Universidad Nacional de Quilmes

Arquitectura de software II

Trabajo Práctico 2 - 2025s1

Jeremias Fuentes - B. Emmanuel Pericon

Julio 2, 2025

Indice

1 Introducción

El presente trabajo práctico tiene como objetivo desarrollar un sistema que pueda obtener información de https://openweathermap.org/, una API gratuita, que nos proveerá datos del clima para luego consultar dicha información según determinados requerimientos. Con este objetivo necesitamos desarrollar dos componentes:

- WeatherLoaderComponent: Consume periódicamente datos de https://openweathermap.org/ y los persiste en una base de datos.
- WeatherMetricsComponent: Consume datos de WeatherLoaderComponent y los expone. Este componente tendrá la responsabilidad de exponer los siguientes reportes mediante una API:
  - Reporte de la temperatura actual
  - Promedio de la temperatura del último día
  - Promedio de la temperatura de la última semana

2 Casos de usos

2.1 CU-WLC (WeatherLoaderComponent)

- Caso de uso: Obtener datos de API externa

Id	CU-WLC-001
Nombre	Obtener datos de API externa
Descripción	Se obtienen los datosObtener datos de API externa de API OpenWeather cada cierta cantidad de tiempo
Precondición	Tener una cuenta y creada una key en OpenWeather para poder tener acceso a la API
Flujo principal	1. Se generan la api key y listado de ciudades a consultar 2. Por medio de una tarea programada cada cierta cantidad de tiempo se hace una petición 3. Se obtienen los datos necesarios 4. Se guardan los valores necesario en la base de datos
Flujo alternativo	En caso de timeout se deja logueado que hubo error y se continua el flujo normal.
Postcondición	Debería haber cada cierta cantidad de tiempo valores guardados en la base de datos (si es que no hubo timeout)

- Caso de uso: Temperatura actual

Id	CU-WLC-002
Nombre	Temperatura actual
Descripción	Se devuelven datos sobre la temperatura actual de una locación
Precondición	-
Flujo principal	1. Se utiliza el query parámetro city para obtener la temperatura actual 2. GET /api/temperature?city=cityName 3. Se devuelve la temperatura actual guardada en BD
Flujo alternativo	3. Si en BD no hay una ultima temperatura con una diferencia menor a la configuracion (`MS_GAP_TIME_FOR_RESEARCH_CURRENT_WEATHER`), se hace la consulta a la API externa 4. Se devuelve la temperatura obtenida de la API externa (si es que no hubo timeout) y se guarda en BD estos datos obtenidos
Postcondición	Se obtienen datos de temperatura actual

- Caso de uso: Temperatura ultimo día

Id	CU-WLC-003
Nombre	Temperatura ultimo día
Descripción	Se devuelve una lista de datos sobre la temperatura del ultimo día de una locación
Precondición	-
Flujo principal	1. Se utiliza el query parámetro city para obtener las temperaturas del ultimo día 2. GET /api/temperature/last-day?city=cityName 3. Se devuelve una lista de datos sobre la temperatura del ultimo día guardadas en BD
Flujo alternativo	-
Postcondición	Se obtienen datos de temperatura del ultimo día

- Caso de uso: Temperatura ultima semana

Id	CU-WLC-004
Nombre	Temperatura ultima semana
Descripción	Se devuelve una lista de datos sobre la temperatura de la ultima semana de una locación
Precondición	-
Flujo principal	1. Se utiliza el query parámetro city para obtener las temperaturas de la ultima semana 2. GET /api/temperature/last-week?city=cityName 3. Se devuelve una lista de datos sobre la temperatura de la ultima semana guardadas en BD
Flujo alternativo	-
Postcondición	Se obtienen datos de temperatura de la ultima semana

2.2 CU-WLC (WeatherMetricsComponent)

- Caso de uso: Reporte de la temperatura actual

Id	CU-WMC-001
Nombre	Reporte de la temperatura actual
Descripción	Se devuelven datos sobre la temperatura actual de una locación
Precondición	-
Flujo principal	1. Se utiliza el query parámetro city para obtener la temperatura actual 2. GET /api/report?city=cityName 3. Se devuelve la temperatura actual
Flujo alternativo	-
Postcondición	Se obtienen datos de temperatura actual

