Évolution et restructuration des logiciels

Rapport

SAR Alexandre

November 2024

Lien vers le dépôt GitHub : https://github.com/texao/tp3_Journalisation

Contents

1 Introduction

Ce rapport documente le développement et l'implémentation d'une application backend simple en Java, conçue pour gérer les profils d'utilisateurs à travers l'observation et l'analyse des logs générés pendant l'exécution. Le projet illustrer des concepts fondamentaux de journalisation logicielle en utilisant la bibliothèque Log4j2 pour collecter des informations. L'objectif est de générer des profils utilisateurs basés sur leurs interactions avec le système (par exemple, opérations de lecture, d'écriture, ou de recherche), tout en garantissant une modularité et une extensibilité du code.

Plusieurs scénarios utilisateurs ont été définis et exécutés afin de simuler des cas d'utilisation. Les données de log générées sont analysées pour construire des profils utilisateurs précis, présentés au format JSON. Ces profils permettent de classifier les utilisateurs selon leurs comportements principaux.

Les autres sections montre en détail l'architecture du système, les outils et bibliothèques utilisés, ainsi que les résultats obtenus.

2 Backend

Le backend implémente la logique nécessaire pour la gestion des utilisateurs, des produits, et des actions associées. Il repose sur une architecture modulaire, facilitant la maintenance et l'ajout de nouvelles fonctionnalités. Les principales composantes incluent la gestion des données utilisateur et produit et l'interface en ligne de commande (CLI).

2.0.1 Actions utilisateurs

Le backend permet aux utilisateurs de réaliser différentes actions simulées sur des produits. Parmi les principales fonctionnalités proposées :

- **Ajout de produit** : Un utilisateur peut ajouter un produit en spécifiant son nom, son prix, et sa date d'expiration.
- **Consultation de produit** : L'utilisateur peut rechercher un produit spécifique ou afficher la liste de tous les produits.
- **Modification de produit** : Un produit peut être mis à jour avec de nouvelles informations.
- **Suppression de produit** : Les utilisateurs peuvent supprimer un produit à l'aide de son identifiant unique.
- **Recherche avancée** : L'utilisateur peut rechercher le produit le plus cher dans la base.

Toutes ces actions sont accompagnées d'un suivi via un système de logs, ce qui permet une traçabilité des opérations.

2.0.2 CLI

Une interface en ligne de commande (CLI) a été développée pour permettre l'interaction avec l'application. Cette interface offre une manière simple d'exécuter les différentes actions utilisateur. Les commandes disponibles incluent :

- `add-product` : Permet d'ajouter un nouveau produit en spécifiant ses attributs.
- `view-product <id>` : Affiche les informations d'un produit à partir de son identifiant.

- `delete-product <id>` : Supprime un produit existant.
- `update-product <id>` : Met à jour les détails d'un produit donné.
- `list-products` : Affiche tous les produits disponibles.
- `find-expensive` : Identifie et affiche le produit le plus cher.

La CLI est conçu pour fournir des messages interactifs, pour une expérience utilisateur efficace.

3 Architecture

L'architecture de cette application est composée de plusieurs parties organisées pour séparer les fonctionnalités .

3.1 Partie principale

Cette partie contient plusieurs classes dont le main de l'application. Elle permet l'exécution en appelant les différentes classes pour :

- Appliquer l'analyse des actions des utilisateurs à l'aide du *Visitor Pattern*.
- Gérer les logs avec *Log4j2*.
- Générer les profils des utilisateurs à partir des données collectées.

3.2 Gestion des logs et des profils utilisateurs

Une classe dédiée gère l'enregistrement des logs à l'aide de la bibliothèque `Log4j2`. Deux autre classe est responsable de la génération des profils utilisateurs en analysant les actions enregistrées. Ces classes sont utilisées pour centraliser la logique de gestion des événements et aussi leur stockage.

3.3 Backend (Partie source)

Le backend contient plusieurs classes : Product, User, UserInterface, ProductRepository. Cette partie encapsule toute la logique métier et sert pour les opérations réalisées par l'application. Les logs sont générés à partir de ces fichiers.

3.4 Répertoires des résultats

Un répertoire spécifique est utilisé pour stocker les fichiers générés par l'application, tels que :

- Les fichiers contenant les actions des utilisateurs sont générer dans un répertoire spécifiques.
- Les profils des utilisateurs générés dans la console.
- Les logs de l'application sont générés à la racine de l'application.

Les fichiers du backend (produits, utilisateurs, etc.) sont générés directement à la racine.

Cette architecture garantit une séparation des fonctionnalités, ce qui facilite la maintenance et le débogage de l'application.

4 Eclipse JDT

Utilisation d'*Eclipse Java Development Tools (JDT)*, un outil permettant d'analyser et de manipuler des projets Java. Eclipse JDT fournit une API pour parcourir les projets, extraire des informations structurelles et effectuer des analyses statiques.

