

# Sistema de Gestión de Tareas Basado en Microservicios con Spring Boot, OAuth2, Docker y Angular.

ABDESAMAD OUBRAHIM AKKOUH



## Contenido

Resumen.....	3
1    Introducción .....	3
1.1    Contexto del problema a resolver.....	3
1.2    Motivación detrás del proyecto .....	4
2    Arquitectura del proyecto .....	4
3    Descripción de los Microservicios .....	5
3.1    User .....	5
3.2    Tarea y Notificación.....	5
3.3    ConfigServer .....	6
3.4    EurekaServer .....	6
3.5    SpringBootAdmin .....	6
3.6    KaycoakAdapter .....	6
3.7    ApiGateway .....	7
3.8    MAVEN .....	7
4    Tecnologías utilizadas.....	7
4.1    Lenguajes de programación .....	7
4.2    Frameworks y herramientas: .....	8
4.3    Bases de datos.....	8
4.4    Contenedores y despliegue.....	8
4.5    Gestión de configuración y descubrimiento: .....	8
4.6    Autenticación y autorización.....	9
4.7    Monitorización .....	9
5    Pruebas y validación.....	9
5.1    Pruebas unitarias.....	9
5.2    Pruebas de integración .....	14
5.3    Errores y excepciones: .....	15
6    Creación, Gestión y Despliegue de Imágenes Docker con Docker Compose.....	16
6.1    Creación de una imagen Docker .....	16
7    Angular .....	17
7.1    Descripción de las Vistas y su Funcionalidad en Angular .....	18
8    Errores y Soluciones .....	25
9    Conclusión .....	26

## Resumen

El proyecto es un sistema de gestión de tareas hecho en una arquitectura de microservicios utilizando Spring Boot, Docker y otras herramientas que explicare más adelante. Está desarrollado para ayudar a los usuarios a organizar sus tareas de forma eficiente, donde también, pueden compartirlas con otros usuarios.

El objetivo es proporcionar una solución que sea segura y fácil de moldear para futuros cambios, y así, facilitar la creación, asignación y seguimiento de tareas maximizando la productividad del usuario.

Algunas de las herramientas que utilizo para garantizar una arquitectura actual y funcional son las siguientes: Config Server, es un microservicio que accede a un repositorio local Git como fuente de datos para una configuración centralizada y asignación de perfiles. También utilizo API Gateway para gestionar las rutas y la seguridad, y Netflix Eureka para asegurar la comunicación entre los microservicios, ya que, con esta herramienta, se registran y descubren servicios. También utilizo Spring Boot Admin para supervisar el sistema. Keycloak para la autenticación y autorización segura. Para la base de datos utilizo PostgreSQL, gestionada con pgAdmin para facilitar la administración y analizar datos. Y Junit para las pruebas y un front en Angular.

Muchas de estas herramientas las consigo desplegar de forma eficiente mediante Docker Compose. También utilizo Docker para crear contenedores independientes para cada uno de los microservicios. Para la documentación y prueba de las APIs, se emplean herramientas como Swagger y Postman.

## 1 Introducción

### 1.1 Contexto del problema a resolver

Hoy en día, muchas personas y equipos se enfrentan a dificultades para gestionar y organizar sus tareas de forma efectiva, ya que faltan herramientas centralizadas y adaptables. Existen soluciones, pero suelen ser costosas o difíciles de implementar en entornos personalizados.

## 1.2 Motivación detrás del proyecto

La principal motivación por la que hago este proyecto es para demostrar mis conocimientos en la arquitectura de microservicios, gestión de contenedores con Docker, tecnologías actuales como Spring Boot, etc. Además, este proyecto está diseñado como una solución backend que puede ser utilizada por un cliente frontend para la gestión de tareas. Ofrece flexibilidad para su integración en proyectos más grandes y se adapta a las necesidades del usuario.

## 2 Arquitectura del proyecto



La arquitectura del proyecto se divide en dos dominios:

### InfraestructuraDomain

Este dominio contiene los microservicios que se encargan de la configuración y gestión de la infraestructura del proyecto, que son ConfigServer, EurekaServer, SpringBootAdmin, ApiGateway y KeycloakAdapter.

### BusinessDomain

Este dominio contiene los microservicios que se encargan de la lógica de negocio, y son User, Notificación Y tarea.

Todos estos microservicios los explicaré más adelante cada uno.

Para simplificar la gestión de dependencias de los microservicios he utilizado carpetas pom padre. Esto me permite centralizar las dependencias comunes, permitiendo así,

que los microservicios hereden automáticamente las configuraciones o dependencias necesarias sin tener que repetir líneas en cada fichero pom de cada microservicio.

Soy consciente de que esta decisión puede afectar la independencia de cada microservicio. Al compartir un mismo POM padre, cualquier pequeño error o cambio en las dependencias comunes puede perjudicar todos los microservicios. Esto puede dificultar tanto el despliegue como el mantenimiento independiente de cada uno.

## **3 Descripción de los Microservicios**

### **3.1 User**

Este microservicio está diseñado para gestionar usuarios dentro del proyecto. Además, está documentado utilizando Swagger OpenAPI para que otros desarrolladores puedan comprender mejor los endpoints y su uso mediante una interfaz visual y bien estructurada.

Este microservicio es el más completo dentro de bussinesdomain, ya que, utilizo una clase que se utiliza para definir un modelo estándar, REC 7807, de respuesta para errores o excepciones. También se manejan de forma global las excepciones, y capturar así, errores lanzados en el microservicio. Mockito y MockMvc para las pruebas unitarias y de integración, etc.

Ahora mismo, me falta una capa que sirva como puente entre el controlador y la entidad. Esto es importante porque hace que el controlador no tenga un acceso directo a la entidad, lo que lo hace más seguro y organizado. Para lograr esto se puede usar un mapeo con DTOs. Esto mejora la seguridad y hace que el proyecto sea más fácil de mantener y entender.

### **3.2 Tarea y Notificación**

Estos dos microservicios son muy parecidos y más simples que el anterior. Gestionan tareas y notificaciones dentro del proyecto. También se estructuran en diferentes paquetes para mantener una organización clara.

### 3.3 ConfigServer

Este microservicio se encarga de centralizar el uso de configuraciones para los demás microservicios. Accede a un repositorio Git local donde se encuentran los archivos de propiedades yaml o properties, permitiendo así, administrar configuraciones por perfil, como local, preproducción y producción.

### 3.4 EurekaServer

Este actúa como un servidor de registro de servicios. Facilita la comunicación entre microservicios permitiendo que se registren y descubran entre ellos. Desde su archivo de propiedades, se configuran parámetros como el puerto, las zonas de disponibilidad y las opciones de registro. Gracias a esto los microservicios cliente pueden registrarse en el servidor y consultar otros ya registrados. Es importante y garantiza que el proyecto sea más flexible y escalable.

Con este accedemos a una interfaz gráfica que nos permite visualizar los microservicios registrados y obtener información sobre ellos.

### 3.5 SpringBootAdmin

Este microservicio monitoriza, administra y supervisa los microservicios del proyecto de forma centralizada. Gracias al servidor de Netflix Eureka descubre automáticamente los microservicios registrados, permitiendo gestionar su estado y operaciones. Nos permite acceder a una interfaz gráfica donde se puede consultar información detallada como el estado de cada microservicio, logs, configuraciones activas, etc, facilitando su administración y asegurando un buen funcionamiento de todo el proyecto.

