

# **DOCUMENTO TÉCNICO: Sistema de Alertas**

**AÑO: 2023**

Entrevistado: Riveros, Sebastián

Entrevistador: D'angelo, Nicolas

## ÍNDICE DE CONTENIDOS

<b>Introducción:</b>	1
<b>Ambiente de desarrollo:</b>	1
<b>Tecnologías utilizadas:</b>	1
<b>Patrones de diseño utilizados:</b>	1
<b>Polimorfismo:</b>	2
<b>Funcionamiento del programa:</b>	2
Premisa importante:	2
<b>Endpoints para probar:</b>	3
<b>Diagrama de clases utilizado para resolver el reto:</b>	5
<b>Consulta SQL:</b>	5
<b>Casos de prueba:</b>	5

## Documentación técnica Woowup - Challenge

### Introducción:

El siguiente documento tiene como fin poder lograr un entendimiento del funcionamiento general y detallado del programa desarrollado para la entrevista técnica para la empresa Woowup.

### Ambiente de desarrollo:

- Sistema operativo: GNU/Linux Pop OS 22.04
- Memoria RAM: 16GB
- Procesador: Intel i7 8va generación.

### Tecnologías utilizadas:

- Java 8
- Spring Boot
- Visual Studio Code
- Thunder Client para pruebas
- Mockito

### Patrones de diseño utilizados:

#### 1. Modelo-Vista-Controlador (MVC)

Riveros Sebastian  
DNI: 42974713

Descripción:

El patrón MVC organiza la aplicación en tres componentes: Modelo (estructura de datos y reglas de negocio), Vista (interfaz de usuario) y Controlador (lógica de negocio). Este enfoque mejora la modularidad y el mantenimiento del código.

Aplicación:

Modelo: Clases de entidad (Usuario, Alerta, Tema).

Controlador: Clases de servicio (UsuarioServiceImpl, AlertaServiceImpl, TemaServiceImpl).

## 2. Inyección de Dependencias

Descripción:

La Inyección de Dependencias desacopla componentes de la aplicación, facilitando la gestión y sustitución de dependencias. En el código, se implementa mediante la anotación `@Autowired` de Spring.

## 3. Objetos de Transferencia de Datos (DTO)

Descripción:

Los DTOs (Objetos de Transferencia de Datos) encapsulan y transfieren información entre subsistemas. En el código, se utilizan (DTOLong, DTOTexto, etc.) para estructurar y desacoplar datos.

# Polimorfismo:

El polimorfismo puede observarse en el uso de interfaces y clases abstractas, lo que permite que objetos de diferentes tipos sean tratados de manera uniforme.

## Aplicación en el Código:

Ejemplos de polimorfismo incluyen la implementación de interfaces como `UsuarioService`, `TemaService`, y `AlertaService`, que permiten un tratamiento uniforme de distintos servicios a través de una interfaz común.

## Funcionamiento del programa:

Para poder apreciar el funcionamiento del programa, de forma muy sencilla, lo primero que se debe realizar, es crear un usuario, crear un tema, suscribir al usuario al tema correspondiente. Luego, crear una alerta, con el usuario y el tema, el manejo de excepciones permitirán evitar errores de ingreso de datos. Una vez creada la alerta, se pueden listar las notificaciones para el usuario, también se pueden marcar como leídas etc. Por otro lado, también podemos realizar consulta sobre las alertas vigentes y podemos dar de baja una alerta.

## Premisa importante:

En el código proporcionado, se ha optado por utilizar nombres en lugar de identificadores (ID) en ciertos métodos. Es importante señalar que esta elección se ha hecho a modo ilustrativo con el objetivo de facilitar la comprensión durante la entrevista y resaltar el funcionamiento del programa de manera más clara.

Es crucial reconocer que, en entornos de producción y mejores prácticas de desarrollo, se recomienda el uso de identificadores únicos (ID) para cada entidad. La utilización de ID proporciona una forma más robusta y consistente de referenciar elementos dentro del sistema, contribuyendo a la integridad y eficiencia del código.