4.1 Visitor

L'analyse des fichiers source repose sur le modèle de visiteur (*Visitor*) proposé par Eclipse JDT, qui permet de parcourir l'arbre syntaxique abstrait (AST) généré à partir du code.

Eclipse JDT parcours tous les fichiers du répertoires contenant le backend et ensuite génère des logs spécifiques aux endroits où des actions sont effectués. Pour chaque fichier analysé, des informations sont collectées et enregistrées dans des fichiers distincts, correspondant aux fichiers du backend.

```
Modified file saved to: Modified_Product.java
Modified file saved to: Modified_User.java
2024-11-25 12:15:42 [main] INFO : User User8 performed: read operation in method getProduct
2024-11-25 12:15:42 [main] INFO : User User1 performed: write operation in method addProduct
2024-11-25 12:15:42 [main] INFO : User User7 performed: write operation in method deleteProduct
2024-11-25 12:15:42 [main] INFO : User User9 performed: write operation in method updateProduct
Modified file saved to: Modified_UserInterface.java
2024-11-25 12:15:42 [main] INFO : User User1 performed: write operation in method addProduct
2024-11-25 12:15:42 [main] INFO : User User8 performed: read operation in method getProduct
2024-11-25 12:15:42 [main] INFO : User User3 performed: write operation in method deleteProduct
2024-11-25 12:15:42 [main] INFO : User User6 performed: write operation in method updateProduct
2024-11-25 12:15:42 [main] INFO : User User9 performed: expensive_search operation in method rechercheProduitLePlusCher
2024-11-25 12:15:42 [main] INFO : User User0 performed: read operation in method displayAllProducts
Modified file saved to: Modified_ProductRepository.java
```

Figure 1: Couplage entre classes

Dans la figure ci-dessus, nous pouvons observer que les fichiers correspondant au backend ont été créés. Ces fichiers contiennent des logs si les fichiers analysés incluent des actions liées aux utilisateurs. Chaque log détaille les événements significatifs détectés au cours de l'exécution. Des logs sont également générés lorsqu'une opération a été faite.

5 Log

La gestion des logs est essentielle pour tracer l'exécution du programme et analyser les comportements du système. L'utilisation de logs permet de suivre les actions des utilisateurs et les opérations effectuées par le backend. Chaque événement est enregistré afin de fournir une information sur le fonctionnement interne de l'application.

5.1 Utilisation de Log4j2

Pour la gestion des logs, j'ai utilisé la bibliothèque *Log4j2*, un framework flexible et performant pour la journalisation en Java. Cette bibliothèque offre de nombreuses fonctionnalités, notamment :

- Une configuration modulaire via un fichier XML ou JSON.
- La possibilité de définir différents niveaux de logs (*INFO*, *DEBUG*, *ERROR*, *WARN*, etc.), permettant de filtrer les informations selon les besoins.
- La prise en charge de plusieurs destinations pour les logs, telles que les fichiers, la console ou des systèmes externes.

Dans ce projet, un fichier de configuration a été défini pour spécifier le format des messages, les niveaux de journalisation, et les destinations des logs. Chaque action utilisateur détectée par le système est enregistrée avec des détails tels que la date, l'heure, et une description de l'événement.

```
<?xml version="1.0" encoding="UTF-8"?>
Bind to grammar/schema...
<Configuration status="WARN">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss} [%t] %-5level: %msg%n" />
        </Console>
    </Appenders>
    <Loggers>
        <Root level="info">
            <AppenderRef ref="Console" />
        </Root>
    </Loggers>
</Configuration>
```

Figure 2: Exemple de configuration de Log4j2

L'exemple de configuration ci-dessus illustre comment les logs affichés dans la console. Tous les messages de log de niveau INFO ou supérieur sont affichés dans la console, avec un format indiquant la date, le niveau de log, et le message. Cette configuration est adaptée pour le débogage et le suivi des activités d'une application sans avoir besoin de fichiers ou d'autres destinations de log.

6 Profiles des utilisateurs

Par la suite, j'analyse les profils des utilisateurs, en décrivant les données fournies au système, leur format, et les résultats obtenus après traitement.

6.0.1 Json


```
{"1":{"expensive_search":2,"read":1,"update":2,"write":3,"delete":2},"2":{"expensive_search":7,"read":5,"update":1,"write":
4,"delete":3},"3":{"expensive_search":0,"read":3,"update":2,"write":1,"delete":3},"4":{"expensive_search":0,"read":3,"update":
3,"write":3,"delete":1},"5":{"expensive_search":3,"read":9,"update":3,"write":4,"delete":1},"6":{"expensive_search":2,"read":
6,"update":4,"write":4,"delete":4},"7":{"expensive_search":4,"read":8,"update":3,"write":3,"delete":2},"8":{"expensive_search":
4,"read":6,"update":5,"write":4,"delete":1},"9":{"expensive_search":5,"read":4,"update":2,"write":5,"delete":4},"10":
{"expensive_search":4,"read":7,"update":2,"write":4,"delete":3}}
```


Figure 3: Données JSON

Ces données JSON représente les opérations effectuées par 10 utilisateurs. Chaque utilisateur est identifié par un numéro unique (par exemple, `"1"`, etc.), et un ensemble d'opérations associées est enregistré pour chaque utilisateur.