### 3.6 KaycoakAdapter

En este microservicio utilizo Keycloak Adapter para gestionar la seguridad de la aplicación implementando autenticación y autorización basadas en el protocolo OAuth2. Keycloak actúa como un servidor de identidad, permitiendo a los usuarios iniciar sesión de forma segura y obtener tokens de acceso para autenticar sus solicitudes. Con esto el microservicio puede validar las credenciales de los usuarios y gestionar sus permisos

sin desarrollar directamente la lógica de seguridad, alojando estas responsabilidades en Keycloak para que el proyecto sea más simple y escalable.

### **3.7 ApiGateway**

Este microservicio dirige todas las solicitudes a los microservicios correspondientes. También se encarga de implementar seguridad, autenticación, autorización, y logging para las solicitudes que llegan al a todo el proyecto.

### **3.8 MAVEN**

Por último, en los archivos pom.xml utilizo dependencias para incluir bibliotecas de trabajo que son necesarios para el buen funcionamiento del proyecto. Uso dependencias tanto de servidor como de cliente. Algunas dependencias importantes son spring-cloud-starter-config, que permite gestionar configuraciones externas y spring-cloud-starter-netflix-eureka-client, que permite la integración con Eureka para el descubrimiento de microservicios. También incluye spring-boot-devtools, que facilita el desarrollo con recarga automática.

## **4 Tecnologías utilizadas**

### **4.1 Lenguajes de programación**

En este proyecto utilizo Java como lenguaje principal para el desarrollo del backend y TypeScript para el front. Quiero añadir que en mi Git Hub podrás encontrar más proyectos que he realizado con distintos lenguajes de programación como javascript, PHP, etc. Tambien encontrarás otros en los que utilizo Bootstrap, html, css, etc.



## 4.2 Frameworks y herramientas:

Los frameworks que utilizo son Spring Boot que es el principal para la creación de microservicios, incluyendo componentes como Spring Data JPA para la gestión de datos y Angular para el front del proyecto.

Para documentar las APIs, y facilitar así, la comprensión para otros desarrolladores que vayan a utilizar el proyecto, he usado Swagger OpenAPI. Y para probar y validar los endpoints de cada controlador rest he utilizado Postman.

## 4.3 Bases de datos

He utilizado PostgreSQL como sistema de gestión de bases de datos relacional por su buen rendimiento. La base de datos es administrada mediante pgAdmin, que permite gestionar las tablas y ejecutar consultas de forma eficiente.

## 4.4 Contenedores y despliegue

He implementado Docker para contenerizar los microservicios mediante un archivo dockerfile en cada microservicio, lo que garantiza un entorno de ejecución consistente. Además, he utilizado Docker Compose para orquestar múltiples contenedores y facilitar la integración entre los diferentes servicios, como la base de datos PostgreSQL, Keycloak y otros microservicios.

## 4.5 Gestión de configuración y descubrimiento:

Para centralizar la configuración de los microservicios uso Spring Cloud Config Server. Utilizo un repositorio Git local como fuente de datos para los perfiles.

Y Netflix Eureka para el registro y descubrimiento de los microservicios, facilitando la comunicación entre ellos.

## 4.6 Autenticación y autorización

Para esto utilizo Keycloak como servidor de autenticación, proporcionando una solución robusta para gestionar usuarios, roles y permisos.

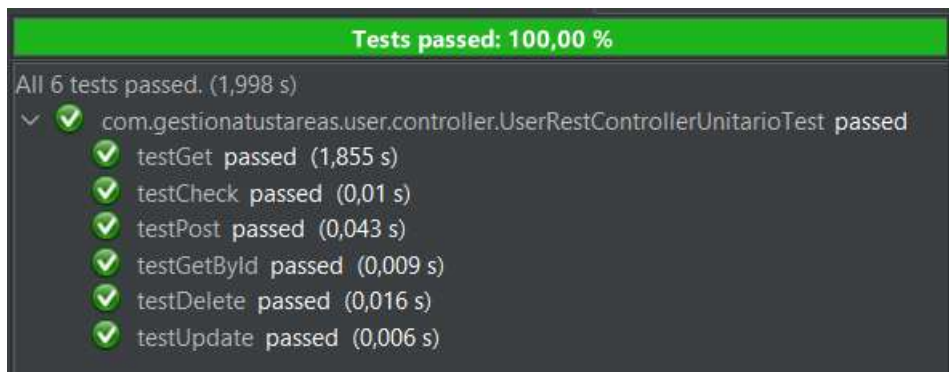
## 4.7 Monitorización

He usado Spring Boot Admin para monitorear el estado y el rendimiento de los microservicios, asegurando así, su correcto funcionamiento.

# 5 Pruebas y validación

## 5.1 Pruebas unitarias

Utilizo Mockito para probar los diferentes endpoints en las pruebas unitarias.



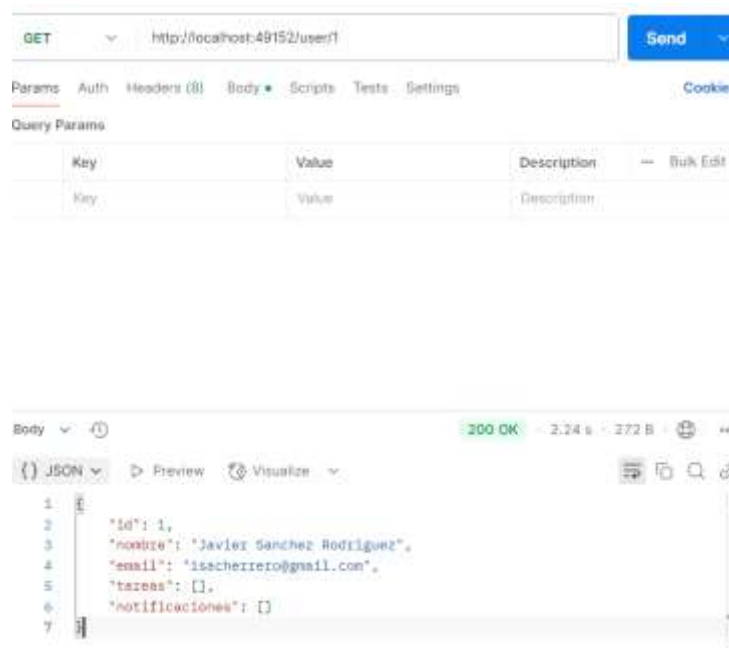
En esta imagen se comprueba que el archivo de las pruebas unitarias no da errores y es un éxito.

Para comprobarlo he probado cada método de la API de los microservicios individualmente para garantizar que las operaciones CRUD y la lógica de negocio funcionan correctamente.

