

# Implementación de la Componente Principal de la Plataforma ENIGMA

Autor: Luis Enrique Saborit González

Tutor: DrC. Daniel Gálvez Lio

Tutor: DrC. Amed Abel Leiva Mederos

# Antecedentes

- En nuestro país, cada día es más importante incorporar la aplicación de la Inteligencia Artificial y la Web Semántica en las aplicaciones relacionadas con el Internet de las Cosas (IoT).
- Se ha trabajado en una plataforma llamada Traffic que combina estos elementos, pero tiene ciertas limitaciones.

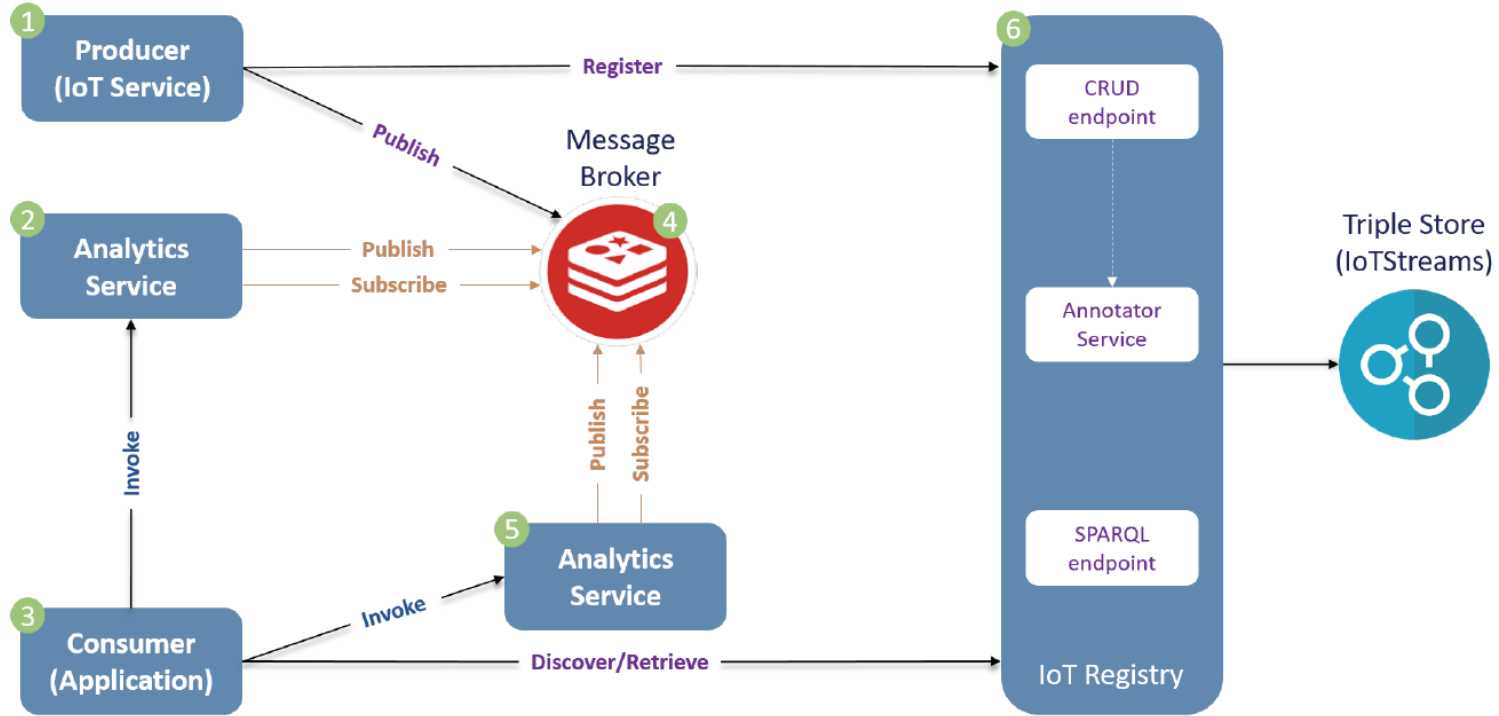
# Objetivo General

Desarrollar la Componente Principal de la plataforma ENIGMA, encargada del procesamiento de flujos de datos, con una arquitectura extensible y escalable.

## Objetivos específicos

1. Establecer los requisitos de diseño, los componentes y la arquitectura de la plataforma ENIGMA.
2. Implementar el Componente Principal de la plataforma.
3. Instanciar el componente desarrollado para un flujo particular.
4. Evaluar el componente desarrollado.