- Caso de uso: Promedio de la temperatura del último día

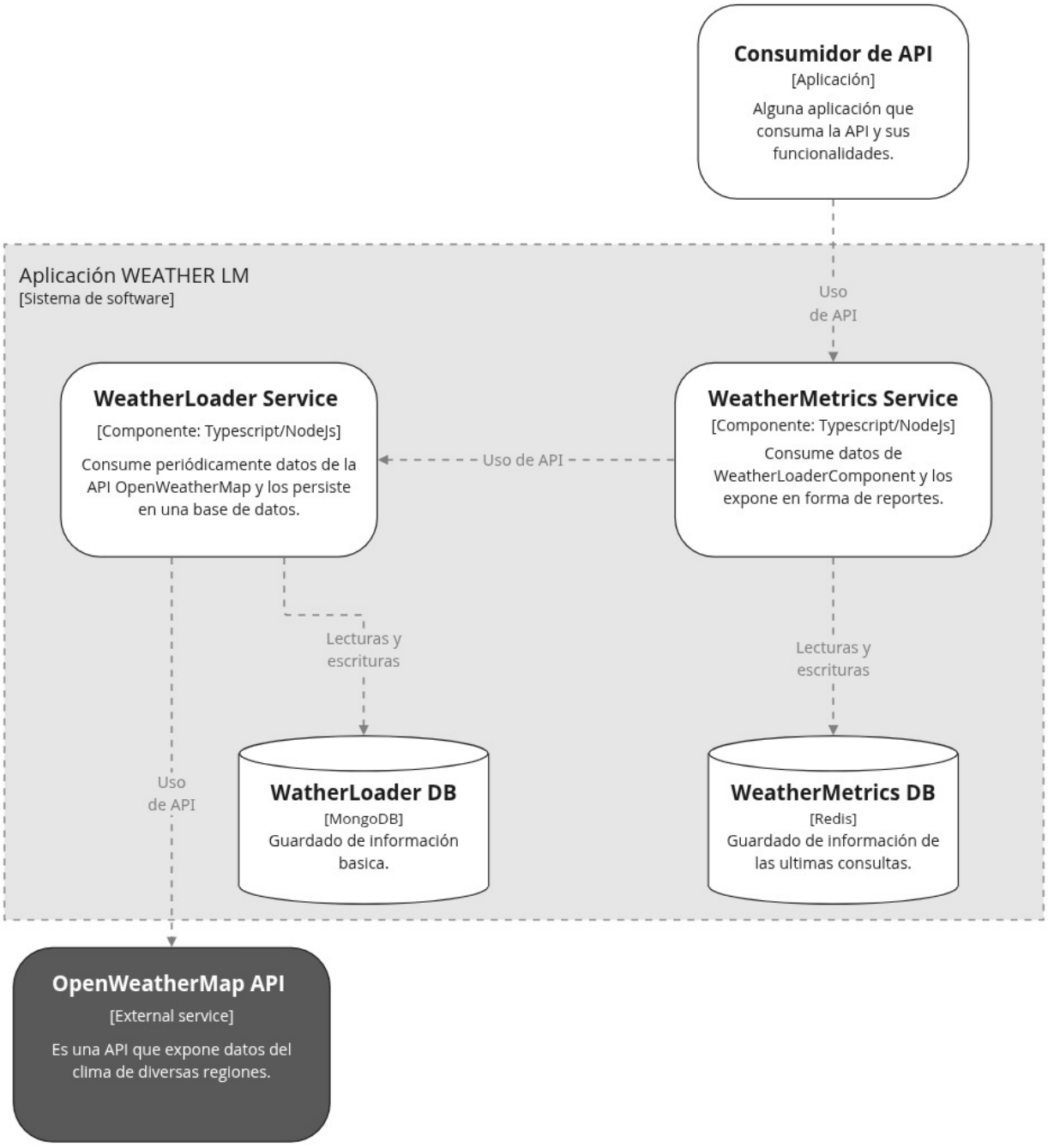
Id	CU-WMC-002
Nombre	Promedio de la temperatura del último día
Descripción	Se devuelve promedio sobre la temperatura del ultimo día de una locación
Precondición	-
Flujo principal	1. Se utiliza el query parámetro city para obtener el promedio de temperaturas del ultimo día 2. GET /api/report/last-day?city=cityName 3. Se devuelve un promedio sobre la temperatura del ultimo día
Flujo alternativo	-
Postcondición	Se obtiene el promedio

