HAI913I

Module : Évolution et restructuration des logiciels

Introduction pour les logiciels de logging et d'Observabilité

Réalisé par :
M lle MEKHNACHE Toudherth – GL M2

Référent :
M r Djamal SIRIAI
M r Bachar RIMA

Promotion 2023/2024

Table des matières

Chapitre 1

Logging et Observabilité dans les Applications Spring Boot

Le lien git : Lien vers le projet GitHub

1.1 Introduction

L'adaptation aux évolutions constantes dans le développement logiciel nécessite une capacité robuste de suivre et d'optimiser les performances des applications. Le logging et l'observabilité permettent de diagnostiquer les problèmes et d'améliorer l'efficacité des systèmes. Ce rapport explore le rôle fondamental du logging et de l'observabilité dans le développement logiciel moderne et leur impact sur l'amélioration de la compréhension de l'état interne des systèmes.

1.2 Contexte

Le logging et l'observabilité sont essentiels pour répondre aux défis de performance, de fiabilité et de scalabilité des applications modernes. Le logging a évolué vers des méthodes structurées, et l'observabilité offre une compréhension globale des systèmes via logs, métriques et traces. L'émergence de normes telles qu'OpenTelemetry souligne l'importance d'une stratégie de logging et d'observabilité bien établie.

1.3 Problématique

La question est de savoir comment implémenter et optimiser les pratiques de logging et d'observabilité pour une meilleure visualisation et analyse des données de performance et d'utilisation, en prenant en compte les spécificités des environnements backend et frontend.

1.4 Logging d'une Application Backend Java pour le Profilage Utilisateur

1.4.1 Implémentation du Backend

L'objectif était de concevoir une application backend Java simplifiée pour la gestion des utilisateurs et des produits, tout en intégrant des fonctionnalités de journalisation avancées. Les développements ont été réalisés conformément au diagramme de classes présenté à la Figure 1.1, assurant ainsi une gestion efficace et un suivi détaillé des opérations via le logging.

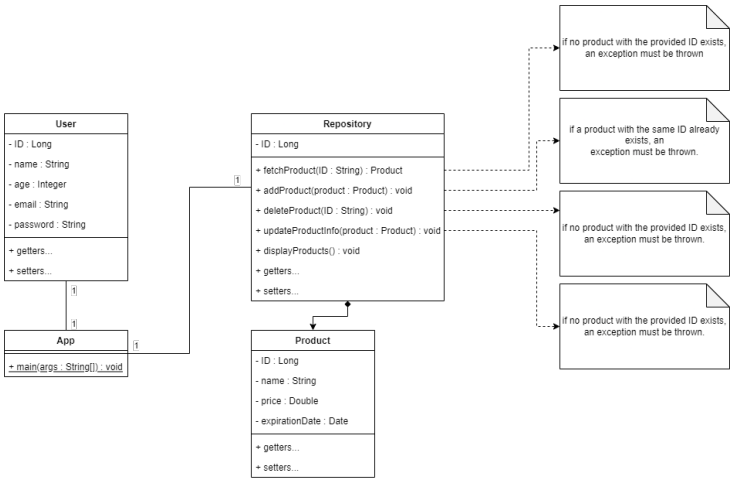

FIGURE 1.1 – Diagramme de classe de l'application

La structure de l'application est illustrée dans la Figure 1.2, montrant l'agencement des composants principaux et leur organisation au sein du projet.

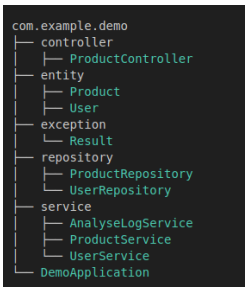

FIGURE 1.2 – Structure de l'application backend

Nous avons mis en œuvre toutes les méthodes requises pour le projet. Les Figures 1.3 et 1.4 illustrent respectivement les méthodes `authenticateUser()` pour l'authentification des utilisateurs et `getAllProducts()` pour récupérer la liste des produits.

