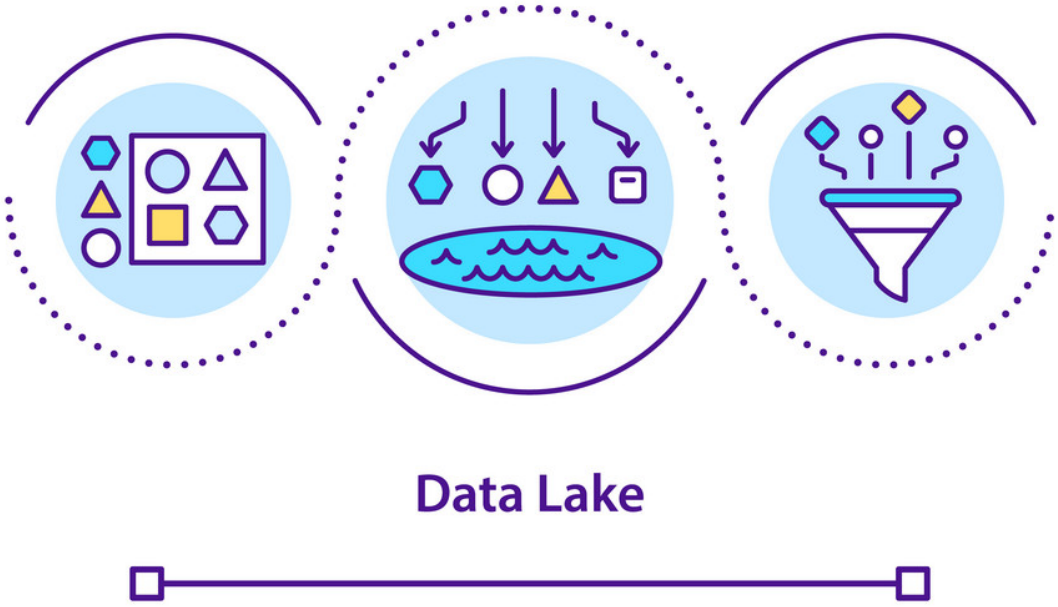
Desarrollo de aplicaciones para ciencia de datos

Primer semestre

PROYECTO DEVELOPING A DATALAKE: MEMORIA

Tercer proyecto



SEGUNDO CURSO

GRADO EN CIENCIA E INGENIERÍA DE DATOS

ESCUELA DE INGENIERÍA INFORMÁTICA

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

ÍNDICE

RESUMEN

Módulo 1: DatalakeModule

El Data Lake es una estructura de almacenamiento que permite almacenar una gran cantidad de datos en su formato original, tanto estructurados como no estructurados. Estos datos pueden ser analizados posteriormente para obtener información valiosa.

En el código proporcionado, se ha creado una clase WeatherDataDownloader que se encarga de descargar los datos meteorológicos de las estaciones de la AEMET (Agencia Estatal de Meteorología) a través de una conexión a su API.

La clase **WeatherDataSaver** se encarga de almacenar los datos descargados en una carpeta llamada "datalake" en ficheros con el nombre de del día.events en formato JSON, y se asegura de que no se dupliquen los datos mediante una copia maestra de todos los datos en un fichero .json.

La clase **WeatherData** es la clase principal del programa, que se encarga de crear una instancia de la clase WeatherDataTask para ejecutar el programa y qué realice la tarea cada hora actualizando los datos del datamart.

WeatherDataTask: se encarga de crear una instancia de cada una de las clases anteriores y de llamar a sus métodos.

RESUMEN

Módulo 2: DatamartModule

Un datamart es una estructura de datos que se utiliza para almacenar información específica de un negocio o industria. En nuestro caso de un datamart en una base de datos SQLite, se trata de una base de datos independiente en la que se almacenan sólo los datos necesarios para el área de negocio específica y se pueden realizar consultas y análisis de esa información de manera más rápida y eficiente.

EventData: esta clase representa los datos de una medición de temperatura, con los siguientes atributos: date, time, place, station, tamax y tamin. También incluye un método para obtener los atributos y para imprimirlos en formato legible.

DataLoader: esta clase se encarga de leer los ficheros .events y almacenar los datos en una estructura temporal (por ejemplo, un ArrayList). También incluye un método para obtener los datos cargados y para imprimirlos en formato legible.

DataProcessor: esta clase se encarga de procesar los datos cargados para buscar las temperaturas máximas y mínimas de cada día. También incluye un método para imprimirlas en formato legible.

DataWriter: esta clase se encarga de crear las tablas en la base de datos SQLite qué llamé "datamart.db" y de escribir los datos de las temperaturas máximas y mínimas en ellas. También incluye métodos para conectarse y desconectarse de la base de datos.

DatamartBuilder: esta clase es la clase principal del programa, que se encarga de crear una instancia de la clase DatamartBuilderTask para ejecutar el programa y qué realice la tarea cada día actualizando los datos del datamart.