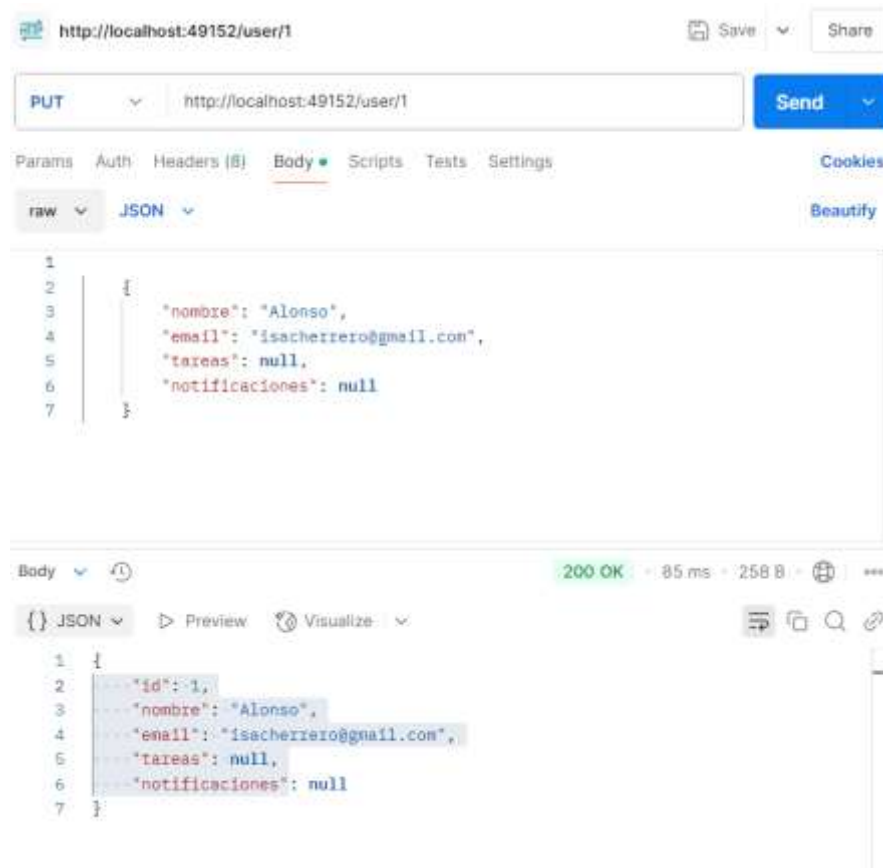
He usado Postman y Swagger para ejecutar las pruebas y verificar las respuestas.

Code	Details
200	<div>Response body</div> <pre>[   {     "id": 1,     "nombre": "Javier Sanchez Rodriguez",     "email": "isacherrero@gmail.com",     "tareas": null,     "notificaciones": null   } ]</pre>

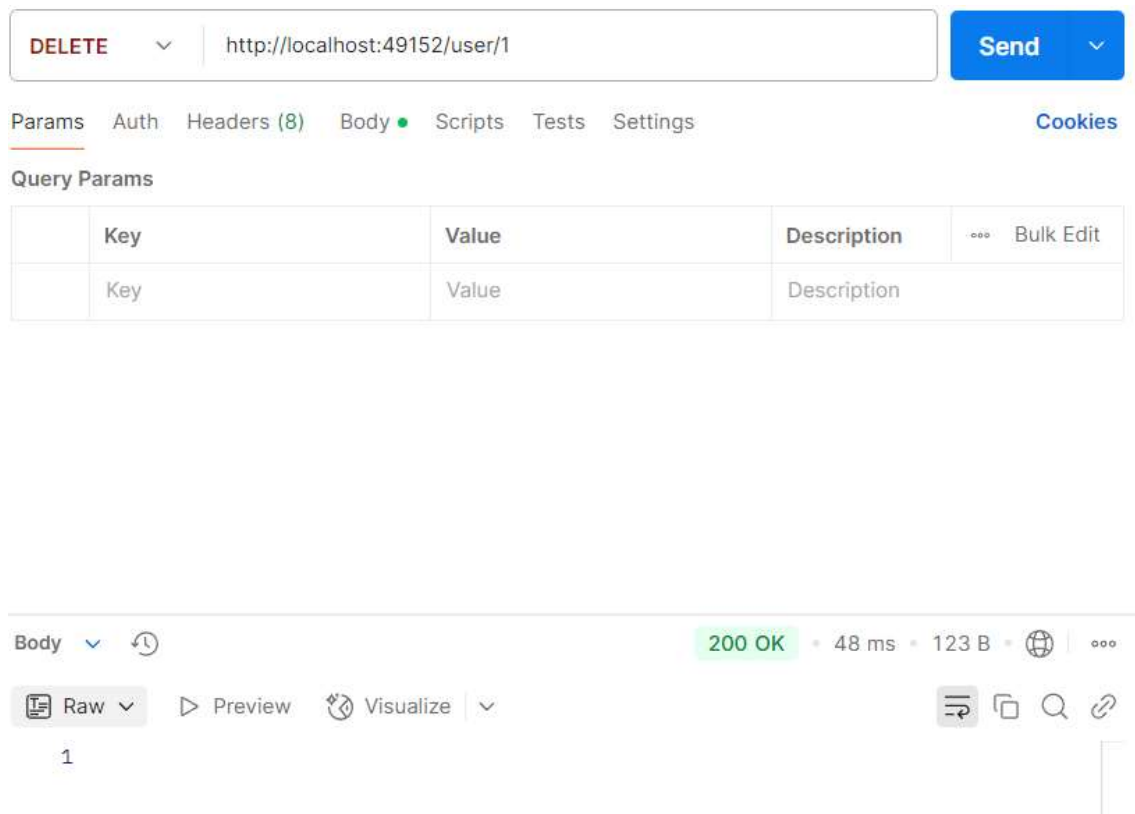
En primer lugar, ejecuto el método POST de la API del microservicio usuario. Después de esto pruebo el método GET donde se muestra la información del usuario guardado, y verificando así, el buen funcionamiento de ambos métodos como se muestra en la imagen.



Después compruebo el método GET `user/{id}`, y como vemos en la imagen, retorna el valor esperado.



Con este método PUT, mi objetivo es cambiar el nombre del usuario con id 1, y como comprobamos en la imagen, el valor retornado es el esperado.



Ahora compruebo el método DELETE. Al eliminar el usuario con Id 1 vemos que el retorno es el esperado ya que nos retorna un código 200.

Estas pruebas, se han llevado a cabo con todos los microservicios de la misma forma que éste.

También quiero añadir que, en este caso, el microservicio User utiliza puerto 49152, garantizando así, que accede al archivo de configuración config-client-prod de Config Server que usa este puerto.



Por último, compruebo el perfil que se está utilizando en ese momento con este método llamado check. Con esta imagen se compruebo que la respuesta retornada es la indicada.

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
APIGATEWAY	n/a (1)	(1)	UP (1) - <a href="#">DESKTOP-V5QHF1D.apigateway.R061</a>
BUSINESSDOMAIN-NOTIFICACION	n/a (1)	(1)	UP (1) - <a href="#">DESKTOP-V5QHF1D.businessdomain-notificacion.R061</a>
BUSINESSDOMAIN-TAREA	n/a (1)	(1)	UP (1) - <a href="#">DESKTOP-V5QHF1D.businessdomain-tarea.R061</a>
CONFIG-SERVER	n/a (1)	(1)	UP (1) - <a href="#">DESKTOP-V5QHF1D.config-server.R061</a>
KEYCLOCK	n/a (1)	(1)	UP (1) - <a href="#">DESKTOP-V5QHF1D.keyclock.R061</a>
SPRINGBOOTADMIN	n/a (1)	(1)	UP (1) - <a href="#">DESKTOP-V5QHF1D.springbootadmin.R061</a>
USER	n/a (1)	(1)	UP (1) - <a href="#">DESKTOP-V5QHF1D.user.R061</a>

