**Proyecto de paradigmas**

**Desarrollo de una API para el Procesamiento de Logs de una Aplicación Spring Boot utilizando Programación Funcional**

**Documentación del Proyecto - Procesamiento de Logs (ProyectoPP)**

**Participantes:**

**Kevin Beita Marín**

**Greivin Narváez Jiménez**

**Francisco Araya Leitón**

**Yutaro Cubero Abe**

**Curso:**

**Paradigmas de programación.**

**Profesor:**

**Allam Chaves Zamora**

**Universidad Nacional de Costa Rica**

**Sede Interuniversitaria de Alajuela**

**2025**

Este proyecto implementa una API en Spring Boot para procesar logs de la aplicación. La API genera diferentes reportes analíticos (errores, tiempos de respuesta, uso de endpoints, alertas críticas y estado general) utilizando técnicas de programación funcional de Java; para así hacerla totalmente funcional para el que desee implementarla en sus propios proyectos.

A continuación, se procederá a dar las instrucciones para el uso y prueba del api y ejecutar la aplicación.

Instrucciones:

1. Abrir la carpeta del proyecto en un ide.

2. En la terminal integrada ejecutar: mvn clean package

3. Ejecutar: mvn spring-boot:run

4.Acceder a la aplicación en http://localhost:8080

5. Abrir Swagger UI en [http://localhost:8080/swagger-ui.html o /swagger-ui/index.html](http://localhost:8080/swagger-ui.html%20o%20/swagger-ui/index.html)

Siguiendo estos pasos, podrá ejecutar con éxito la aplicación.

Ahora bien, por motivos de documentación, se procederá a dar evidencia de cada endpoint del api, con sus respectivas capturas de pantalla. Estas capturas, son directamente desde Swager UI, donde se puede mostrar de manera visual la ejecución de cada endpoint y su salida en formato JSON.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.Reporte de errores:

Patrón de fondo

El contenido generado por IA puede ser incorrecto.Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.Mostrando lista de errores agrupados:

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.Aplicación

El contenido generado por IA puede ser incorrecto.Top3 mostrando los 3 errores más frecuentes:

Interfaz de usuario gráfica, Patrón de fondo

El contenido generado por IA puede ser incorrecto.Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.Mostrando errores agrupados por hora:

**Tiempo de respuestas:**

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.Interfaz de usuario gráfica, Aplicación, Teams

El contenido generado por IA puede ser incorrecto.Mostrando promedio, min, max, mediana:

Texto

El contenido generado por IA puede ser incorrecto.Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.Mostrando distribución por endpoint:

Forma, Rectángulo

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación, Teams

El contenido generado por IA puede ser incorrecto.**Uso de endpoints:**

Mostrando top y least usados:

Texto

El contenido generado por IA puede ser incorrecto.

Rectángulo

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación, Teams

El contenido generado por IA puede ser incorrecto.Mostrando conteo por método HTTP:

Patrón de fondo

El contenido generado por IA puede ser incorrecto.

**Alertas y eventos críticos:**

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.Mostrando lista de alertas críticas:

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Mostrando cantidad de alertas:**Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.**

**Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.**

**Estado de la aplicación:**

Forma, Patrón de fondo, Rectángulo

El contenido generado por IA puede ser incorrecto.Interfaz de usuario gráfica, Aplicación, Teams

El contenido generado por IA puede ser incorrecto.Mostrando estado general:

**Datos de prueba desde Postman**

**Entrada:**

GET<http://localhost:8080/api/usage/by-method>

**Salida:**

Error 404 Not Found

**Entrada:**

POST http://localhost:8080/api/reports/usage/by-method

**Salida:**

Error 405 Method Not Allowed

**Entrada:**

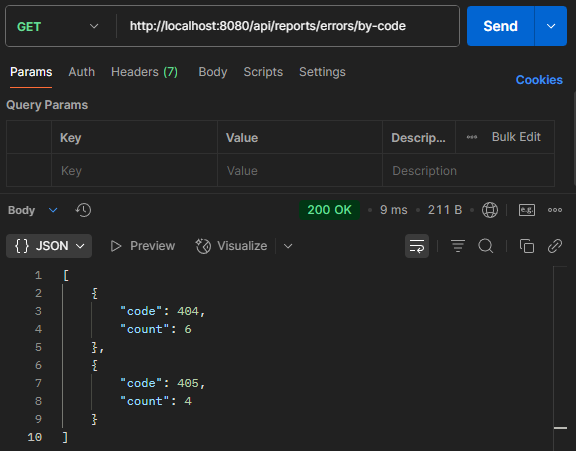
GET http://localhost:8080/api/reports/errors/by-code

**Salida:**

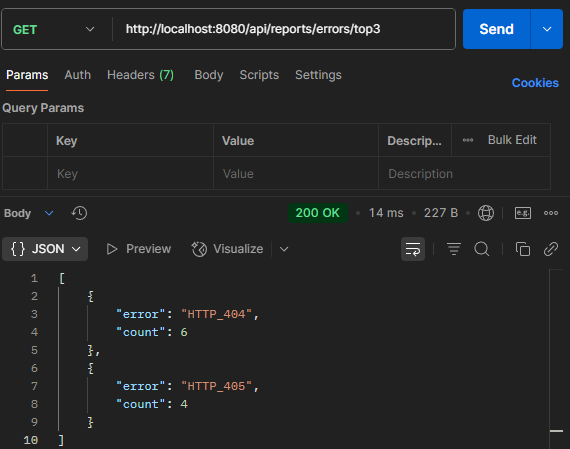
Código 200 OK

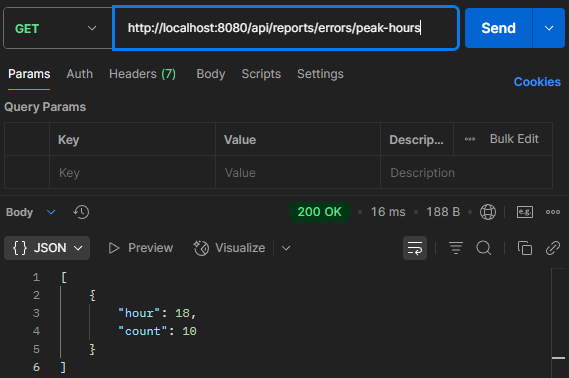
**Nota: ‘**Los datos iniciales puede reiterarse con el fin de aumentar los contadores deceventos en los endpoint’