DatamartBuilderTask: se encarga de crear una instancia de cada una de las clases anteriores y de llamar a sus métodos

RESUMEN

Módulo 3: ApiModule

Una API REST con Spark es una forma de crear una interfaz de programación de aplicaciones (API) que utiliza el framework Spark para manejar solicitudes y respuestas HTTP.

Spark es un marco de trabajo de código abierto para procesamiento de big data en cluster que se utiliza para analizar grandes conjuntos de datos. Utilizando Spark, se pueden crear aplicaciones RESTful que manejan grandes cantidades de datos y ofrecen un rendimiento escalable y rápido como es nuestro caso.

En resumen, una API REST con Spark es una forma de crear una interfaz para interactuar con una base de datos o sistema de almacenamiento de datos utilizando el framework Spark para manejar solicitudes y respuestas HTTP, permitiendo un rendimiento escalable y rápido.

La clase **TemperatureController** : La clase TemperatureController se encarga de manejar las peticiones y devolver la información en formato JSON, y se apoya en la clase Database para acceder a la base de datos y Gson para devolver la información en formato JSON.

La clase tiene dos métodos principales: getMaxTemperature y getMinTemperature, los cuales se encargan de manejar las peticiones GET correspondientes. Cada uno de estos métodos toma los parámetros "from" y "to" de la petición, los utiliza para realizar una consulta a la base de datos a través de la instancia de la clase Database, y devuelve el resultado en formato JSON.

El método toJsonArray se encarga de convertir un conjunto de resultados de una consulta a base de datos a un objeto JsonArray que puede ser convertido fácilmente a una cadena JSON.

La clase **Database**: esta clase se encarga de realizar las consultas a la base de datos y devolver los resultados en formato JSON. Tiene dos métodos getMaxTemperature y getMinTemperature que utilizando SQLiteDataSource y JDBC, realizan las consultas a la base de datos y devuelve el resultado en formato JSON.

La clase **TemperatureAPI** es la clase principal, es la encargada de inicializar y configurar las rutas de la API, utilizando las clases Database y TemperatureController para manejar las peticiones y devolver la información respectiva.

RECURSOS UTILIZADOS

Entornos de desarrollo utilizados

- Intellij
- Maven

Herramientas de control de versiones

- Git
- GitHub

Herramientas de documentación

- JDK 11 Documentation
- Maven Central
- Google
- TaletApiTester
- Json Documentation
- Sqlite Documentation

DISEÑO

Patrones y principios de diseño utilizados

En este proyecto hay un uso de clases y métodos, lo que sugiere el uso del patrón de diseño de programación orientada a objetos (OOP, Object-Oriented Programming).

He intentado tener en cuenta los 5 principios SOLID de diseño de aplicaciones de software de la siguiente manera:

Separación de responsabilidades: El diseño se divide en tres paquetes distintos que tienen responsabilidades específicas. Esto permite que cada paquete se enfoque en una tarea específica, lo que facilita la mantenibilidad y escalabilidad del código.

Patrón de diseño Builder: Este patrón ayuda a construir objetos complejos a partir de una clase base y varios objetos auxiliares. En este caso, se podría aplicar para construir los objetos con toda la información recolectada en el scraping y parseada.

CONCLUSIONES

En general, el proyecto se enfoca en la recolección, procesamiento y almacenamiento de datos meteorológicos y proporciona una interfaz para acceder a estos datos de forma organizada y estructurada..

LÍNEAS FUTURAS

En cuanto a líneas futuras lo siguiente es lo qué se me ocurre:

- **Análisis de datos**: Una vez que los datos están almacenados en el datamart, se podrían realizar análisis para obtener información valiosa y útil. Esto podría incluir el análisis de tendencias en las temperaturas, comparar temperaturas de diferentes estaciones, etc.
- **Visualización de datos**: Una vez que se han realizado análisis, se podrían crear visualizaciones para presentar los datos de forma fácil de entender. Esto podría incluir gráficos, mapas, etc.
- **Integración con otros sistemas**: El proyecto podría integrarse con otros sistemas, como un sistema de alertas para avisar a los usuarios de condiciones climáticas extremas.
- **Mejora en la escalabilidad**: Puede ser necesario mejorar la escalabilidad del proyecto para poder manejar una gran cantidad de datos, ya sea mediante la implementación de un sistema de almacenamiento distribuido o mediante la optimización de los procesos existentes.
- **Mejora en la seguridad**

BIBLIOGRAFÍA

- https://docs.oracle.com/en/java/javase/11/
- https://jsoncrack.com/editor
- https://www.sqlitetutorial.net/sqlite-java/create-database/
- https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/Map.html#of()
- https://sparkjava.com/
- https://opendata.aemet.es/centrodedescargas/altaUsuario
- https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/Map.html#of()

CONTRAPORTADA

Autor

Mara Pareja del Pino

Fecha de creación de la memoria

21/12/2022

Versión y revisión

Versión 1.0