- Caso de uso: Promedio de la temperatura de la última semana

Id	CU-WMC-003
Nombre	Promedio de la temperatura de la última semana
Descripción	Se devuelve promedio sobre la temperatura de la ultima semana de una locación
Precondición	-
Flujo principal	1. Se utiliza el query parámetro city para obtener el promedio de temperaturas de la ultima semana 2. GET /api/report/last-week?city=cityName 3. Se devuelve un promedio sobre la temperatura de la ultima semana
Flujo alternativo	-
Postcondición	Se obtienen el promedio

3 Tipo de arquitectura: Hexagonal

Tenemos dos componentes donde cada uno tiene una ocupación particular **WeatherLoaderComponent** y **WeatherMetricsComponent**. Pasamos a especificar endpoints de cada componente:



- **WeatherLoaderComponent:**

  Servidor de Nodejs. Se encarga de la obtención de los datos directamente con la API OpenWeatherMap, guardado de estos datos en una base de datos MongoDB. También expone algunos endpoints para hacer uso de estos datos.

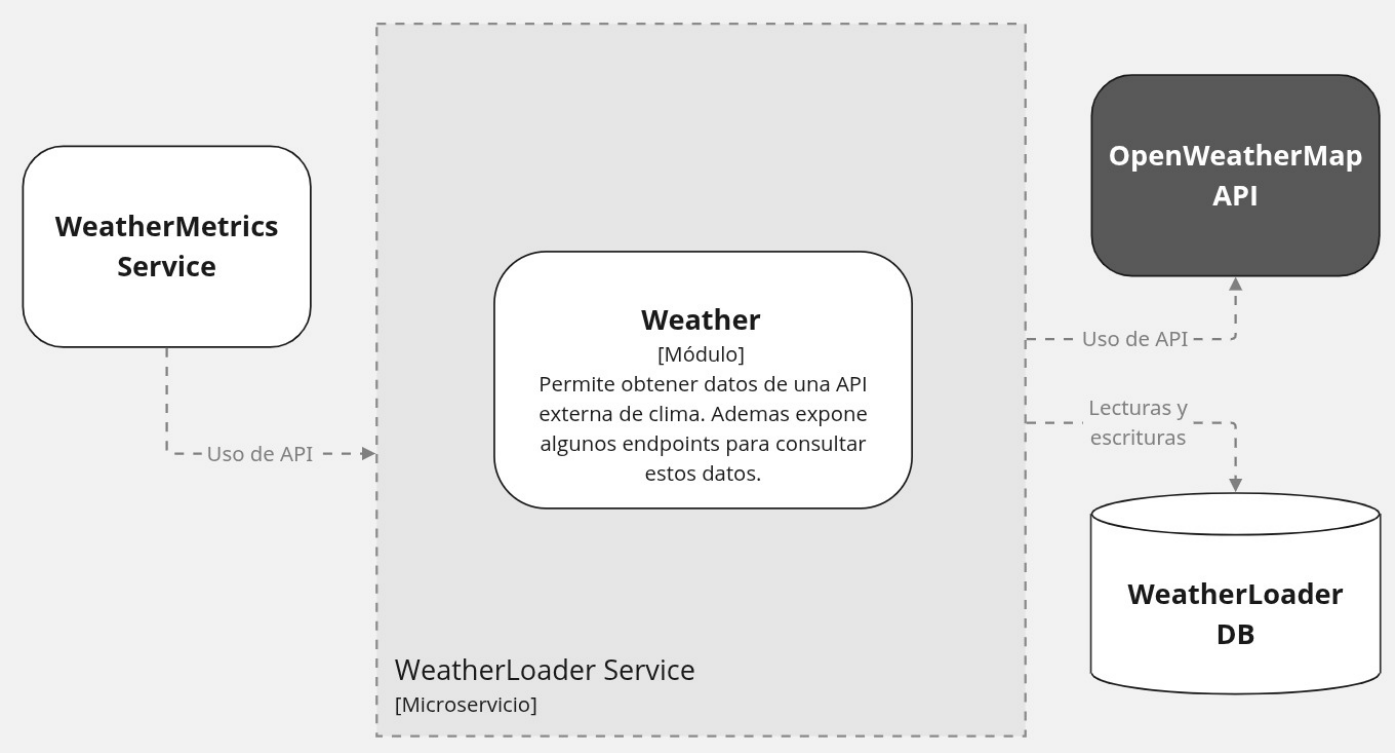
  Comprendido por un módulo:

  – **Weather:** Obtiene los datos de la API externa, guarda estos datos y los expone.

