PROYECTO FINAL: AEMET

Desarrollo de Aplicaciones para Ciencia de Datos (DACD)

2º GRADO DE CIENCIA E INGENIERÍA DE DATOS (GCID)
ESCUELA DE INGENIERÍA INFORMÁTICA (EII)
Universidad de las Palmas de Gran Canaria (ULPGC)

Autor

José Gabriel Reyes Rodríguez

Fechas de desarrollo del proyecto

desde el 31-12-2022 hasta el 13-12-2022

Versiones

1

Revisiones de la memoria

0 revisiones (por tutoría)

4 revisiones (propias)

Índice:

Resumen:

Para este trabajo se han tenido que realizar 3 módulos diferentes, estos son:

El módulo Feeder: Este módulo se encarga de recoger los datos de Aemet y guardarlos en un datalake de ficheros ".events", en donde cada evento contiene la fecha en formato LocalDateTime, la ubicación del sensor, el identificador del sensor, la temperatura del aire, la temperatura máxima y la temperatura mínima. Estos datos se guardan en formato JSON. Este módulo contiene una interfaz llamada WeatherSensor junto con un a implementación de la misma llamada AemetSensor, la cual tiene la misión de leer los datos provenientes de AEMET, devolviéndolos en una lista de objetos de la clase del modelo Weather, también posee una interfaz lamada DataLake que se encarga de las operaciones propias del manejo del datalake como lo es escribir en él y obtener los datos del mismo. En la implementación hecha, seescriben todos los datos que no están presentas en los ficheros tanto del día de hoy como del día de ayer.

El módulo Datamart: Este módulo se encarga de construir la base de datos con sus respectivas tablas para la temperatura máxima y la temperatura mínima. Para la base de datos tengo que recalcar que la forma en la que implementé la misma es basándome en el pensamiento de que una base de datos está compuesta por tablas y por tanto, la interfaz de la base de datos solo contiene 2 operaciones, las cuáles son obtener las tablas pedidas en el problema y por tanto, en la implementación de esta interfaz, la clase SqliteDataBase tiene un constructor en donde se crean estas tablas en el constructor de la base de datos y aparte de las operaciones de la interfaz también se encarga de ofrecer un método para obtener la conexión a la base de datos. Esta stabals cuentan a su vez con una interfaz con las operaciones para crear la tabla, insertarle los datos y obtener los datos de las tablas. De esta manera, mi intención es que, si se llegase a necesitar una nueva tabla, se crea una nueva implementación de la interfaz y añadirla al constructor e la base de datos junto con su getter. Esto lo hice porque previamente tenía todo en una misma clase y me resultaba difícil su mantenimiento y de esta manera me parecía más fácil, además de que así conseguí que, por ejemplo, para hacer un insert no fuese necesario pasarle como parámetro la dirección de la base de datos ya que esta dirección es responsabilidad de la base de datos y no de las propias tablas (ya que veo grosso modo que una base de datos es como un almacén de tablas). Además, se ha implementado una clase llamada DatabaseTablesGetter que se encarga de encapsular las tablas de la base de datos para que, en el caso de hacer una nueva tabla, no se vea afectada la interfaz Database y por ende las implementaciones de esta. La responsabilidad de añadir esta nueva tabla pertenecería a la clase DataBaseTablesGetter y actuaría como puente.

El módulo TemperatureService: Este módulo se encarga de la APIRest, en donde tiene una clase modelo WeatherForApi que nos hace de puente entre el modelo Weather anterior con más datos de los necesarios para nuestras peticiones a objetos Weather con solo la información necesaria para suplir la petición en JSON, en donde a las peticiones se le sdevuelve una lista de JSON Strings, uno por día. Cuenta con un controller para inicializar este proceso en caso de querer añadirse nuevas peticiones de otro tipo que no sean "GET".

También cabe destacar lo siguiente:

-El módulo Datalake para ejecutarse necesita como parámetro la APIKey

- Si se abre el proyecto desde el IntellIJ ya cuenta con una Run Configuration con los tres módulos y el argumento (La APIKey) añadida en la configuración (se llama RunProjectV2). La APIKey se pasa como argumento del programa.

-Importante: para poder usar el RunProjectV2 hay que cambiar los directorios de los artefactos (el "working directory" y el "Path To JAR" para ajustarlo a su dirección cambiando el "C:\Users\Gabri\IdeaProjects"). No he podido encontrar una manera para hacer que se creen los artefactos con una dirección relativa.

-En vez de un TimerTask empleo un ScheduledExecutorService(documentación en al biobliografía) para hacer que el programa se ejecute cada hora y además el módulo Datamart tiene un delay de 16 segundos ya que el Feeder se acaba construyendo después del datamart en algunas ejecuciones y me parece más interesante si el Feeder se ejecuta antes que el Datamart sea cual sea el orden de ejecución a la hora de construir los módulos.

-Durante la ejecución si un módulo ya ha terminado (Feeder o Datamart) se hace un SOUT para indicar su finalización (desconozco si es contraproducente, pero durante el desarrollo me pareció útil).

