Dufour Mathias
Apcher Mathéo
Nolland Léo
Chaouche Rayan
Bairouki Yssam
Neveu Benjamin

20 Mars 2025

Plateforme de Gestion de Portefeuilles Financiers en Temps Réel

Projet Scala GSI
Programmation Fonctionnelle

Mme Zaouche Djaouida

Table des matières

Introduction **2**
Algorithmes Financiers **3**
Détection des tendances du marché (MarketTrendDetector) 3
Principe de l'EMA 3
Mise en œuvre dans l’application 4
Calcul de la valeur nette des actifs (NetAssetValueCalculator) 4
Formule du NAV 4
Mise en œuvre dans l’application
Lancement du wallet manager 4
Architecture du projet 10
Déploiement 10
Problème rencontrés 12

Introduction

Avec l'émergence des investissements financiers accessibles (actions, crypto-monnaies, ETF, etc.), les investisseurs ont besoin d'outils modernes pour suivre et analyser leurs portefeuilles en temps réel. Ce projet vise à développer une application financière permettant aux utilisateurs de surveiller la valeur de leurs actifs, de recevoir des notifications sur les événements du marché et de simuler des stratégies d'investissement.

L'application repose sur un système multi-agents basé sur Akka, qui assure une gestion efficace de la concurrence et des mises à jour en temps réel. Une interface web accompagne le système pour offrir une expérience utilisateur fluide et interactive.

L'objectif est de créer un système distribué, résilient et réactif capable d'afficher en temps réel les performances des portefeuilles d'investissement, d'intégrer des flux de données de marché et d'exécuter des algorithmes d'analyse financière. Ces algorithmes permettent de calculer les performances historiques, d'évaluer les risques et la volatilité, et de détecter des opportunités d'investissement à l'aide d'indicateurs financiers comme le RSI, MACD et EMA.

Ce rapport détaillera l'architecture du système, les choix technologiques effectués, ainsi que les algorithmes financiers développés et leurs résultats.

Algorithmes Financiers

L'application implémente plusieurs algorithmes financiers permettant d'analyser les portefeuilles des utilisateurs et de fournir des insights sur les tendances du marché, la valeur des actifs et la rentabilité ajustée au risque. Ces algorithmes utilisent des données de marché en temps réel pour offrir des analyses pertinentes et exploitables

Détection des tendances du marché (MarketTrendDetector)

L'algorithme de détection des tendances repose sur l'Exponential Moving Average (EMA), un indicateur utilisé en analyse technique pour lisser les variations de prix et identifier la direction d'une tendance.

Principe de l'EMA

L'EMA est une moyenne pondérée des prix historiques où les valeurs récentes ont plus d'importance que les anciennes. La formule de l'EMA est :

$$EMA_t = \alpha \cdot P_t + (1 - \alpha) \cdot EMA_{t-1}$$

où :

- EMA_t est l'EMA au temps t,
- P_t est le prix actuel,
- $\alpha = \frac{2}{n+1}$ est le facteur de lissage, avec n la période de calcul.

Mise en œuvre dans l'application

L'algorithme calcule deux EMA :

- EMA court terme (5 périodes)
- EMA long terme (10 périodes)

En comparant ces deux valeurs, il détermine la tendance du marché :

- Si EMA court > EMA long, la tendance est haussière.
- Si EMA court < EMA long, la tendance est baissière.
- Si les valeurs sont proches, la tendance est neutre.

Le système récupère les prix via le FinancialRepository, puis applique cette logique pour détecter les tendances d'un actif.

Calcul de la valeur nette des actifs (NetAssetValueCalculator)

La valeur nette des actifs (Net Asset Value, NAV) est une métrique clé pour évaluer la valeur d'un portefeuille en temps réel. Elle correspond à la somme des valeurs actuelles de tous les actifs détenus par un utilisateur.

Formule du NAV

$$NAV = \sum_{i=1}^{n} P_i$$

où :

- P_i est le prix du $i^{ème}$ actif dans le portefeuille.

