Desarrollo de Aplicaciones para Ciencia de Datos

2º Grado en Ciencia e Ingeniería de Datos

Universidad de las Palmas de Gran Canaria, EII

DEVELOPING A DATALAKE WITH AEMET DATA

Ricardo Juan Cárdenes Pérez

RICARDO JUAN CÁRDENES PÉREZ

12/01/2023

ÍNDICE

RESUMEN

En esta memoria se explica brevemente el funcionamiento, diseño y arquitectura de un programa que se encarga de recolectar datos relacionados con la temperatura de diversas estaciones meteorológicas españolas y, pasando por una fase de construcción de un DataMart especializado, es capaz de proporcionar a los usuarios mediante una API Web información de las temperaturas máximas y mínimas en Gran Canaria a lo largo de los días.

RECURSOS UTILIZADOS

La plataforma escogida para llevar a cabo el desarrollo de este proyecto a sido IntelliJ por todas las facilidades que nos brinda a la hora de programar, además de permitirnos trabajar con el control de artefactos Maven. Este proyecto logra su correcto funcionamiento gracias a tres dependencias importadas, entre las cuales se encuentran las siguientes:

- JDBC-SQLite: esta dependencia hace posible la conexión con la base de datos principal del programa.
- Spark: nos permite crear una API web en nuestro servidor local LocalHost.
- Gson: nos permite obtener la serialización en formato JSON de cualquier objeto que queramos.

Este proyecto utiliza Git como control de versiones, y cuenta con un repositorio público en GitHub al cual puede accederse mediante el siguiente enlace

https://github.com/ricardocardn/Aemet

DISEÑO

Este proyecto consta de tres módulos o aplicaciones independientes, las cuales no saben de la existencia la una de la otra, pero que trabajan de forma cooperativa para recoger, filtrar y mostrar los datos proporcionados por la API de Aemet a nuestros usuarios. Estos tres módulos que en conjunto dan una funcionalidad útil a nuestra aplicación son los siguientes:

- AemetFeeder: Este módulo, como su propio nombre indica, es el que nos proporciona los datos recogidos por Aemet y ofrecidos a nosotros por su API. Es el encargado de guardar todos los eventos de tiempo, en nuestro código llamados WeatherEvent(s), por días en nuestro DataLake. Es el encargado, además, de filtrar los datos de entrada para guardar en nuestro sistema de información únicamente aquellos datos tomados con sensores situados en la isla de Gran Canaria.

  Esta tarea asignada al AemetFeeder se realiza cada hora, gracias al uso de las clases Java TimerTask, que permite definir una tarea a ejecutarse, y Timer, que permite ejecutar la tarea en cuestión cada cierto tiempo.

PARA QUE ESTE MÓDULO FUNCIONE ES IMPORTANTE APORTAR SU APIKEY DE LA API AEMET COMO ARGUMENTO AL MÉTODO MAIN DEL MÓDULO

Nota: la API de Aemet, al hacer una petición GET, nos devuelve todos los eventos tomados desde que comenzó el día hasta la hora actual. Por ello, cada vez que accedemos y tomamos datos de la API borramos el contenido ya existente de ese día del DataLake e insertamos el que acabamos de obtener, pues estos ya contienen los datos de horas anteriores. Si quisiera guardarse lo anterior, bastaría con pasarle true al instanciar el objeto fileWriter que escribe en el DataLake mediante su parámetro append del constructor y no se sobre escribirían los datos. No obstante, en el enunciado de la práctica no se especifica la estructura interna de los ficheros del DataLake, por lo que se supone que existe libertad en la forma en la que se guarda esta información siempre y cuando se satisfagan los propósitos principales de la práctica.

- AemetDM (de AemetDataMart): Esta es la parte de nuestra aplicación encargada de generar un DataMart con los datos previamente guardados en el DataLake. Este módulo, periódicamente (cada hora), se encarga de actualizar la información de nuestro DataMart a medida que se insertan nuevos datos en el DataLake. El sistema de información utilizado para esto consiste en una base de datos relacional con tecnología SQLite. Dentro de esta base de datos encontramos las tablas MaxTemp y MinTemp, las cuales disponen de los eventos con máxima y mínima temperatura en Gran Canaria por día, respectivamente.

