Backend de una aplicación de Lista de Tareas con Autenticación

Introducción:
Este proyecto consiste en el desarrollo de una aplicación de lista de tareas con funcionalidades de autenticación de usuarios. La aplicación permitirá a los usuarios registrarse, iniciar sesión, agregar tareas, marcarlas como completadas eliminarlas y podrán generar un reporte con el nombre de la tarea, la descripción, fecha de creación y el status.

Tecnologías utilizadas:
Se utiliza Java 17 como lenguaje de programación y Spring Boot 3.3.0 como framework
Se integra una base de datos MySQL para almacenar la información de usuarios y tareas
Se utiliza Spring Security para la autenticación de usuarios, junto con JSON Web Token (JWT)
Se utiliza el algoritmo de hash bcrypt para garantiza la seguridad de las credenciales de los usuarios almacenadas en la base de datos
Se integra RabbitMQ como un intermediario de mensajes para facilitar la comunicación entre los distintos componentes de la aplicación
Se utiliza Firebase Cloud Messaging para enviar notificaciones push en tiempo real a los usuarios.
Se integra Redis como una base de datos en memoria para la caché de datos.

Inicio:
Estas instrucciones te ayudarán a obtener una copia del proyecto en tu máquina local para propósitos de desarrollo y prueba.

Prerrequisitos:
-Docker Desktop
https://www.docker.com/products/docker-desktop/

-RabbitMQ
Puedes utilizar la imagen de RabbitMQ descargándola desde Docker Hub:

```
docker pull rabbitmq:3.13.1-management
```

Crear y ejecutar un contenedor RabbitMQ a partir de la imagen descargada:

```
docker run --rm -it -p 15672:15672 -p 5672:5672 rabbitmq:3.13.1-management
```

El puerto 15672 se utiliza para acceder a la interfaz de usuario de RabbitMQ, mientras que el puerto 5672 es el puerto de conexión AMQP que se utiliza para comunicarse con RabbitMQ.

-Redis
Puedes utilizar la imagen de Redis descargándola desde Docker Hub:

```
docker pull redis
```

Crear y ejecutar un contenedor de Redis a partir de la imagen descargada:

```
docker run -p 6379:6379 redis
```

Para levantar el proyecto:
-Clona este repositorio en tu máquina local.
-Para la correcta ejecución del programa los contenedores de RabbitMQ y Redis deben de estar en ejecución.
-En inteliji IDEA clic en el botón "Run" (Ejecutar) en la barra de herramientas o presionar Shift + F10.

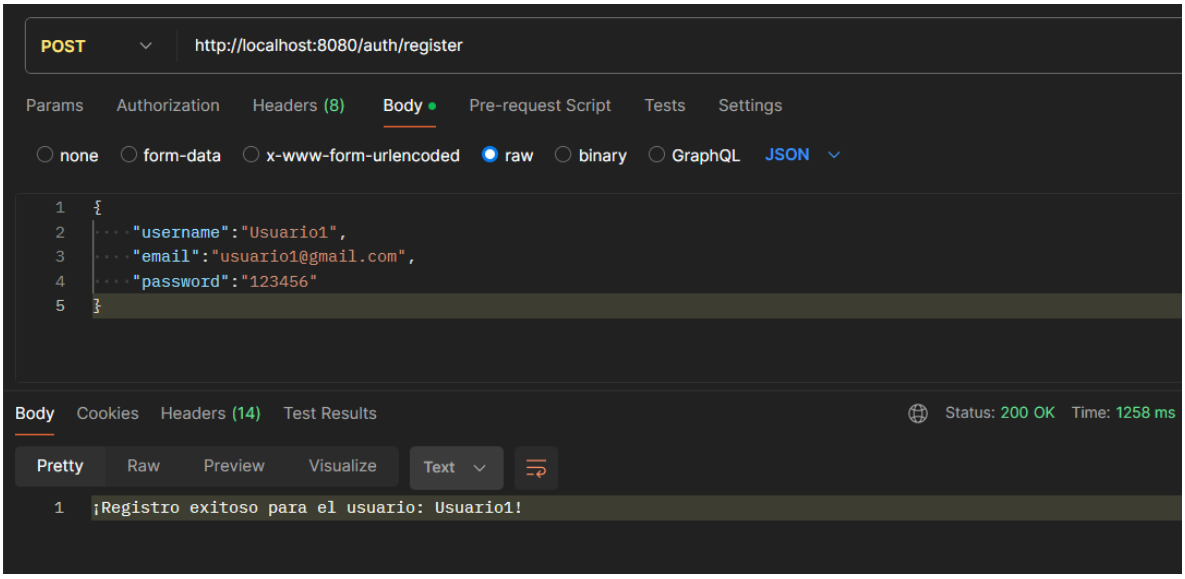
Ejecutando las pruebas:

Pruebas en POSTMAN

Registro de usuario

POST http://localhost:8080/auth/register

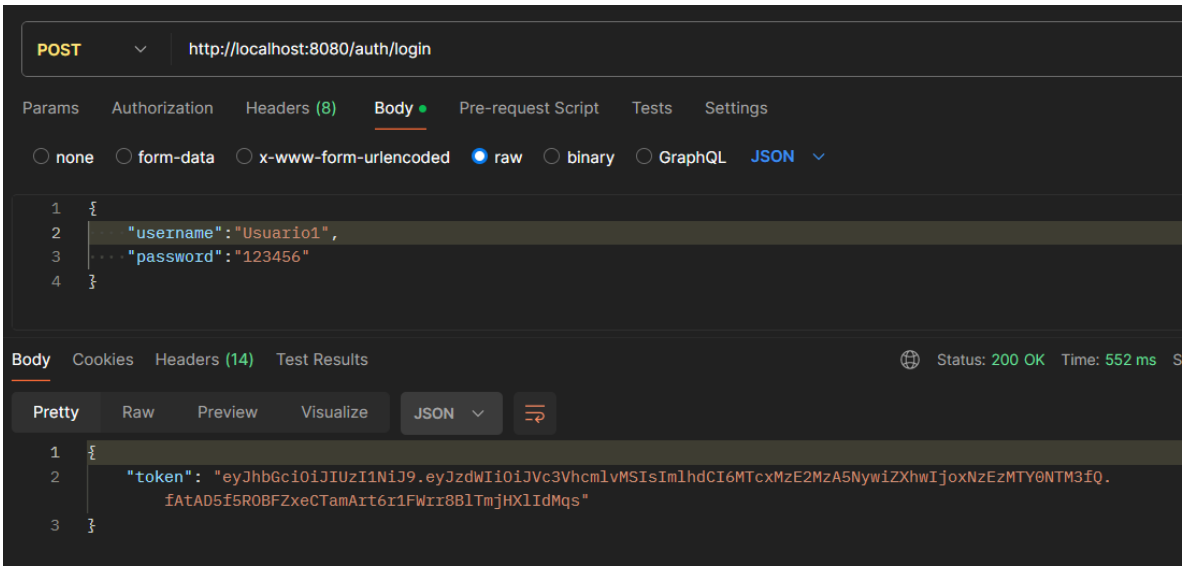
Este método verifica si el nombre de usuario ya está en uso, y si no lo está toma la información de registro de un nuevo usuario, la guarda en la base de datos, genera un token de autenticación para ese usuario y devuelve un mensaje de confirmación:



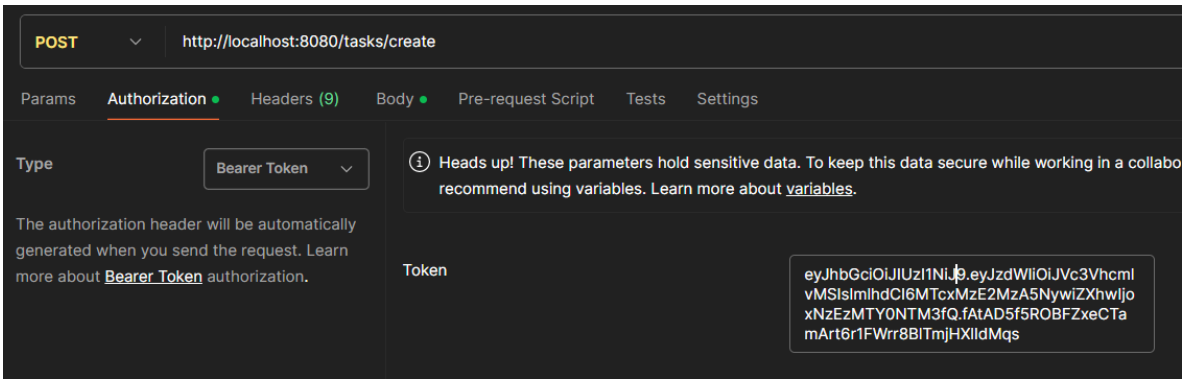
Login de usuario

POST http://localhost:8080/auth/login

Este método toma las credenciales de inicio de sesión del usuario, las utiliza para autenticar al usuario en el sistema, genera un token de autenticación para el usuario autenticado y devuelve el token como parte de la respuesta de inicio de sesión:



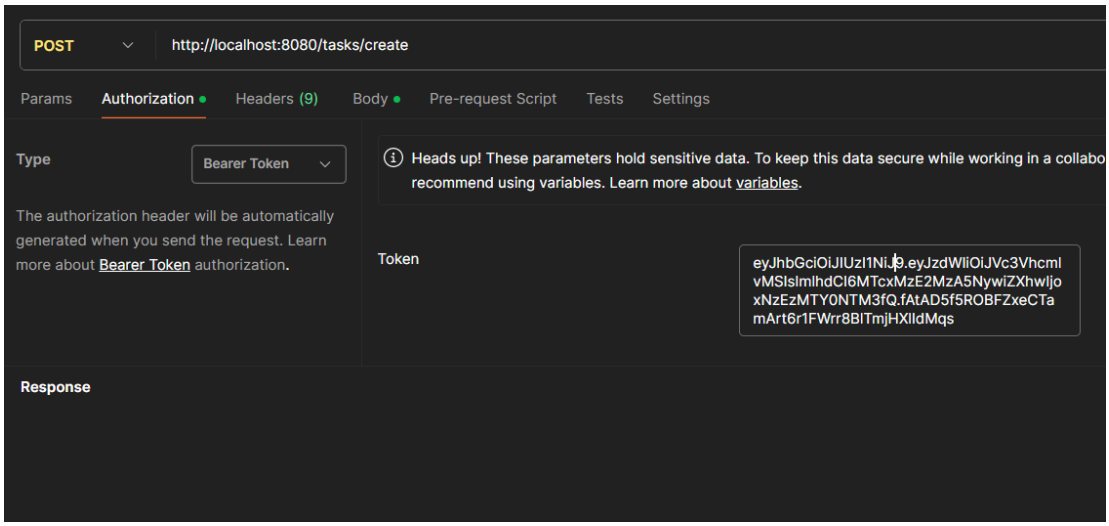
Importante: Para autenticar al usuario en rutas protegidas, se debe incluir el token generado en el inicio de sesión en la autorización de la solicitud HTTP con el tipo "Bearer Token".

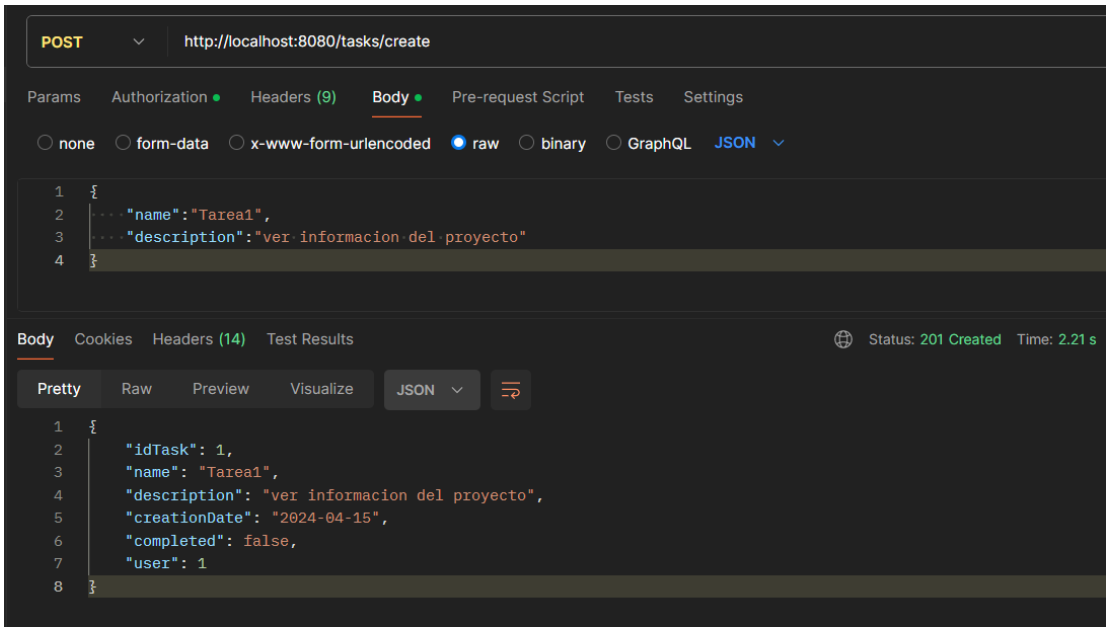


Crear una tarea

POST http://localhost:8080/tasks/create

Este método permite crear nuevas tareas asociadas a usuarios autenticados en el sistema para guardar la información en la base de datos:

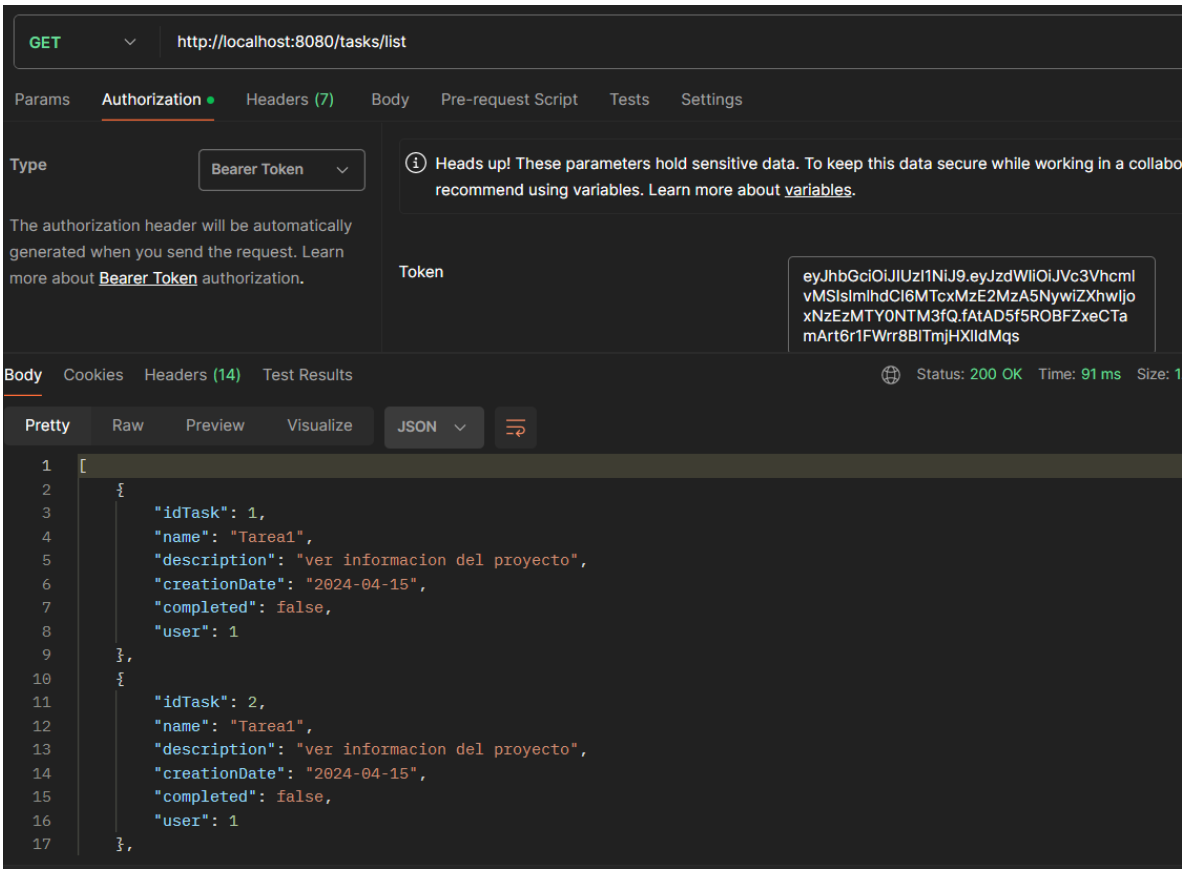




Listar las tareas del usuario

GET http://localhost:8080/tasks/list

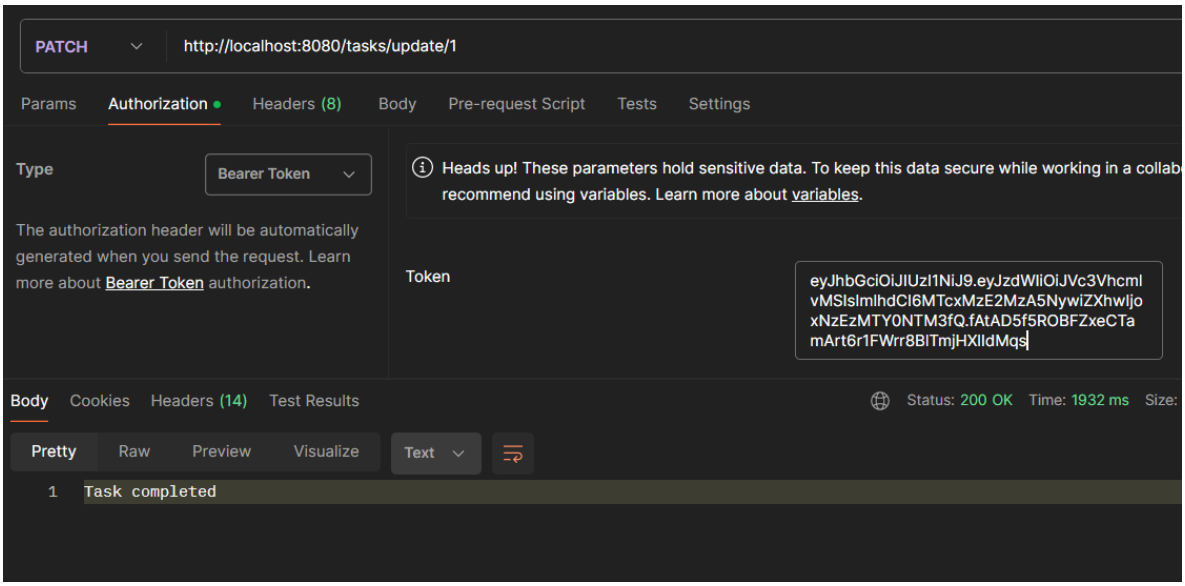
Este método recupera todas las tareas pendientes de completar asociadas a un usuario específico y las devuelve como una lista. Si el usuario no existe en la base de datos, devuelve una lista vacía:



