PROYECTO FINAL AEMET

Desarrollo de Aplicaciones para la Ciencia de Datos

UNIVERSIDAD DE LAS PALMAS DE GRANCANARIA
ESCUELA DE INGENIERÍA INFORMÁTICA
2º curso en el Grado de Ciencia e Ingeniería de Datos

Susana Suárez Mendoza

14/01/2023

Versión 1.1

Resumen.

Este proyecto tiene como objetivo poner en práctica el diseño de una arquitectura lambda estudiada en clase, con un datalake para registrar eventos y un datamart vinculado a una APIRest para realizar consultas.

Consta de 3 módulos:

1. Módulo “feeder”. Este módulo es el encargado de descargar cada hora los datos de todas las estaciones meteorológicas de Gran Canaria desde el webservice de la AEMET y guardarlos en una carpeta llamada “datalake” con ficheros de eventos diarios.
2. Módulo “datamart”. Este módulo es el encargado de crear y alimentar un datamart a partir de los datos contenidos en el datalake con las temperaturas máximas y mínimas de cada día en la isla. En la base de datos llamada “datamart.db” se crearán dos tablas, una para la temperatura máxima y otra para la mínima, en las que se guardarán: fecha, hora, lugar, identificador de la estación y la temperatura.
3. Módulo “apiRest”. Este módulo es el encargado de ofrecer una API REST para realizar consultas de los sitios con mayor temperatura y menor temperatura de la isla en un rango de días con las siguientes rutas:
    a. GET/v1/places/with-max-temperature?from={date}&to={date}
    b. GET/v1/places/with-min-temperature?from={date}&to={date}

Índice

1. Recursos utilizados.

❖ Entornos de desarrollo.

Para la implementación de toda la funcionalidad del proyecto se ha utilizado el entorno de desarrollo integrado, perteneciente a JetBrains, IntelliJ IDEA Edu utilizando el lenguaje de programación Java.

❖ Herramientas de control de versiones.

Las herramientas de control de versiones son herramientas de software que ayudan a gestionar los cambios en el código fuente a lo largo del tiempo. La herramienta utilizada en este proyecto ha sido GIT y los cambios son alojados en la plataforma Github.

En este caso, solo se podrá ver una rama de desarrollo ya que es la primera versión de nuestra aplicación. Sin embargo, en esta rama se podrán ver todos los cambios y desarrollo de los diferentes módulos planteados.

❖ Herramientas de documentación.

Para poder implementar algunas funciones ha sido necesario documentarse previamente con las siguientes herramientas:

1. Timer Task → Java.util.TimerTask class in Java - GeeksforGeeks
2. API Rest de AEMET → AEMET OpenData
3. Información contenida en el PDF situado en el campus.

2. Diseño.

❖ Arquitectura del sistema.

Para la resolución de este sistema se ha diseñado una arquitectura lambda con un feeder de datos de entrada correspondientes a AEMET, un datalake como almacenamiento por lotes, un datamart alimentado del datalake y una API REST de consulta de los datos contenidos en el datamart.

❖ Principios de diseño.

