DATALAKE TO DATAMART

2022-2023

13 enero

Universidad de Las Palmas de Gran Canaria

Escuela de Ingeniería Informática

Grado en Ciencia e Ingeniería de Datos

ULPGC
Universidad de
Las Palmas de
Gran Canaria

Autor:

Óscar Rico Rodríguez

Versiones:

0.0 [10/12/2022]

Creación e implementación del modulo feeder.

1.0 [12/12/2023]

Creación e implementación del modulo datarmart-provider.

2.0 [13/12/2023]

Creación e implementación del modulo temperature-service.

3.0 [20/12/2023]

Mejoras en el programa, el modulo feeder recoge los parametros como argumentos.

4.0 [09/01/2023] FINAL

Todos los modulos se ejecutan a la vez.

Resumen

Datalake to Datamart

Para este proyecto se nos pedía realizar un programa que recogiera datos de la api de aemet sobre la temperatura del aire en Gran Canaria, estos se guardan en un datalake que después fuera usado para guardar temperaturas máximas y minimas en un datamart y este alimentara un webservice donde he realizado una API REST mediante Java Spark.

Contenido

Mi programa consta de tres modulos, cada uno posee una serie de clases, interfaces y records necesarias para realizar su función.

- feeder.
- Datamart-provider.
- Temperature-service.

Funcionamiento

Al ejecutar el programa los 3 modulos comenzarían a funcionar.

Feeder recoge 5 parametros como argumentos (AEMET apikey, min lat, max lat, min lon, max lon) y se encargaría de recoger cada hora todos los datos posibles del día anterior y el día actual, filtrar los datos por la zona asignada en los argumentos y almacenar en el datalake los datos extraidos y que no se encuentren ya almacenados de AEMET, para guardarlos en su correspondiente fichero del tipo “YYYYmmDD.events” siendo YYYYmmDD el año, mes y día en el que ocurrió el evento.

Datamart-provider se encarga de leer los datos del datalake, filtrar las temperaturas máximas y mínimas de cada día y almacenarlas en el datarmat en 2 tablas “MaxTemp” (para temperaturas máximas) y “MinTemp” (para temperaturas mínimas).

Temperature-service se encarga de leer los datos almacenados en el datamart para dar respuesta a las peticiones de los usuarios las respuestas serán en formato JSON (ya que se trata de una API REST) y se le mostrará al cliente en el cuerpo de la petición siempre y cuando los datos introducidos sean correctos. En caso de petición incorrecta, se lanzará una excepción.

Anotaciones

Para el desarrollo de esta aplicación he tenido en cuenta los siguientes aspectos:

El modulo datarmat tiene en su TimerTask un delay de 10 segundos, que es algo más de lo que puede tardar el modulo feeder en realizar su función, para que así la ejecución del datamart sea más efectiva.

Uso del programa

Los parametros de las llamadas del webservice para filtrar las temperatures deben seguir el formato “YYYY-mm-DD” para poder ser filtradas correctamente, ejemplo:

Solicitud con parametros correctos:

http://localhost:4567/v1/places/with-max-temperature?from=2023-01-03&to=2023-01-07


```
[
  {date: "2023-01-04", time: "17:00", station: "LA ALDEA DE SAN NICOLAS", location: "C619Y",…},
  {date: "2023-01-05", time: "12:00", station: "LAS PALMAS G.C. SAN CRISTÓBAL",…},
  {date: "2023-01-06", time: "13:00", station: "LAS PALMAS G.C. SAN CRISTÓBAL",…},
  {date: "2023-01-07", time: "14:00", station: "LA ALDEA DE SAN NICOLAS", location: "C619Y",…}
```


Solicitud con parametros incorrectos (Saltaría error 404):

http://localhost:4567/v1/places/with-max-temperature?from=2023/01/03&to=2023/01/07

```
No se dispone de datos para estas fechas, o el formato es incorrecto [YYYY-mm-DD].
```

Solicitud sin parametros:

http://localhost:4567/v1/places/with-max-temperature


```
{date: "2023-01-13", time: "10:00", station: "MOGAN (PUERTO RICO)", location: "C629Q",…}
```

Índice

Recursos utilizados 4
Diseño 5
Patrón y principio de diseño usado 5
Diagrama de clases 6
Conclusiones 6
Líneas Futuras 6
Bibliografía 7

Recursos utilizados

Entorno de desarrollo

- IntelliJ

Controlador de Versiones

- Git

Editor de texto

- Word

Alojamiento del Proyecto

- GitHub

Diseño

Patrón y principio de diseño usado

Como patrón de diseño para el programa decidí usar un Movel View Controller (MVC) ya que es un patrón sencillo, escalable y mantenible. Por lo que no lo vi mala opción.