    Obtención de datos: API [JSON/HTTPS]
    Se genera una tarea programada utilizando la librería **node-cron**, la cual nos deja correr la consulta a la API externa cada cierta cantidad de tiempo.

    Interfaz de comunicación: API [JSON/HTTPS]

    * **GET /temperature?city=cityName** - Obtención de temperatura actual
    * **GET /temperature/last-day?city=cityName** - Obtención de lista de temperaturas del ultimo dia
    * **GET /temperature/last-week?city=cityName** - Obtención de lista de temperaturas de la ultima semana



- **WeatherMetricsComponent:**

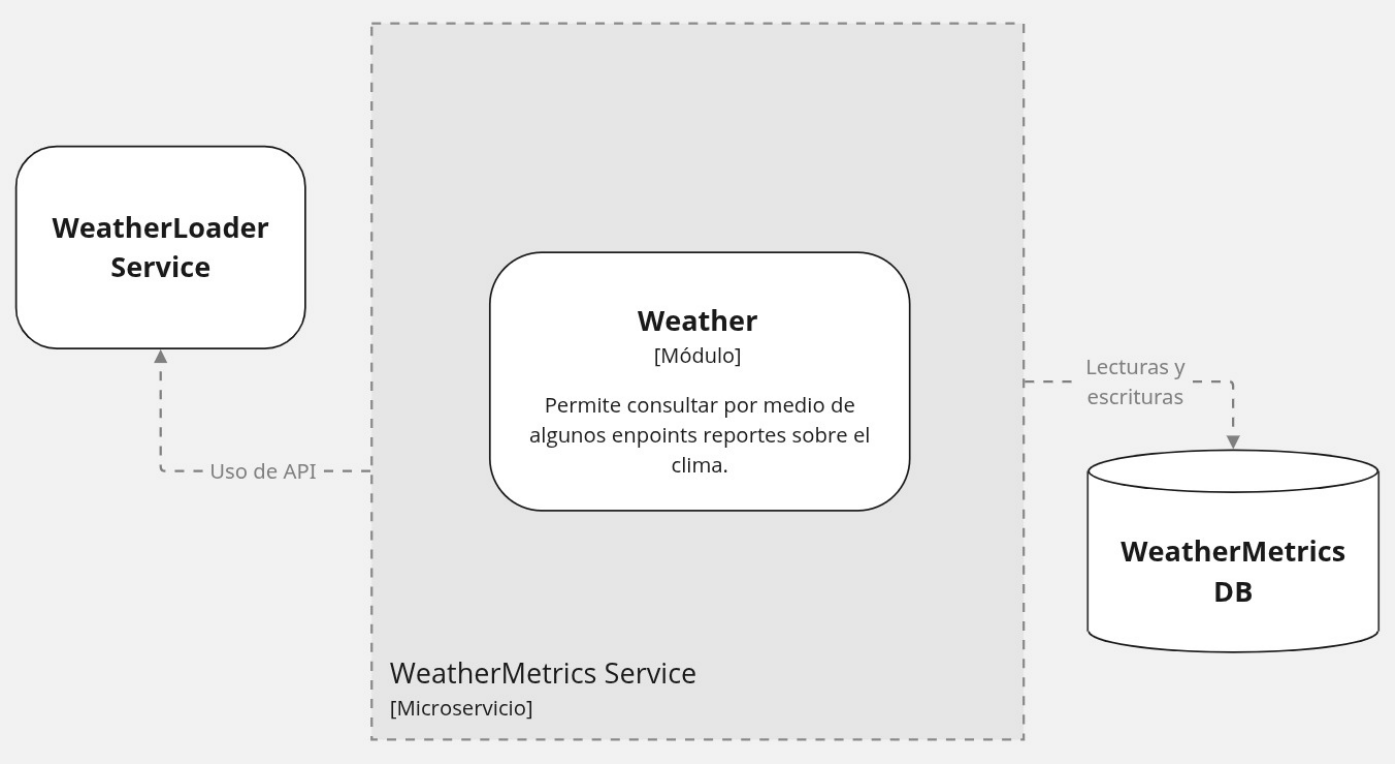
  Servidor de Nodejs. Se encarga de exponer los datos obtenidos desde el **WeatherLoaderComponent**.