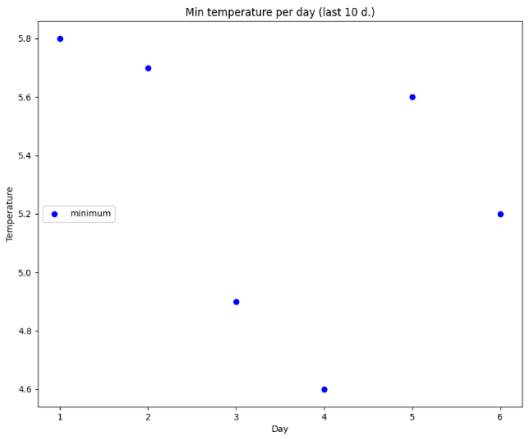
- API: Esta es la única de las tres aplicaciones que tendrá interacción alguna con el usuario final al que va dirigida nuestro proyecto. Este módulo se encarga de desplegar un servidor mediante Spark el cual ofrece una API RESTFul para que los usuarios puedan consular (únicamente) los datos del DataMart. Esta API ofrece respuestas en formato JSON o HTML, en función del path utilizado. Para obtener el formato JSON en las respuestas, basta con usar los path definidos en el enunciado de esta práctica. Mientras, para visualizar los datos en versión HTML, basta introducir /html después de la versión de la API a utilizar. Así, para obtener todas las temperaturas mínimas tomadas hasta el día de hoy, bastaría con introducir en el navegador el siguiente enlace:

  http://localhost:8086/v1/html/places/with-min-temperature?to={date}

  Siendo date la fecha de hoy y estando escrita en el formato dd-MM-yyyy.

- Stats: este se trata de un módulo desarrollado en el lenguaje de programación Python que incluye un programa sencillo para generar gráficos con las mediciones máximas y mínimas de los últimos 10 días medidos. Un ejemplo

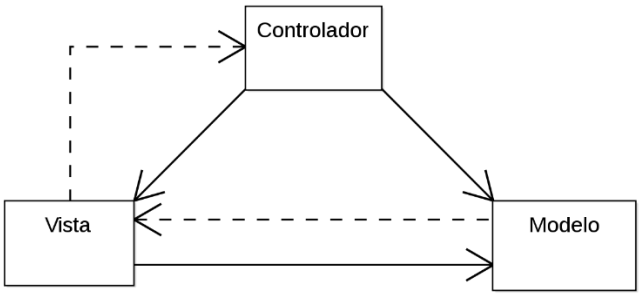
podría ser el siguiente, que nos muestra las menores temperaturas de los días de los cuales disponemos datos:



Para este proyecto hemos pensado en optar por el uso de un **estilo arquitectónico** que divide la lógica de la aplicación en dos paquetes para cada una de las tres aplicaciones: Model y Controller.

- Model: este es el componente central de nuestro arquitectura. Contiene la representación abstracta de la información con el que el sistema opera.
- Controller: dedicado a responder a eventos, ya sean internos o provocados por el usuario a partir de la interacción con la API RESTFul, e invoca peticiones al modelo cuando se solicita alguna información.

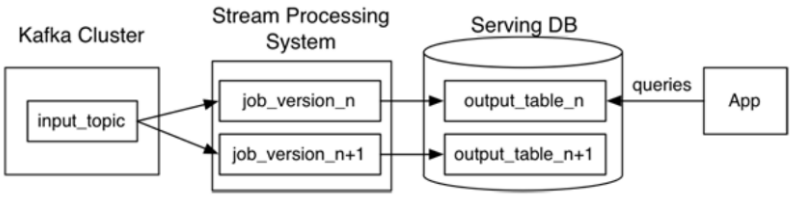
En esta arquitectura, el paquete de control depende del modelo pero jamás el modelo del control. En el módulo de la API se incluye, además, un paquete llamado web, que actuaría como el paquete View y ajustándose, por tanto, a la arquitectura MVC (Model View Controller).



En cuanto a la **arquitectura del sistema**, este sistema está montado mediante una arquitectura Kappa. Pese a que de la API esté disponible al usuario en tiempo real, el sistema no es reactivo a la entrada de datos mediante la API Aemet, pues espera una hora desde la última toma de datos para obtener datos nuevos. Además, estos

datos o eventos que obtenemos son, aunque muy rápidamente, procesados por lotes. Por ello, y porque todos los flujos de datos de la aplicación atraviesan una misma ruta de acceso, decimos que la arquitectura implementada no es Lambda, sino Kappa.