**Cantidad de errores por código**

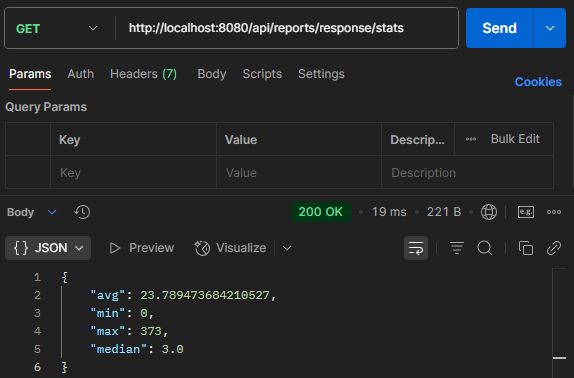
****

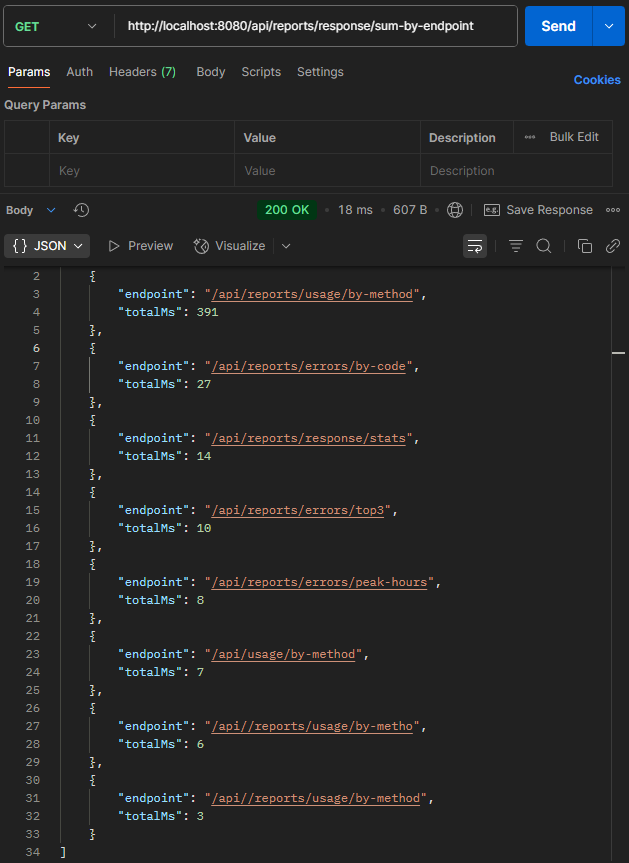
**Top 3 errores más frecuentes**

****

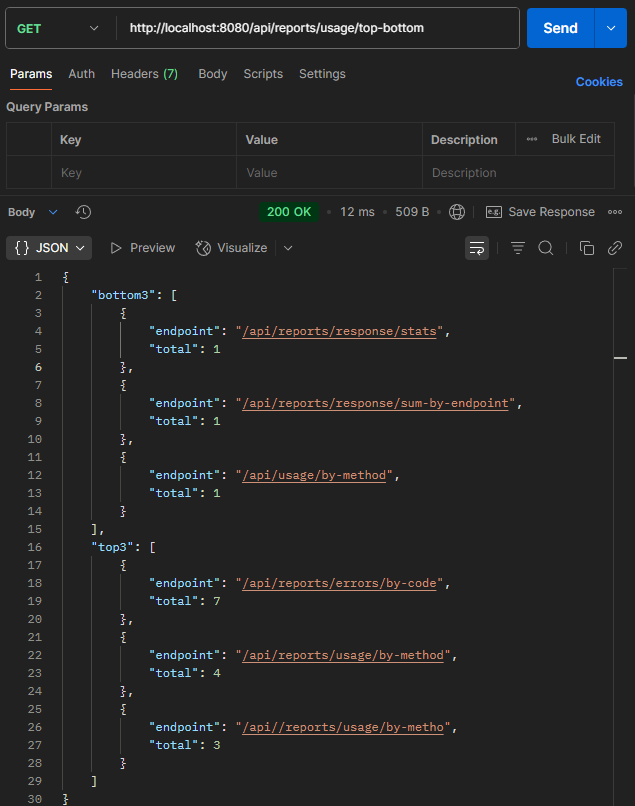
**Horas pico donde suceden más errores**