  Comprendido por un módulo:

  - **Weather:** Obtiene los datos de **WeatherLoaderComponent** y los expone.

    Interfaz de comunicación: API [JSON/HTTPS]

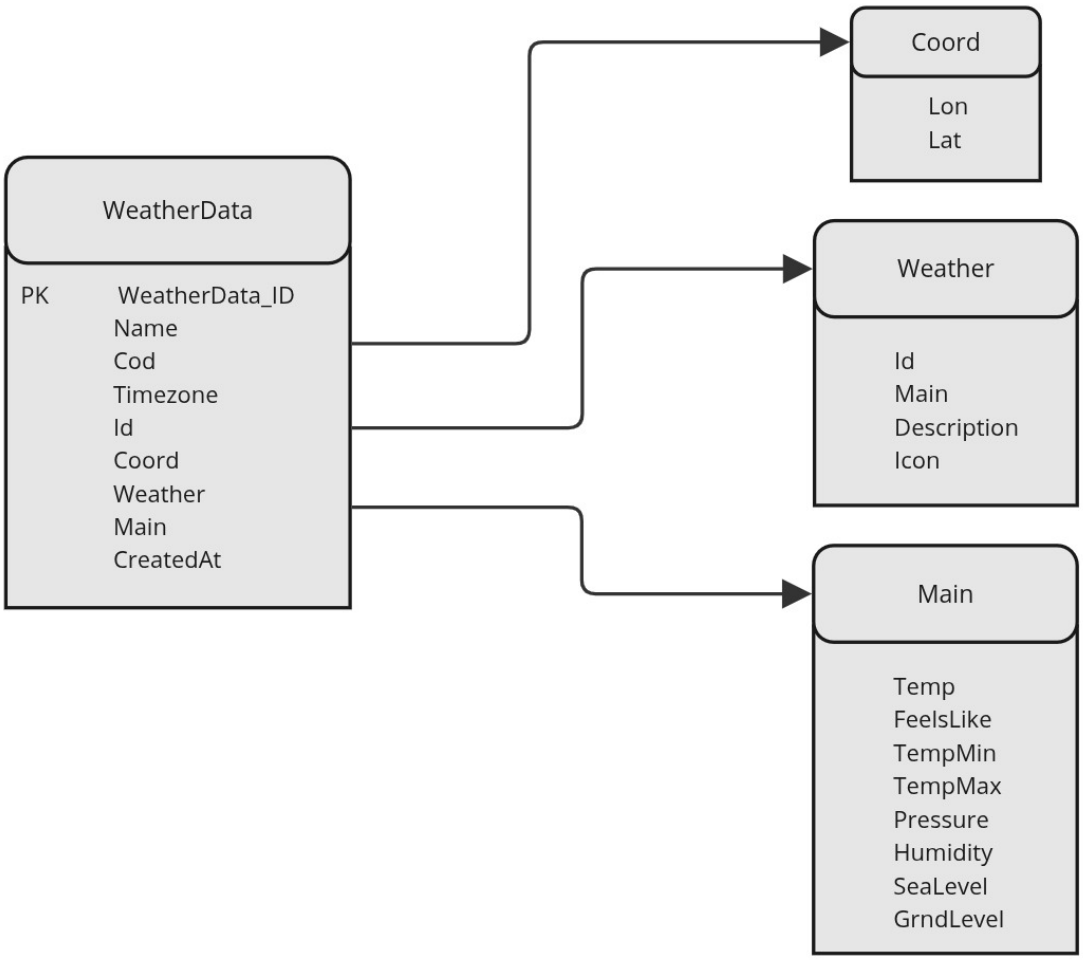
    * **GET /report/current-temperature?city=cityName** - Obtención de temperatura actual
    * **GET /report/last-day?city=cityName** - Obtención promedio temperaturas del ultimo dia
    * **GET /report/last-week?city=cityName** - Obtención promedio temperaturas de la ultima semana