# Traffic, un sistema para la detección de eventos de tráfico basado en ontologías

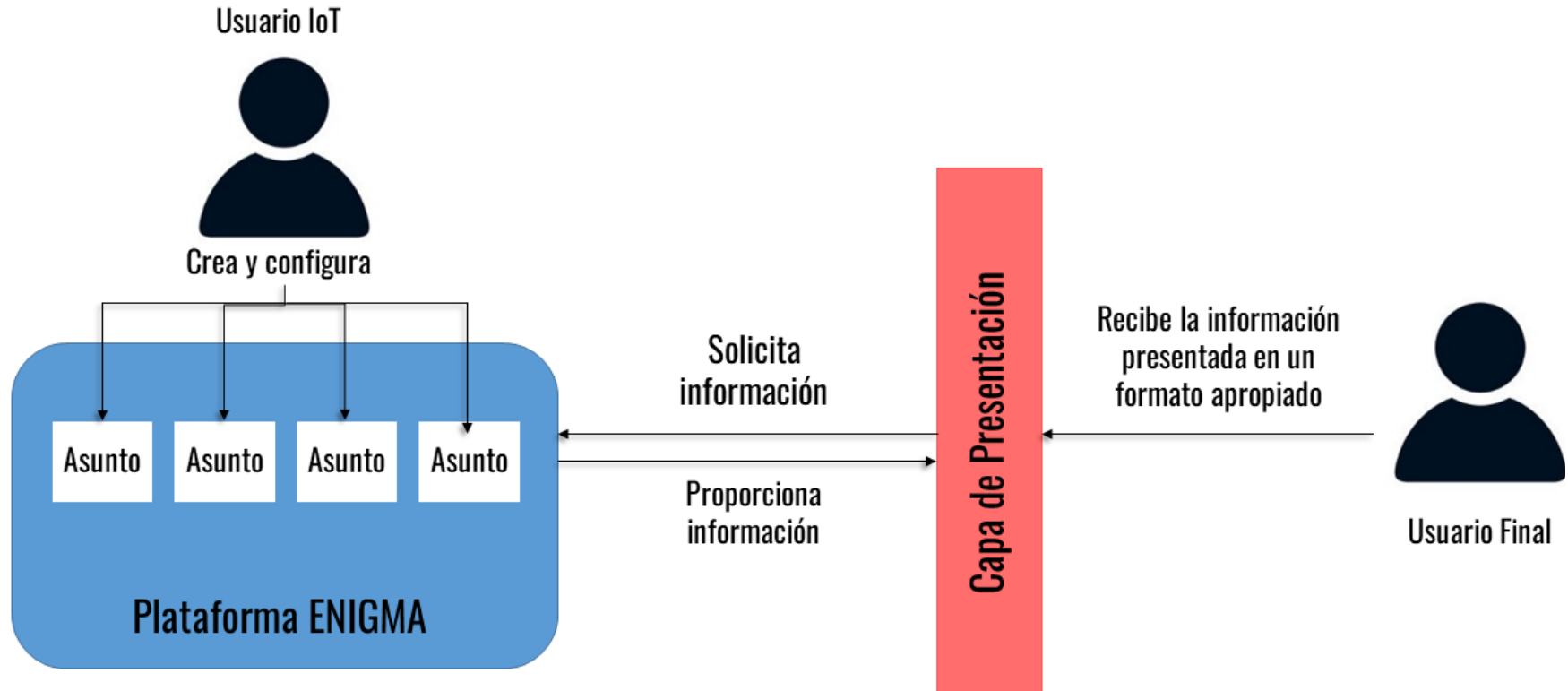


# Traffic: Valoración del Diseño

- Es un sistema para el procesamiento de los datos de sensores ubicados en la ciudad de Bruselas, en Bélgica.
- El sistema es flexible y se puede adaptar a otras fuentes de datos, pero es necesario modificar el código fuente.
- No es capaz de manejar varias fuentes de datos a la vez.
- Publica los eventos en una base de datos Redis.
- Posee una componente de analíticas (*Gaussian Mixture Models*).
- Se enriquece la información de los resultados registrándolos en un *triplestore*.

