si prefieres ejecutar tu aplicación en un emulador en lugar de un dispositivo físico, puedes usar los siguientes comandos:

iOS (requiere macOS y Xcode) `bash expo run:ios`

Android

```
1  expo run:android
```

Compilar la Aplicación Para compilar tu aplicación y generar un archivo APK o IPA, puedes usar los siguientes comandos:

Compilar para Android

```
1  eas build --platform android
```

Compilar para iOS

```
1  eas build --platform ios
```

Nota: Para compilar para iOS, necesitas una cuenta de desarrollador de Apple y una máquina con macOS.

Análisis

Diagrama de Arquitectura de Aplicación

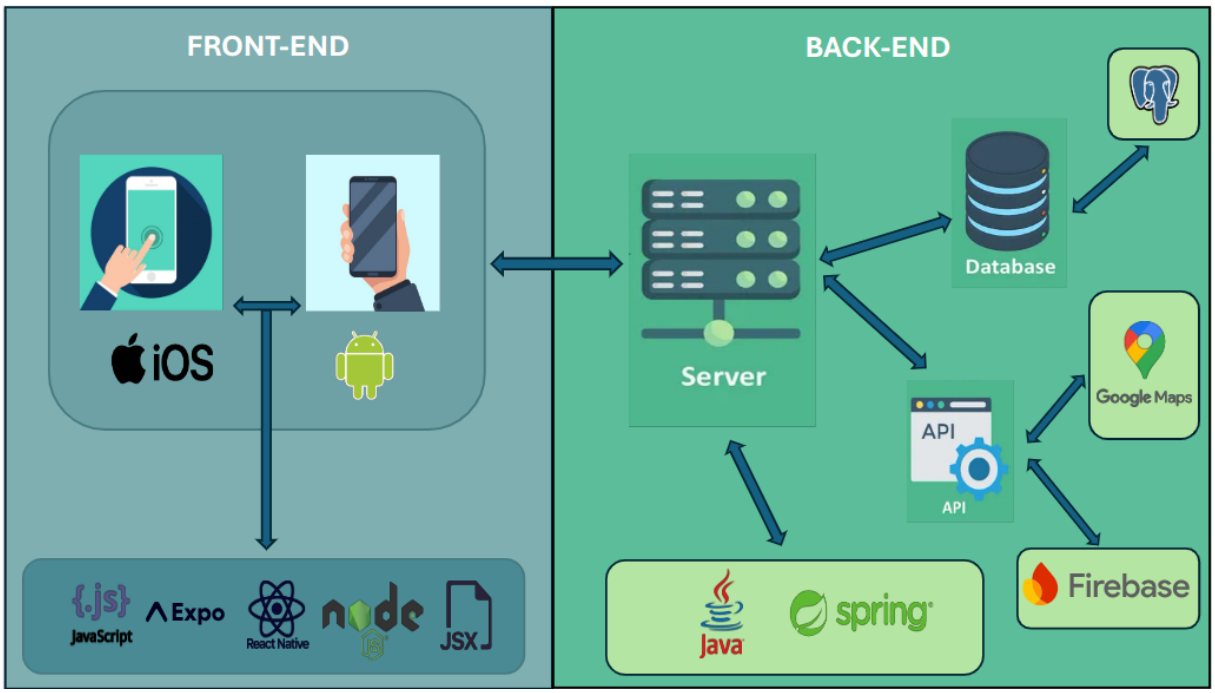

Figure 11: Diagrama de Arquitectura

Componentes Principales:

1. **Cliente (Frontend)**:
    - Aplicación móvil desarrollada con React Native y Expo.
    - Utiliza componentes reutilizables y gestiona el estado localmente.
    - Se comunica con el Backend a través de API REST.
2. **Servidor (Backend)**:
    - Desarrollado con Spring Boot.
    - Expone una API RESTful para manejar operaciones CRUD.
    - Implementa seguridad con Spring Security para autenticación y autorización.
    - Se conecta a una base de datos PostgreSQL para almacenar y recuperar datos.
    - Interactúa con Firebase para almacenar y gestionar imágenes.
3. **Base de Datos**:

- PostgreSQL utilizada para almacenar información clave como usuarios, restaurantes, reservas, menús y calificaciones/opiniones.
- Almacena el nombre de imágenes alojadas en Firebase Storage.

4. **Firebase**:
    - **Firebase Storage**: Utilizado para almacenar imágenes relacionadas con los restaurantes.
    - **Firebase Admin SDK**: Integrado con el Backend para la gestión de archivos en Firebase Storage y autenticación adicional.