4 Modelo de datos

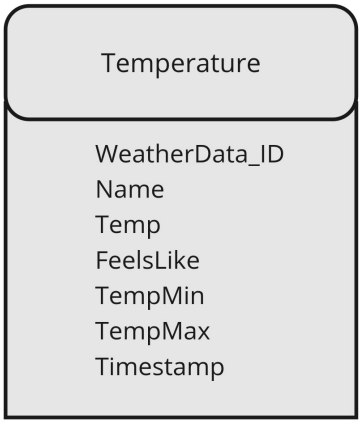
- **WeatherLoader DB:**

  Utilizaremos un tipo NoSQL (MongoDB). La elección de este tipo de base de datos es por su gran consistencia, facilidad de aprendizaje y en este caso nos ofrece el nivel de manejo de lecturas/escrituras que nos alcanza. Almacenaremos los datos que necesitamos, obtenidos desde la API externa, para después consumirlos de ser necesario.
  Este modelo de datos **WeatherData** se simplifica para cuando se expone por endpoints a **Temperature**.



- **WeatherMetrics DB:**

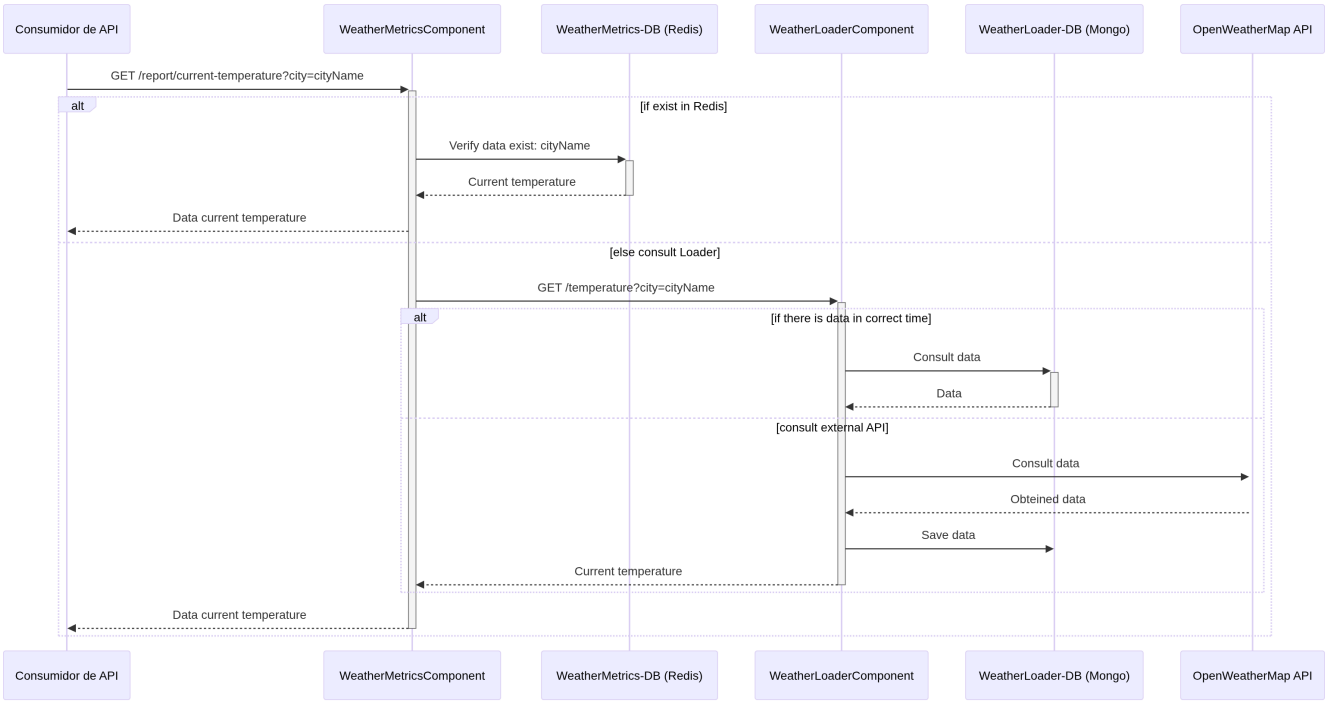
  Utilizaremos un tipo NoSQL clave-valor (Redis). La elección de este tipo de base es por su velocidad y eficiencia, y que también soporta múltiples estructuras de datos que posiblemente necesitemos guardar.
  Este modelo de datos **Temperature** es el expuesto por el **WeatherLoaderComponent**.