FIGURE 1.3 – Méthode d'authentification des utilisateurs

FIGURE 1.4 – Méthode pour obtenir la liste des produits

1.4.2 Outil Logging (SLF4J)

1.4.2.1 Definition

SLF4J (Simple Logging Facade for Java) est une façade de logging pour Java, conçue pour fournir une interface de logging simple mais extensible aux applications Java. Elle sert d'abstraction pour différentes bibliothèques de logging, telles que Log4j, java.util.logging, et logback, permettant aux développeurs de plugger n'importe laquelle de ces bibliothèques sous-jacentes au runtime sans avoir à changer le code source de l'application.

SLF4J fournit un ensemble minimal mais suffisant de fonctionnalités de logging, notamment la capacité de logger des messages à différents niveaux de sévérité (comme ERROR, WARN, INFO, DEBUG, et TRACE). Cela permet aux développeurs de catégoriser et de filtrer les logs en fonction de leur importance ou de leur utilité.

1.4.2.2 Pourquoi SLF4J

— Indépendance vis-à-vis de l'Implémentation.
— Simplicité et Facilité d'Utilisation, ce qui rend l'intégration du logging dans les applications Java rapide et sans complication.
— LF4J est souvent considérée comme une meilleure pratique dans le développement Java car elle offre une interface de logging standardisée et bien soutenue par la communauté.
— SLF4J est compatible avec plusieurs frameworks et bibliothèques Java, ce qui en fait un choix idéal pour de nombreux environnements de développement.
— SLF4J, les développeurs peuvent facilement basculer entre différentes bibliothèques de logging ou utiliser plusieurs d'entre elles dans le même projet si nécessaire.
— SLF4J est conçu pour être performant, avec un faible impact sur les performances de l'application, notamment grâce à son mécanisme de substitution de chaîne de caractères efficace.

1.5 La traçabilité avec des logs

Pour notre projet Spring Boot, nous avons choisi d'utiliser Spoon, un outil d'instrumentation de code Java, pour intégrer des déclarations d'impression de logs (LPS). Cette méthode nous a permis d'enregistrer de manière précise les interactions des utilisateurs avec notre base de données et de profiler les utilisateurs en fonction de leurs activités.

Nous avons défini trois profils d'utilisateurs basés sur leurs interactions avec la base de données. Pour illustrer notre architecture, nous avons représenté notre approche comme le montre la Figure 1.5.

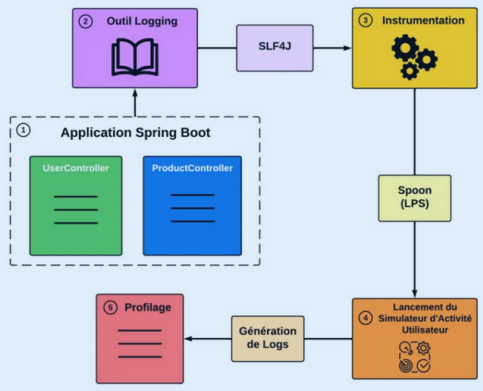

FIGURE 1.5 – Schéma de l'application de traçabilité des logs avec Spoon et LPS

1.5.1 Instrumentation avec Spoon LPS

Spoon est un outil d'analyse et de transformation de code Java. Il est ici utilisé pour l'instrumentation, modifiant dynamiquement le code source pour y intégrer des instructions de logging. Cela permet de journaliser de manière structurée les actions effectuées par les contrôleurs de notre application.

