Desarrollo de aplicaciones para ciencia de datos

Primer semestre

PROYECTO SCRAPER: MEMORIA

Segundo proyecto

SEGUNDO CURSO

GRADO EN CIENCIA E INGENIERÍA DE DATOS

ESCUELA DE INGENIERÍA INFORMÁTICA

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

ÍNDICE

RESUMEN

En resumen, ha desarrollado un servicio web que extrae información de una página web (Booking) en HTML utilizando la biblioteca jsoup, convierte esa información a formato JSON y la proporciona a través de una API de consulta, se pueden realizar las peticiones mediante la extensión de Chrome Talend API Tester. Por ejemplo:

http://localhost:4567/hotels/gloria-palace-amadores/comments

El programa funciona en local y mediante una interfaz de usuario en modo texto la cuál especifica mejor las posibles peticiones “Get”.

La estructura del proyecto consta de tres paquetes:

El paquete "**modelo**" almacena las clases que representan los datos obtenidos del scraping. Es decir la localización del hotel, servicios, puntuaciones y los comentarios de los clientes.

El paquete "**controller**" contiene la clase que maneja las peticiones GET a la API. Esta clase es responsable de recibir la petición, procesar y devolver una respuesta en formato JSON.

El paquete "**scraping**" contiene la clase que se utiliza para realizar el scraping de los datos. Esta clase utiliza la biblioteca jsoup para extraer información de la página web y almacenarla en objetos del modelo.

Además, también hay una clase **Main** que es la clase principal donde se ejecuta la aplicación y se define el punto de entrada.

Este diseño permite separar las diferentes responsabilidades del sistema en diferentes paquetes y clases, lo que facilita la mantenibilidad y escalabilidad del código. Además, al tener una capa de controlador y otra de scraping se aísla y el cambio en una de las capas no afecta a la otra.

RECURSOS UTILIZADOS

Entornos de desarrollo utilizados

- Intellij
- Maven

Herramientas de control de versiones

- Git
- GitHub

Herramientas de documentación

- JDK 11 Documentation
- Maven Central

DISEÑO

Patrones y principios de diseño utilizados

He intentado aplicar un estilo arquitectónico en mi código, escogí el MVC, es decir el Model View Controller, ya que es el que hemos visto en clase y el que entendí mejor además me parece

una un estilo arquitectónico adecuado para este proyecto ya que crea independencia de funcionamiento. Facilita el mantenimiento en caso de errores. Ofrece maneras más sencillas para probar el correcto funcionamiento del sistema y permite el escalamiento de la aplicación en caso de ser requerido.

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 (modelo, controlador y scraping) 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 Model-View-Controller (MVC): Este patrón divide el código en tres capas: modelo, vista y controlador. El modelo representa los datos y la lógica del negocio, la vista es la interfaz de usuario, en mi caso por línea de comandos y el controlador se encarga de recibir las peticiones del usuario y de actualizar la vista. En este proyecto se puede apreciar esta estructura en los paquetes modelo, controller y Main.

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

El proyecto desarrollado es un servicio web que extrae información de una página web utilizando la biblioteca jsoup y la proporciona a través de una API en formato JSON. El diseño del proyecto se basa en separar las responsabilidades en diferentes paquetes y clases, siguiendo principios de diseño de software como SOLID y patrones de diseño como MVC y Builder.

LÍNEAS FUTURAS

- Implementar validación de entradas para los parámetros de las peticiones GET en el controlador para prevenir posibles errores o ataques.
- Implementar caching de los datos scrapeados para mejorar el rendimiento de la aplicación, especialmente si se está haciendo scraping de la misma página varias veces.
- Ampliación del alcance del scraping, si se desea obtener más información de la página web o scraping más páginas.
- Implementar autenticación y autorización en la API para restringir el acceso solo a usuarios autorizados.
- Implementar la persistencia de los datos scrapeados en una base de datos para tener un histórico de los datos.
- Integración de pruebas automatizadas en el proyecto para asegurar que el código funciona correctamente y para facilitar la detección de errores.

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()

CONTRAPORTADA

Autor

Mara Pareja del Pino

Fecha de creación de la memoria

21/12/2022

Versión y revisión

Versión 1.0