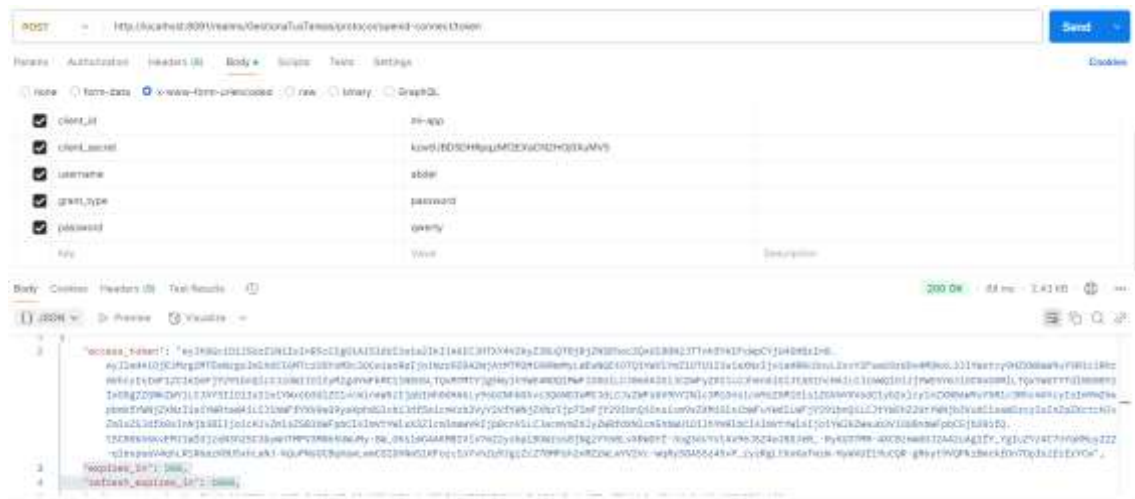
Gracias a esta imagen, comprobamos también que todos los microservicios se registran correctamente en el servidor de Eureka.



Aquí también comprobamos que se registran en el de Spring Boot Admin correctamente.

```
INFO 17412 --- [apigateway] [ctor-http-nio-3] o.g.setups.globalpostfiltering : Global Post Filter ejecutado
INFO 17412 --- [apigateway] [ctor-http-nio-3] r.n.http.client.HttpClientOperations : [03d2c3e2-1, 1:/192.168.5.2:56708]
INFO 17412 --- [apigateway] [ctor-http-nio-3] o.a.c.g.h.RoutePredicateHandlerMapping : Route matched: user_service
INFO 17412 --- [apigateway] [ctor-http-nio-3] o.a.c.g.h.RoutePredicateHandlerMapping : Mapping [exchange: PORT, http://192.168.5.2:56708] Mapped to org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter
INFO 17412 --- [apigateway] [ctor-http-nio-3] o.a.c.g.handler.FilteringWebHandler : Sorted gatewayFilterFactories: []
INFO 17412 --- [apigateway] [ctor-http-nio-3] o.g.setups.globalprefiltering : Global prefilter ejecutado
INFO 17412 --- [apigateway] [ctor-http-nio-3] f.f.h.o.ObservedHttpRequestHttpHeadersFilter : Will instrument the HTTP request
INFO 17412 --- [apigateway] [ctor-http-nio-3] f.f.h.o.ObservedHttpRequestHttpHeadersFilter : Client observation [name=http.request, id=03d2c3e2-1, 1:/192.168.5.2:56708]
INFO 17412 --- [apigateway] [ctor-http-nio-3] r.n.http.client.HttpClientOperations : [03d2c3e2-2, 1:/192.168.5.2:56708]
INFO 17412 --- [apigateway] [ctor-http-nio-3] f.f.h.o.ObservedResponseHttpHeadersFilter : Will instrument the response
INFO 17412 --- [apigateway] [ctor-http-nio-3] f.f.h.o.ObservedResponseHttpHeadersFilter : The response was handled for observation
INFO 17412 --- [apigateway] [ctor-http-nio-3] o.g.setups.globalpostfiltering : Global Post Filter ejecutado
```

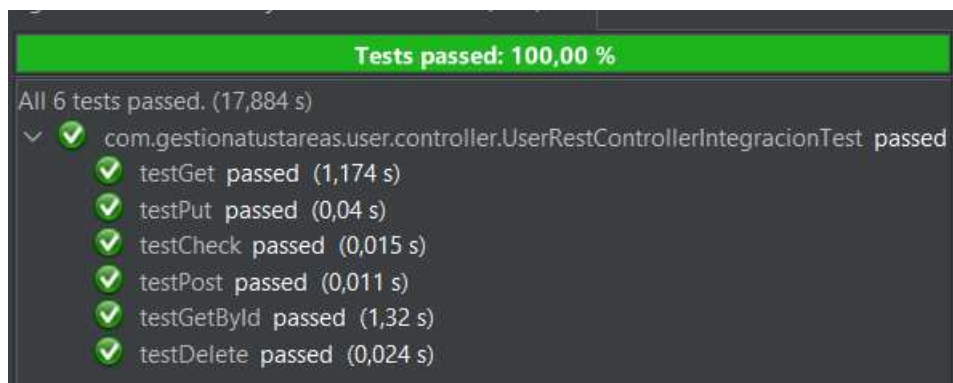
Gracias a esta imagen comprobamos que las solicitudes http pasan por el ApiGateway ya que se activa el GlobalPostFiltering y GlobalPretFiltering.



Al hacer una petición al servidor de keycloak para solicitar el token de acceso con los datos de autenticación que se ven en la imagen, lo devuelve con éxito.

## 5.2 Pruebas de integración

Se utiliza MockMvc para probar el controlador sin necesidad de levantar el servidor simulando llamadas HTTP y validando las respuestas, códigos de estado y cuerpos JSON.



Como muestra la imagen, el resultado es el esperado.

También he realizado una prueba específica desde el microservicio User, donde se valida la interacción con los microservicios Tarea y Notificación gracias al método full. Este método accede a los demás microservicios para visualizar a los usuarios con sus tareas y notificaciones correspondientes.

Code	Details
200	<p>Response body</p> <pre>{   "id": 1,   "nombre": "Javier Sanchez Rodriguez",   "email": "isacherrero@gmail.com",   "tareas": [     {       "id": 1,       "titulo": "string",       "descripcion": "string",       "estado": "PENDIENTE",       "fechaCreacion": "2025-01-14T23:06:08.306",       "fechaVencimiento": "2025-01-14T23:06:08.306",       "propietariosIds": [         1       ]     }   ],   "notificaciones": [     {       "id": 1,       "usuario_id": 1,       "tipo": "CREACION",       "mensaje": "hola soy una notificación",       "fechaCreacion": "2025-01-14T23:06:32.031",       "fechaVencimiento": "2025-01-14T23:06:32.031",       "leido": true     }   ] }</pre>

Aquí comprobamos que la conexión es correcta ya que la respuesta retornada es la esperada.

### 5.3 Errores y excepciones:

Las respuestas de los errores y el manejo de excepciones han sido comprobadas para asegurar que los mensajes sean claros y estructurados, siguiendo el estándar REC 7807.

```
{
  "type": "TECNICO",
  "title": "Input-Output-error",
  "code": "1024",
  "detail": "Method parameter 'id': Failed to convert value of type 'java.lang.String' to required type 'long'; For input string: \"0,1\"",
  "instance": null
}
```



Así, por ejemplo.