Un 'Processor' de Spoon a été implémenté pour générer automatiquement le code nécessaire à l'instrumentation avec SLF4J, comme illustré dans l'exemple de la Figure 1.6.

```
public class LoggingProcessor extends AbstractProcessor<CtMethod<?>> {

    @Override
    public void process(CtMethod<?> method) {
        CtClass<?> parentClass = method.getParent(CtClass.class);
        if (parentClass != null && "ProductController".equals(parentClass.getSimpleName())) {
            String methodName = method.getSimpleName();

            // Ignorer les méthodes spécifiques de User (authentification , add user, getUser, deconnexion)
            if (isUserMethod(methodName)) {
                return; // Ignorer cette méthode
            }
            addLoggerFieldsIfAbsent(parentClass);
            if (isGetProductByIdMethod(method)) {
                // for price logs
                replaceGetProductByIdMethod(method);
            } else {
                //handleMethodLogging(method, parentClass);
                // for write & read methodes
                handleReadWriteMethodLogging(method);
            }
        }
    }
```

FIGURE 1.6 – La méthode LoggingProcessor Spoon

Les logs générés grâce à Spoon sont illustrés dans la Figure 1.7, mettant en évidence les informations clés extraites lors de l'exécution des méthodes.

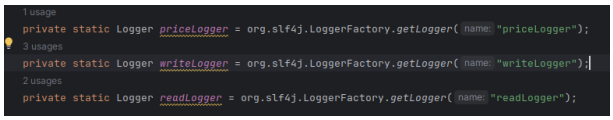

FIGURE 1.7 – Les logs générés par l'instrumentation Spoon dans productController

La méthode 'getAllProducts()' pour récupérer la liste des produits est présentée dans la Figure 1.8 apres l'instrumentation avec spoon.

FIGURE 1.8 – La méthode getAllProducts() apres instrumentation avec spoon

Les LPS ont été spécifiquement conçus pour capturer le contexte d'opération, à savoir le comment, le quand, le où, le quoi et le pourquoi, fournissant ainsi une richesse de données pour le profilage et l'analyse ultérieurs.

1.5.2 Lancement du Simulateur d'Activité Utilisateur

Pour tester le bon fonctionnement de notre système de logging, nous avons definit plusieurs utilisateurs de system. Ces simulateurs (utilisateurs) ont permis de générer des interactions fictives avec notre système afin de valider la journalisation des opérations et d'assurer l'intégrité de notre instrumentation.

Nous avons conduit des tests avec plusieurs profils d'utilisateurs fictifs pour recueillir diverses déclarations de logs (LPS) enregistrées dans des fichiers .log.

Les logs suivants montre des exemples de données journalisées recueillis lors de ces tests, illustrant les actions effectuées par les utilisateurs du système.


```
{
  "timestamp": "2024-01-08T19:32:26.073",
  "level": "INFO",
  "logger": "readLogger",
  "file": "ProductController.java",
  "message": "getProducts",
  "User": "douda",
  "UserID": "658fe6aa72ab2f6fdb6e127a"
}

{
"timestamp": "2024-01-04T22:15:58.042",
"level": "INFO",
"logger": "priceLogger",
"file": "ProductController.java",
"message": "getProductById",
"User": "Gaetan",
"UserID": "659570b852f4d900d2ac7706",
"Product": "Ordinateur",
"ProductID": "659591feff910f65b555f96f",
"Price": 1300.99
}
```


Ces logs démontrent l'efficacité de notre implémentation de logging, en nous permettant de tracer avec précision les activités des utilisateurs et d'identifier des points d'intérêt spécifiques pour le profilage et l'analyse des performances.

1.5.3 Profilage :

Les données de journalisation collectées peuvent être utilisées pour le profilage, où les données sont analysées pour comprendre le comportement des utilisateurs ou pour diagnostiquer des problèmes dans l'application.

nous avons adopté une approche systématique pour analyser les logs générés et construire des profils d'utilisateurs détaillés. Cela implique les étapes suivantes :