5 Diagramas de secuencia

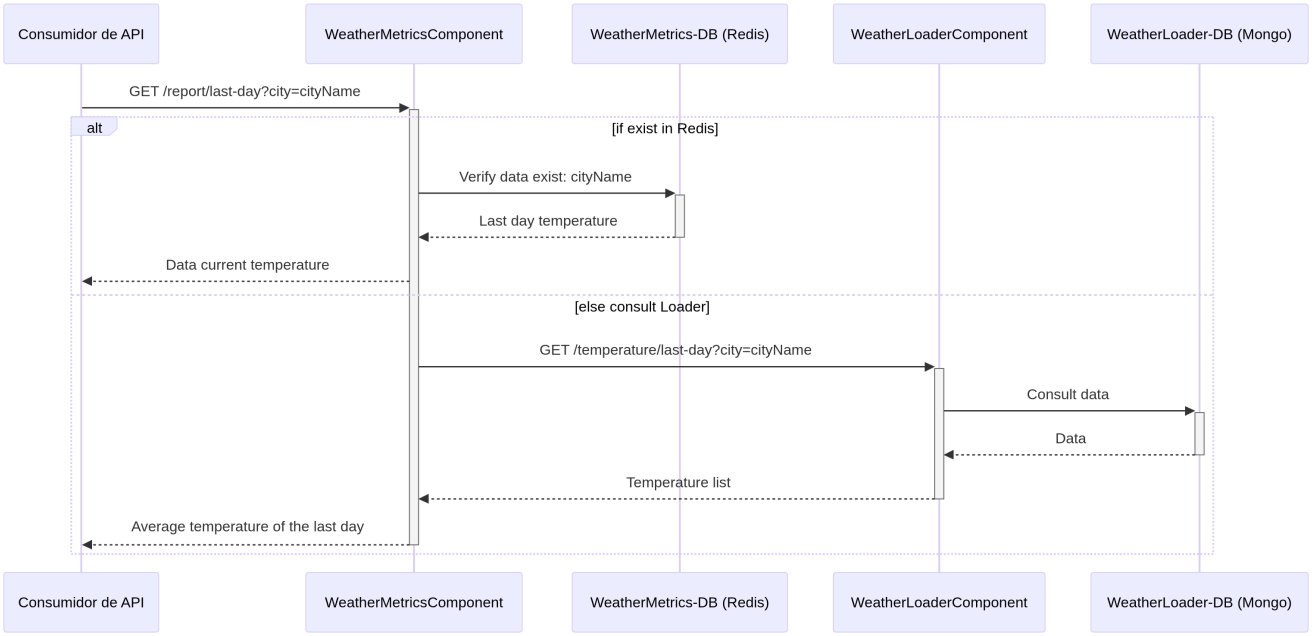
1. Reporte de la temperatura actual (CU-WMC-001)

   - El consumidor de la API consulta la temperatura actual
   - El WeatherMetricsComponent verifica si la consulta existe en cache:
     - Si existe se devuelven esos datos
     - Si no, se consulta al WeatherLoaderComponent
   - El WeatherLoaderComponent consulta en su BD (hay un tiempo configurable para esta verificación)
     - Si esta en tiempo, devuelve esos datos
     - Si no, se consulta a la API externa
   - En caso de consultar la API externa, al obtener los datos se guardan en DB y se devuelven al WeatherMetricsComponent
   - Se devuelven los datos de la temperatura actual al consumidor de la API



2. Promedio de la temperatura del último día (CU-WMC-002)

   - El consumidor de la API consulta el promedio de la temperatura del ultimo dia
   - El WeatherMetricsComponent verifica si la consulta existe en cache:
     - Si existe se devuelven esos datos
     - Si no, se consulta al WeatherLoaderComponent
   - El WeatherLoaderComponent consulta en su BD, devuelve la lista de temperaturas del ultimo dia
   - El WeatherLoaderComponent con los resultados calcula el promedio
   - Se devuelve el promedio al consumidor de la API



6 Estrategias de tolerancia a fallos

6.1 Time-out

- **WeatherLoaderComponent**: Configurado en axios al llamar a OpenWeatherMap.
- **WeatherMetricsComponent**: Configurado en axios para consultas al otro componente.

