Proyecto datalake to datamart

Titulación: Grado en Ciencia e Ingeniería de Datos

Universidad: Universidad de Las Palmas de Gran Canaria

Asignatura: Desarrollo de Aplicaciones para Ciencia de Datos

Escuela: Escuela de Ingeniería Informática

Cynthia Quintana Reyes

Ultima versión de la memoria: 13-01-2023

Resumen

El proyecto es un sistema de gestión de datos meteorológicos que tiene como objetivo recopilar, procesar y dar acceso a los datos meteorológicos de Gran Canaria. El sistema se implementó utilizando la versión 1.8 de Java y se desarrolló en el entorno de programación IntelliJ. El proyecto consta de tres programas o módulos principales:

Un programa de recogida de datos que descarga los datos de todas las estaciones meteorológicas cada hora desde el webservice de AEMET. Los datos recopilados se almacenan en una carpeta llamada "datalake" en formato json. Cada evento registra la marca de tiempo, la estación, el lugar y la temperatura del aire.

Un programa de procesamiento de datos que usa los datos almacenados en el lago de datos para construir un datamart con las temperaturas máximas y mínimas de cada día en la isla. El datamart se almacena en una base de datos SQLite y consta de dos tablas: una para temperaturas máximas y otra para temperaturas mínimas. Cada registro incluye la fecha, hora, lugar, estación y valor de la temperatura.

Un programa API REST que permite a los usuarios consultar las temperaturas más altas y bajas en la isla dentro de un rango de días. El programa utiliza el datamart creado por el programa anterior para proporcionar la información solicitada.

El objetivo principal de este proyecto es proporcionar una manera fácil y organizada de acceder a los datos meteorológicos de Gran Canaria. El sistema permitirá a los usuarios recuperar y analizar datos de manera rápida y eficiente, lo que podría ser útil para diversas aplicaciones relacionadas con el clima. El proyecto también servirá como una herramienta educativa útil para aprender sobre la recopilación de datos, el procesamiento y el desarrollo de API.

índice

Recursos utilizados

Entorno de desarrollo: IntelliJ IDEA se utilizó como entorno de desarrollo principal para este proyecto. Es un entorno de desarrollo integrado (IDE) potente y ampliamente utilizado que proporciona un conjunto completo de herramientas para desarrollar aplicaciones Java, incluida la edición de código, la depuración y las pruebas. IntelliJ IDEA es conocido por su interfaz fácil de usar y su capacidad para integrarse con varios sistemas de control de versiones, lo que lo convierte en una opción ideal para este proyecto.

Control de versiones: Git se utilizó como sistema de control de versiones para este proyecto. Git es un sistema de control de versiones distribuido que permite que varios desarrolladores trabajen en la misma base de código simultáneamente, mientras realiza un seguimiento de todos los cambios realizados en el código. Esto permite una fácil colaboración, reversiones y administración de diferentes versiones del código.

Dependencias:

1.sqlite-jdbc (3.39.3.0): SQLite JDBC Driver, es una biblioteca Java que permite que las aplicaciones Java interactúen con las bases de datos SQLite. Se utilizó en este proyecto para acceder y manipular las bases de datos SQLite utilizadas para almacenar el datamart.

2.gson (2.10): Gson es una biblioteca Java que se puede usar para convertir objetos Java en su representación JSON. También puede volver a convertir de JSON a su objeto Java equivalente. Se utilizó en este proyecto para convertir datos entre objetos Java y formato JSON.

3.json (20220924): JSON (Notación de objetos de JavaScript) es un formato de intercambio de datos liviano que es fácil de leer y escribir para los humanos y fácil de analizar y generar para las máquinas. Se utilizó en este proyecto para manejar datos JSON.

4.jsoup (1.11.3): jsoup es una biblioteca de Java para trabajar con HTML del mundo real. En este proyecto se utilizó para analizar documentos HTML y XML obtenidos del webservice de AEMET.

5.spark-core (2.9.4): Spark es un marco web Java simple y liviano que se usó para crear la API REST de este proyecto. Permite a los desarrolladores crear fácilmente aplicaciones y servicios web.

Estas dependencias se usaron para manejar bases de datos SQLite, convertir objetos Java hacia y desde JSON, analizar documentos HTML y XML y crear API REST respectivamente. También se utilizaron para facilitar la manipulación de datos y para interactuar con el webservice de AEMET.

Diseño

En el módulo feeder, las clases WeatherDataLoader y WeatherDataSaver utilizan el patrón Singleton para garantizar que solo se cree una instancia de cada clase durante la vida útil del programa. Esto se hace haciendo que los constructores sean privados y proporcionando un método estático que devuelve la instancia única de la clase.

La clase WeatherDataSaver usa el patrón Observer para programar actualizaciones de los datos, programa una tarea para que se ejecute después de cierto tiempo y llama al método de actualización.

En cuanto a la gestión de la memoria, la clase WeatherDataSaver utiliza un conjunto de objetos WeatherEvent para realizar un seguimiento de los eventos que ya se han guardado en el archivo. Esto evita la duplicación de eventos en el archivo.

Además, la clase WeatherDataLoader usa el patrón Adapter para adaptar los datos de la API al objeto WeatherEvent.

Además, el uso de la interfaz DataLoader permite cambiar la implementación de la carga de datos, sin afectar el resto del código.

El módulo datamart-builder es responsable de construir un datamart a partir del lago de datos creado por el módulo Feeder. Utiliza la base de datos SQLite para almacenar los datos.

La clase Controller usa el patrón Iterator para iterar sobre los archivos en un directorio dado y analizar los eventos en cada archivo usando la clase EventParser. Luego usa la clase SQLiteAemetDatabase para conectarse a la base de datos SQLite, crear tablas si no existen e insertar los eventos analizados en las tablas.