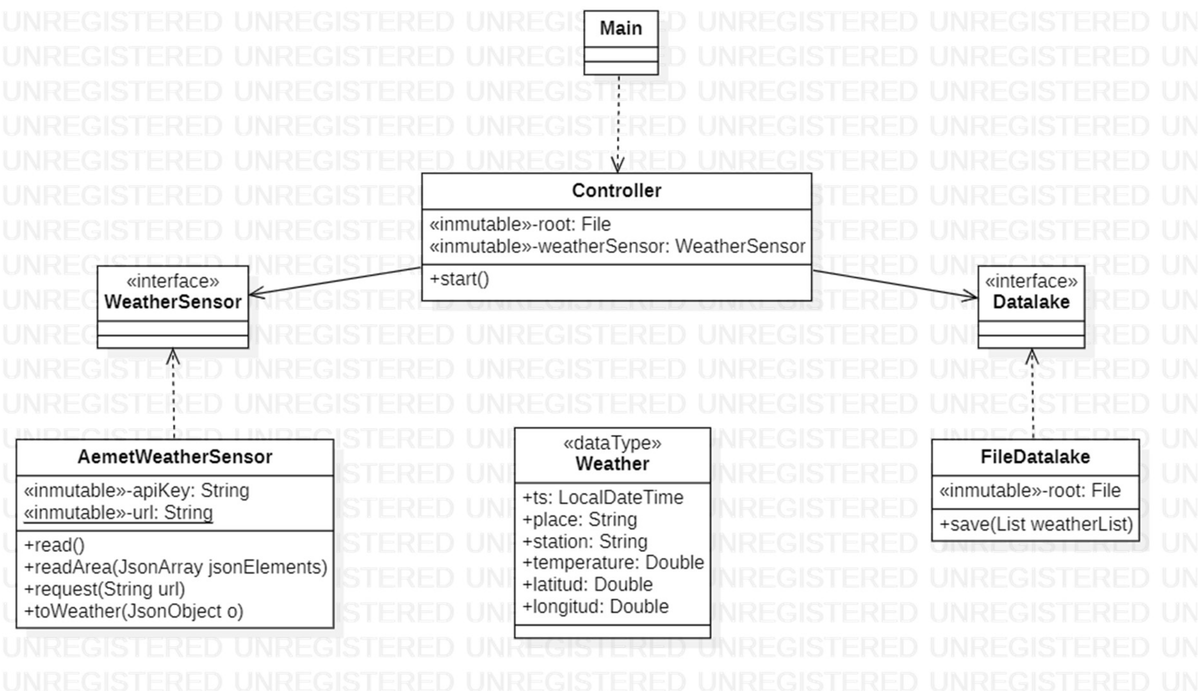
- Responsabilidad única. Cada clase de nuestro trabajo tiene una única responsabilidad. Por ejemplo, tenemos una clase que sólo es responsable de pasar a formato JSON los datos para poder ser distribuidos a través de la API.

- Abierto a la ampliación, cerrado a la modificación. Debido a la gran cantidad de interfaces podemos ampliar nuestro dominio y además de los datos de AEMET, tomar más datos de entrada de diferentes fuentes, crear bases de datos en otros lenguajes o incluso diferentes API REST para su consulta.

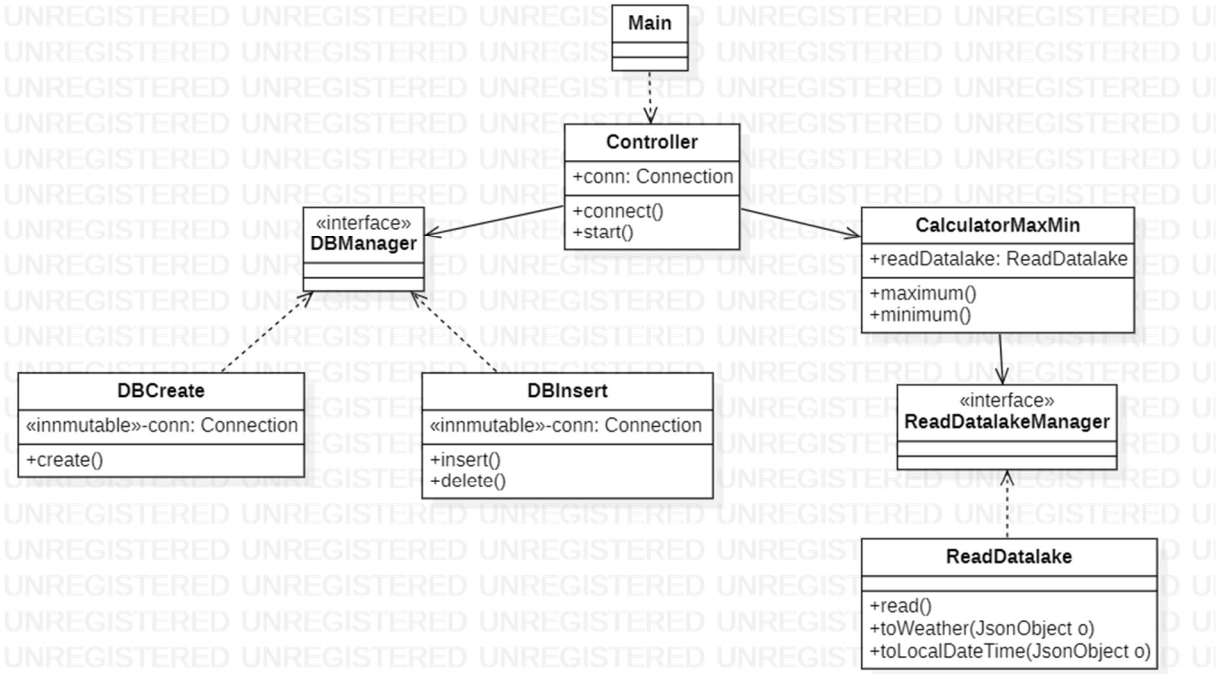
- Inversión de dependencia. Las clases de bajo nivel no dependen de clases de alto nivel sino de interfaces.