- **Parsing des Logs :** Nous avons utilisé un parser JSON pour convertir chaque ligne des fichiers de log en objets JSON. Ces objets JSON permettent une manipulation aisée des données de log et facilitent l'extraction des informations pertinentes.
- **Reification des LPS :** Chaque log est considéré comme une instance de Log Printing Statement (LPS). Nous avons défini un modèle pour les LPS qui inclut les champs comme le timestamp, l'identité de l'utilisateur, le type d'événement, et les informations supplémentaires de l'action.
- **Construction des LPS :** En utilisant le patron de conception Builder, nous avons implémenté une classe UserProfileBuilder qui permet de construire progressivement un profil utilisateur en ajoutant des informations à chaque fois qu'une action est détectée dans les logs.
- **Agrégation des LPS :** Pour chaque utilisateur, les actions sont comptabilisées et agrégées pour former un profil utilisateur complet, qui inclut le total des actions lues, écrites et les recherches des produits les plus chers.
- **Stockage et Affichage des Profils :** Les profils utilisateurs agrégés sont stockés sous forme d'objets JSON, ce qui facilite leur sérialisation pour le stockage dans des systèmes de fichiers ou des bases de données. Ils peuvent également être affichés à travers des interfaces API pour une consultation et une analyse ultérieures.

Le résultat de ce processus est une collection de profils utilisateurs qui reflète fidèlement les activités de chaque utilisateur au sein de l'application sont presenter dans la figure 1.9, permettant ainsi une analyse comportementale et une meilleure prise de décision basée sur les données.

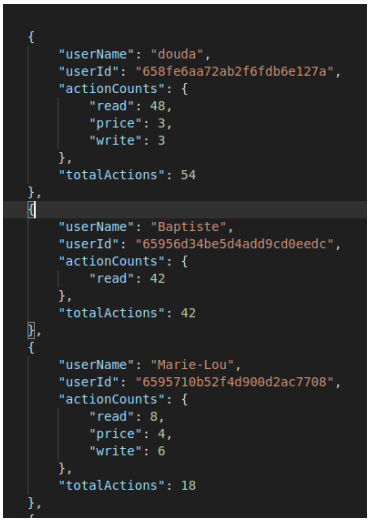

FIGURE 1.9 – Les profiles utilisateurs

En résumé, ce schéma montre comment une application Spring Boot est équipée d'un système de journalisation à l'aide de SLF4J et comment Spoon est utilisé pour injecter automatiquement des instructions de logging dans le code source de l'application. Les

logs générés sont ensuite utilisés pour le profilage, ce qui pourrait impliquer l'analyse des patterns d'utilisation, la performance de l'application, ou d'autres métriques importantes pour les développeurs.

1.6 Structure de l'application

La structure de notre projet semble bien organisée et suit les conventions typiques d'un projet Spring Boot. Voici un résumé des points saillants de votre structure de projet :

- **Controller :** Contient les contrôleurs (ProductController, ProfileController). Ils traitent les requêtes HTTP entrantes.
- **Entity :** Contient vos entités (Product, User, UserProfile).
- **Exception :** Cet emplacement serait utilisé pour les classes d'exception personnalisées comme la classe Result.
- **Repository :** Contient les interfaces de repository (ProductRepository, UserRepository).
- **Service :** Contient les classes de service (AnalyseLogService, ProductService, UserService). Les services contiennent la logique métier et l'interaction avec les repositories.
- **Spoonparsers :** ce dossier contient des classes liées au parsing ou à la manipulation de code avec la bibliothèque Spoon comme LoggingProcessor et SpoonRunner.
- **Builder :** il contient la classe UserProfileBuilder.
- **DemoApplication :** C'est la classe principale Spring Boot.

Cette structure de l'application est representer comme le montre la figure 1.10.

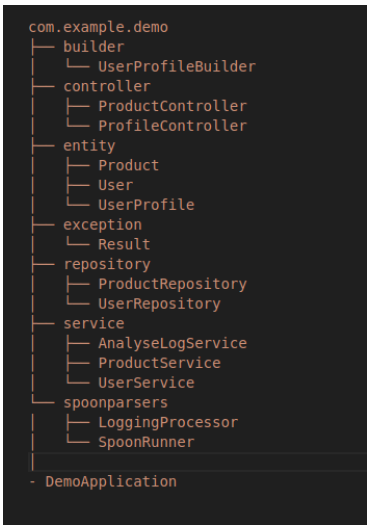

FIGURE 1.10 – Structure de l'application back-end

1.7 Conclusion

En Conclusion, ce rapport a démontré l'importance cruciale d'un système de logging et d'observabilité sophistiqué dans le développement d'applications modernes. L'utilisation de SLF4J pour le logging et de Spoon pour l'instrumentation du code a permis de créer des profils d'utilisateurs détaillés, améliorant ainsi la compréhension et la surveillance de