6.2 Circuit Breaker

En ambos componentes utilizamos **opossum** para envolver las llamadas, de esta manera se previene que el sistema falle repetidamente y de ser asi, tener conocimiento de ello.

6.3 Request cache

En el componente **WeatherMetricsComponent** utilizamos **Redis** como base de datos clave-valor. Para cada tipo de consulta se utiliza un clave de cache distinta, ademas de un TTL (Time-To-Live) acorde a la consulta. La estrategia seria **Cache-Aside**. El componente consulta primero la cache, si hay un cache miss, consulta al otro componente, actualiza la cache y devuelve el resultado.

7 Estrategia de observabilidad

7.1 Log aggregation

Para realizar el logging en nuestros componentes utilizamos **Pino**, el cual envia en formato JSON los logs a **Loki** por medio de **Promtail**. Logueamos peticiones HTTP (en WeatherExpressController), inicio/fin de tareas cron, datos cargados, errores en cualquier capa, llamadas a servicios externos (OpenWeatherMap, BD). Todo integrado para verse en **Grafana**.

7.2 Metrics aggregation

Utilizamos el cliente de **Prometheus** (prom-client) para exponer un endpoint en nuestros componentes y asi poder scrapear los datos para mostrarlos en **Grafana**.

7.3 Distributed tracing

Usamos **OpenTelemetry** y sus herramientas automáticas para Nodejs. Se exporta a **Tempo** que esta también integrado a **Grafana**.

7.4 Alerting

Nuestras herramientas en este caso son:

- **Prometheus**: Recolecta todas las métricas de las aplicaciones y de la infraestructura.
- **Grafana alerting**: Un motor de alertas nativo de Grafana.
- **Canales de notificación**: Grafana puede enviar las alertas a múltiples destinos: como Slack o Telegram.

Para evitar sobrecarga de alertas (recibir tantas notificaciones que terminas ignorándolas), nos vamos a basar en estos 3 puntos:

- Alerta sobre síntomas, no causas: Es mejor alertar sobre lo que el usuario experimenta (ej. "la API está lenta o devuelve errores") que sobre la posible causa (ej. "el uso de CPU es del 80Un alto uso de CPU solo es un problema si causa un impacto real.

- Cada alerta debe ser accionable: Si recibimos una alerta, se debe saber qué hacer. Si no hay una acción clara, la alerta es solo ruido.
- Niveles de severidad:
  - **warning** (advertencia) para algo que no está bien y podría convertirse en un problema.
  - **critical** (Crítico): El servicio está fallando o a punto de fallar, entonces requiere acción inmediata.

Pasamos a listar algunas alertas que pensamos clave para cada uno de los componentes:

- **WeatherLoaderComponent**

- Tasa de errores alta (critical):
  Mide el porcentaje de peticiones a la API que están fallando (códigos HTTP 5xx), es el indicador mas claro de que el servicio esta roto. Mayor al 5% de errores durante 2/3 minutos podría ser la condición adecuada para la notificación.
- Latencia alta (warning):
  Mide el tiempo que tarda la API en responder, específicamente el percentil 95 (p95). Para notificar nuestra condición seria si la latencia p95 es mayor a 2 segundos durante 5 minutos.

- **WeatherMetricsComponent**

- Datos desactualizados (critical):
  Se mide si ha pasado demasiado tiempo desde la última vez que se cargaron datos exitosamente. Se lanza alerta cuando la diferencia de tiempo es 2 veces mayor a la que tenemos configurada de consulta.
- Fallos consecutivos en la carga (warning):
  Mide si el job de carga de datos falla varias veces seguidas. Mas de 3 fallos dentro de 1 hora, seria una alerta adecuada.
- Circuit Breaker Abierto (warning):
  Medimos el tiempo en el cual el circuito esta abierto, si este estado dura mas de 5 minutos se notificaría.

También se pueden agregar a otros niveles como **alertas de infraestructura**:

- Contenedor caído (critical):
  Si Prometheus no puede "scrapear" métricas de uno de los contenedores durante 1 minuto o mas se notifica que el contenedor esta caído.