La decisión de utilizar nombres en este contexto específico es con fines pedagógicos y no refleja necesariamente las prácticas ideales en un entorno de desarrollo real. En un ambiente de producción, se insta a seguir las mejores prácticas y emplear identificadores adecuados para garantizar la calidad y mantenibilidad del código.

## Endpoints para probar:

Una vez ejecutado el programa, tanto con Postman como Thunder Client, es posible probar la respuesta de los endpoints, los cuales son los que se mencionan a continuación:

### **AlertaController:**

Endpoint: POST /crearAlerta

Riveros Sebastian  
DNI: 42974713

Descripción: Crea una nueva alerta utilizando la información proporcionada en el cuerpo de la solicitud.

Body:

```
{  
  "nombreAlerta": "ATENTOS AL ZONDA",  
  "nombreTema": "tema5",  
  "descripcionTema": "Gran tema",  
  "esDirigidaParaTodos": false,  
  "nombreUsuario": "Seba",  
  "nombreTipoAlerta": "URGENTE"  
}
```

Endpoint: POST /bajaAlerta

Descripción: Da de baja una alerta según la información proporcionada en el cuerpo de la solicitud.

Body:

```
{  
  "nombreAlerta": "ATENTOS AL ZONDA1"  
}
```

Endpoint: GET /consultarAlertas

Descripción: Consulta y devuelve la lista de alertas existentes.

### **TemaController:**

Endpoint: POST /crearTema

Descripción: Crea un nuevo tema utilizando la información proporcionada en el cuerpo de la solicitud.

Body:

### **UsuarioController:**

Endpoint: POST /suscripcionTema

Descripción: Suscribe a un usuario a una alerta según la información proporcionada en el cuerpo de la solicitud.

Body:

```
{  
  "nombreUsuario": "Juan5",  
  "idTema": "1"  
}
```

Endpoint: GET /listarNotificacionesUsuario

Descripción: Lista las notificaciones del usuario especificadas en el cuerpo de la solicitud.

Body:

Riveros Sebastian  
DNI: 42974713

```
{  
  "cadena": "Seba"  
}
```

Endpoint: POST /marcarAlertaLeida

Descripción: Marca una notificación como leída según la información proporcionada en el cuerpo de la solicitud.

Body:

```
{  
  "id": 1  
}
```

Parámetros: localhost:8080/marcarAlertaLeida?nombreUsuario=Seba

Endpoint: POST /registrarUsuario

Descripción: Registra un nuevo usuario utilizando la información proporcionada en el cuerpo de la solicitud.

Body:

```
{  
  "nombreUsuario": "Juan",  
  "apellidoUsuario": "Riveros"  
}
```

Se adjunta el link del archivo JSON con los cuerpos de cada solicitud para facilitar la prueba del programa:

[https://drive.google.com/file/d/1XTaLMMS\\_I1xTBubzVSZ8pDwJx\\_IGINv/view?usp=sharing](https://drive.google.com/file/d/1XTaLMMS_I1xTBubzVSZ8pDwJx_IGINv/view?usp=sharing)

Diagrama de clases utilizado para resolver el reto:



Figura 1: Diagrama de clases

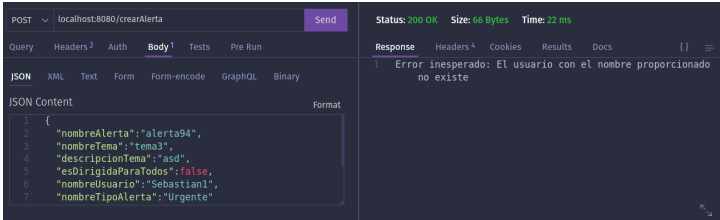
## Consulta SQL:

```
SELECT C.ID, C.Nombre, C.Apellido
FROM Clientes C
JOIN Ventas V ON C.ID = V.Id_cliente
WHERE V.Fecha >= DATEADD(MONTH, -12, GETDATE())
GROUP BY C.ID, C.Nombre, C.Apellido
HAVING SUM(V.Importe) > 100000;
```