Mise en œuvre dans l'application

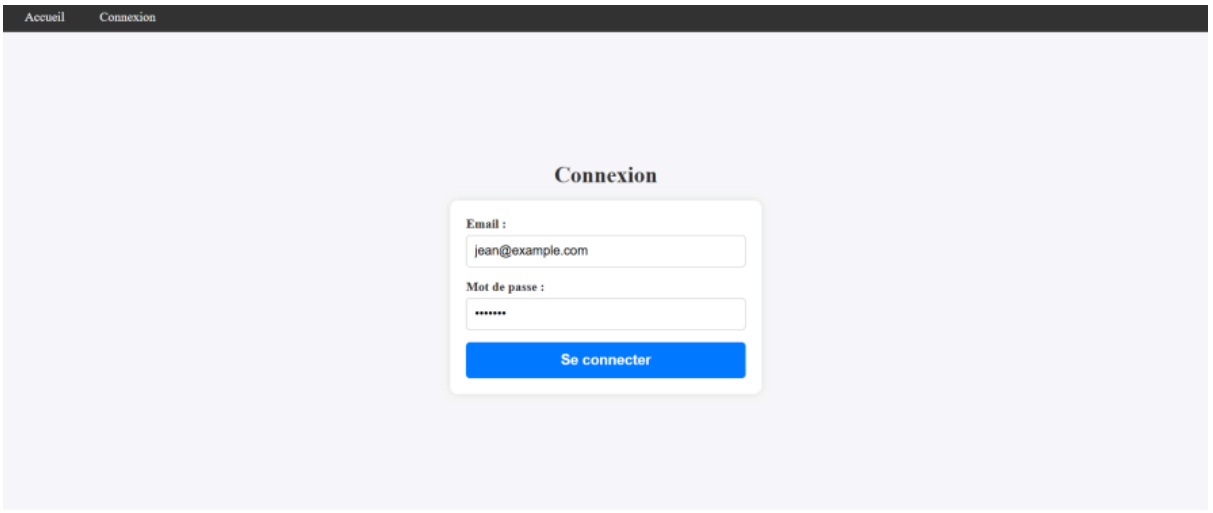
L'algorithme récupère la liste des actifs détenus par un utilisateur via le FinancialRepository, puis additionne les prix actuels de ces actifs pour obtenir la valeur totale du portefeuille. Si un prix n'est pas disponible, une valeur de 0 est utilisée pour éviter les erreurs de calcul.

Cet indicateur est mis à jour en temps réel pour fournir aux utilisateurs une vision précise de la valeur de leurs investissements.

Lancement du Wallet Manager

Page Home :

Page de bienvenue simple et efficace.