-También me gustaría mencionar que el proyecto lo subí tarde al GitHub y hay commits desde que el trabajo ya funcionaba.

Recursos utilizados

Entornos de desarrollo

Intellij IDEA versión educativa edición 222.4167.41

Herramientas de control de versiones

Git

Herramientas de documentación

Microsoft Word.

Extras:

Talented API tester

Diseño

En cuanto al diseño, el módulo Feeder se basa en un MVC en donde el modelo estaría recogido en el AemetSensor y el Weather ya que obtienen los datos de AEMET en forma de “Weather”, la vista sería el FileDataLake ya que en él se pueden ver los eventos recogidos de AEMET y los guarda, y el controlador sería el Controller, ya que tiene la ejecución del programa con el ScheduledExecutorService.

En el módulo Datamart seguiría un MVP ya que se encarga de preparar los datos del Datalake para su uso por el TemperatureService haciendo uso del modelo del DataLake.

En el módulo TemperatureService, seguiría un MVC ya que tiene su propia clase WeatherForApi para conseguir un objeto más preciso para nuestras necesidades, la vista sería la implementación de la API en Spark y el controlador el Controller que inicializa la API.

Como conjunto el proyecto seguiría un MVP en donde:

El modelo sería el Feeder ya que recoge y guarda todos los eventos recogidos de AEMET.

La vista sería el Datamart, en donde se preparan los datos para su fácil acceso y en donde se puede apreciar los resultados que serán usados por el TemperatureService.

Y por último el presentador sería el TemperatureService ya que establece bajo que peticiones y formato serán accesibles los datos del Datamart para el usuario.

Por otra parte, también consideraría correcto verlo como un MVC en donde el modelo seguiría siendo el DataLake pero la vista sería el TemperatureService ya que es la manera que tiene el usuario de ver la información recogida en la base de datos (Datamart) y el controller sería el Datamart ya que se encarga de preparara los datos del DataLake para su posterior consulta por el TemperatureService.

Diagrama de clases del Feeder(imagen original en el repositorio de Github):

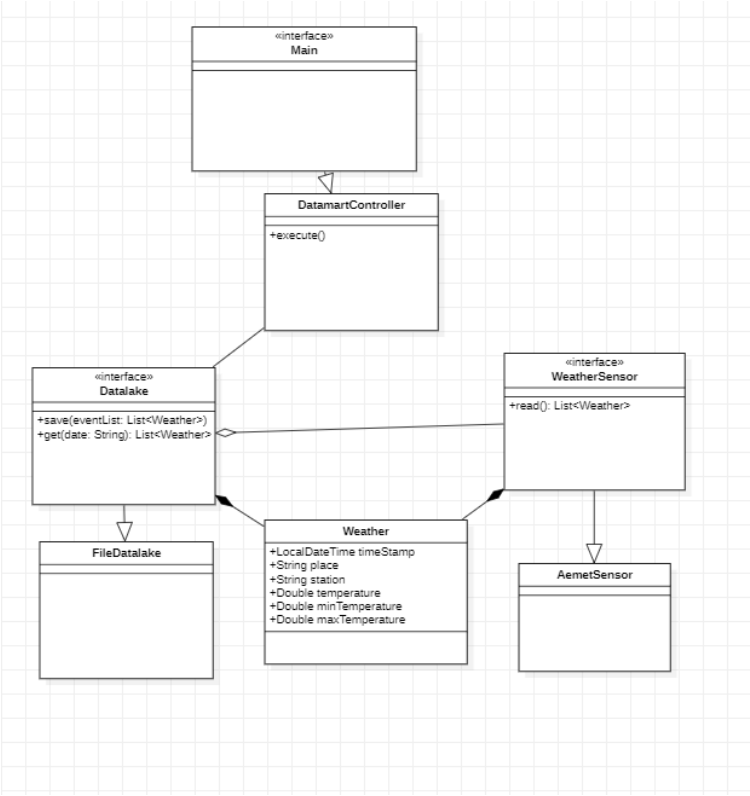


Diagrama de clases del Datamart(imagen original en el repositorio de Github)