Completar una tarea

PATCH http://localhost:8080/tasks/update/{idTask}

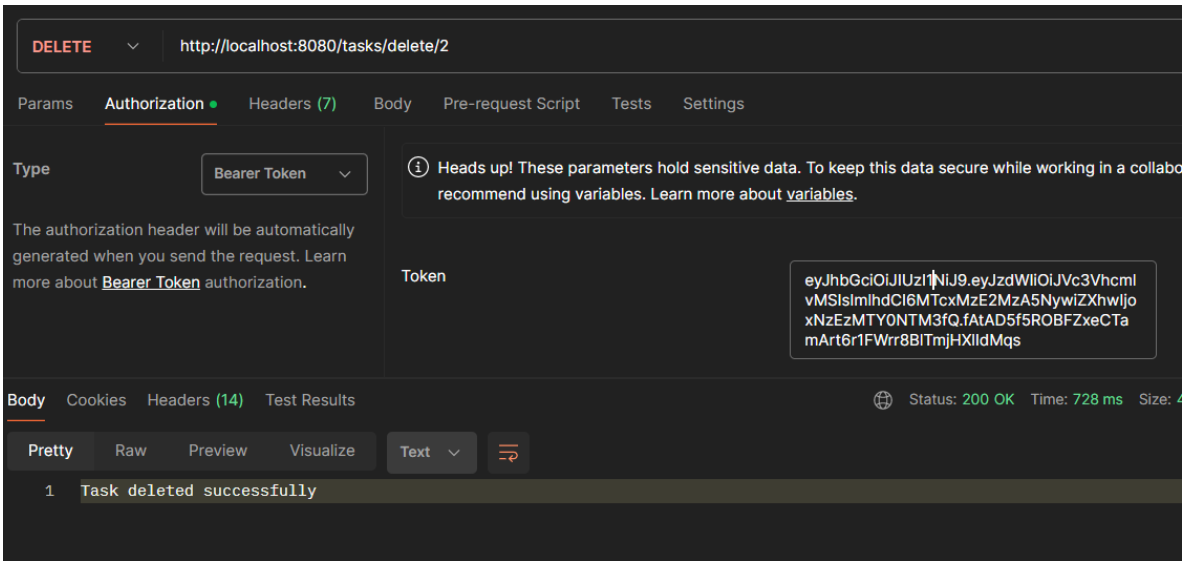
Este método actualiza el estado de una tarea especificada por su ID como completada (true) y devuelve true si la tarea fue actualizada correctamente, o false si la tarea no se encontró en la base de datos.



Eliminar una tarea

DELETE http://localhost:8080/tasks/delete/{idTask}

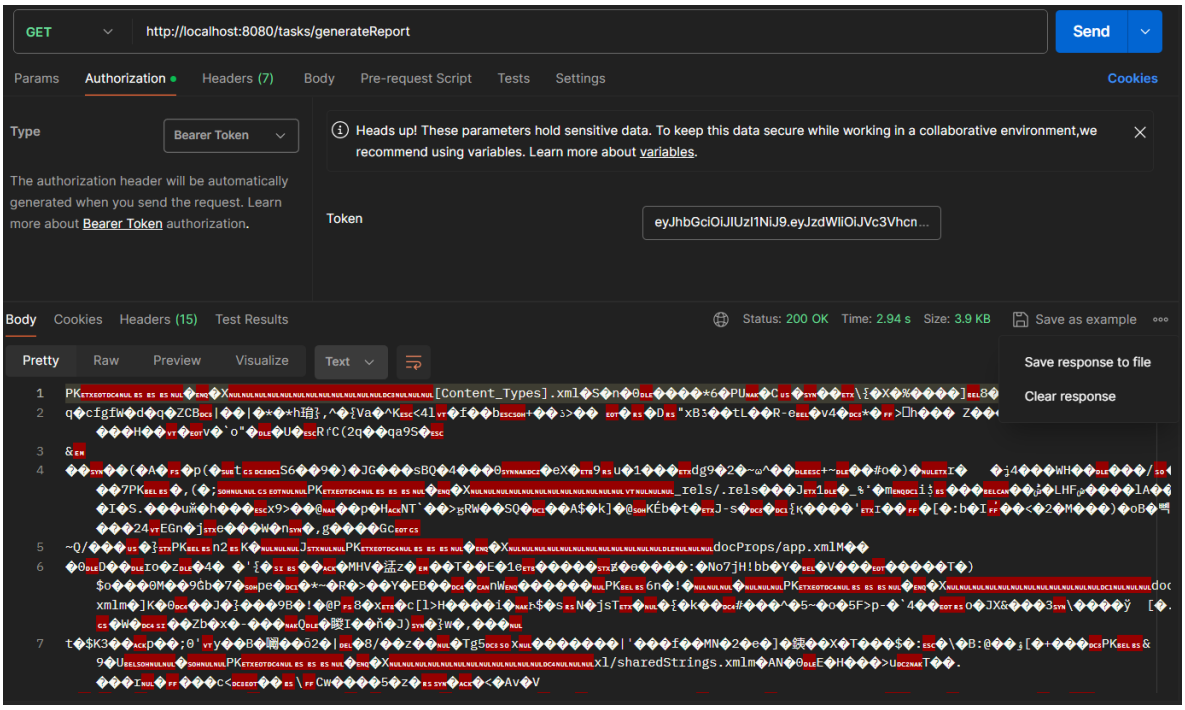
Este método elimina una tarea específica según su ID y devuelve true si la tarea fue eliminada correctamente, o false si la tarea no se encontró en la base de datos:



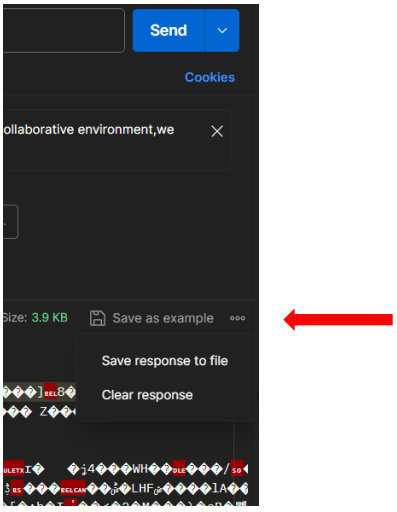
Generar un reporte en Excel

GET http://localhost:8080/tasks/generateReport

Este método consulta las tareas de un usuario, las convierte en un informe de Excel y devuelve el informe como un arreglo de bytes. Si no se encuentra el usuario en la base de datos, se lanza una excepción:



Para descargar el archivo en formato Excel, elige “Save response to file”:



Pruebas en RabbitMQ

Para monitorear y administrar las colas y el intercambio de mensajes en RabbitMQ, puedes acceder a la interfaz de administración de RabbitMQ.