El MVC es un modelo divido en 3 capas donde cada cual cumple una función espéficica.

- View. Parte de la aplicación donde se contempla la interfaz gráfica. En esta capa se obtiene información sobre lo que quiere el usuario (Eventos) y se muestra la información del Modelo.
- Model (Lógica de negocio). Parte de la aplicación donde se relacionan los datos con los que se van a trabajar. Es el encargado de mapear las actividades del mundo real a la forma en la que se va a modificar la información.
- Controller. Parte de la aplicación que responde a los eventos del usuario y decide que view mostrar al usuario dependiendo de la solicitud recibida (Se puede ver como la conexión entre el model y el view).

En cuanto al principio de diseño, he seguido la guía de Clean Code que se nos proporcionó en el Campus Virtual.

Clean Code es una filosofía de desarrollo de software que consiste en aplicar técnicas simples que facilitan la escritura y lectura de un código.

Algunas de las principales prácticas para tener un código limpio son las siguientes:

1. Los nombres son importantes
    - Debe ser preciso, sin preocupación sobre el tamaño del nombre, es preferible claridad antes que brevedad.
2. Regla del boy scout
    - Esta regla indica que el código debe quedar más limpio de lo que estaba antes de ser editado
3. Métodos breves y descriptivos
    - Debemos evitar el anidamiento excesivo (complejidad ciclomática). Esto se consigue asignando a cada método una única funición.
4. DRY (Don´t Repeat Yourself)
    - Evitar el código duplicado. El código repetido entre diferentes clases, se pasa a clases abstractas.
5. Comentar solamente lo necesario
    - El código bien escrito se explica solo, sin necesidad de comentarios.
6. Tratamiento de errores
    - Para un código limpio, tratar las excepciones de forma correcta es un gran paso.
7. Tests limpios
    - Realizar tests es una etapa muy importante en la programación, un código solo se considera limpio después de haber sido válidada con varias pruebas.

Diagrama de clases

Imágenes en el repositorio

Diagrama de clases feeder [diagramas_de_clase/feeder_diagrama.png]

Diagrama de clases datamart [diagramas_de_clase/ datamart _diagrama.png]

Diagrama de clases webservice [diagramas_de_clase/ webservice _diagrama.png]

Conclusiones

Durante el desarrollo de esta actividad he afianzado conocimientos con el funcionamiento de las arquitecturas lambda, el uso de streams y, la escritura y lectura de archivos con java. Ha sido un trabajo bastante entretenido y que ha sufrido muchos cambios desde su inicio hasta el proyecto final por la toma de ciertas decisiones que para mi gusto lo hacían un programa más robusto.

En cuanto a experiencias en el proyecto me ha gustado bastante, ya que gracias a él he aprendido cosas sobre el uso de TimerTask, la realización de un programa multimodular y el manejo de fechas.

Líneas Futuras

Bajo mi punto de vista si, quisiéramos evolucionar este producto hasta convertirlo en un producto comercializable, debería tener en cuenta los siguientes aspectos:

- Posibilidad de filtrar áreas por nombre.
- Posibilidad de conseguir temperaturas medias en un día o en un periodo de días.
- Posibilidad de comparar periodos de días para ver cual fue más caluroso o frio.

Bibliografía

MVC

¿Qué es patron MVC y cómo funciona?

MVC Explained in 4 Minutos

Clean Code

Clean Code: Código limpio, ¿qué es?

Guía de estilo en Java y buenas prácticas para un código limpio

Java

Cómo hacer un TimerTask

Manejo de fechas en java

Leer y escribir archivos en java