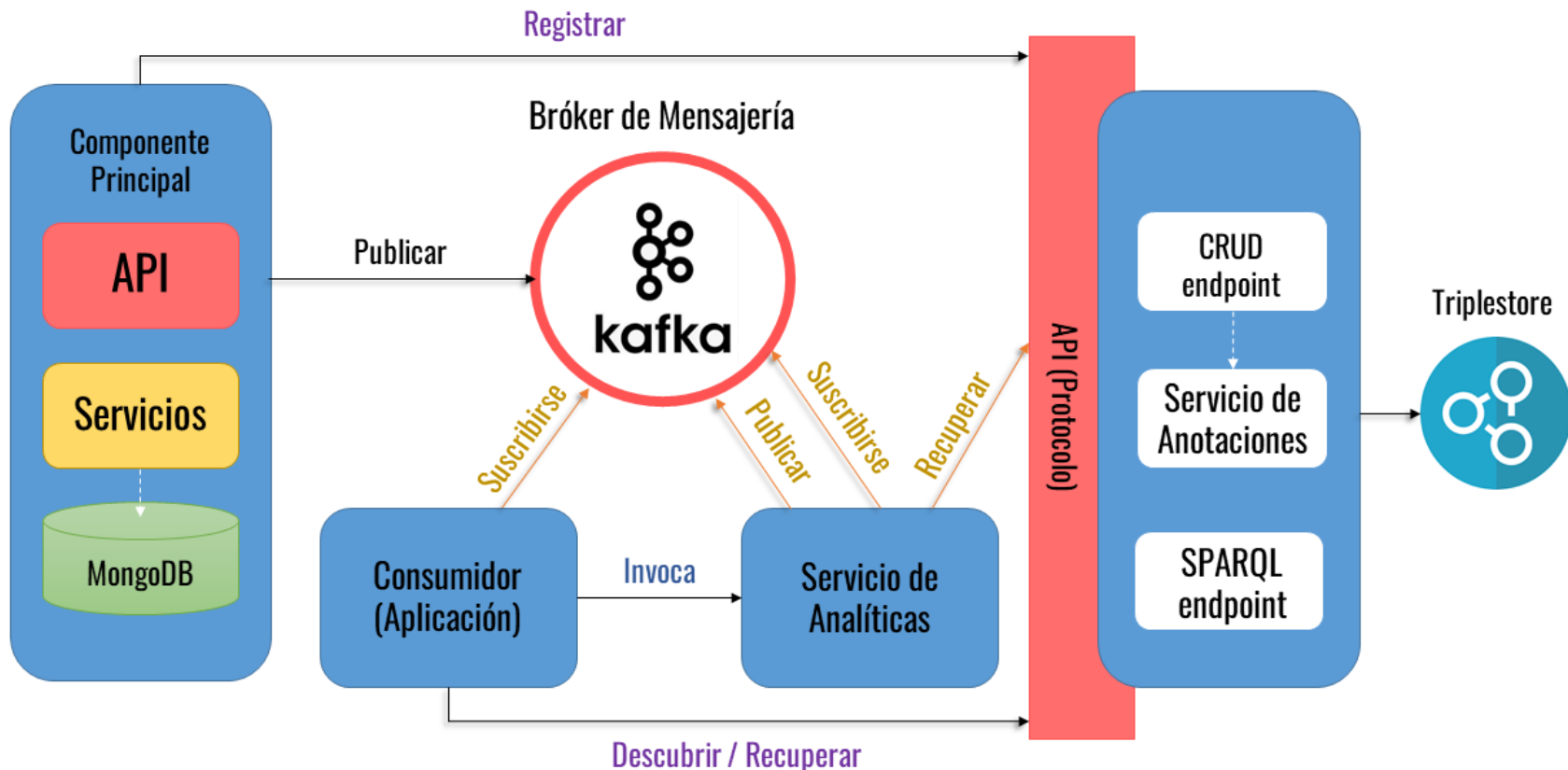
# **Diseño de la plataforma ENIGMA**

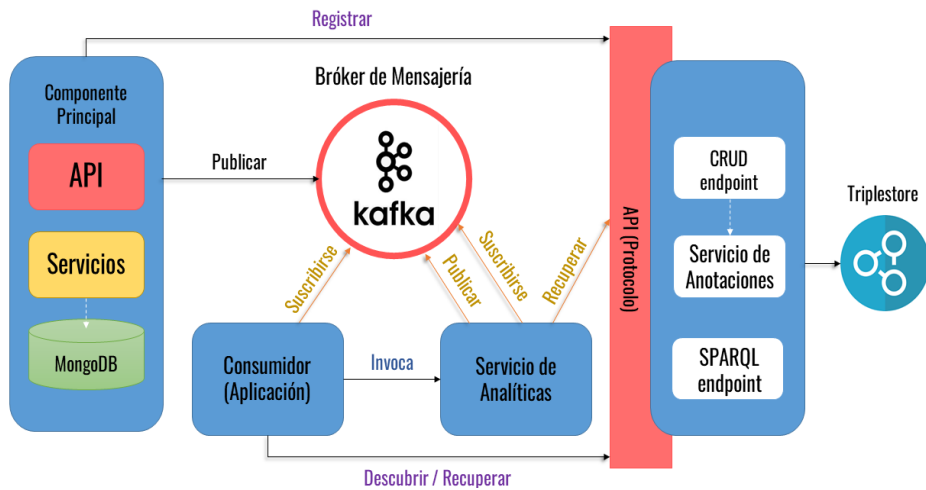
# Diagrama de Contexto de la plataforma ENIGMA





# Diseño de los componentes de la plataforma ENIGMA



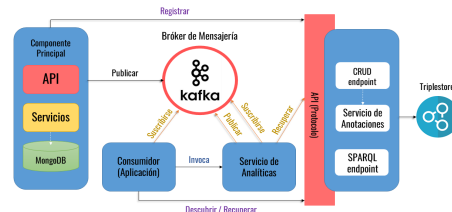


# Principales Modificaciones

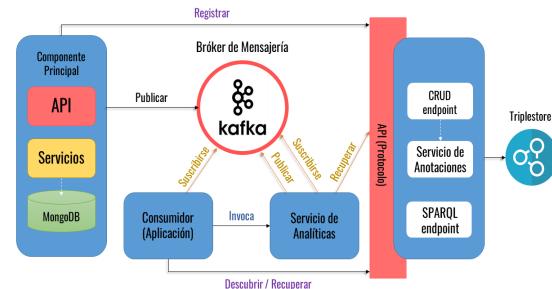
# Generalidad en la plataforma ENIGMA

Para que la plataforma pueda procesar datos de cualquier fuente, y varias instancias a la vez, se implementan las siguientes modificaciones:

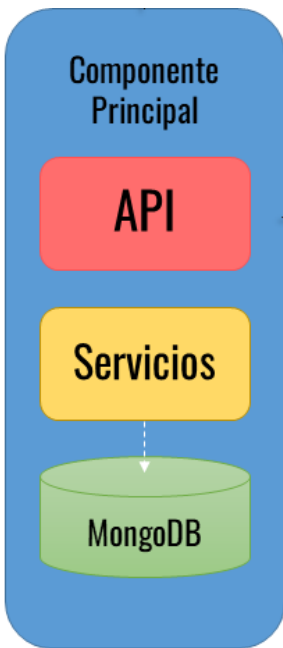
- Se propone definir un protocolo para la comunicación entre los componentes de Servicio de Analíticas externos y la plataforma.
- Se expone una API HTTP a la cual pueden publicarse los datos de los sensores, streams, asuntos y observaciones.
- Se propone la creación del Componente Registro IoT de tal forma que pueda procesarse cualquier tipo de datos.
- Se implementa una interfaz que define el comportamiento de los tipos de datos de las observaciones. Varias clases implementan dicha interfaz.



# Otras modificaciones



- Se propone utilizar Apache Kafka en lugar de Redis en el bróker de mensajería.
- Se creó un sistema de asuntos para organizar los datos de las diferentes instancias dentro de la plataforma.