Interacciones:

- **Cliente (Frontend)**:
    - Realiza peticiones HTTP a las rutas definidas en el Backend para obtener y enviar datos.
    - Recibe URLs de imágenes almacenadas en Firebase a través de respuestas del Backend.
- **Servidor (Backend)**:
    - Recibe peticiones HTTP del Cliente y las enruta a los controladores correspondientes.
    - Gestiona la lógica de negocio y la integración con la base de datos PostgreSQL.
    - Utiliza Firebase Admin SDK para almacenar y gestionar imágenes en Firebase Storage.
- **Base de Datos**:
    - Almacena datos estructurados relacionados con usuarios, restaurantes, reservas, menús y calificaciones/opiniones.
    - Almacena nombre de las imágenes en PostgreSQL relacionadas con los registros de restaurantes.

Diagrama de Casos de Uso

El Diagrama de Casos de Uso es una herramienta fundamental en el desarrollo de software que permite representar las interacciones entre los usuarios (actores) y el sistema. En el contexto del “Sistema de Gestión de Restaurantes” (TastyGo), se han identificado varios actores principales: Cliente, Admin y Empresario.

Este diagrama visualiza cómo cada actor interactúa con la aplicación para realizar diversas acciones:

```
@startuml Diagrama de casos de uso

left to right direction
skinparam packageStyle rectangle

actor Cliente as "Cliente"
actor Admin as "Admin"
actor Empresario as "Empresario"

rectangle "Aplicación TastyGo" {
    usecase "Listar Restaurantes" as ListarRestaurantes
    usecase "Ver Detalles del Restaurante" as VerDetallesRestaurante
    usecase "Hacer Reserva" as HacerReserva
    usecase "Listar Menús" as ListarMenus
    usecase "Buscar Restaurantes" as BuscarRestaurantes
    usecase "Editar Reserva" as EditarReserva
    usecase "Hacer Comentarios" as HacerComentarios
    usecase "Añadir a Favoritos" as AnadirFavoritos

    usecase "Añadir Restaurante" as AnadirRestaurante
    usecase "Editar Restaurante" as EditarRestaurante
    usecase "Añadir Menús" as AnadirMenus
    usecase "Editar Menús" as EditarMenus
    usecase "Ver Reservas" as VerReservas
    usecase "Editar Reservas" as EditarReservas

    usecase "Gestionar Usuarios" as GestionarUsuarios
    usecase "Gestionar Reservas" as GestionarReservas
    usecase "Gestionar Menús" as GestionarMenus


    usecase ( registro / login ) as Login

    ListarRestaurantes .. Login
    VerDetallesRestaurante ..Login
    HacerReserva ..Login
    ListarMenus ..Login
    BuscarRestaurantes .. Login
    EditarReserva .. Login
    HacerComentarios .. Login
    AnadirFavoritos ..Login
    AnadirRestaurante .. Login
    EditarRestaurante .. Login
    AnadirMenus .. Login
    EditarMenus .. Login
    VerReservas .. Login
    EditarReservas .. Login
    GestionarUsuarios .. Login
    GestionarReservas .. Login
    GestionarMenus .. Login
```

```
51
52
53
54      Cliente --> ListarRestaurantes
55      Cliente --> VerDetallesRestaurante
56      Cliente --> HacerReserva
57      Cliente --> ListarMenus
58      Cliente --> BuscarRestaurantes
59      Cliente --> EditarReserva
60      Cliente --> HacerComentarios
61      Cliente --> AnadirFavoritos
62
63      Empresario --> AnadirRestaurante
64      Empresario --> EditarRestaurante
65      Empresario --> AnadirMenus
66      Empresario --> EditarMenus
67      Empresario --> VerReservas
68      Empresario --> EditarReservas
69
70      Admin --> GestionarUsuarios
71      Admin --> GestionarReservas
72      Admin --> GestionarMenus
73
74
75  }
76
77  @enduml
```

Este diagrama muestra cómo cada tipo de usuario interactúa con la aplicación, desde realizar reservas y gestionar menús hasta administrar usuarios y reservas. Proporciona una vista clara de las funcionalidades que el sistema ofrece y cómo se distribuyen entre los diferentes roles de usuarios.

El uso de UML, en particular los Diagramas de Casos de Uso, facilita la comprensión y comunicación de los requerimientos y funcionalidades del sistema tanto para desarrolladores como para usuarios finales.