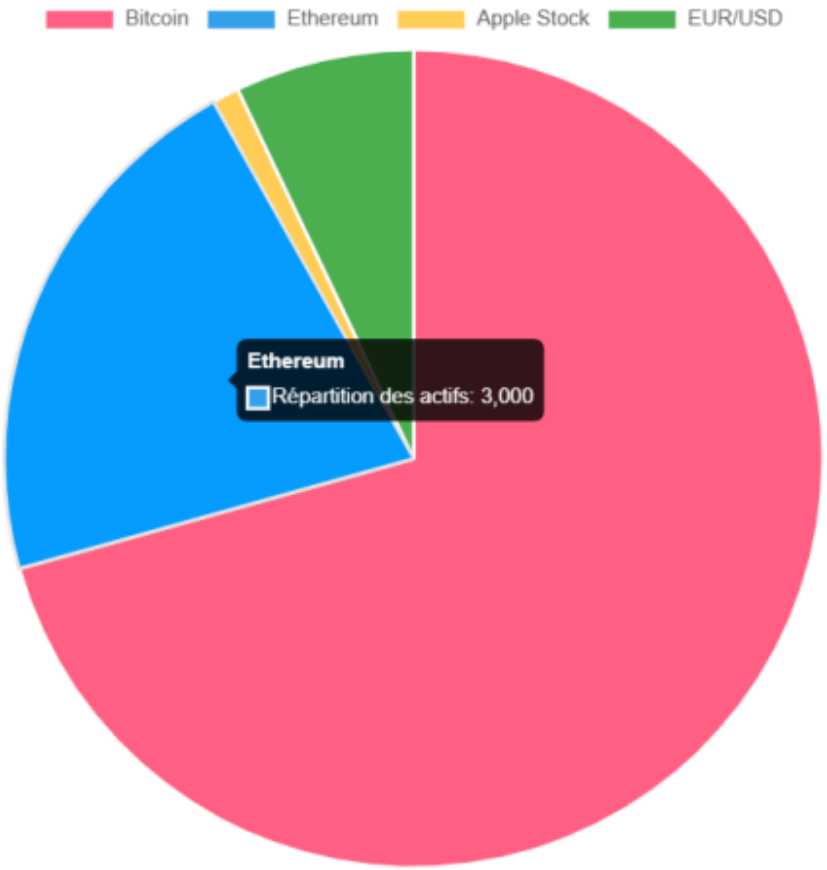
Page Login:

Cette page permet aux utilisateurs de se connecter à leur compte. Elle affiche un formulaire où ils doivent entrer leur email et leur mot de passe. Lors de la soumission, une requête est envoyée à l'API d'authentification pour vérifier les identifiants. Si la connexion réussit, le token et l'ID utilisateur sont enregistrés dans le contexte d'authentification, puis l'utilisateur est redirigé vers le tableau de bord. En cas d'erreur, un message d'avertissement s'affiche.

Page AjouterModifierActif :

Cette page permet aux utilisateurs d'ajouter ou modifier des actifs dans leur portefeuille. Lors du chargement, elle récupère automatiquement la liste des actifs détenus par l'utilisateur via une API, en vérifiant son authentification. L'utilisateur peut sélectionner un actif existant ou en ajouter un nouveau, puis entrer une quantité. Lors de la validation, la page envoie une requête au serveur pour enregistrer ou mettre à jour l'actif, puis actualise les données. Un message de confirmation est affiché en cas de succès.