## 6 Creación, Gestión y Despliegue de Imágenes Docker con Docker Compose

## 6.1 Creación de una imagen Docker

Gracias al Dockerfile genérico que tiene cada microservicio puedo crear las imágenes.

```
C:\Users\usuario>cd C:\Users\usuario\OneDrive\Escritorio\ProyectosMicroservicios\GestionaTusTareas\infraestructuradomain
\springBootAdmin
C:\Users\usuario\OneDrive\Escritorio\ProyectosMicroservicios\GestionaTusTareas\infraestructuradomain\springBootAdmin>
```

En primer lugar, accedo a la ubicación del microservicio, que, en este caso, es SpringBootAdmin. Aquí que es donde se encuentra el archivo Dockerfile.

[illegible]

Posteriormente pasamos a crear la imagen con el comando `docker build` pasando un argumento a la variable `JAR_FILE`. Esta variable se crea como argumento en el `Dockerfile` y se hace para para que éste sea flexible reutilizable.

Como comprobamos en la imagen, se crea correctamente. Esto se hace con todas las imágenes Docker que he creado.

```
C:\Users\usuario\OneDrive\Escritorio\ProyectosMicroservicios\GestionaTareas\businessdomain\tarea>docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
tarea_image	latest	8388b433acf1	6 seconds ago	517MB
user_image	latest	e27f73a737eb	5 minutes ago	517MB
eureka-server_image	latest	116e448c9734	5 minutes ago	455MB
apigateway_image	latest	8aa02d803693	6 minutes ago	452MB
notificacion_image	latest	b7de19a25e86	6 minutes ago	517MB
springbootadmin_image	latest	45d2579cd2ce	7 minutes ago	461MB
keycloakadapter_image	latest	809e3d3c1e0e	8 minutes ago	439MB
configserver_image	latest	c064572c5dcf	11 minutes ago	454MB

Aquí se muestran todas las imágenes de los microservicios.

```
C:\Users\usuario\OneDrive\Escritorio\ProyectosMicroservicios\GestionaTusTareas\infrastructuremain\springbootAdmin>cd C:\Users\usuario\OneDrive\Escritorio\ProyectosMicroservicios\GestionaTusTareas
```

Ahora accedo a la ubicación que se muestra en la imagen ya que es donde se encuentra el archivo de Docker Compose.