Abre tu navegador web e ingresa a la siguiente dirección:
http://localhost:15672
Inicia sesión con las siguientes credenciales:
Username: guest
Password: guest

La interfaz de administración te permitirá visualizar el estado de las colas, verificar el flujo de mensajes y realizar otras tareas de administración relacionadas con RabbitMQ.

En el archivo application.properties, se definen las variables de configuración necesarias para enviar y recibir mensajes a través de RabbitMQ, utilizando la cola, la clave de enrutamiento y el intercambio especificados.

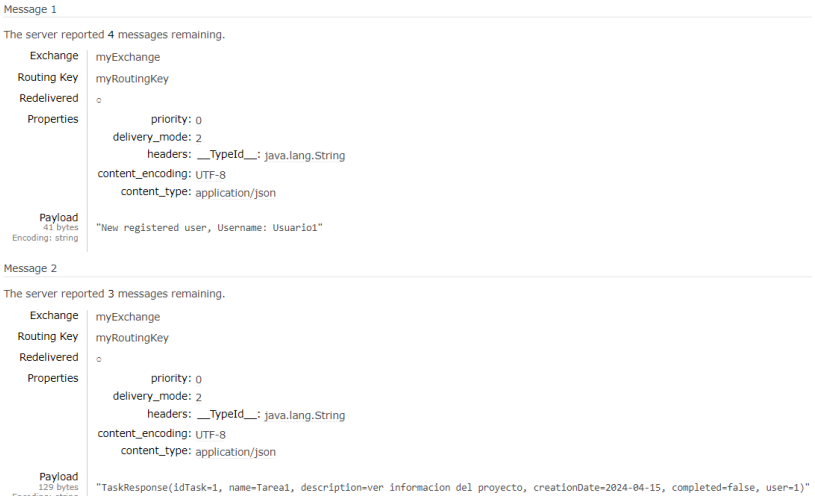
```
#RabbitMQ
rabbitmq.queue.name.task=myQueue
rabbitmq.routing.key.task=myRoutingKey
rabbitmq.exchange.name=myExchange
```

En las peticiones HTTP excepto getUserTasksController y loginController, se envía un mensaje a RabbitMQ por medio de la clase Producer:

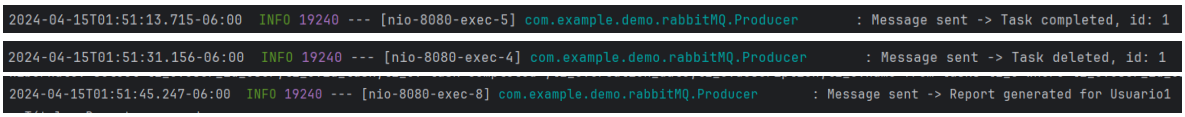
Logs:

```
2024-04-15T01:50:37.992-06:00  INFO 19240 --- [nio-8080-exec-1] com.example.demo.rabbitMQ.Producer       : Message sent -> New registered user, Username: Usuario1
2024-04-15T01:50:58.212-06:00  INFO 19240 --- [nio-8080-exec-6] com.example.demo.rabbitMQ.Producer       : Message sent -> TaskResponse(idTask=1, name=Tarea1,
 description=ver informacion del proyecto, creationDate=2024-04-15, completed=false, user=1)
```

En RabbitMQ Maganamet se visualiza de esta forma:

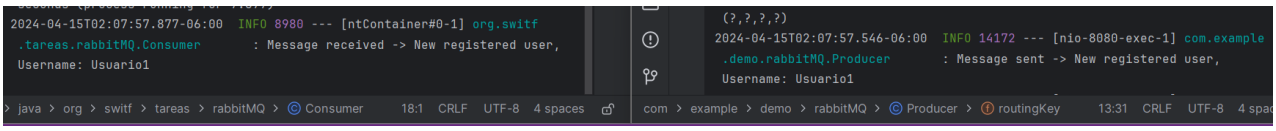


Logs:



En RabbitMQ Maganamet se visualiza de esta forma:

También se puede crear en otro programa un Consumer que este escuchando a la cola myQueue para poder enviar información de un programa a otro, con la misma configuración de la clase RabbitMQ de este programa



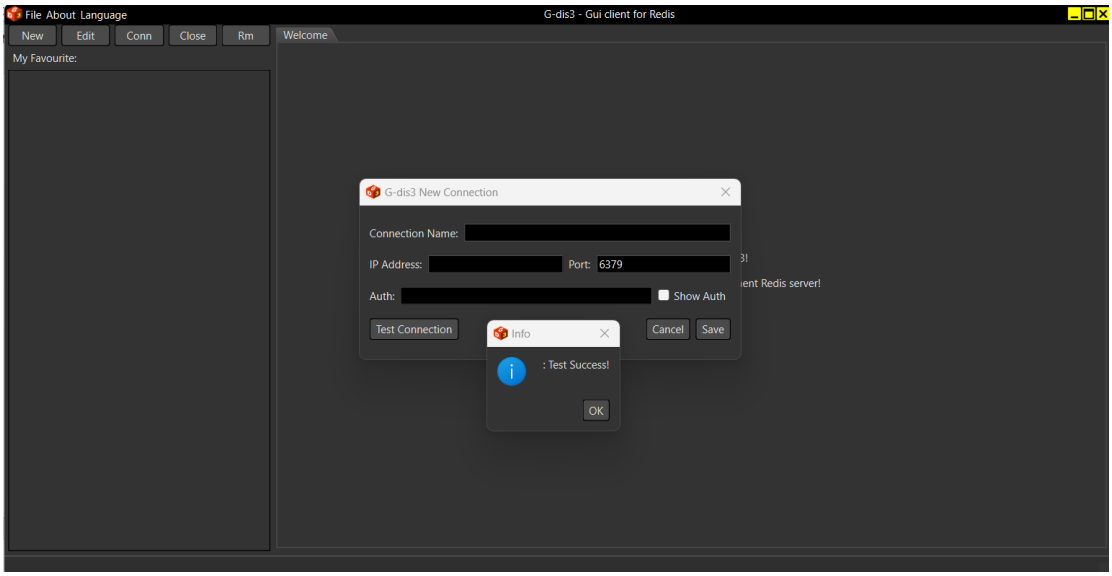
Pruebas de Redis

Para llevar a cabo las pruebas de Redis, utilicé una aplicación llamada G-dis3, la cual se puede descargar desde el siguiente enlace:
https://apps.microsoft.com/detail/9np9rdgqhcld?hl=es-ar&gl=AR