## Casos de prueba:

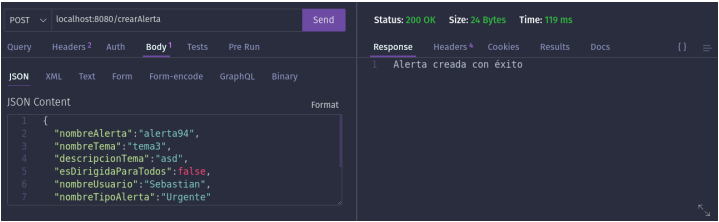
Nombre del caso	Alta usuario ya existente
Descripción	Se realiza el alta a un usuario con un nombre de usuario ya existente
Datos de prueba	nombreUsuario: Sebastian
Resultado esperado	Se espera que NO se dé el alta del usuario y que lance una excepción

Resultado obtenido	No se da de alta al usuario y se lanza una excepción
Evidencias	
RESULTADO DE PRUEBA	ÉXITO

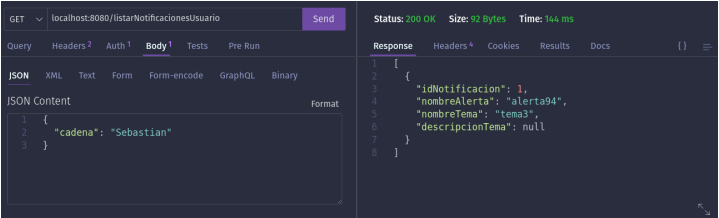
Nombre del caso	Crear una alerta con usuario NO existente
Descripción	Se realiza el alta de una alerta, asignando un usuario que no existe a la misma
Datos de prueba	nombreUsuario: Sebastian
Resultado esperado	Se espera que NO se dé el alta de la alerta y que lance una excepción
Resultado obtenido	No se da de alta la alerta y lanza una excepción
Evidencias	
RESULTADO DE PRUEBA	ÉXITO

Nombre del caso	Leer notificación de usuario
Descripción	Se realiza la lectura de una notificación para que no aparezca mas como SIN VER
Datos de prueba	nombreUsuario: Sebastian idNotificación: 1
Resultado esperado	Se espera que al leer la notificación, deje de aparecer en la lista de notificaciones del usuario.
Resultado obtenido	El usuario lee la notificación y la misma deja de aparecer en la lista de notificaciones.
Evidencias	1- Se crea la Alerta para Sebastian

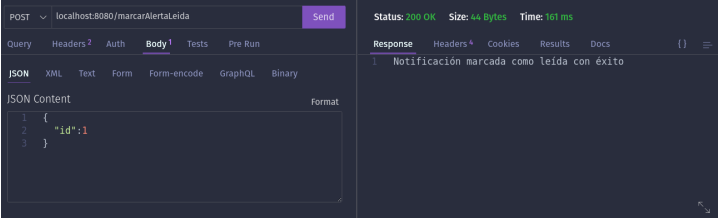




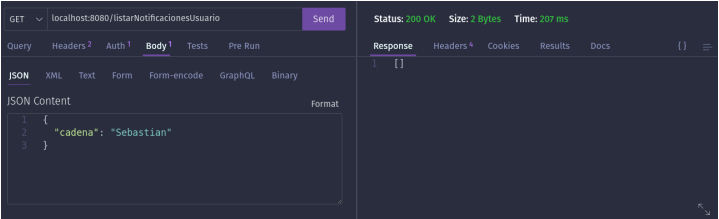
2-Se listan las notificaciones para Sebastian



3-Ejecutamos el método de leer notificación e ingresamos el idNotificacion



4-Volvemos a listar las notificaciones



RESULTADO DE PRUEBA

ÉXITO