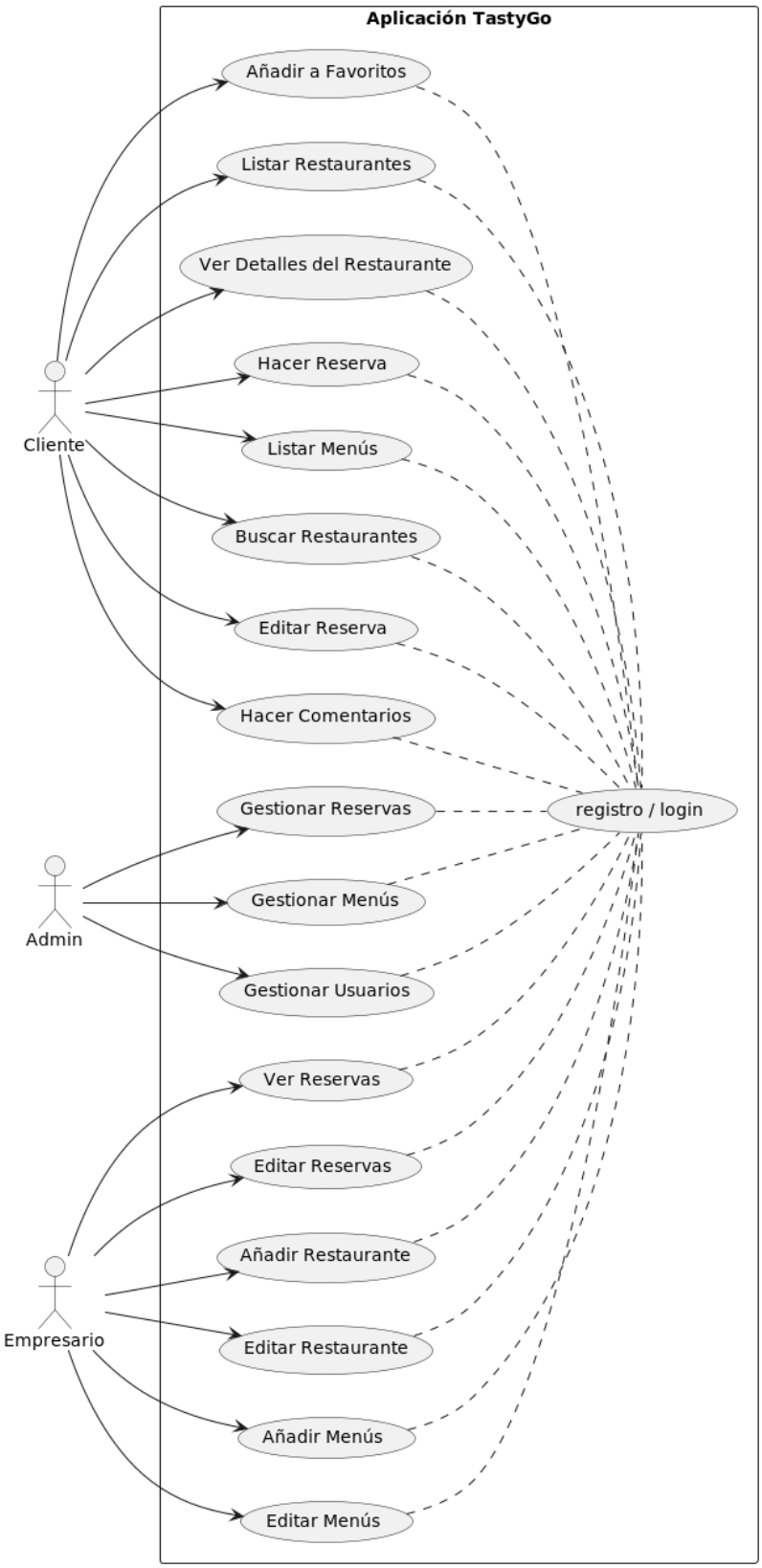

Figure 12: Diagrama Casos de Usos

Diagrama Entidad-Relación

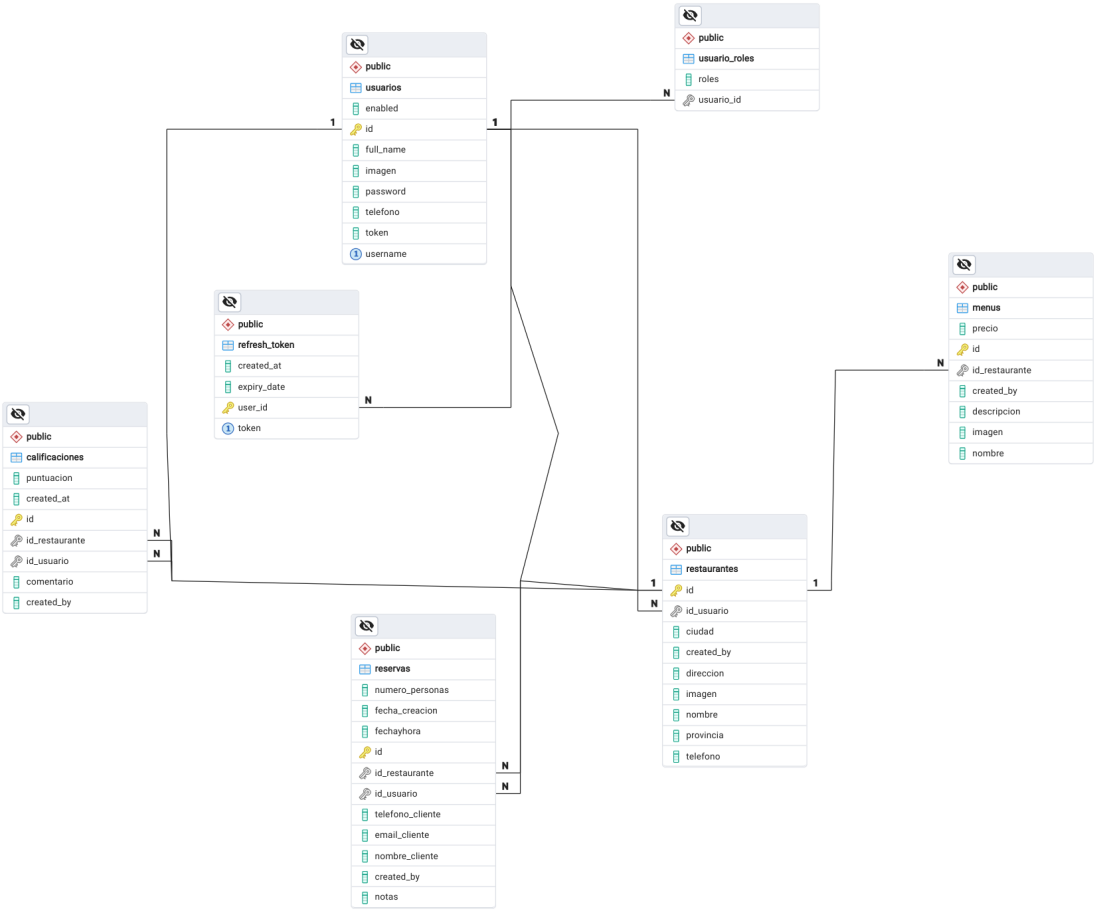

Figure 13: Diagrama Entidad-Relación

El diagrama entidad-relación describe la estructura de la base de datos y las relaciones entre las entidades principales, tales como `usuarios`, `restaurantes`, `reservas`, `menu` y `calificaciones_opiniones`.

Diagrama de Clases

```
@startuml

   enum UsuarioRol {
      ADMIN
      USUARIO
      EMPRESARIO
   }

   class Usuario {
      - id: Long
      - password: String
      - username: String
      - fullName: String
      - telefono: String
      - imagen: String
      - token: String
      - enabled: boolean
      - roles: Set<UsuarioRol>
      - createdBy: String
      - accountNonExpired: boolean
      - accountNonLocked: boolean
      - credentialsNonExpired: boolean
      + getAuthorities(): Collection<? extends GrantedAuthority>
      + isAccountNonExpired(): boolean
      + isAccountNonLocked(): boolean
      + isCredentialsNonExpired(): boolean
      + isEnabled(): boolean
      + getUsername(): String
   }

   class Restaurante {
      - id: Long
      - usuario: Usuario
      - nombre: String
      - ciudad: String
      - provincia: String
      - telefono: String
      - imagen: String
      - direccion: String
      - createdBy: String
   }

