Document de l'architecture logicielle pour une Application de Gestion de Portefeuille Financier

YANG Chen

29/11/2023

1 Introduction

Ce document présente l'architecture logicielle proposée pour une application de gestion de portefeuille financier. L'architecture suit le modèle MVC (Modèle-Vue-Contrôleur) pour une séparation claire des préoccupations et une meilleure maintenabilité du code.

2 Couche Frontale (Côté Client) :

2.1 Interface Utilisateur (UI) :

— Développez avec JavaFX pour créer une interface graphique utilisateur réactive et intuitive. L'UI suivra le modèle MVC et inclura :

Une interface de création et de gestion de portefeuille (Vue).
Un affichage de la valeur des actifs en temps réel (Vue).
Une fonctionnalité de consultation de données historiques (Vue).
Une visualisation graphique des actifs et des valeurs du portefeuille (Vue).
Des paramètres utilisateur pour la personnalisation et la gestion des préférences (Contrôleur).

2.2 Modèle MVC :

— Le modèle MVC sera utilisé pour séparer les préoccupations dans l'interface utilisateur :

Modèle : Gère les données et la logique de l'application. Interagit avec la base de données et effectue des opérations telles que les requêtes et les mises à jour des données.

Vue : Présente les données à l'utilisateur dans un format spécifique et s'occupe de l'affichage de l'interface utilisateur.

Contrôleur : Fait le lien entre l'utilisateur et le système, gérant l'entrée de l'utilisateur, les interactions avec le modèle et la mise à jour de la vue.

3 Couche Backend (Côté Serveur) :

3.1 Interface de Programmation d'Application (API) :

— Utilisez le framework Java Spring pour construire des API RESTful qui :

Gèrent l'inscription des utilisateurs, la connexion et la gestion des profils (Contrôleur).

Prendent en charge les opérations de portefeuille telles que la création, la gestion, le clonage, l'ajout/suppression d'actifs (Modèle).

Intègrent avec des fournisseurs de données de marché financier externes via des API publiques (Modèle).

Permettent l'importation de données à partir de plateformes de transaction externes (par exemple, Coinbase, Binance) (Modèle).

4 Couche de Stockage des Données :

4.1 Base de Données :

— Utilisez SQLite pour stocker les informations utilisateur, les données de portefeuille, les actifs et les enregistrements de transactions. Ce choix offre une solution légère et facile à déployer pour les applications de petite à moyenne taille.

4.2 Cache Local :

— Implémentez des mécanismes de mise en cache pour stocker temporairement les données fréquemment accédées afin d'améliorer la performance et l'efficacité. Le cache sera géré par le Modèle, fournissant un accès rapide aux données pour la Vue.

5 Couche Intermédiaire :

5.1 Middleware de Sécurité :

— Incluez des mesures de sécurité telles que :

Le chiffrement des données sensibles au repos et en transit pour protéger les informations financières et personnelles des utilisateurs.

Une authentification et une autorisation sécurisées pour l'accès à l'application, utilisant des mécanismes tels que les jetons JWT ou OAuth.

5.2 Middleware de Performance :

— Intégrez des outils pour assurer :

Des temps de réponse rapides pour les interactions de l'UI, en utilisant des techniques telles que le chargement paresseux et l'optimisation des requêtes.

Des mises à jour régulières et ponctuelles des données financières, idéalement avec un délai d'une minute, grâce à des systèmes de messagerie ou de file d'attente asynchrone.

6 Infrastructure de Développement & Déploiement :

6.1 Système de Contrôle de Version :

— Emploie Git avec une plateforme comme GitHub pour la gestion du code source, avec une stratégie claire de branching et de pull request. Cela facilite la collaboration entre les développeurs et le suivi des changements.

6.2 Intégration Continue/Déploiement Continu (CI/CD) :

— Mettez en place des pipelines CI/CD pour les tests automatisés et le déploiement. Cela permettra d'assurer la qualité du code et de faciliter les déploiements réguliers sur les environnements de production.

7 Tests & Documentation :

7.1 Tests Unitaires :

— Utilisez JUnit ou des cadres similaires pour écrire des tests unitaires couvrant les fonctionnalités principales et les cas limites. Les tests unitaires seront essentiels pour le Modèle et le Contrôleur, garantissant que la logique métier et les interactions de l'utilisateur fonctionnent comme prévu.

7.2 Documentation :

— Maintenez une documentation de code complète en utilisant Javadoc et des commentaires en ligne pour les segments de logique complexes. Cela aidera les nouveaux développeurs à comprendre rapidement le code et facilitera la maintenance à long terme de l'application.