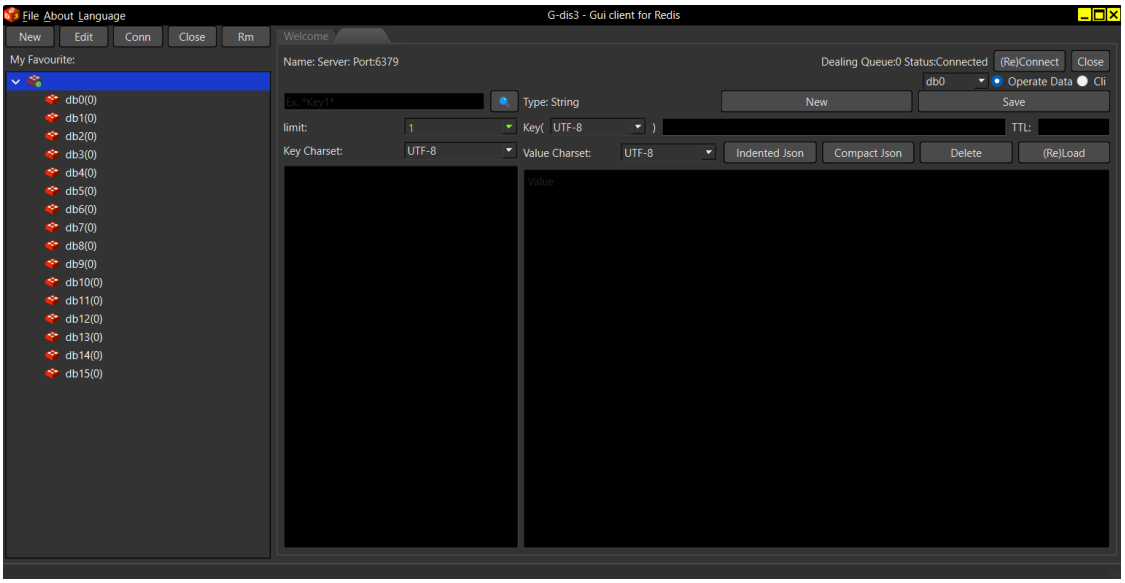
Instrucciones:

Agregar una conexión:

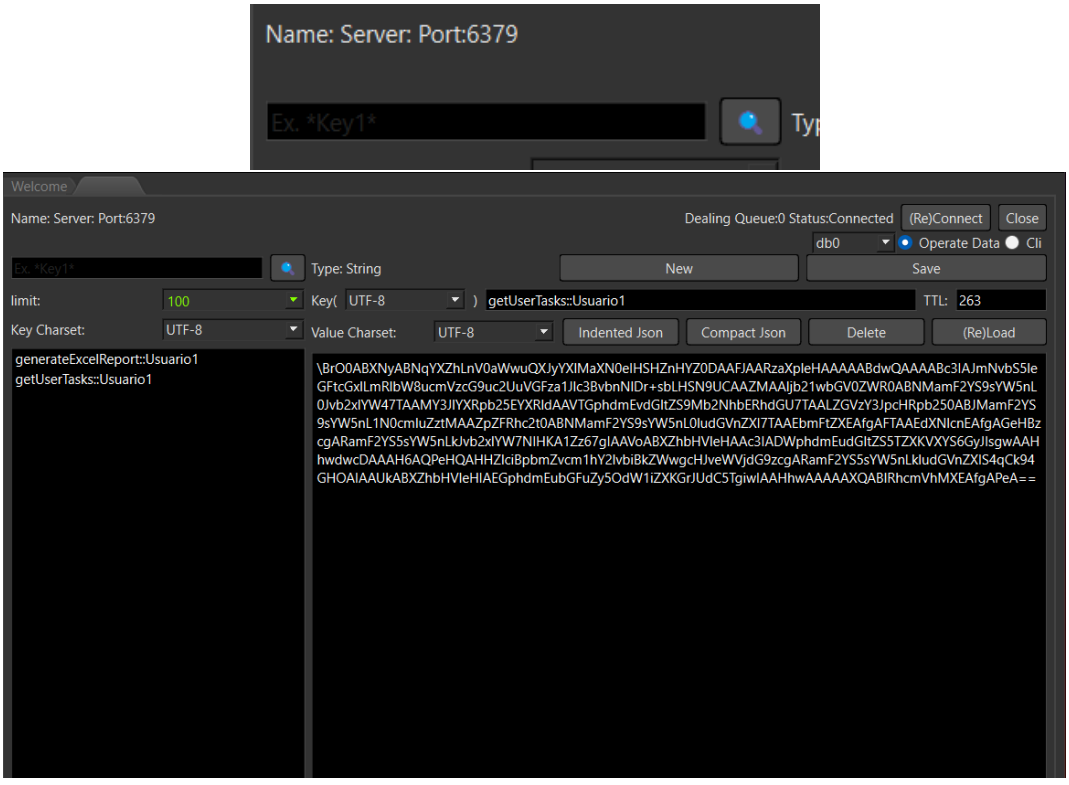
Abre la aplicación G-dis3 y haz clic en el botón 'Nuevo'.
Se abrirá una ventana donde deberás ingresar la información de conexión. En el campo "Port", coloca 6379. Luego, haz clic en "Test Connection" para verificar la conexión y finalmente en "Save" para guardar la configuración:



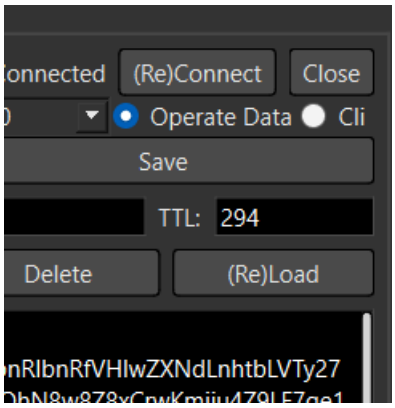
Seleccionar una conexión:
En la sección "My favourite", selecciona al menos una conexión para comenzar a utilizarla:



Verificación de métodos en la memoria de Redis:
Una vez que hayas configurado la aplicación G-dis3, utiliza la lupa en color azul del panel derecho para buscar y visualizar los métodos que están almacenados en la memoria de Redis:



En la esquina superior derecha de la interfaz de usuario, se muestra el tiempo restante que le queda a la petición actual en la memoria caché de Redis. Esto proporciona una indicación visual de cuánto tiempo falta antes de que los datos sean invalidados y se recarguen desde la fuente de datos original:



La aplicación está configurada para que los datos en la caché expiren después de 5 minutos. Esto significa que, una vez transcurrido este tiempo, los datos almacenados en Redis serán eliminados automáticamente y la próxima vez que se soliciten, serán recargados desde la fuente de datos original. Esta configuración ayuda a mantener los datos actualizados, evitando que se almacenen en caché indefinidamente y asegurando que los usuarios siempre accedan a la información más reciente:

```
@Bean
public CacheManager cacheManager(RedisConnectionFactory redisConnectionFactory){
    RedisCacheConfiguration redisCacheConfiguration = RedisCacheConfiguration.defaultCacheConfig()
            .entryTtl(Duration.ofMinutes(5));
    return RedisCacheManager.builder(redisConnectionFactory)
            .cacheDefaults(redisCacheConfiguration)
            .build();
}
```

Además, cada vez que se realiza una acción que afecta los datos, como crear una nueva tarea, marcar una tarea como completada o eliminar una tarea, se invalidará la caché de Redis:

```
@CacheEvict(cacheNames = {"getUserTasks", "generateExcelReport"}, allEntries = true)
```

Esta anotación indica a Spring que elimine la entrada relacionada en la caché de Redis después de que se complete la acción. De esta manera, se garantiza que la caché se mantenga actualizada y refleje los cambios realizados en los datos en tiempo real.

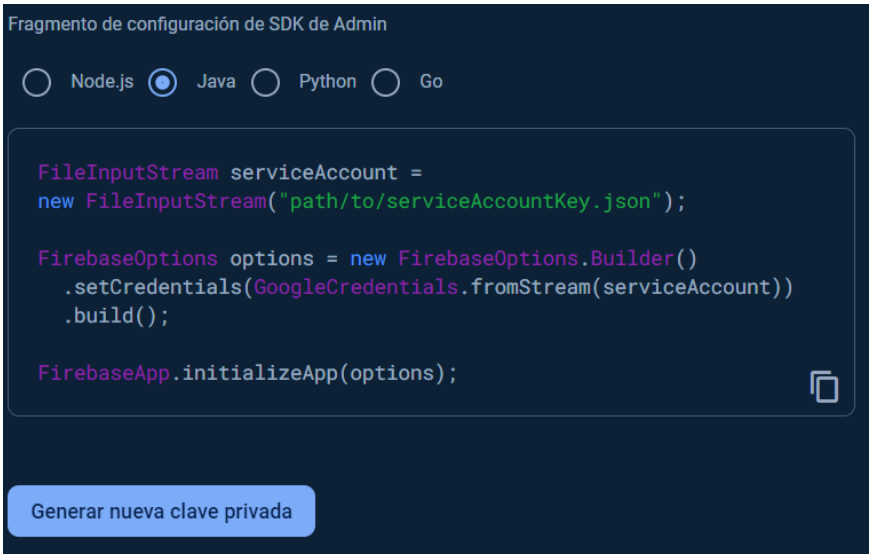
Pruebas de Firebase Cloud Messaging:

Visita la dirección:

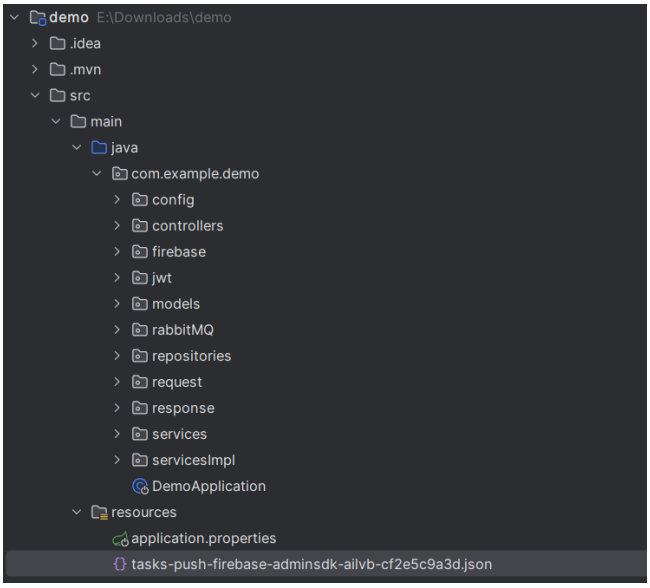
https://firebase.google.com

Dirígete a Firebase Console y crea un proyecto de Firebase Cloud Messaging llamado "tasks-push".

Genera una nueva clave privada en la sección de Configuración del proyecto en Firebase Console.



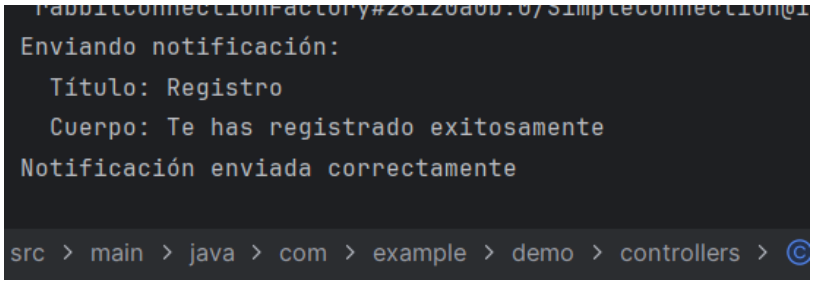
Mueve o copia el archivo JSON a la ubicación src/main/resources dentro del proyecto:



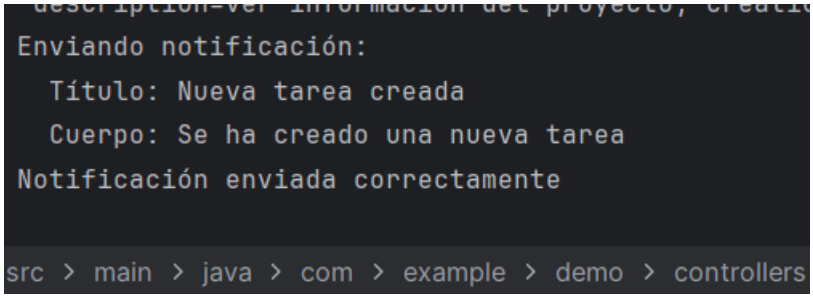
En el application.properties copia y reemplaza por el nombre de tu archivo:

```
#Firebase
gcp.firebase.service-account=classpath:tasks-push-firebase-adminsdk-ailvb-cf2e5c9a3d.json
```