**Estadísticas de tiempos de respuesta**

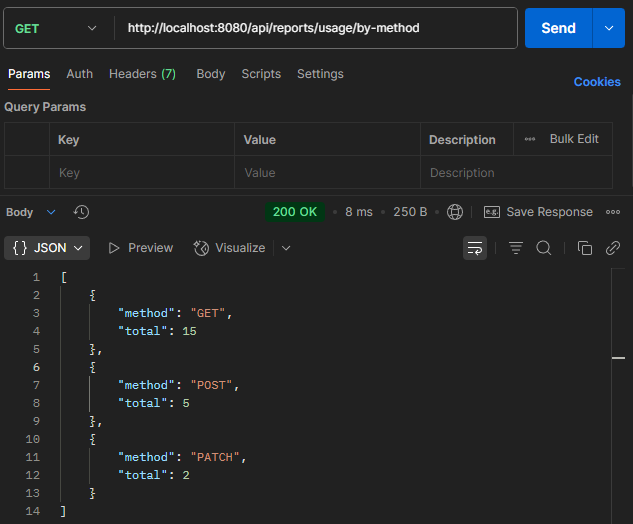
****

**Suma de milisegundos empleados por endpoint**

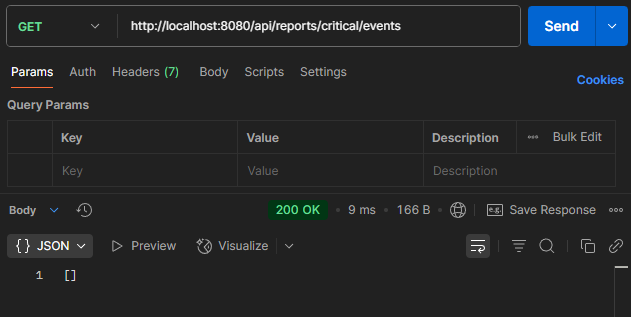
**Top 3 endpoints más y menos utilizados**

****

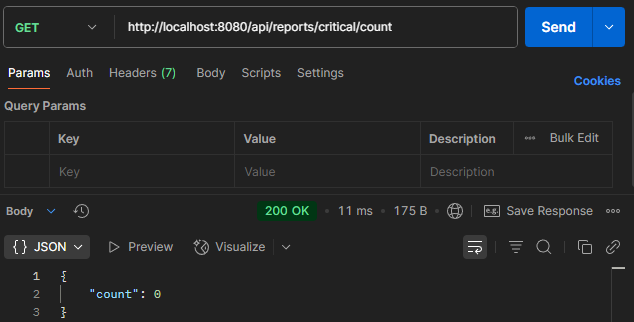
**Cantidad de usos por método**

****

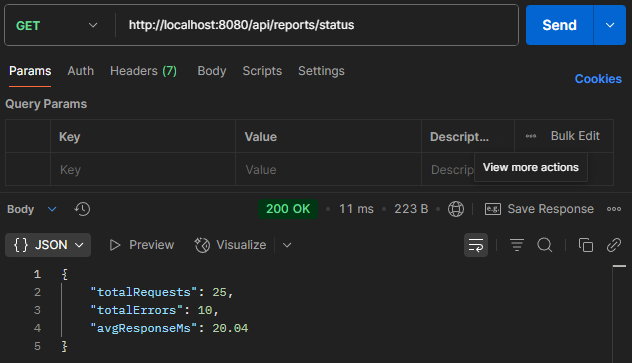
**Eventos críticos**

****

**Contador de eventos críticos**

****

**Estado general de la aplicación**

****

A su vez, también se procede a mostrar la programación funcional aplicada en la aplicación. Por lo que el proyecto utiliza programación funcional en Java para procesar colecciones de logs. Se emplean pipelines de Streams para filtrar, mapear y agrupar datos sin usar bucles tradicionales. Se promueve la inmutabilidad y se evita modificar colecciones externas dentro de las operaciones de stream.

Ejemplo incluido en el código (cálculo de mediana):

**List<Long> sorted = durations.stream()  
 .sorted()  
 .collect(Collectors.toList());  
double median = (sorted.size() % 2 == 1)  
 ? sorted.get(sorted.size()/2)  
 : (sorted.get(sorted.size()/2 - 1) + sorted.get(sorted.size()/2)) / 2.0;**

El proyecto de Procesamiento de Logs implementado en Spring Boot cumple con los objetivos planteados, ofreciendo una API robusta y funcional para el análisis de registros de aplicación. Gracias al uso de técnicas de programación funcional en Java, se logra un procesamiento eficiente de grandes volúmenes de datos, promoviendo la inmutabilidad y evitando efectos secundarios sobre colecciones externas.

Los diferentes reportes generados —errores, tiempos de respuesta, uso de endpoints, alertas críticas y estado general— permiten una visualización clara y estructurada de la información relevante para la gestión y monitoreo de la aplicación. Asimismo, la integración con Swagger UI facilita la interacción directa con los endpoints, proporcionando una experiencia de prueba y documentación accesible y completa.

En definitiva, esta implementación constituye una herramienta versátil y reutilizable para desarrolladores que requieran analizar logs de manera precisa y eficiente, destacando por su facilidad de uso, claridad en los reportes y adopción de buenas prácticas de programación funcional en Java.