l'application. Cependant, l'efficacité de ces outils repose sur leur intégration judicieuse, soulignant la nécessité d'une stratégie de logging équilibrée qui optimise à la fois la collecte de données et les performances de l'application. La mise en œuvre réussie de ces pratiques souligne leur rôle essentiel dans le maintien de la qualité et de la fiabilité des systèmes logiciels.

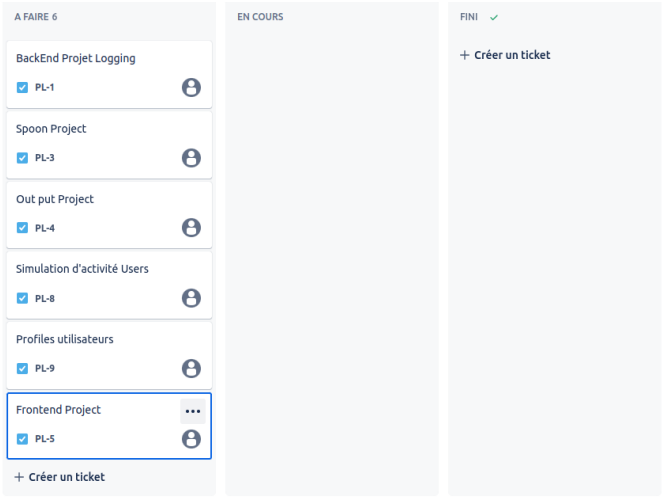

FIGURE 1.11 – Les taches de la premiere semaine

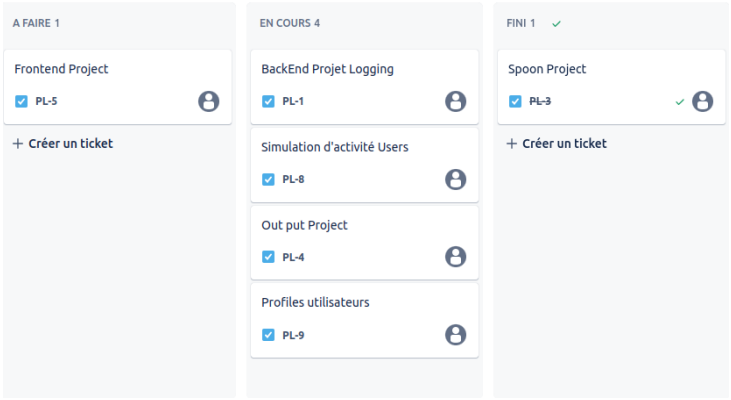

FIGURE 1.12 – Les taches de la deuxieme semaine

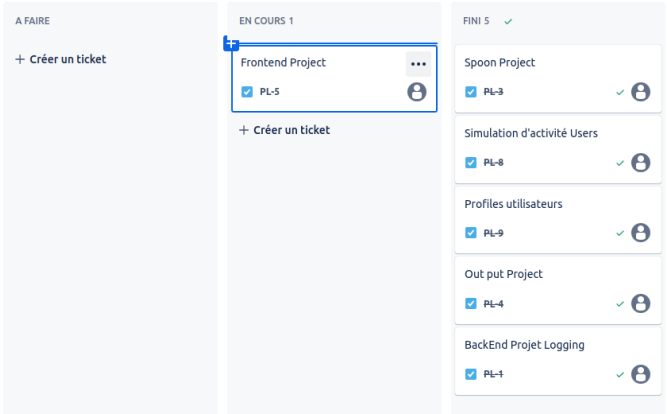

FIGURE 1.13 – Les taches de la troisieme semaine