   class Reserva {
      - id: Long
      - usuario: Usuario
      - restaurante: Restaurante
      - fechaYHora: LocalDateTime
      - nombreCliente: String
      - telefonoCliente: String
```

```
50          - emailCliente: String
51          - numeroPersonas: int
52          - notas: String
53          - createdBy: String
54          - fechaCreacion: LocalDateTime
55      }
56
57      class RefreshToken {
58          - id: Long
59          - user: Usuario
60          - token: String
61          - expiryDate: Instant
62          - createdAt: Instant
63      }
64
65      class Menu {
66          - id: Long
67          - restaurante: Restaurante
68          - createdBy: String
69          - nombre: String
70          - descripcion: String
71          - precio: Double
72          - imagen: String
73      }
74
75      class Calificacion {
76          - id: Long
77          - usuario: Usuario
78          - restaurante: Restaurante
79          - puntuacion: Double
80          - comentario: String
81          - createdBy: String
82          - createdAt: LocalDateTime
83      }
84
85      Usuario "1" -- "0..*" Reserva : Realiza
86      Usuario "1" -- "1" RefreshToken : Tiene
87      Usuario "1" -- "0..*" Calificacion : Realiza
88      Usuario "1" -- "0..*" Restaurante : Propietario
89
90      Restaurante "1" -- "0..*" Menu : Tiene
91
92  @enduml
```

