AMT Projet Rapport final - Plateforme De Trading

Bijelic Alen & Bogale Tegest

Professeur : Chapuis Bertil
Assistant : Gambin Dorian

28.01.2024

Table des matières

Plateforme de trading

Introduction

Pour ce dernier laboratoire, nous avons mis en place une plateforme fictive d'échange de crypto-monnaies.Nous avons choisi d'utiliser la technologie Hilla, qui nous a permis de créer un Frontend en React avec les composants qu'elle intègre et un Backend en Spring Boot. Nous avons ainsi eu l'occasion d'explorer spring, un framework largement utilisé dans de nombreux postes de l'industrie .
Par ailleurs, nous avons implémenté une communication par message en utilisant JMS, notamment pour récupérer les prix des crypto-monnaies.

Description

En tant qu'utilisateur, je peux

- Accéder à la liste de crypto monnaies listées sur le site
- Créer un compte et me connecter
- Acheter et vendre des crypto monnaies
- Ajouter et retirer des fonds

Le site dispose les pages suivantes:

- Page d'accueil avec le choix de se loger ou de s'enregistrer
- Page de login et register suivant le choix fait dans la page d'accueil
- Page de trading avec une grille présentant de manière structurée les crypto-monnaies disponibles sur notre site, avec leurs noms, symboles et prix actuels. En cliquant sur la grille, une interface interactive s'affiche à droite de l'écran pour l'achat ou vente.
    - On peut vendre si on dispose d'un certain nombre de crypto monnaies
    - On peut acheter si on dispose d'assez de fond
- Page de compte
    - permettant à un utilisateur d'ajouter ou de retirer ses fonds
        - ces fonds sont fictifs, on "retire" autant qu'on peut et on "ajoute" autant qu'on veut (1'000'000$ maximum)
        - les fonds seront en dollar
    - un graphique en fromage de l'état actuel de ses possessions
    - Une liste simple des trades effectués

Services

Les services implémentent la logique métier de la plateforme et sont accédé par le frontend. Pour l'accès, cela se fait grâce à l'annotation @BrowserCallable qui permet de générer les fichiers TypeScript pour pouvoir accéder à la méthode. De plus, il faut spécifier les droits d'accès: @PermitAll, @RolesAllowed, @DenyAll, @AnonymousAllowed (

https://hilla.dev/docs/react/guides/security/configuring#security-options) sinon les méthodes ne seront pas accessibles.
Une chose que nous avons remarqué est que l'injection de dépendance n'est pas recommandée directement (en utilisant @Autowired sur un attribut d'une classe) mais il est préférable d'injecter depuis un constructeur (https://stackoverflow.com/questions/40620000/spring-autowire-on-properties-vs-constructor, https://odrotbohm.de/2013/11/why-field-injection-is-evil/).

Repositories

Les repositories permettent d'interroger la base de donnée de trois manières différents: avec les méthodes fourni par la superclasse JpaRepository, de définir une requête selon le nom de la signature de la méthode dans l'interface ou d'ajouter l'annotation @Query sur la signature de la méthode. Nous avons surtout utilisé les deux dernières méthodes. Deux requêtes ont été créées avec l'annotation @Query car nous avions besoin d'effectuer des mapping vers une DTO, ainsi que des calculs de sommes selon le type de trade.

Authentification

L'authentification et les autorisations ont été implémentés lors de la création du projet en spécifiant l'option –auth (https://hilla.dev/docs/react/start/quick#creating-with-basic-security). Nous avons obtenu les entités AbstractEntity, User et Role. Cela nous a permis d'avoir une authentification déjà implémentée et de nous concentrer sur les autres fonctionnalités importantes du projet.

Messaging

JMS est utilisé afin de récupérer les prix des cryptomonnaies listés sur notre site. Pour chaque cryptomonnaies, nous mettons à jour le champ lastPrice dans la table CryptoCurrency. Nous avons utilisé l'API gratuite de Coinbase (https://api.coinbase.com/v2/prices/<SYMBOL>-USD/spot).

Tests

Les tests ont été réalisés avec JUnit et Mockito. JUnit a été utilisé pour tester les Repositories et ainsi vérifier l'insertion et la lecture des données. Nous avons utilisé Mockito pour vérifier le bon fonctionnement des Services, notamment les fonctionnalités essentielles comme le JMS et la mise à jour des prix. Mockito a été utilisé pour mocker la réponse API du prix et la réponse d'un repository.
Nous avions eu des erreurs lors de la mise en place des tests avec des erreurs peu habituelles et cela venait du fait qu'il manquait cette annotation de classe: @TestInstance(TestInstance.Lifecycle.PER_CLASS).

Technologies

Nous avons utilisé le framework ‘Hilla’**Hilla** (https://hilla.dev/) qui utilise React pour le frontend et Spring Boot pour le Backend. L’’API Coinbase (https://coinbase.com/) a été utilisée pour récupérer le prix des cryptomonnaies listées sur notre site.

Par ailleurs, nous avons utilisé la Base de données Postgres et de Spring Data JPA.