```
C:\Users\usuario\OneDrive\Escritorio\ProyectosMicroservicios\GestionaTusTareas>docker compose up -d
[+] Running 12/12
 ✓ Network gestionatustareas_default Created
 ✓ Container postgres Started
 ✓ Container id-eureka Started
 ✓ Container id-keycloakServer Started
 ✓ Container pgadmin Started
 ✓ Container bd-tarea Started
 ✓ Container bd-user Started
 ✓ Container id-notificación Started
 ✓ Container id-configserver Started
 ✓ Container id-springbootadmin Started
 ✓ Container id-apigateway Started
 ✓ Container id-keycloakadapter Started
```

Por último, se levanta en segundo plano los servicios en el archivo de docker-compose, que acceden a sus respectivas imágenes, gracias al comando `docker compose up -d`.

## 7 Angular

Para el frontend del backend he decidido usado Angular y he trabajado con Standalone Components. Esto elimina la necesidad de declarar los componentes dentro de módulos. Cada componente es independiente, lo que hace más fácil la reutilización de código.

En este caso implemento un sistema de autenticación donde los usuarios pueden iniciar sesión utilizando su nombre y contraseña, que se verifican por Keycloak desde el back. Para la seguridad de las rutas, se emplea un AuthGuard, permitiendo el acceso solo a usuarios autenticados.

Además, he desarrollado un chatbot interactivo que permite gestionar usuarios. Dependiendo del mensaje enviado, el chatbot puede devolver una lista de usuarios dentro de un kodal, crear o editar usuarios, facilitando la administración sin necesidad de navegar manualmente por la interfaz.

Junto a este chatbot se encuentra otro chat que se conecta a un modelo pre entrenado de Hugging Face. Este chatbot accede a una API externa de Hugging Face, usando un token personal con el objetivo de interactuar con una IA. Pasaré a explicar el segundo con la siguiente imagen.

Mi objetivo con este front es demostrar mis habilidades utilizando Angular para interactuar con una API, en este caso, la de gestión de usuarios. La aplicación está diseñada para mostrar cómo puedo consumir y utilizar los datos de una API utilizando Angular.

Quiero añadir que en mi Git Hub podrás encontrar proyectos donde demuestro mis habilidades con frameworks como Laravel y cakePHP, en lenguajes como Javascript y Php. También en HTML, CSS y Bootstrap.

## 7.1 Descripción de las Vistas y su Funcionalidad en Angular



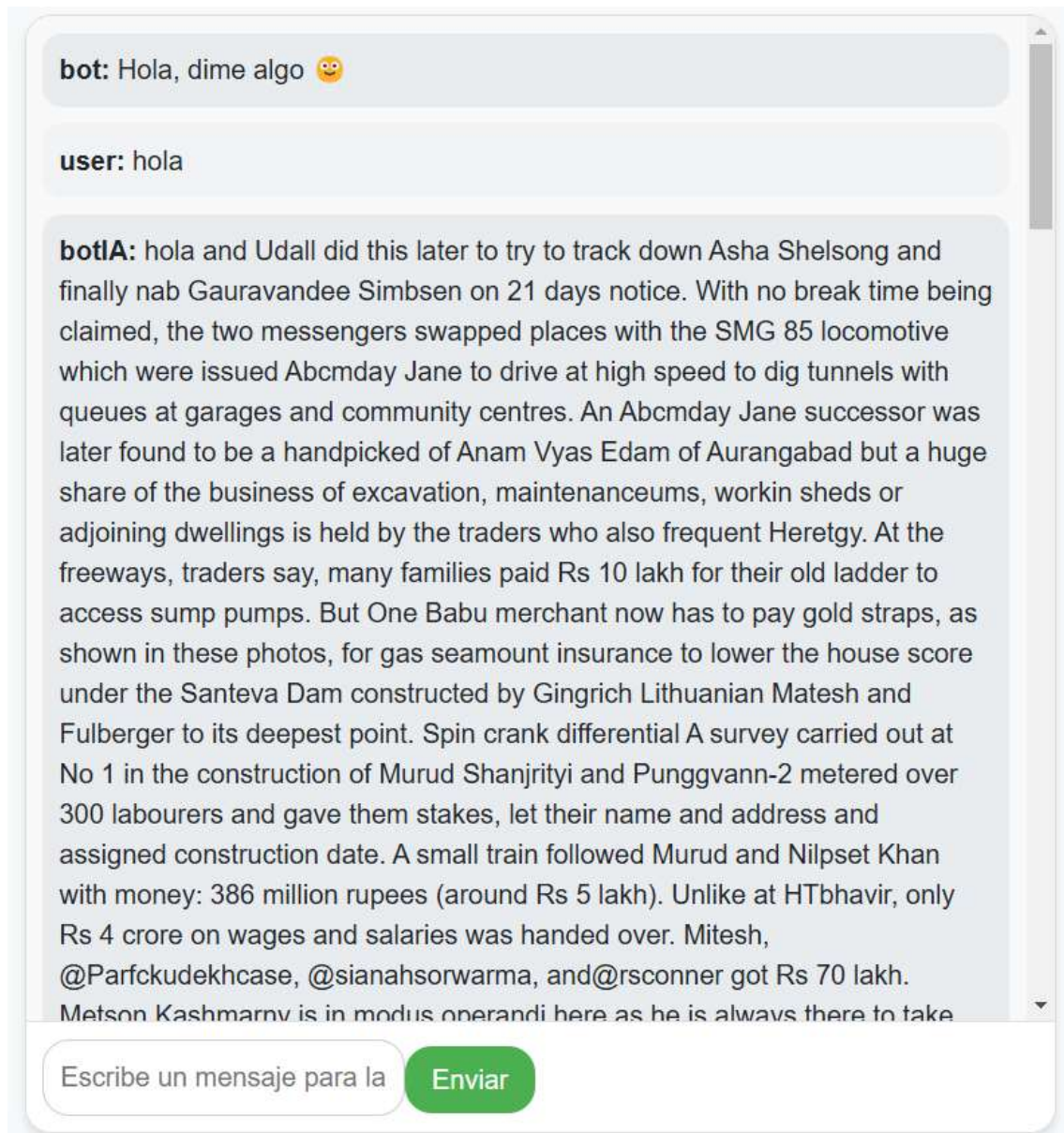
Esta sería la vista del Login, donde el usuario puede autenticarse con usuario y contraseña.



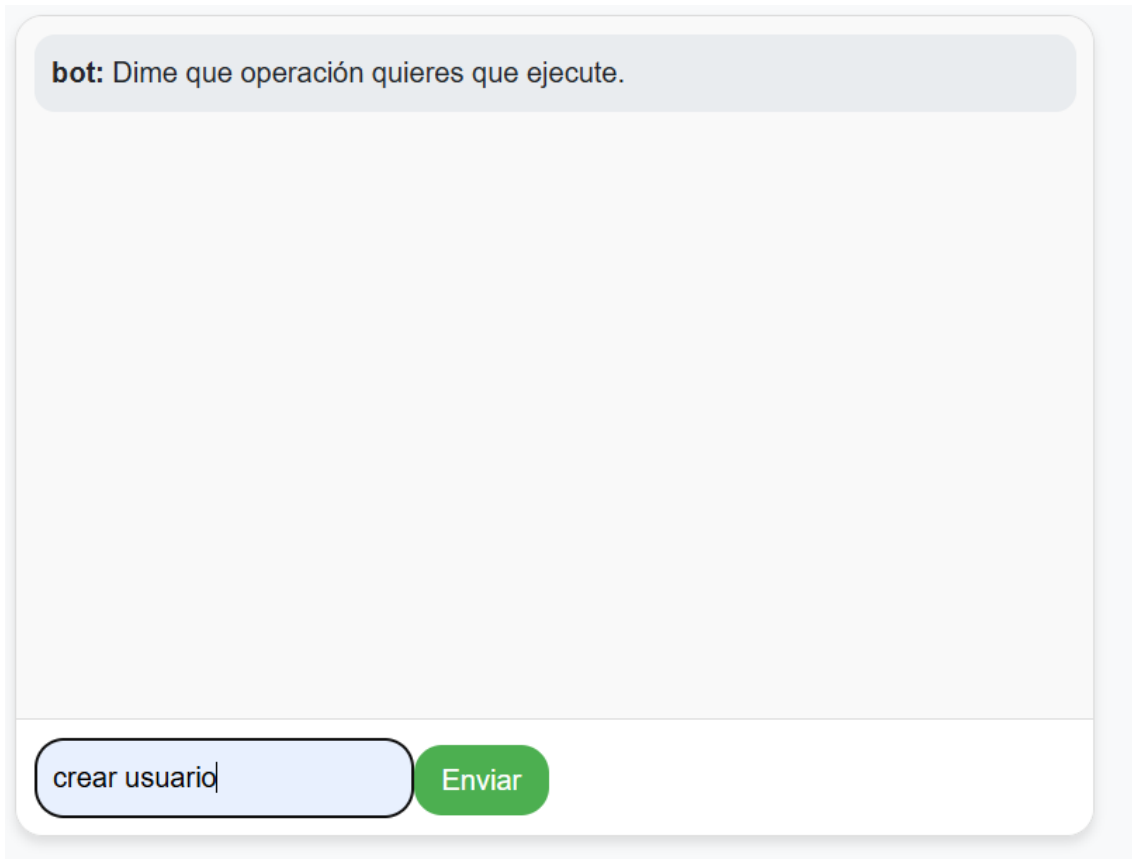
Cuando se hace click en el botón "Ingresar" aparece un indicador de carga circular de Angular Material.



Al ser autorizado, accede a esta vista principal del proyecto. Son dos Chatbots con diferentes funcionalidades como ya comenté. Uno se encarga de realizar operaciones CRUD, y el otro interactuar con una IA. Pasaré a explicar el segundo con la siguiente imagen:



Cuando se le envía un mensaje “hola” por ejemplo, el Chatbot envía esta entrada al modelo de Hugging Face, el cual, devuelve una respuesta aleatoria. Esto es porque estoy utilizando un modelo gratuito y no es eficiente.



bot: Dime que operación quieres que ejecute.

crear usuario

Enviar

En el primer Chatbot, al enviar un mensaje donde ponga “crear usuario” se abre un modal donde nos muestra un formulario para registrar un usuario.



**Crear Usuario** ×

**Nombre:**

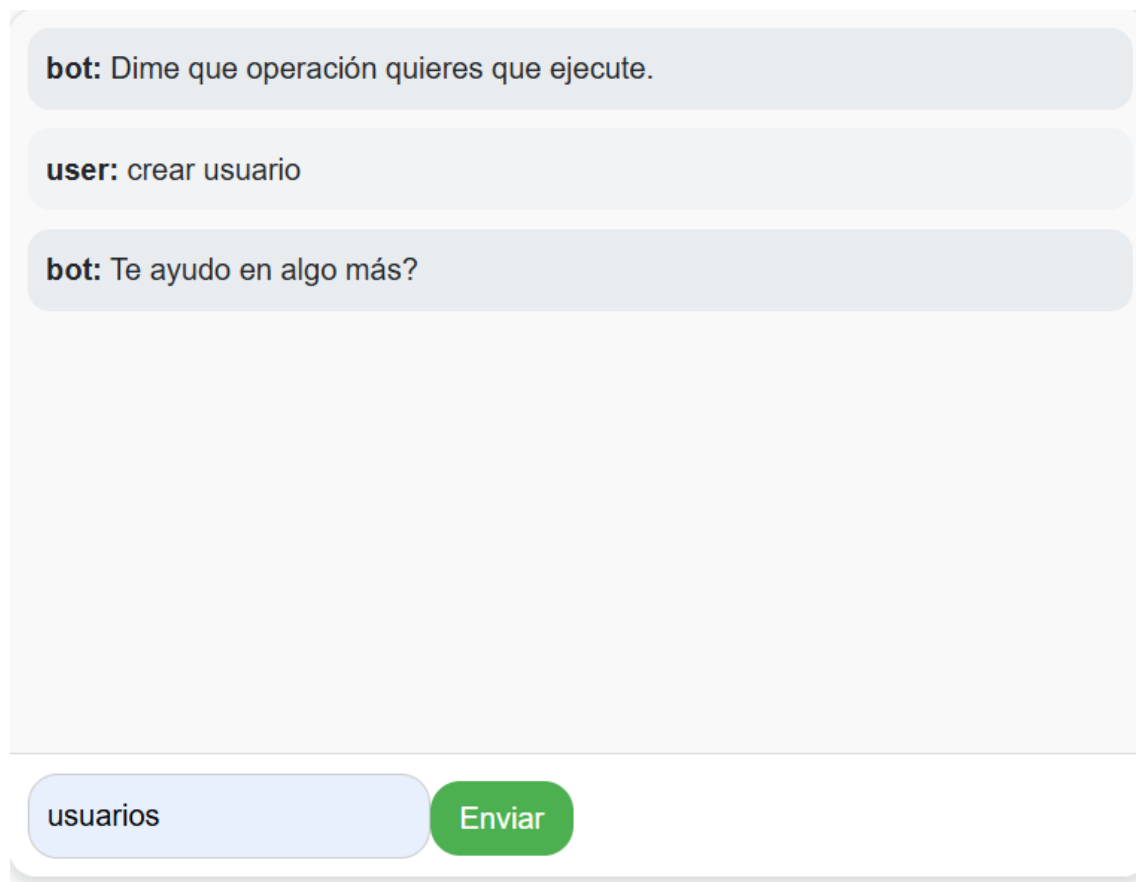
Abdel

**Correo:**

abdel.oub9@gmail.com