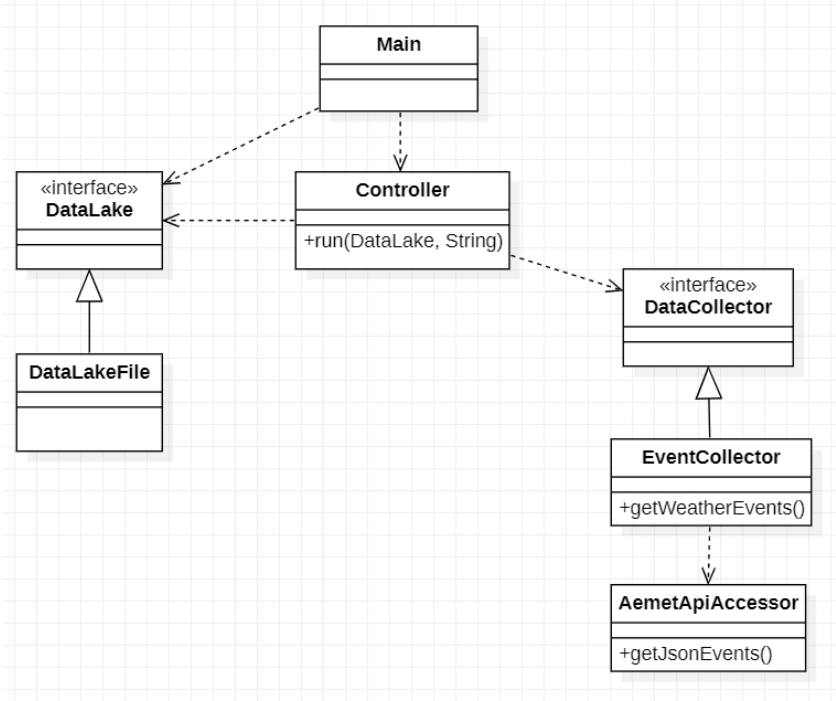
Ej. de arquitectura kappa:



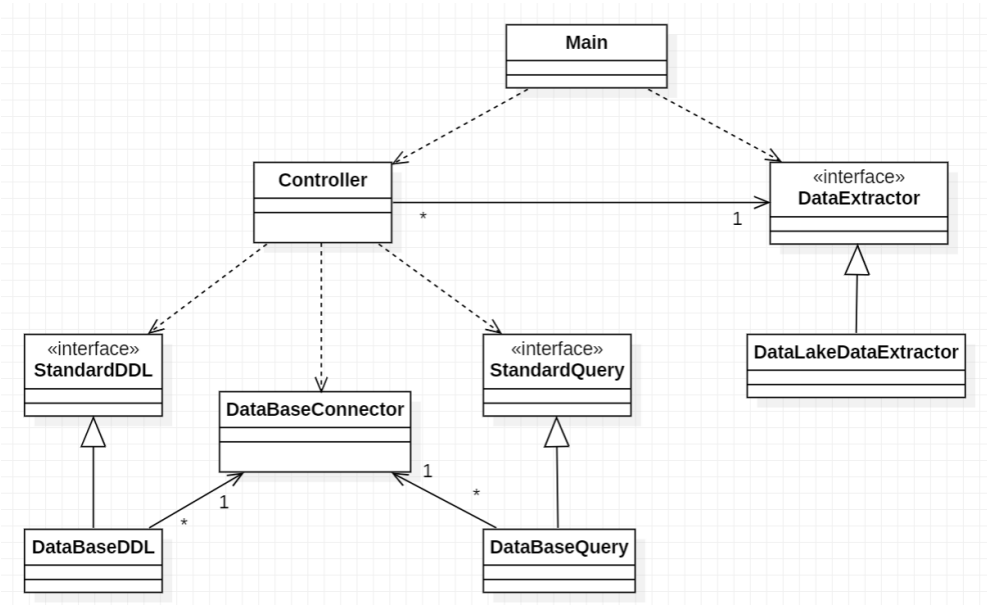
Principios de Diseño

En las siguientes imágenes podemos visualizar de forma gráfica como están organizadas las clases de cada una de las aplicaciones, mediante diagrama de clases UML.