Data model

UML

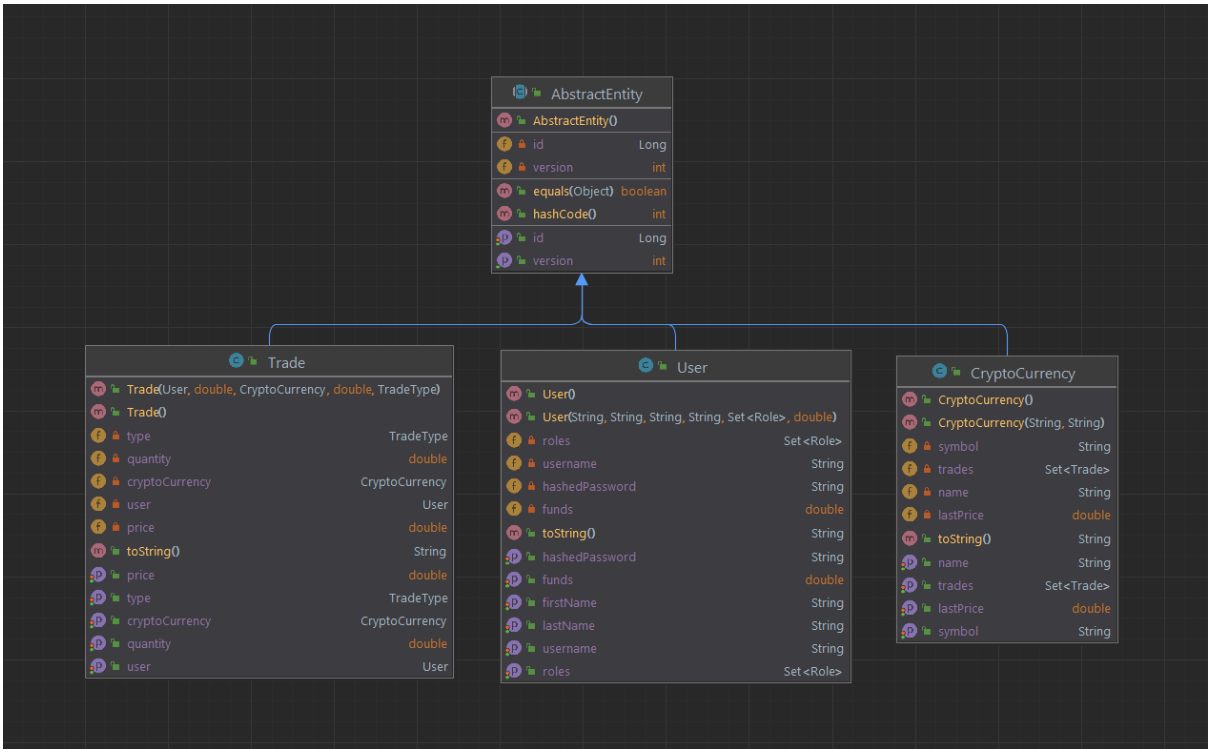
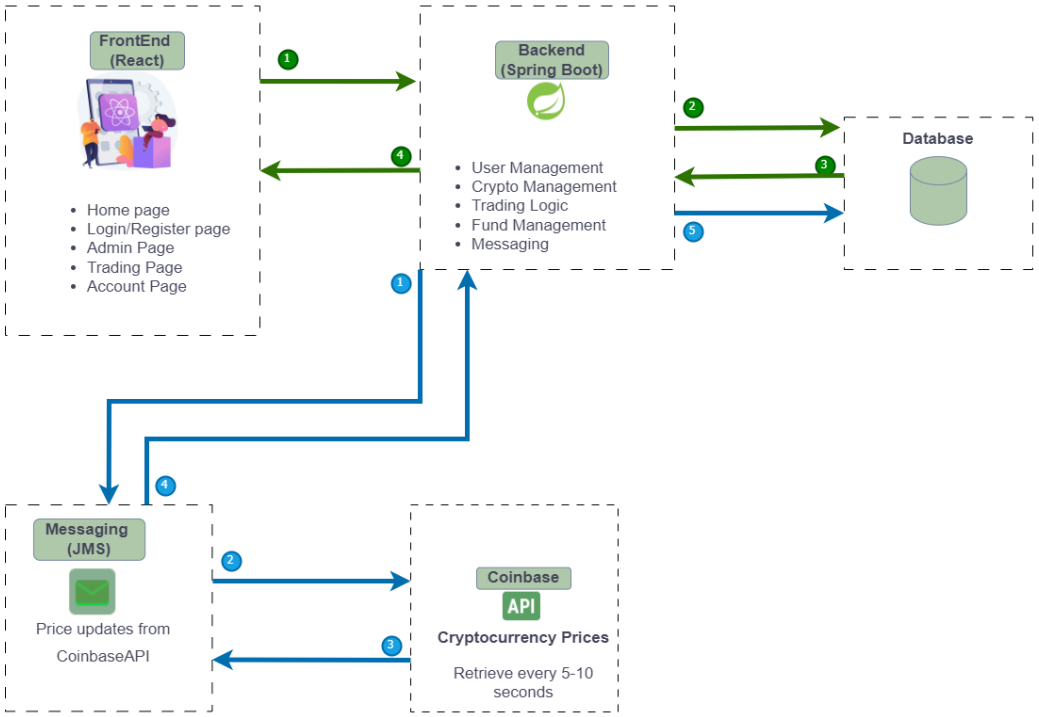


Diagramme d'architecture



Dans ce diagramme, nous avons illustrer deux choses:

- Les communications clients, en vert, par exemple pour récupérer les derniers prix des cryptos (mais convient pour toutes autres actions)
    - Le Frontend effectue une requête GET au Backend (1)
    - Le Backend récupère le dernier prix de la crypto dans la DB (2 et 3)
    - Cette valeur est retournée au Frontend (4)
- Les communications avec l'API Coinbase, en bleu
    - Le Backend va demander le dernier prix (1)
    - Le PriceProducer va aller récupérer le prix grâce à l'API Coinbase(2 et 3)
    - Le PriceConsumer met à jour le prix d'un crypto monnaie (4 et 5)