# Implementación de la Componente Principal

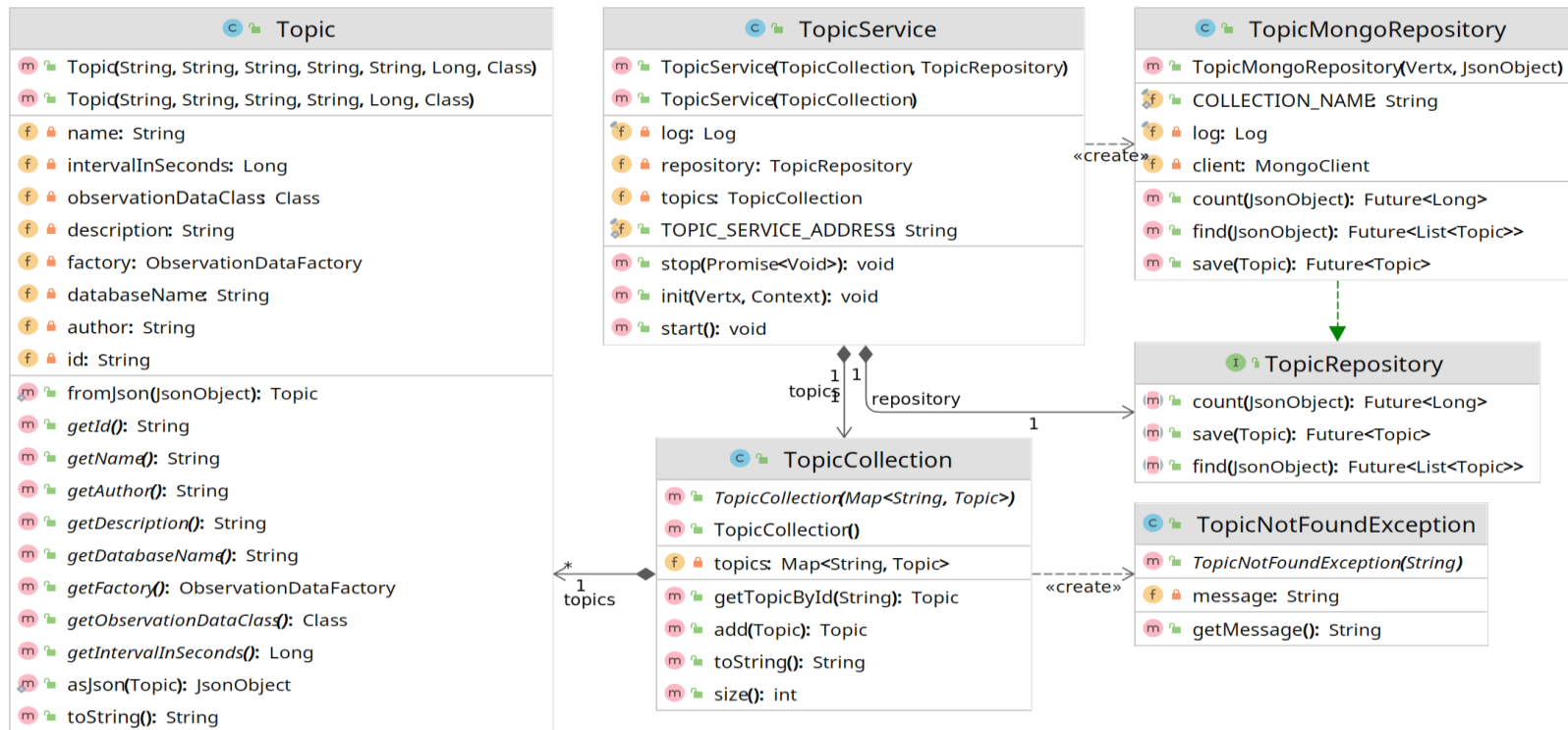
# Servicios implementados en la Componente Principal

El Componente Principal está escrito en lenguaje Java y se usa el kit de herramientas Vert.x para lograr que sus operaciones se realicen de forma asíncrona. Está compuesto por varios servicios, como son:

- **ApiService:** este servicio expone una API HTTP a la cual se pueden publicar nuevos sensores, streams, observaciones y asuntos.
- **SensorService:** es un servicio que se ocupa de gestionar los sensores con los que trabaja cada asunto del sistema.
- **StreamService:** se encarga de gestionar los streams que generan los sensores.
- **ObservationService:** este servicio se encarga de gestionar las observaciones y publicarlas al bróker de mensajería.
- **ObservationLoggingService:** es un servicio que registra todas las observaciones que son procesadas por el sistema en un repositorio MongoDB, para su posterior análisis.
- **TopicService:** es el servicio que se encarga de crear, modificar, buscar y eliminar los asuntos.

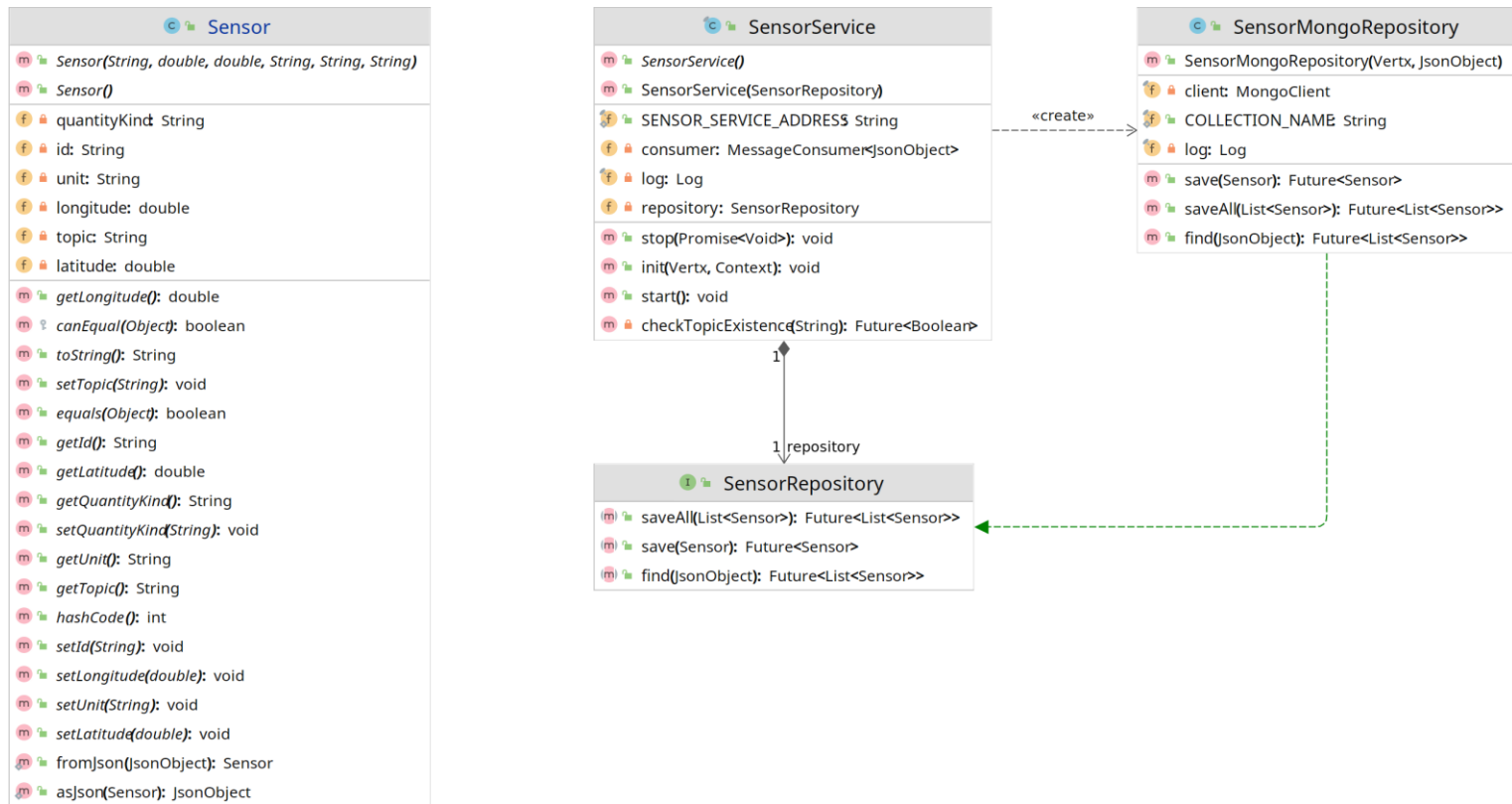
# Principales Clases de la Implementación en el Componente Principal:

## Asuntos



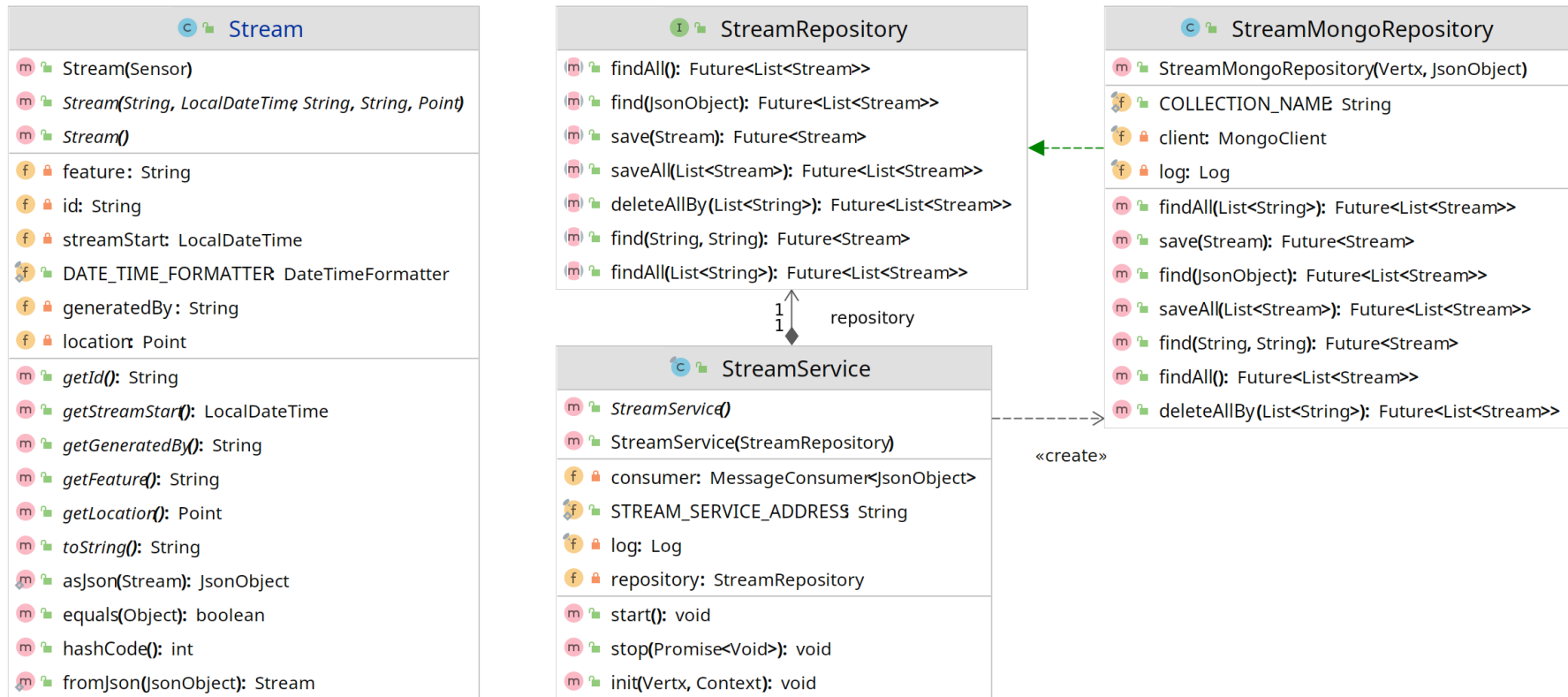
# Principales Clases de la Implementación en el Componente Principal:

## Sensores

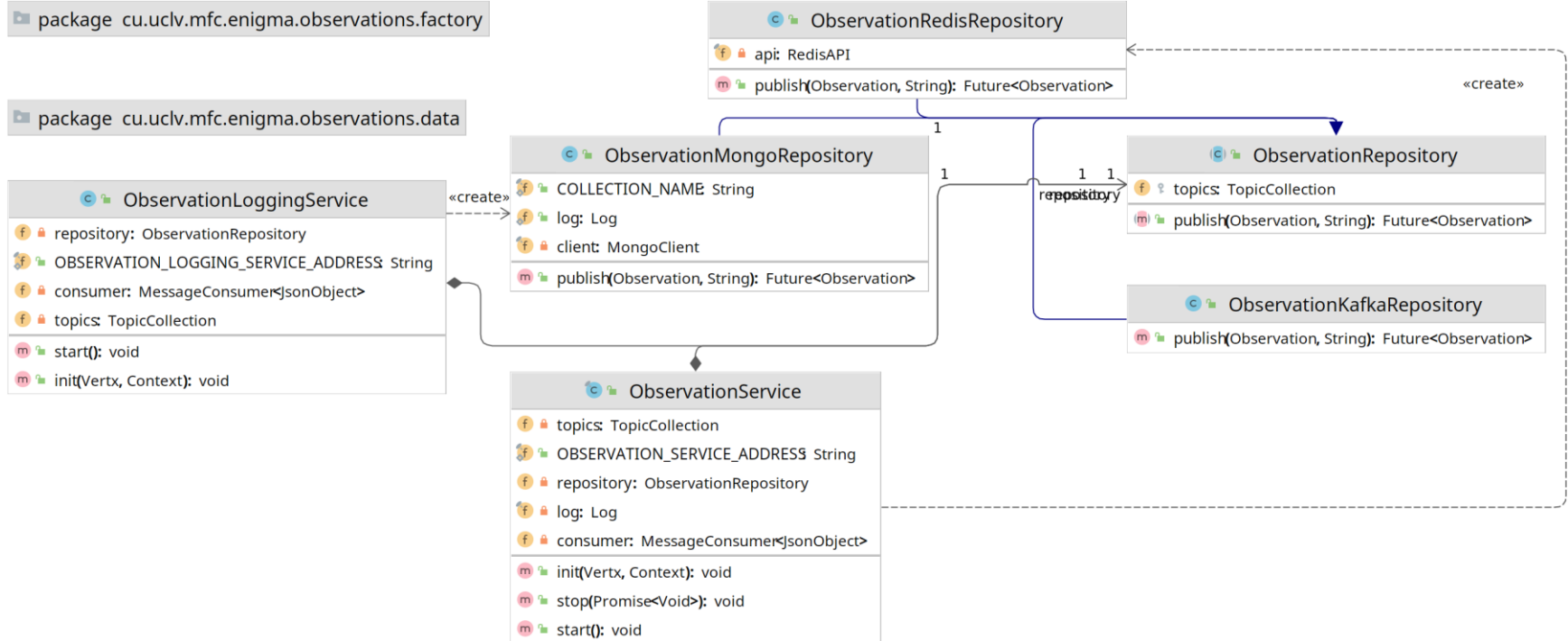




# Principales Clases de la Implementación en el Componente Principal: Streams



# Principales Clases de la Implementación en el Componente Principal: Observaciones

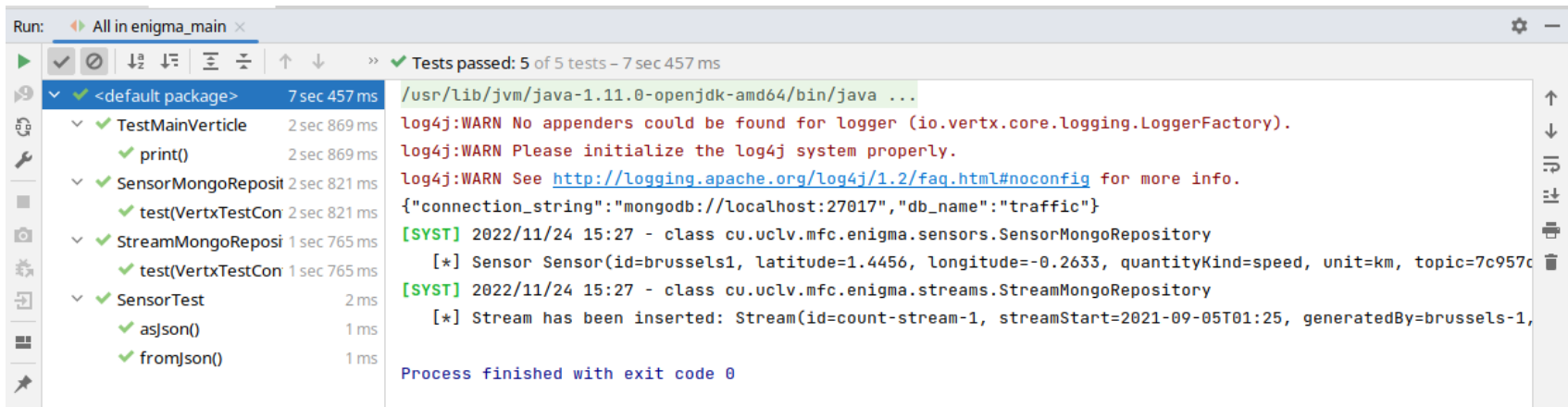


**Pruebas Realizadas**

# Pruebas Unitarias

Las pruebas unitarias se enfocan en características y operaciones específicas. En este caso, se comprueba que un sensor o un stream se agregue al repositorio de forma satisfactoria y que los datos se conviertan hacia y desde el formato JSON.

En el proceso, se utilizó JUnit5, un framework para pruebas unitarias en Java. Además se utilizaron herramientas de Vert.x para realizar las pruebas al código que se comporta de forma asíncrona.



The screenshot shows the 'Run' window of an IDE, titled 'All in enigma\_main'. The top bar indicates 'Tests passed: 5 of 5 tests - 7 sec 457 ms'. The test results are listed in a tree view on the left, and the corresponding log output is shown on the right.

Test Name	Duration
<default package>	7 sec 457 ms
TestMainVerticle	2 sec 869 ms
print()	2 sec 869 ms
SensorMongoReposit	2 sec 821 ms
test(VertxTestCon	2 sec 821 ms
StreamMongoReposi	1 sec 765 ms
test(VertxTestCon	1 sec 765 ms
SensorTest	2 ms
asJson()	1 ms
fromJson()	1 ms