Crear Usuario

Registramos el usuario con los siguientes datos y al darle a la X el modal se cierra.



The screenshot shows a chat window with a light gray background. It contains three messages in rounded rectangular bubbles. The first bubble is light blue and contains the text "bot: Dime que operación quieres que ejecute.". The second bubble is light gray and contains the text "user: crear usuario". The third bubble is light blue and contains the text "bot: Te ayudo en algo más?". At the bottom of the chat window, there is a light blue input field containing the text "usuarios" and a green button labeled "Enviar".

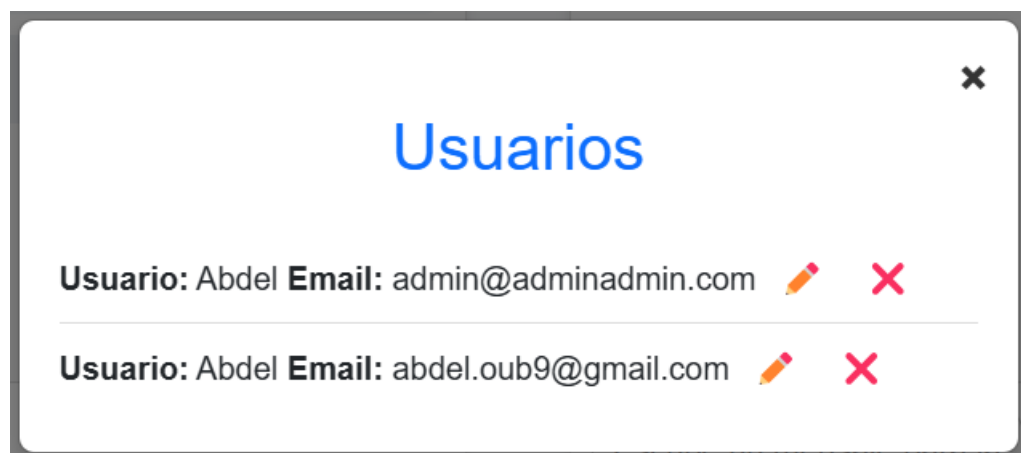
bot: Dime que operación quieres que ejecute.

user: crear usuario

bot: Te ayudo en algo más?

usuarios Enviar

El Bot muestra un mensaje que dice “Te ayudo en algo más?”. Para visualizar todos los usuarios, se le debe enviar un mensaje donde ponga “usuarios”.



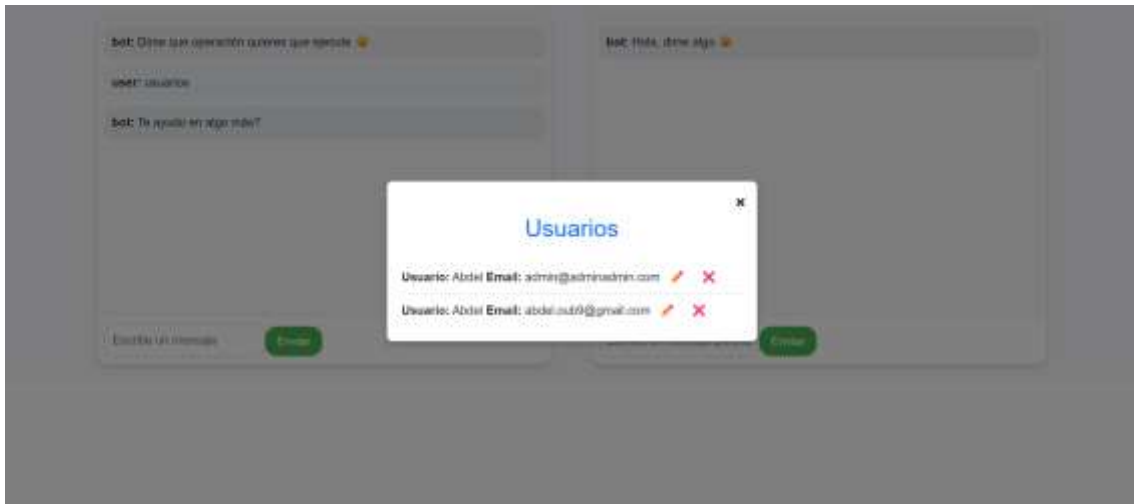
The screenshot shows a modal window with a white background and a gray border. It has a close button (X) in the top right corner. The title "Usuarios" is centered at the top in blue. Below the title, there is a list of users. Each user entry consists of the text "Usuario: Abdel" followed by "Email: " and an email address. To the right of each email address are two icons: a pencil icon and a red X icon. The first user entry is "Usuario: Abdel Email: admin@adminadmin.com" and the second is "Usuario: Abdel Email: abdel.oub9@gmail.com".

Usuarios

Usuario: Abdel Email: admin@adminadmin.com ✎ ✖

Usuario: Abdel Email: abdel.oub9@gmail.com ✎ ✖

Y muestra los usuarios existentes. Como podemos ver, el usuario antes creado se muestra correctamente.



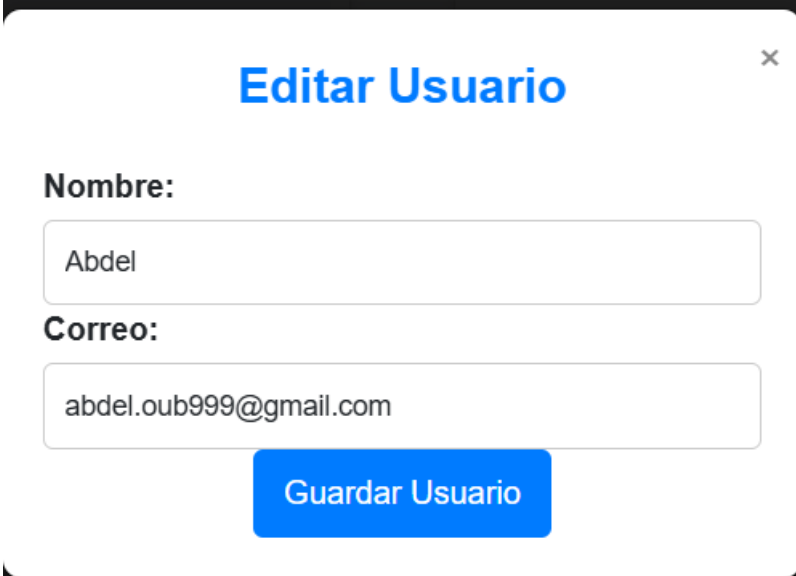
Aprovecho para mostrar el tamaño de los modales que se abren en la página.

Al hacer click en el lápiz se abrirá otro modal donde se podrán editar los datos de dicho usuario.

A screenshot of a modal window titled "Editar Usuario". It contains two input fields: "Nombre:" with the value "Abdel" and "Correo:" with the value "abdel.oub9@gmail.com". Below the fields is a blue button labeled "Guardar Usuario".

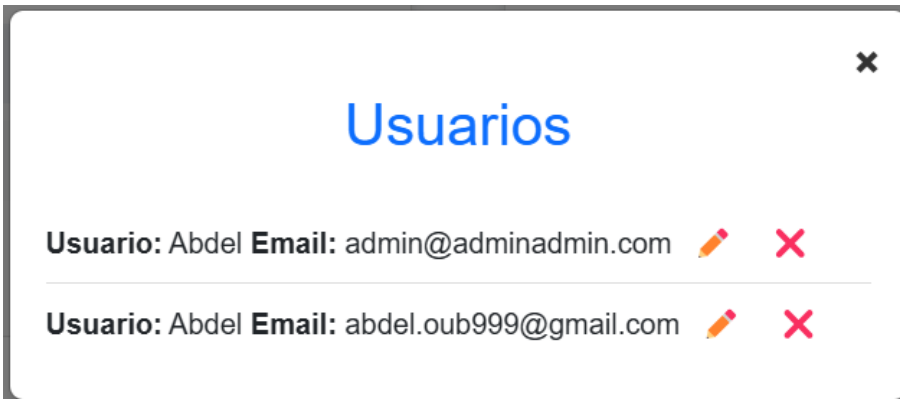
Verificamos que los datos del usuario aparecen correctamente en el formulario.





A modal window titled "Editar Usuario" with a close button (X) in the top right corner. It contains two input fields: "Nombre:" with the value "Abdel" and "Correo:" with the value "abdel.oub999@gmail.com". Below the fields is a blue button labeled "Guardar Usuario".

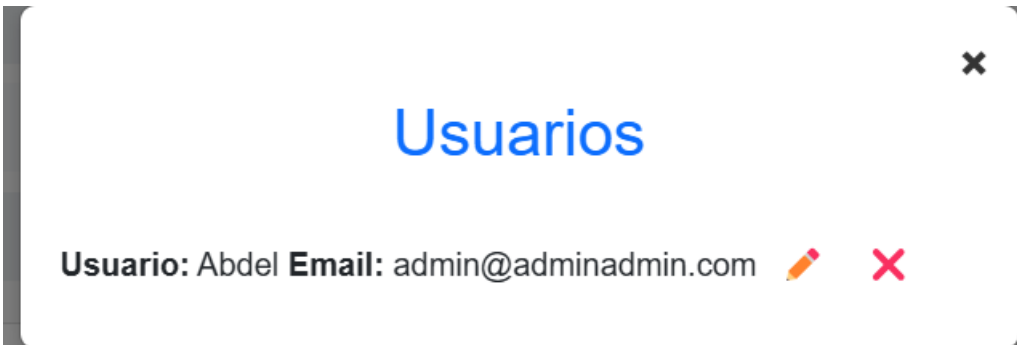
En este caso modifiko el email para comprobar que el formulario funciona correctamente.



A modal window titled "Usuarios" with a close button (X) in the top right corner. It displays a list of two users. Each user entry shows the username "Usuario: Abdel" and the email "Email: admin@adminadmin.com" and "Email: abdel.oub999@gmail.com" respectively. To the right of each email is a pencil icon and a red X icon.

Al cerrar el modal después de editar el usuario se refresca la lista de usuarios. Como podemos observar el usuario ha sido modificado correctamente.

Para eliminar el usuario se pulsa la X



A modal window titled "Usuarios" with a close button (X) in the top right corner. It displays a list of one user. The user entry shows the username "Usuario: Abdel" and the email "Email: admin@adminadmin.com". To the right of the email is a pencil icon and a red X icon.