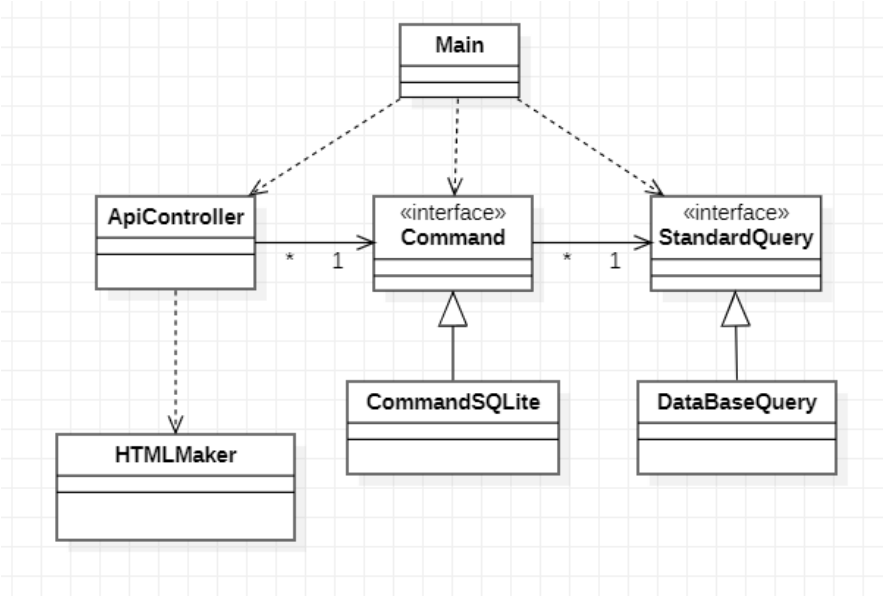
- **AemetFeeder**



- **AemetDM**



- **API**



Bastaría con observar el nombre de las clases para ver que cumplimos correctamente el principio SOLID de responsabilidad única, pues existe una clase para cada función que realiza nuestra aplicación. Por ejemplo, existen tres clases para gestionar la base de datos, de las cuales una se encarga de crear la conexión, otra de crear e insertar datos en las tablas y otra de realizar consultas a las mismas.

Se cumple además el principio de sustitución de Liskov, pues utilizamos interfaces para todas aquellas partes del programa que puedan ser modificadas o incluso cambiadas por otras en la vida útil del mismo. De esta forma, si cambiamos de sistema de base de datos nos bastará con crear dos nuevas clases, una para la inserción y creación de datos y otra para la consulta de ellos, las cuales implementen las interfaces *StandardDDL* y *StandardQuery*, respectivamente.

También podemos apreciar cómo se satisface el principio de inversión de dependencia, pues clases abstractas no dependen de clases de bajo nivel, como es el caso de la clase Controller del módulo AemetDM.

El paquete modelo de cada módulo únicamente incluye una clase, WeatherEvent o TempEvent, que representa un evento de tiempo y que, pesa a por lo general ser los eventos objetos inmutables, hemos creado como mutables por diversas razones de desarrollo.

Conclusiones

Me ha parecido un trabajo muy interesante en el que sin duda hemos puesto en práctica todos los conocimientos que hemos ido adquiriendo a lo largo de la asignatura. Me pare super interesante como podemos crear tres aplicaciones independientes las cuales trabajan en armonía para proporcionar una funcionalidad realmente útil y que, además, esas aplicaciones puedan estar programadas en lenguajes totalmente distintos.

Líneas Futuras

Como líneas futuras, me gustaría publicar este servicio web en internet, de forma que cualquier usuario pueda acceder a él desde cualquier parte. Una forma de llevar esto acabo puede ser creando un túnel HTTP entre el servidor local e internet. Una forma de llevar esto acabo es usando la tecnología Ngrok. Para ello, basta con descargarse la aplicación desde su sitio web principal y ejecutarla en la terminal con el comando

/user/{carpeta donde se encuentra ngrok.exe}> ngrok http {port}

Una vez ejecutemos, nos aparecerá lo siguiente en la terminal

Donde podemos ver como nos proporciona una url pública mediante la cual podemos acceder a nuestra API desde internet , con la única condición de que en el ordenador donde se ejecute el comando ngrok se haya montado una API en el servidor localhost con el puerto 8086, en este caso.

Me gustaría también aprender a usar AWS para de esta forma poder ejecutar directamente mi aplicación en una JVM ya desplegada en un servidor conectado a internet. De esta manera, no tendría que estar ejecutando la aplicación en un ordenador

concreto para que se pueda acceder a la API, pues ya se está ejecutando en una Java Virtual Machine de Amazon.

Me gustaría también crear una parte de la API que fuera privada. Para ello he visto que existen algoritmo de autenticación por tokens bastante seguros como OAuth2, pero tengo que ver como implementarlo con Spark.

Bibliografía

- https://docs.oracle.com/javase/7/docs/api/
- https://opendata.aemet.es/dist/index.html
- https://sparkjava.com/documentation
- https://www.sqlitetutorial.net/sqlite-java/
- https://mvnrepository.com/repos/central

Ricardo Juan Cárdenes Pérez

Universidad de las Palmas de Gran Canaria