❖ Diagramas de clases.

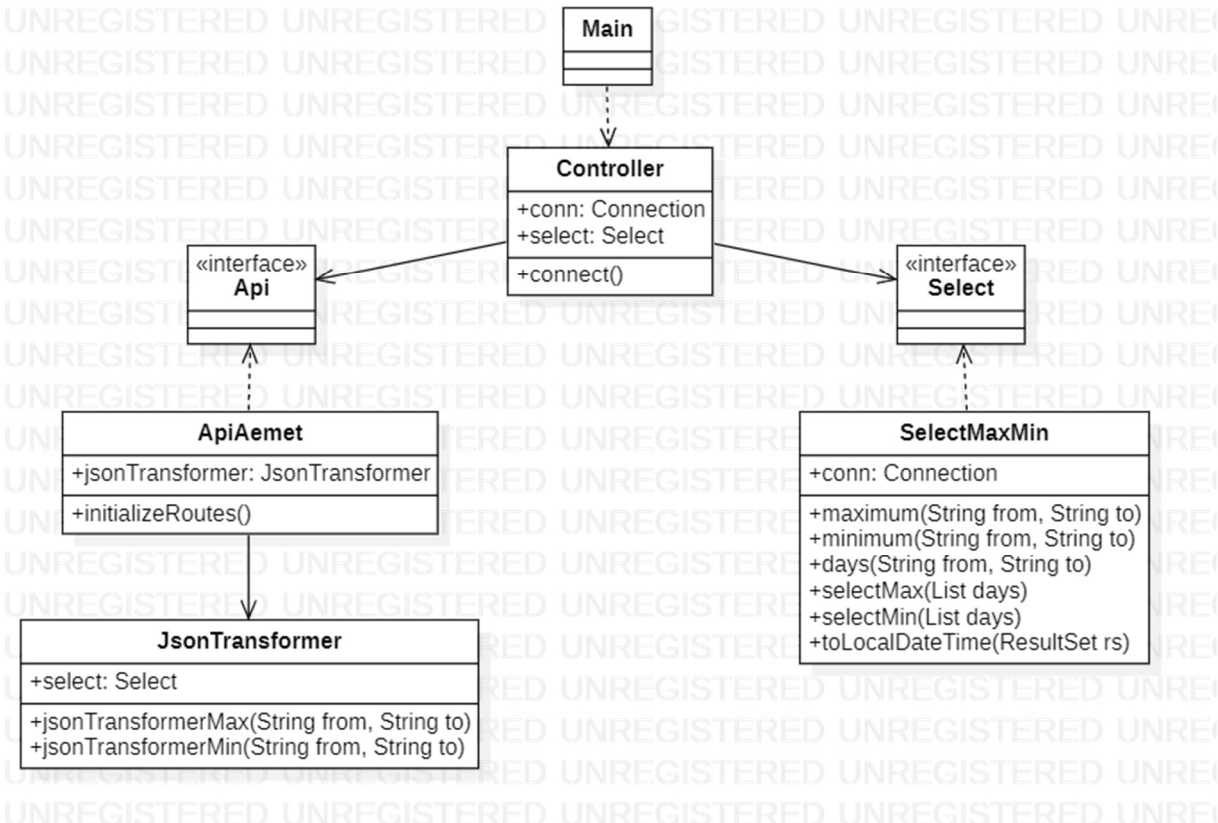
- Feeder.