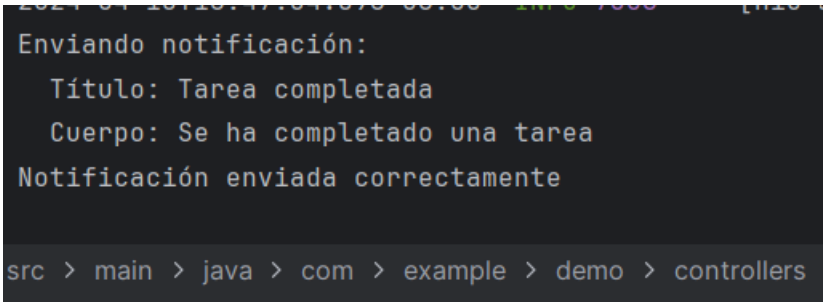
El controlador AuthController, después de que un usuario se registre exitosamente, se envía una notificación al usuario indicando que el registro ha sido exitoso:



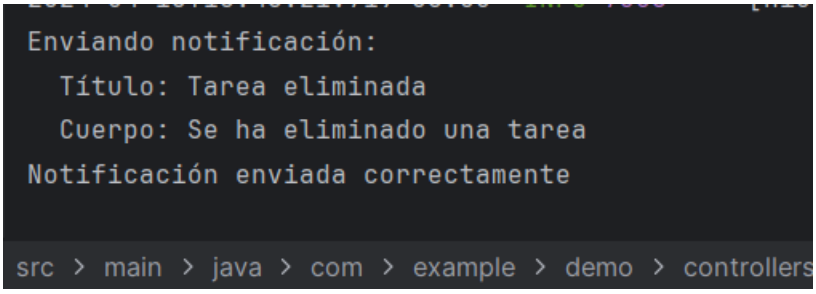
El controlador TaskServiceController, después de crear una nueva tarea, se envía una notificación al usuario informándole sobre la creación exitosa de la tarea.



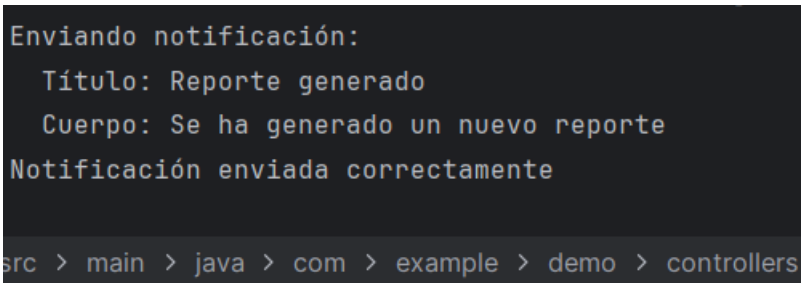
Cuando una tarea se marca como completada, se envía una notificación al usuario indicando que la tarea ha sido completada:



Tras eliminar una tarea, se envía una notificación al usuario comunicando que la tarea ha sido eliminada:



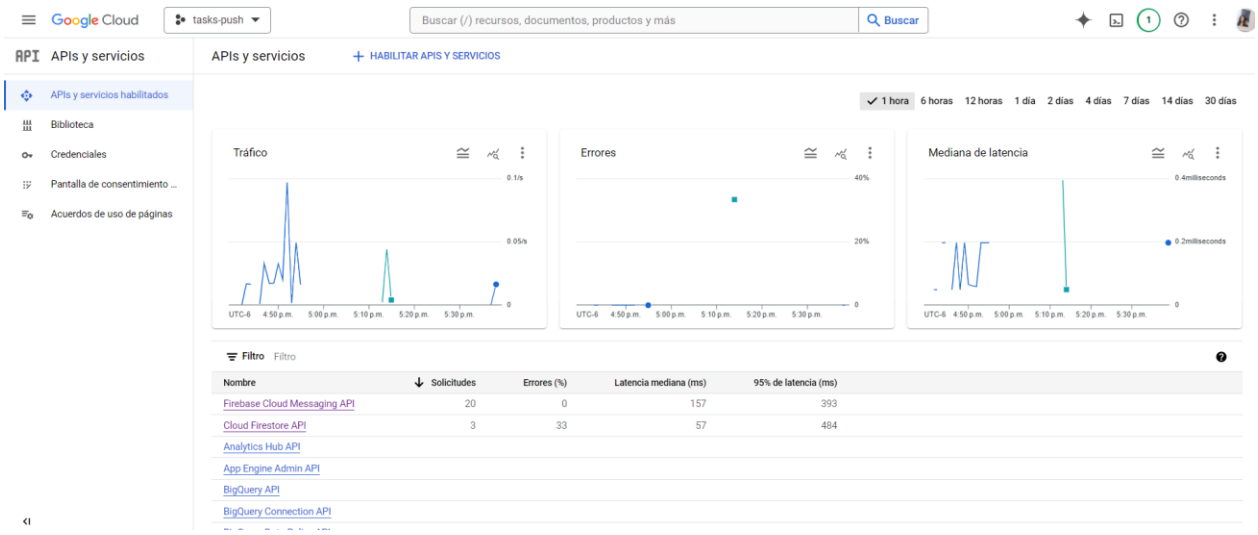
Y por ultimo, después de generar un informe de tareas, se notifica al usuario sobre la generación exitosa del informe:



Para comprobar el envío de las notificaciones, puedes acceder a la consola de Google Cloud Platform en la siguiente dirección:

https://console.cloud.google.com/

Una vez allí, selecciona el nombre de tu proyecto, en este caso "tasks-push". Luego, navega hasta la sección de "APIs y Servicios" donde podrás realizar una variedad de tareas relacionadas con el envío y la gestión de notificaciones push. Podrás obtener estadísticas detalladas sobre el número total de notificaciones enviadas, entregadas con éxito, fallidas, así como el número de notificaciones abiertas por los usuarios, entre otros datos importantes para evaluar el rendimiento de tus campañas de notificaciones.



Construido con
JAVA 17
Spring boot 3.3.0
Maven

Autores
Cintia Favila.