La clase SQLiteAemetDatabase usa el patrón Singleton para garantizar que solo se cree una instancia de la clase durante la vida útil del programa. También utiliza el patrón Connector para conectarse a la base de datos SQLite y el patrón Command para crear las tablas e insertar datos en la base de datos.

En cuanto a la gestión de la memoria, la clase SQLiteAemetDatabase realiza un seguimiento de la temperatura máxima y mínima de cada día en un Hashmap, de esta forma evita duplicar los mismos datos.

Además, el uso de la clase EventParser permite cambiar la implementación del análisis de datos, sin afectar el resto del código.

El módulo temperature-api es un servicio web que permite a los clientes recuperar datos de temperatura de una base de datos SQLite. El módulo consta de dos clases principales: TemperatureData y TemperatureService.

La clase TemperatureData es un POJO simple (Plain Old Java Object) que contiene los datos para una sola lectura de temperatura. Tiene campos privados para la fecha, la hora, el lugar, la estación y el valor, así como métodos públicos de obtención y configuración para cada campo.

La clase TemperatureService es responsable de conectarse a la base de datos SQLite, consultar datos de temperatura y devolver los datos en formato JSON utilizando la biblioteca GSON. Tiene dos métodos principales, getMaxTemperature y getMinTemperature, que se utilizan para recuperar los valores de temperatura máxima y mínima respectivamente. Ambos métodos toman un objeto de solicitud y respuesta como parámetros, que proporciona el marco Spark. Estos métodos utilizan los parámetros de consulta "desde" y "hasta" en el objeto de solicitud para especificar el rango de fechas para recuperar los datos de temperatura.

La clase TemperatureService también usa el patrón Singleton, de modo que solo se crea una instancia de la clase para toda la aplicación. El patrón DAO (objeto de acceso a datos) se utiliza para TemperatureService, que separa la lógica de acceso a datos de la lógica empresarial de la aplicación.

El módulo utiliza el principio de encapsulación para ocultar los detalles de implementación de las clases TemperatureData y TemperatureService del resto de la aplicación. También sigue los principios SOLID para que el módulo sea fácil de mantener y ampliar. Y sigue el principio DRY (Don't Repeat Yourself) en los métodos getMaxTemperature y getMinTemperature, que tienen una estructura similar y solo se diferencian por la consulta utilizada.

Conclusión

En conclusión, el sistema de gestión de datos meteorológicos desarrollado en este proyecto fue una solución integral que tenía como objetivo recopilar, procesar y dar acceso a los datos meteorológicos de Gran Canaria. El sistema se implementó utilizando Java versión 1.8 y se desarrolló en el entorno de programación IntelliJ.

Durante el desarrollo de este proyecto, hemos aprendido las siguientes lecciones:

-La importancia de separar la lógica de acceso a datos de la lógica de negocio de la aplicación utilizando el patrón DAO.

-La importancia de usar librerías y frameworks como SQLite, GSON, JSoup y Spark para una solución robusta y eficiente.

-Los beneficios de usar los principios SOLID y Encapsulation para hacer que el módulo sea fácil de mantener y ampliar.

-El poder de usar el patrón Singleton para garantizar que solo se cree una instancia de una clase durante la vida útil del programa.

Mirando hacia atrás, si tuviera que comenzar el proyecto de nuevo, consideraría usar una versión más moderna de Java y posiblemente un marco web más moderno como Spring Boot. Además, también consideraríamos usar una solución de almacenamiento de datos más poderosa y escalable como una base de datos NoSQL.

En general, este proyecto sirvió como una valiosa experiencia de aprendizaje, permitiéndonos obtener experiencia práctica en la recopilación, el procesamiento y el desarrollo de API de datos. También proporcionó una herramienta educativa útil para aprender sobre varios conceptos de programación y mejores prácticas.

Líneas futuras

Para convertir el sistema de gestión de datos meteorológicos en un producto comercializable, se pueden considerar las siguientes líneas futuras:

Integrar el sistema con más fuentes de datos, como otros servicios meteorológicos, redes sociales y dispositivos IoT, para brindar una visión más completa de las condiciones climáticas en la isla.

Desarrollar una aplicación web o móvil fácil de usar que permita a los usuarios acceder y visualizar fácilmente los datos meteorológicos.

Implementar técnicas avanzadas de análisis y aprendizaje automático para proporcionar información y predicciones adicionales basadas en los datos recopilados.

Desarrollar un modelo basado en suscripción que permita a los usuarios acceder a funciones premium, como alertas personalizadas y análisis de datos históricos.

Permitir el acceso a los datos a través de una API pública que podría ser utilizada por otras aplicaciones y servicios relacionados con el clima.

Desarrollar una arquitectura más robusta y escalable que pueda manejar una gran cantidad de solicitudes y datos.

Desarrollar un sistema más seguro mediante la implementación de mecanismos adecuados de autenticación y autorización para proteger los datos.

Agregar más soporte de idiomas para que el servicio sea más inclusivo.

Agregue más opciones de visualización de datos para que los datos sean más accesibles e informativos para los usuarios.

Implementar mecanismos de monitoreo y alerta para detectar y solucionar proactivamente cualquier problema que pueda surgir en el sistema.

En general, al implementar estas líneas futuras, el sistema de gestión de datos meteorológicos se puede transformar en un producto potente, confiable y fácil de usar que brinda información valiosa y predicciones sobre las condiciones climáticas en la isla. Esto podría ser de gran valor para una amplia gama de usuarios, desde investigadores y meteorólogos hasta turistas y residentes locales.