- Datamart.



- Api.



3. Conclusiones y anotaciones.

❖ Lecciones aprendidas.

Como aprendizaje me llevo lo siguiente:

- Conocimientos sobre los datos en abierto proporcionados por a AEMET.
- Creación de un directorio datalake y creación de archivos de eventos.
- Construcción de una API Rest con parámetros query.

❖ Anotaciones por módulos.

- Feeder.

En primer lugar, para el funcionamiento de dicho módulo es necesario pasar por parámetros la apiKey dada por AEMET, sino será imposible la obtención de los datos a través del método GET.

En segundo lugar, La API porporcionada por AEMET devuelve todos los datos diarios desde las 0:00 hasta la hora actual por lo que debido a este formato he optado por guardar todos los datos horariamente en vez de actualizar ya que así optimizamos el tiempo del programa y evitamos las comprobaciones horarias para ahorrar espacio en tiempo de ejecución con los diversos métodos correspondientes.

En el caso que el programa tuviese que actualizar los datos horariamente bastaría con hacer un método que tome la información contenida y la nueva de llegada y sólo escriba en el fichero aquella que no esté escrita.

- Datamart.

Para la ejecución correcta de este módulo es necesario haber ejecutado previamente el módulo anterior ya que si no existe el fichero diario será imposible obtener esos datos para poder hacer los cálculos planteados en la base de datos.

Además, este módulo se ejecuta horariamente como el feeder salvo que tiene un minuto de delay puesto que así da tiempo a que el feeder obtenga los nuevos datos previamente.

- Api Rest.

Es importante tener en cuenta que esta API está proporcionando datos del datamart por lo que, si las fechas no están disponibles en nuestra base de datos, tampoco lo estarán en nuestra API. Por tanto, esta API está pensada para su ejecución en dos o tres días después de la obtención de los diferentes eventos en el datalake y datamart.

4. Líneas futuras.

Este producto puede evolucionar de la siguiente manera:

En primer lugar, la obtención de datos se podría llegar a registrar eventos no sólo de AEMET sino de otros sensores de la empresa o incluso de otras API del tiempo de Canarias para poder comparar los resultados de dichos sensores.

Por tanto, podríamos crear directorios dentro de nuestro datalake correspondientes a cada fuente de datos para poder clasificarlos.

En segundo lugar, el datamart podría tener diferentes tablas combinando todos los dataos de las fuentes. Por ejemplo, además del registro mínimo y máximo podría tener una tabla más que nos indique la media diaria o incluso la realización de predicciones temporales futuras para los siguientes días y a medida que se obtengan, ir comparando si las predicciones eran correctas.

En tercer lugar, la API podría mostrar no solo la máxima temperatura o mínima, sino que, gracias a lo dicho anteriormente en el datamart, podría visualizar las predicciones en cada lugar.

Por último, todo lo anterior se podría comercializar a un usuario final a través de una interfaz gráfica en la que se muestren gráficas sobre predicciones de lluvia, temperaturas, etc, para agricultores y así poder llevar mejor un control sobre sus cultivos y ahorrar recursos de agua.