Y se verifica que el usuario es eliminado con éxito.

## 8 Errores y Soluciones

Desarrollando este proyecto me he tenido muchos errores que impedían el buen funcionamiento de algunas funcionalidades. Gracias a los registros de logs e investigación, he podido identificar y solucionar estos problemas. Son los siguientes:

- Error en la configuración de @Column en el microservicio de Notificaciones: Declaré un booleano por defecto en español pensando que era una simple descripción, lo que causó que H2 generara incorrectamente la tabla de notificaciones. Eliminando esta configuración se resolvió el problema.
- Problema con Keycloak y la petición del token: Keycloak rechazaba la solicitud del token hasta que he desactivado la confirmación del usuario en la vista del servidor de Keycloak.
- Conexión incorrecta entre Keycloak Adapter y Eureka dentro de Docker Compose: Estaba refiriéndome a Keycloak con el puerto externo en vez del puerto interno dentro del Docker Compose, lo que daba errores de conexión entre los microservicios.
- Error en Keycloak Adapter por la clave secreta incorrecta: Copié mal el keycloak.client-secret, lo que daba fallos en la autenticación. Arreglar este problema ha requerido un poco de tiempo, ya que no aparecía claramente en los logs.
- Problema con la anotación @LoadBalanced en WebClient: Al llamar a los métodos full o getById, que se comunican con los microservicios de tareas y notificaciones di por hecho que ya había configurado @LoadBalanced en WebClient.Builder, lo que generaba errores de conexión utilizando Eureka.  
\_\_\_\_\_1
- Error en API Gateway al acceder a los endpoints de cada microservicio: se corrigió al quitar esta línea - Method=GET.POST.PUT.DELETE del archivo de propiedades.
- Problema con CORS al conectar el front con el back: Intenté varias soluciones sin éxito hasta que encontré la configuración correcta para permitir el acceso entre ambos en el archivo de propiedades.
- Error en el request para el login en Keycloak: La estructura de la solicitud POST /login no coincidía con la que esperaba el método en Keycloak/login se solucionó cambiando @RequestParam por @RequestBody.

- Error 404 en llamadas directas a Keycloak: Intentaba acceder a `http://KEYCLOAK/roles` en vez de <http://KEYCLOAK/keycloak/roles> y daba un error 404 ya que no lo encontraba.
- Falta de Authorization en llamadas autenticadas: En algunos métodos en angular faltaba agregar `HttpHeaders().set('Authorization', Bearer ${token})` para que ApiGateway los acepte cuando pasen por el filtro de autenticación de keycloak.. También he faltaba mapear correctamente las respuestas para obtener solo lo necesario.

## 9 Conclusión

Este proyecto demuestra la implementación de una arquitectura moderna de microservicios utilizando Spring Boot, Docker, Keycloak y PostgreSQL, etc. Cada microservicio está diseñado para cumplir una función específica asegurando modularidad y mantenimiento eficiente. Además, se añaden pruebas unitarias, integración con herramientas de monitoreo y contenedores para asegurar un entorno eficiente.

El uso de Angular para el front, aunque sea sencillo, refleja mi compromiso con el aprendizaje continuo de frameworks modernos y la capacidad de integrarlos con sistemas backend.

Este proyecto refuerza mis habilidades técnicas y mi capacidad para desarrollar soluciones completas y bien documentadas. Están listas para su implementación en entornos reales.