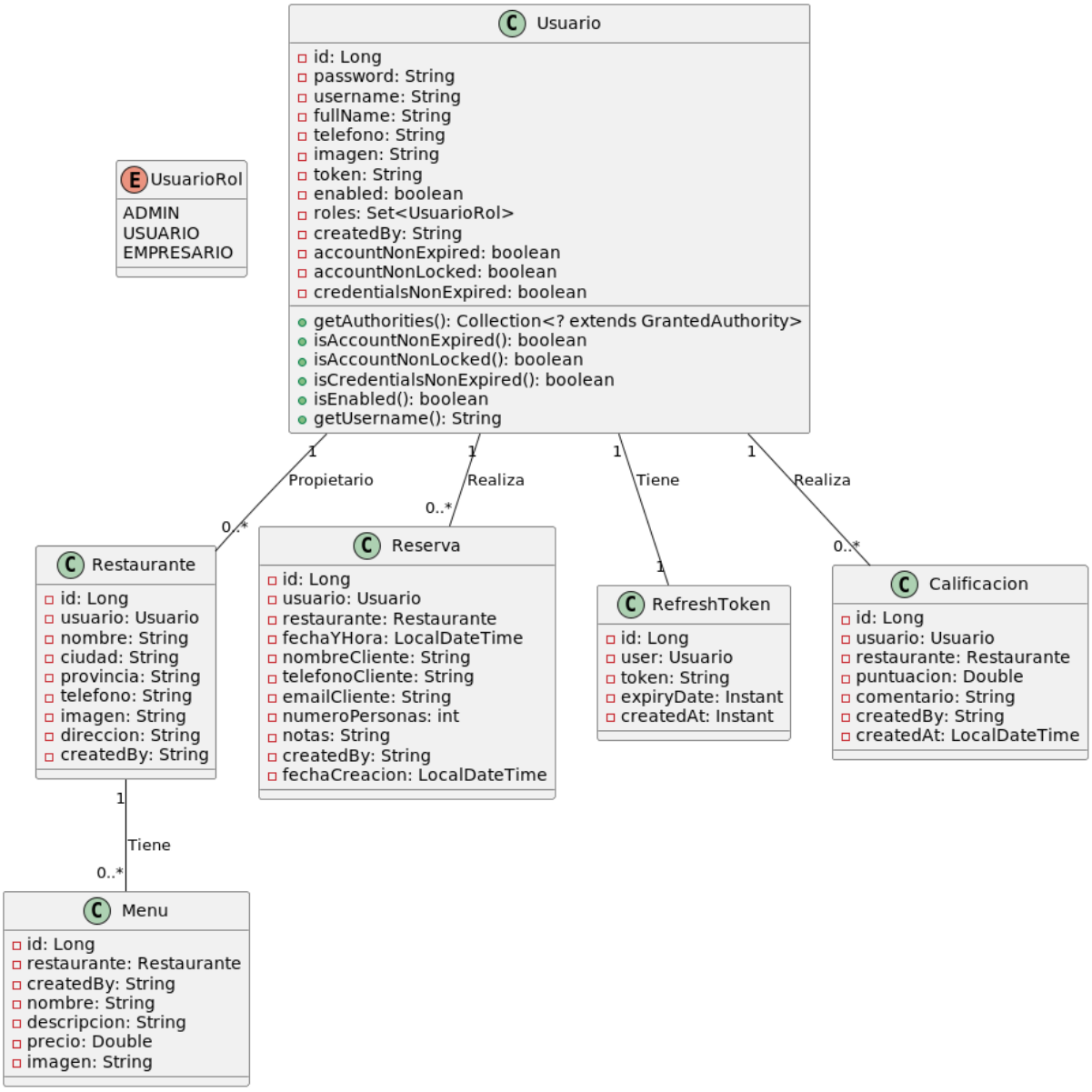

Figure 14: Diagrama de clase

Clases del Sistema de Gestión de Restaurantes

Clase Usuario: Representa un usuario del sistema con todos los atributos necesarios para la autenticación y gestión de información personal.

Atributos:

- `id`: Identificador único del usuario.
- `password`: Contraseña del usuario.
- `username`: Nombre de usuario único.
- `fullName`: Nombre completo del usuario.
- `telefono`: Número de teléfono del usuario.
- `imagen`: URL de la imagen del usuario.
- `token`: Token de autenticación del usuario.
- `enabled`: Indica si la cuenta del usuario está habilitada.
- `roles`: Conjunto de roles (`UsuarioRol`) asignados al usuario.
- `createdBy`: Usuario que creó el registro del usuario.

Métodos:

- `getAuthorities()`: Devuelve las autoridades del usuario para la autenticación.
- Métodos de verificación de la vigencia de la cuenta (`isAccountNonExpired()`, `isAccountNonLocked()`, `isCredentialsNonExpired()`, `isEnabled()`).

Clase Restaurante: Representa un restaurante registrado en el sistema.

Atributos:

- `id`: Identificador único del restaurante.
- `usuario`: Usuario propietario del restaurante.
- `nombre`: Nombre del restaurante.
- `ciudad`: Ciudad donde se encuentra el restaurante.
- `provincia`: Provincia donde se encuentra el restaurante.
- `telefono`: Número de teléfono del restaurante.
- `imagen`: URL de la imagen del restaurante.
- `direccion`: Dirección física del restaurante.
- `createdBy`: Usuario que creó el registro del restaurante.