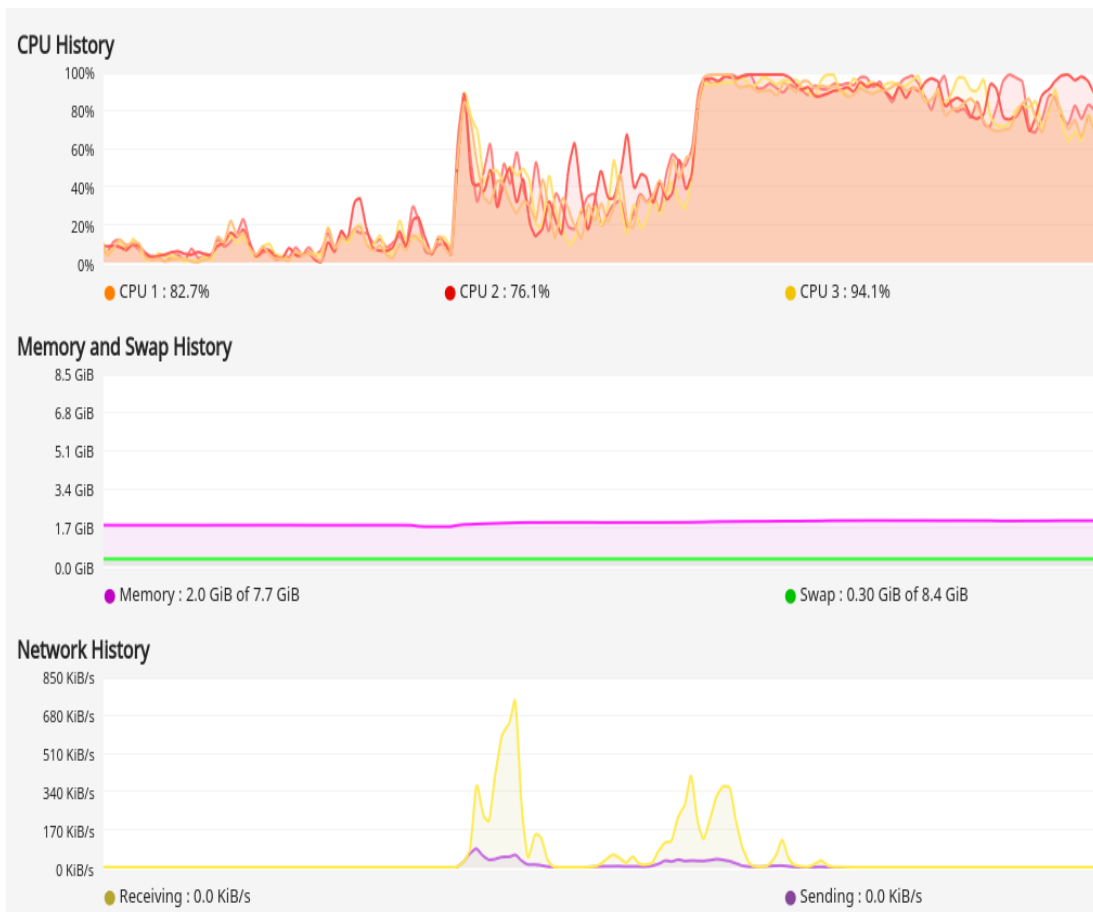
The log output on the right shows the following messages:

```
/usr/lib/jvm/java-1.11.0-openjdk-amd64/bin/java ...
log4j:WARN No appenders could be found for logger (io.vertx.core.logging.LoggerFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
{"connection_string":"mongodb://localhost:27017","db_name":"traffic"}
[SYST] 2022/11/24 15:27 - class cu.uclv.mfc.enigma.sensors.SensorMongoRepository
[*] Sensor Sensor(id=brussels1, latitude=1.4456, longitude=-0.2633, quantityKind=speed, unit=km, topic=7c957c
[SYST] 2022/11/24 15:27 - class cu.uclv.mfc.enigma.streams.StreamMongoRepository
[*] Stream has been inserted: Stream(id=count-stream-1, streamStart=2021-09-05T01:25, generatedBy=brussels-1,
Process finished with exit code 0
```

# Pruebas de Rendimiento

También se realizaron pruebas de rendimiento al sistema:

- Se simula el trabajo de la plataforma en un único caso de uso. Para ello se usa un script escrito en Python que crea un asunto, hace peticiones a la fuente de datos y publica esta a la API.
- En la segunda prueba se crearon veinte instancias en la plataforma. Para cada instancia se hizo un trabajo similar al de la primera prueba. Las peticiones y las publicaciones a la plataforma de todos los casos de uso se realizan al mismo tiempo.



# Conclusiones

Como resultado de este trabajo se arriban a las conclusiones siguientes:

- Se establecen como requisitos principales de la plataforma ENIGMA la adaptabilidad y extensibilidad, esta plataforma está integrada por cinco componentes contruidos en forma de servicios por lo que esta arquitectura permite un despliegue distribuido de sus componentes.
- Se han implementado las clases y los servicios que integran la Componente Principal de la plataforma ENIGMA y a manera de evaluación del diseño y la implementación se ha instanciado este componente para procesar un flujo concreto.
- Se definió y aplicó una estrategia de evaluación para la componente implementada, los resultados de esta evaluación basadas en pruebas fueron satisfactorias.

# Recomendaciones

Para dar continuidad a este trabajo se recomienda:

- Realizar pruebas del Componte Principal en entornos más reales y diversos.
- Trabajar para lograr que el Componente Principal pueda procesar información de cualquier tipo de dato definido por el usuario que adapta la plataforma a su caso específico.
- Continuar con la implementación de los restantes componentes de la plataforma.