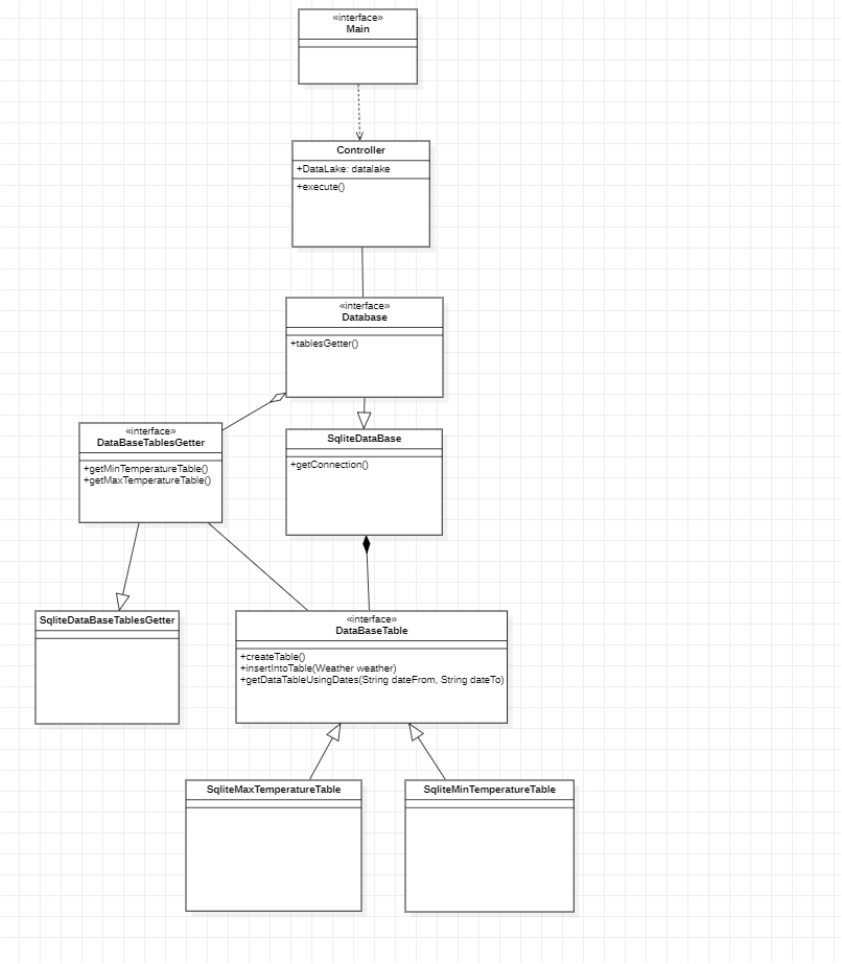
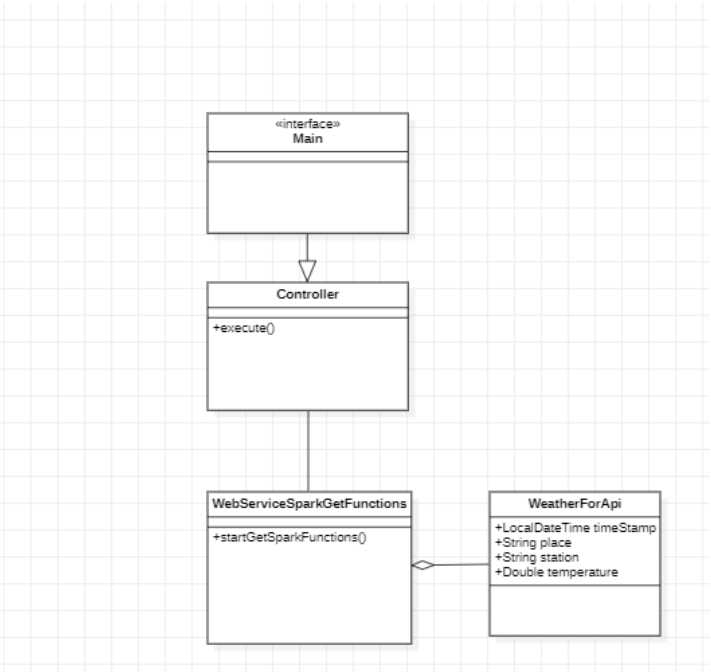


Diagrama de clases de TemperatureService(imagen original en el repositorio de Github):



Conclusiones

Con este proyecto he aprendido a organizar mejor el código y he empezado a preocuparme más en la calidad de la solución gracias a la experiencia anterior obtenida en DACD y en el proyecto de IS. Con este proyecto también he aprendido a usar mejor los ficheros ya que este tipo de operaciones siempre me ha costado más comprenderlas y a usar mejor el entorno de desarrollo, sobre todo por la parte del "Run Configuration".

Si volviese a empezar de cero usaría desde el principio el git y GitHub ya que anteriormente solo estaba haciendo commits sin push y a raíz de la corrección del trabajo de Spotify me di cuenta de que era commit & push (pernsaba que con el commit ya valía) y separaría el proyecto en paquetes desde el inicio.

Líneas futuras

Una idea para el futuro podría ser que, con los datos obtenidos de AEMET, hacer predicciones del tiempo a unas horas determinadas por el usuario y un lugar concreto de los sensores y en base a esos datos hacer recomendaciones sobre el abrigo que sería aconsejable llevar a esas horas en ese lugar en concreto

Bibliografía

ScheduledExecutorService

Java Timer vs ExecutorService? - Stack Overflow
ScheduledExecutorService (Java Platform SE 8) (oracle.com)

sqlite tutorial:
https://www.sqlitetutorial.net/sqlite-replace-statement/

Datatypes In SQLite
SQLite Date & Time - How To Handle Date and Time in SQLite (sqlitetutorial.net)

run configuration:
https://www.jetbrains.com/idea/guide/tutorials/hello-world/creating-a-run-configuration/

guía de patrones de donde tome inspiración para estructurar la base de datos (no se si lo que tengo hecho encaja con alguno):

Design Pattern - Overview (tutorialspoint.com)

FileWriter:

java write data to file without erasing the old content - Stack Overflow