Clase Reserva: Representa una reserva realizada por un usuario en un restaurante específico.

Atributos:

- `id`: Identificador único de la reserva.
- `usuario`: Usuario que realiza la reserva.
- `restaurante`: Restaurante donde se realiza la reserva.
- `fechaYHora`: Fecha y hora de la reserva.
- `nombreCliente`: Nombre del cliente que realiza la reserva.
- `telefonoCliente`: Teléfono de contacto del cliente.
- `emailCliente`: Correo electrónico del cliente.
- `numeroPersonas`: Número de personas incluidas en la reserva.
- `notas`: Notas adicionales sobre la reserva.
- `createdBy`: Usuario que creó la reserva.
- `fechaCreacion`: Fecha y hora de creación de la reserva.

Clase RefreshToken: Representa un token de actualización asociado a un usuario para autenticación.

Atributos:

- `id`: Identificador único del token.
- `user`: Usuario al que pertenece el token.
- `token`: Valor único del token.
- `expiryDate`: Fecha de expiración del token.
- `createdAt`: Fecha de creación del token.

Clase Menu: Representa un menú asociado a un restaurante.

Atributos:

- `id`: Identificador único del menú.
- `restaurante`: Restaurante al que pertenece el menú.
- `createdBy`: Usuario que creó el menú.
- `nombre`: Nombre del menú.
- `descripcion`: Descripción del menú.
- `precio`: Precio del menú.
- `imagen`: URL de la imagen del menú.

Clase Calificacion: Representa la calificación y comentario de un usuario sobre un restaurante.

Atributos:

- `id`: Identificador único de la calificación.
- `usuario`: Usuario que realiza la calificación.
- `restaurante`: Restaurante calificado.
- `puntuacion`: Puntuación otorgada al restaurante.
- `comentario`: Comentario realizado por el usuario.
- `createdBy`: Usuario que creó la calificación.
- `createdAt`: Fecha y hora de creación de la calificación.

Relaciones

Relación Usuario - Reserva

- Un usuario puede realizar varias reservas (1 a 0..*).

Relación Usuario - RefreshToken

- Un usuario puede tener asociado un token de actualización (1 a 1).

Relación Usuario - Calificación

- Un usuario puede realizar varias calificaciones de restaurantes (1 a 0..*).

Relación Usuario - Restaurante

- Un usuario puede ser propietario de uno o varios restaurantes (1 a 0..1).

Relación Restaurante - Menu

- Un restaurante puede tener varios menús (1 a 0..*).

Implementación

Explicación del Código Relevante

GenerateToken() En el siguiente fragmento de código en Java se muestra la función `generateToken`, la cual es responsable de generar un token JWT para autenticación y autorización de usuarios en la aplicación:

```java
public String generateToken(Usuario user) {
        Date tokenExpirationDateTime =
            Date.from(
                Instant.now().plus(Duration.ofDays(30))
                        .atZone(ZoneId.systemDefault())
                        .toInstant()
            );

        return Jwts.builder()
                .setHeaderParam("typ", TOKEN_TYPE)
                .setSubject(user.getId().toString())
                .setIssuedAt(new Date())
                .setExpiration(tokenExpirationDateTime)
                .signWith(secretKey)
                .compact();
}
```

Funcionamiento:

- `Fecha de Expiración`: La función calcula la fecha de expiración del token sumando 30 días a la fecha y hora actuales.
- `Construcción del Token JWT`: Utiliza la biblioteca JJWT para construir el token JWT con los siguientes parámetros:
  - `Header (typ)`: Tipo del token, en este caso “JWT”.
  - `Subject`: Identifica al usuario dentro del token, utilizando el ID del usuario convertido a string.
- `Issued At (iat)`: Fecha y hora en que se emitió el token.
- `Expiration (exp)`: Fecha y hora de expiración del token.
- `Firma del Token`: Utiliza una clave secreta (secretKey) para firmar el token, asegurando su autenticidad y seguridad. Esta función es crucial para generar tokens seguros que permitan la autenticación de usuarios en la aplicación de manera eficiente y segura, utilizando estándares de seguridad modernos como JWT.

ExceptionControllerAdvice()

Controlador de Excepciones para Autenticación El siguiente fragmento de código Java muestra un controlador de excepciones (`ExceptionControllerAdvice`) utilizado en una aplicación Spring Boot para manejar excepciones relacionadas con la autenticación:

```java
@RestControllerAdvice
public class ExceptionControllerAdvice {

    @ExceptionHandler({AuthenticationException.class})
    public ResponseEntity<?> handleAuthenticationException(
        AuthenticationException ex, HttpServletRequest request) {
         return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
                 .header("WWW-Authenticate", "Bearer")
                 .body(ErrorDetails.of(
                         "error",
                         "Error de autenticación: " + ex.getMessage(),
                         request.getRequestURI()));

    }
}
```

`@RestControllerAdvice`: Esta anotación marca esta clase como un consejo global para manejar excepciones en controladores REST dentro de la aplicación Spring Boot.