Chaque opération est comptabilisée selon son type :

- **read** : Nombre d'opérations de lecture effectuées.
- **write** : Nombre d'opérations d'écriture effectuées.
- **expensive_search** : Nombre de recherches complexes effectuées.
- **update** : Nombre d'opérations de mise à jour effectuées.
- **delete** : Nombre d'opérations de suppression effectuées.

De plus, ces opérations correspondent aux événements spécifiques suivants :

- `rechercheProduitLePlusCher` : Recherche du produit le plus cher (*expensive_search*).
- `updateProduct` : Mise à jour des informations d'un produit (*update*).
- `deleteProduct` : Suppression d'un produit (*delete*).
- `addProduct` : Ajout d'un nouveau produit (*write*).
- `displayAllProducts` : Affichage de tous les produits disponibles (*read*).

Le tableau ci-dessous montre les opérations effectuées par l'utilisateur `"1"` telles qu'enregistrées dans le fichier JSON. Cet utilisateur a réalisé un ensemble d'actions variées :

Pour l'utilisateur `1`, voici le nombre d'opérations effectuées :

- **expensive_search** : 2 – cet utilisateur a effectué 2 recherches du produit le plus cher.
- **read** : 1 – une action de lecture a été réalisée.
- **update** : 2 – l'utilisateur a mis à jour 2 produits.
- **write** : 3 – trois ajouts de produits ont été effectués.
- **delete** : 2 – deux produits ont été supprimés.

6.0.2 Resultats

```
Product{id='0c93105f-ec7f-4cd0-ab98-be9a4e93bd54', name='Produit75', price=104.915645646825, expirationDate=2025-10-31}
Product{id='2393192c-8e0d-473a-9e5f-fe47a9c44ec8', name='Produit74', price=67.31412359975388, expirationDate=2025-04-29}
Product{id='d3b31005-421c-4273-b840-52c8ba3b47c0', name='Produit6', price=60.3489976252535, expirationDate=2025-07-18}
2024-11-25 19:16:14 [main] INFO : récupération du produit avec id 3f4fd6d7-167c-4c75-9383-af77642842b3 reussit
Profiles saved to src/main/java/tp3_hai913/UserProfile/userProfiles.json
Utilisateurs qui lisent principalement : [1, 4, 5, 6, 7, 8, 9, 10]
Utilisateurs qui écrivent principalement : [1, 2]
Utilisateurs qui recherchent principalement des produits chers : [3]
```

Figure 4: Profils des utilisateurs

D'après les données collectées, nous avons pu observer les comportements suivants des utilisateurs en fonction des types d'opérations qu'ils effectuent principalement :

- **Utilisateurs qui lisent principalement :** Les utilisateurs `1`, `4`, `5`, `6`, `7`, `8`, `9`, et `10` sont ceux qui réalisent principalement des opérations de lecture (`read`). Ces utilisateurs sont plus s'intéresser à consulter les informations disponibles qu'à modifier ou ajouter de nouvelles données.

- **Utilisateurs qui écrivent principalement :** Les utilisateurs `1` et `2` sont ceux qui réalisent principalement des opérations d'écriture (`write`). Cela peut indiquer que ces utilisateurs sont impliqués dans l'ajout ou la mise à jour des produits dans le système.
- **Utilisateurs qui recherchent principalement des produits chers :** L'utilisateur `3` est celui qui effectue principalement des recherches pour connaître le produit le plus cher (`expensive_search`). Cela montre que cet utilisateur se concentre sur la recherche des produits les plus chers.
- L'utilisateur `1` fais partie des utilisateurs qui lisent et écrivent car il effectue autant d'action en lecture qu'en écriture.

Ces résultats permettent de mieux comprendre les comportements des utilisateurs dans le système, ce qui peut être utile pour l'optimisation des ressources ou la personnalisation des fonctionnalités proposées.

7 Conclusion

Le projet a permis de mettre en pratique les principes de journalisation logicielle dans une application backend en Java. L'utilisation de Log4j2 était utile pour capturer et structurer les événements d'exécution facilitant ainsi l'analyse et la construction de profils utilisateurs détaillés.

Les résultats obtenus montrent la capacité du système à catégoriser efficacement les utilisateurs selon leurs actions principales (lecture, écriture, ou recherche). Les profils utilisateurs en format JSON offrent une solution légère et portable pour le stockage et l'exploitation des données. L'approche adoptée met en avant l'importance d'une architecture bien définie et de scénarios d'exécution diversifiés pour garantir la représentativité des utilisateurs.