# Implementación de la Componente Principal de la Plataforma ENIGMA

Autor: Luis Enrique Saborit González

Tutor: DrC. Daniel Gálvez Lio

Tutor: DrC. Amed Abel Leiva Mederos



# Pregunta 1

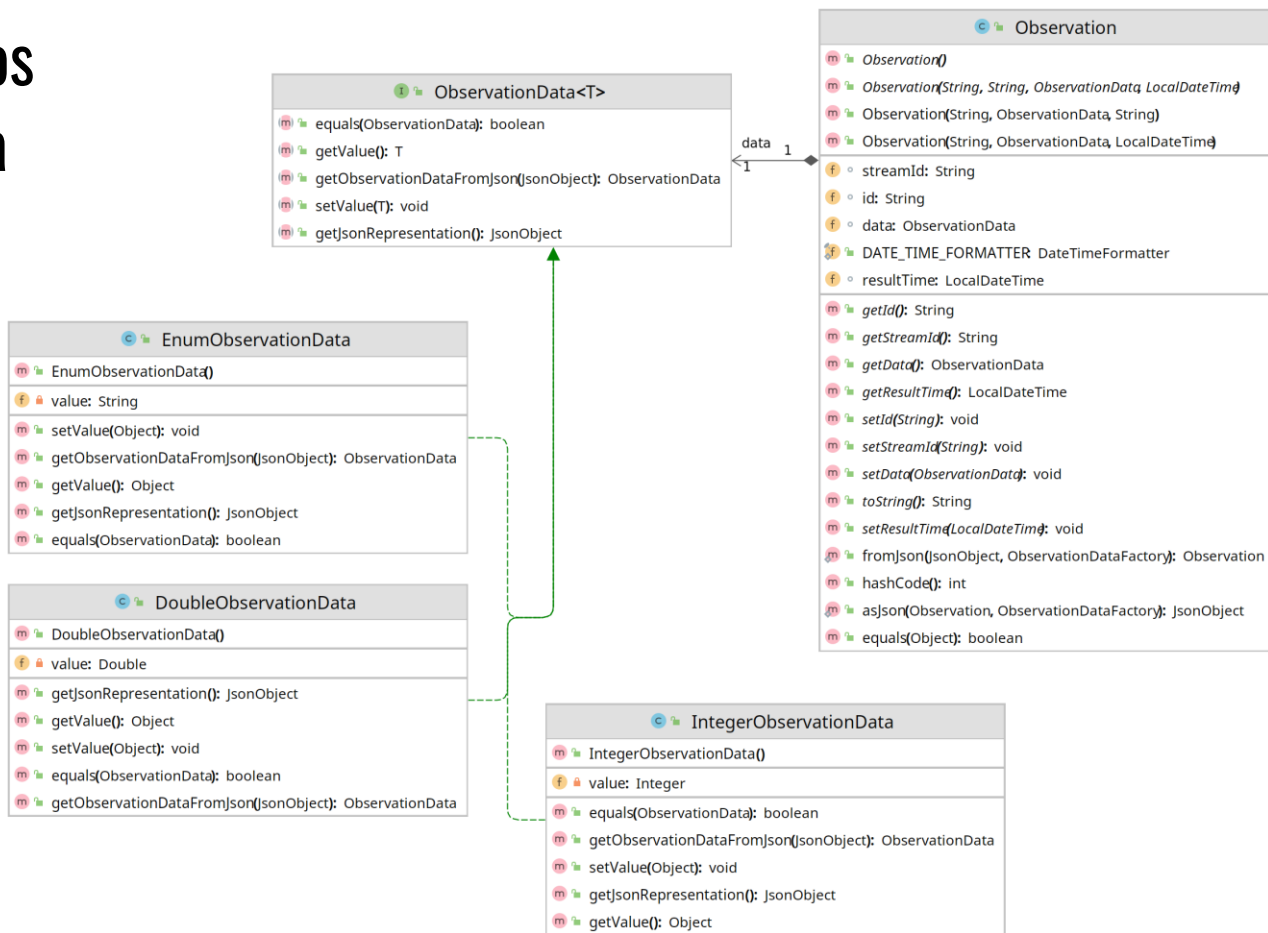
¿Cómo se logra en el sistema propuesto la flexibilidad de aceptar datos de las mediciones de los sensores independientemente de su tipo? Ejemplifique

R/

- Se expone una API HTTP a la cual pueden publicarse los datos de los sensores, streams, asuntos y observaciones en formato JSON, de tal forma que se puede introducir cualquier tipo de datos a la plataforma.
- Se propone implementar los demás componentes estableciendo un protocolo similar para la comunicación y el procesamiento de los datos.

# Generalización de los Tipos de Datos en la Plataforma

- Se ha implementado la interfaz **ObservationData**, que define el comportamiento de los tipos de datos de las observaciones. Varias clases implementan dicha interfaz: **DoubleObservationData**, **IntegerObservationData** y **EnumObservationData**.



## Pregunta 2

En el documento se menciona que una desventaja de Apache Kafka frente a Redis es su lentitud. Sin embargo, el sistema propuesto debe estar preparado para soportar un volumen de datos de grandes dimensiones. No incide esto en su rendimiento. ¿Se realizaron pruebas?

R/ El uso de Apache Kafka está motivado principalmente por la falta de escalabilidad de un bróker que use Redis. Kafka está diseñado para manejar grandes volúmenes de datos con mayor tolerancia a los fallos, mientras que Redis, al cargar sus datos directamente a la memoria RAM de la computadora, no resulta tan fiable en este sentido.

La desventaja de Kafka, al ser ligeramente más lento, es un costo que se puede despreciar en favor de las ventajas que ofrece.

# De Redis a Apache Kafka

Se propone sustituir el bróker basado en Redis, por uno basado en Apache Kafka. Entre las principales ventajas de este sistema de transmisión de mensajes están:

- Múltiples consumidores pueden procesar los datos al mismo tiempo, algo de lo que carece Redis.
- Permite procesar flujos de eventos a medida que se producen o de forma retrospectiva.
- Pensado para manejar grandes cantidades de datos con una mayor tolerancia a fallos. Permite utilizar tantos servidores como sea necesario y mantener hasta terabytes de datos durante un período de retención más largo que Redis.
- Su principal desventaja es su lentitud en comparación con Redis, debido a que guarda los mensajes incluso después de entregarlos.

En conclusión, se propone usar Kafka en busca de mayor escalabilidad, fiabilidad, alto rendimiento y tolerancia a fallos.

## Pregunta 3

En las pruebas de rendimiento solo se muestran los recursos empleados, no así los tiempos de respuesta. Incluya en las mismas los tiempos de respuesta del sistema.

R/ Hemos modificado los scripts de las pruebas de rendimiento para calcular los tiempos de respuesta de la API HTTP del Componente Principal.

Los resultados fueron:

- El máximo tiempo de respuesta de una petición a la API, la cual crea un asunto es 0.031 segundos.
- El máximo tiempo de respuesta de una petición a la API, la cual publica un sensor es : 0.821 segundos.
- El máximo tiempo de respuesta de una petición a la API, la cual publica un stream: 0.915 segundos.

Hay que destacar que todas las peticiones, que son aproximadamente 4340, se realizan a la vez.

# Implementación de la Componente Principal de la Plataforma ENIGMA

Autor: Luis Enrique Saborit González

Tutor: DrC. Daniel Gálvez Lio

Tutor: DrC. Amed Abel Leiva Mederos