`@ExceptionHandler({ AuthenticationException.class })`: Este método es invocado cuando se produce una excepción del tipo `AuthenticationException`, indicando un error de autenticación.

Respuesta de Error (`ResponseEntity`):

- **Estado HTTP (`HttpStatus.UNAUTHORIZED`):** Devuelve un código de estado HTTP 401 (Unauthorized) indicando que la solicitud no tiene autorización adecuada.
- **Cabecera `WWW-Authenticate`:** Agrega una cabecera `WWW-Authenticate` con el valor "Bearer", indicando que se espera un token de autenticación en formato Bearer en las solicitudes futuras.
- **Cuerpo de la respuesta (`body`):** Utiliza la clase `ErrorDetails.of` para crear un objeto de detalles de error que contiene:
  - Tipo de error ("error").
  - Mensaje de error específico ("Error de autenticación:" + ex.getMessage()), que concatena el mensaje de la excepción `AuthenticationException`.
  - URI de la solicitud actual (`request.getRequestURI()`), que proporciona la ruta URI que originó la excepción.

Este controlador de excepciones asegura que las excepciones de autenticación sean manejadas de manera adecuada, devolviendo respuestas HTTP claras y proporcionando información útil para diagnosticar y solucionar problemas relacionados con la autenticación en la aplicación Spring Boot.

CustomUserDetailsService() Este servicio `CustomUserDetailsService` es crucial en la configuración de seguridad de Spring, asegurando que los detalles del usuario sean cargados correctamente durante el proceso de autenticación y autorización en la aplicación.

```
1     @Service("userDetailsService")
2     @RequiredArgsConstructor
3     public class CustomUserDetailsService implements UserDetailsService
          {
4
5         @Autowired
6         private final UsuarioService userService;
7
8         @Override
9         public UserDetails loadUserByUsername(String username) throws
              UsernameNotFoundException {
10            return userService.findByUsername(username)
11                    .orElseThrow(() -> new UsernameNotFoundException("
                      No se encontro usuario con el username: " +
                      username));
12        }
13    }
```

`@Service("userDetailsService")`: Marca la clase `CustomUserDetailsService` como un servicio gestionado por Spring con el nombre "userDetailsService". Este servicio implementa la interfaz `UserDetailsService`, que es esencial para cargar detalles de usuario durante el proceso de autenticación.

`@RequiredArgsConstructor`: Lombok genera un constructor para la clase que inicializa automáticamente el campo **`final`** `UsuarioService userService`. Este servicio se utiliza para interactuar con la capa de persistencia y recuperar los detalles del usuario según el nombre de usuario proporcionado.

`loadUserByUsername(String username)`: Método implementado de la interfaz UserDetailsService que carga los detalles del usuario basado en el nombre de usuario proporcionado. Utiliza el servicio `UsuarioService` para buscar un usuario por su nombre de usuario. Si no se encuentra ningún usuario, se lanza una excepción `UsernameNotFoundException` con un mensaje adecuado.

UsuarioController() Este controlador `UsuarioController` proporciona endpoints RESTful para manejar operaciones relacionadas con los usuarios, como obtener una lista de usuarios almacenados en la base de datos. Utiliza inyección de dependencias para acceder a los servicios y componentes necesarios para ejecutar estas operaciones de manera eficiente y segura.

```
1   @RestController
2   @RequestMapping("/api")
3   public class UsuarioController {
4
5       @Autowired
6       private UsuarioService usuarioService;
7
8       @GetMapping("/usuarios")
9       public ResponseEntity<UsuarioResponse<List<UserResponse>>>
            obtenerTodosLosUsuarios() {
10          try {
11              List<UserResponse> usuarios = usuarioService.
                    findAllUsuarios();
12              if (usuarios == null || usuarios.isEmpty()) {
13                  UsuarioResponse<List<UserResponse>>
14                          response = new UsuarioResponse<>(
15                          "error", "No hay usuarios almacenados en la
                                base de datos", null);
16                  return ResponseEntity.status(HttpStatus.NOT_FOUND).
                        body(response);
17              }
18              UsuarioResponse<List<UserResponse>> response = new
                    UsuarioResponse<>(
19                      "ok", "Usuarios listados con éxito", usuarios);
20              return ResponseEntity.ok(response);
21          } catch (Exception e) {
22              UsuarioResponse<List<UserResponse>> response = new
                    UsuarioResponse<>(
23                      "error", "Ocurrió un error al listar los
                            usuarios: " + e.getMessage(), null);
24              return ResponseEntity.status(HttpStatus.
                    INTERNAL_SERVER_ERROR).body(response);
25          }
26      }
27  }
```

`@RestController`: Esta anotación marca la clase `UsuarioController` como un controlador de Spring que maneja las solicitudes REST.

`@RequestMapping("/api")`: Especifica que todas las solicitudes HTTP manejadas por este controlador deben comenzar con `/api` en su ruta URL.

Inyección de Dependencias: - `@Autowired private UsuarioService usuarioService;`: Inyecta una instancia del servicio `UsuarioService`, que se utiliza para interactuar con los datos de los usuarios en la capa de servicio.

`@GetMapping("/usuarios"):` Este método maneja las solicitudes GET dirigidas a `/api/usuarios`. Retorna una respuesta HTTP que contiene una lista de usuarios en formato JSON.

Funcionalidad del Método: - Intenta obtener todos los usuarios utilizando `usuarioService.findAllUsuarios()`. - Si no se encuentran usuarios (`usuarios == null || usuarios.isEmpty()`), devuelve una respuesta con estado HTTP 404 (Not Found) y un mensaje indicando que no hay usuarios almacenados. - Si se encuentran usuarios, devuelve una respuesta con estado HTTP 200 (OK) y la lista de usuarios junto con un mensaje de éxito. - Si ocurre alguna excepción durante el procesamiento, captura la excepción, devuelve una respuesta con estado HTTP 500 (Internal Server Error) y un mensaje indicando el error ocurrido.

Conclusiones

Durante el desarrollo de este proyecto he adquirido conocimientos y experiencias significativas que no solo han enriquecido mi entendimiento sobre el desarrollo multiplataforma, sino que también han moldeado mi enfoque hacia futuros proyectos:

- **Aprendizajes Nuevos:**
  - He explorado en profundidad el desarrollo multiplataforma utilizando tecnologías como React Native con Expo y Spring Boot, comprendiendo la importancia de escribir código eficiente y adaptable a diferentes sistemas operativos desde una sola base.
  - Mejoré mis habilidades en la gestión de bases de datos, implementando estrategias efectivas para la organización y recuperación de datos de manera óptima.
- **Reflexión sobre Decisiones:**
  - En retrospectiva, reconocí la importancia de una planificación inicial más detallada, lo que habría facilitado un desarrollo más fluido y eficiente.
  - Aunque las decisiones tecnológicas fueron acertadas en su mayoría, aprendí la necesidad de mantenerme actualizado con las últimas tecnologías y evaluaciones continuas para optimizar aún más el rendimiento y la experiencia del usuario.
- **Próximos Pasos:**
  - Mi compromiso es mejorar continuamente el Sistema de Gestión de Restaurantes, integrando nuevas funcionalidades como pagos en línea, análisis avanzado de datos y optimización de procesos operativos.
  - Exploraré oportunidades para expandir la aplicación a otras plataformas y dispositivos, para alcanzar una mayor audiencia y utilidad.
  - Continuaré fortaleciendo mis conocimientos en seguridad cibernética y protección de datos, asegurando la confianza y seguridad de los usuarios en cada paso del desarrollo.

Este proyecto no solo fue una oportunidad para desarrollar una aplicación funcional, sino también un viaje de aprendizaje y crecimiento profesional que me prepara para desafíos futuros en el dinámico campo del desarrollo de software móvil y multiplataforma.

Bibliografía

Referencias

1. Stack Overflow. (2024, 15 de junio). Introduction to Cross-Platform Mobile Development.

   https://stackoverflow.com/questions/34103167/what-is-cross-platform-mobile-development

2. TechCrunch. (2023, 10 de mayo). Benefits of Cross-Platform Mobile Development.

   https://techcrunch.com/2023/05/10/benefits-of-cross-platform-mobile-development/

3. React Native. (2024, 15 de junio). Learn Once, Write Anywhere.

   https://reactnative.dev/

- Curso para hacer api-rest-segura-spring-boot-jwt : https://openwebinars.net/academia/aprende/api-rest-segura-spring-boot-jwt/.
- Curso para hacer api-rest-spring-boot https://openwebinars.net/academia/aprende/primera-api-rest-spring-boot/.
- Curso para mejorar respuestas api-rest-spring-boot https://openwebinars.net/academia/aprende/json-views-mejorar-respuesta-api-rest-spring-boot/.
- Curso creación de aplicaciones con react-spring-boot https://openwebinars.net/academia/aprende/desarrollar-app-spring-boot-react/.
- Curso git https://openwebinars.net/academia/aprende/git/.
- Curso java 8 https://openwebinars.net/academia/aprende/java-8-desde-cero/.
- Curso React-intermedio https://openwebinars.net/academia/aprende/react-intermedio/.