Répartition des Actifs

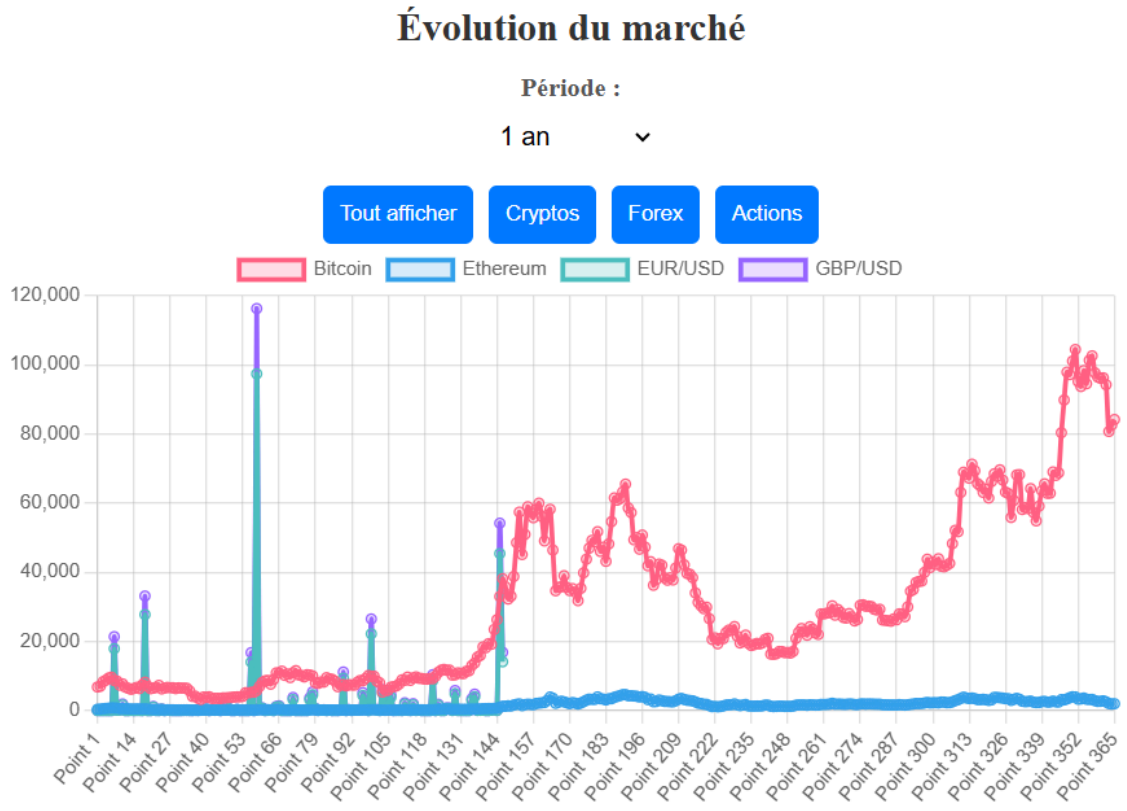


Métriques du Portefeuille

NAV : 14150.00 $
Market Trend : 10000.00

Page Dashboard :

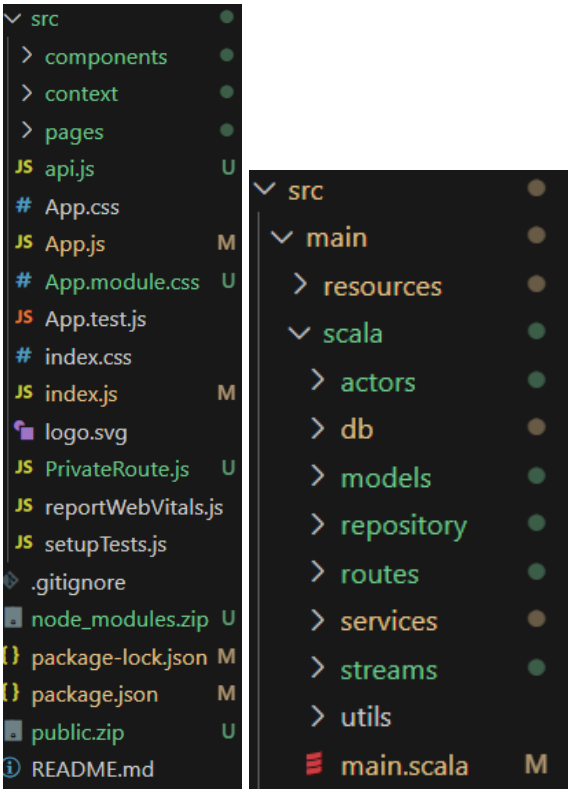
Cette page affiche un tableau de bord pour gérer un portefeuille d'investissement. Elle vérifie si l'utilisateur est authentifié et récupère son portefeuille depuis une API. Si l'utilisateur n'est pas connecté, il est redirigé vers la page de connexion. Le tableau de bord comprend plusieurs visualisations : un graphique de répartition des actifs, des indicateurs financiers du portefeuille, un graphique des tendances du marché et un suivi des mouvements du marché. Un bouton permet d'accéder à la page d'ajout ou de modification d'actifs.



Le code définit un composant React FilteredMarketChart qui affiche un graphique en ligne de l'évolution des actifs financiers (cryptos, forex, actions) en fonction d'une période sélectionnée (1 jour, 1 semaine, 1 mois et 1 an). Il utilise Chart.js pour le rendu du graphique

et le contexte MarketDataContext pour récupérer et gérer les données de marché en temps réel. L'utilisateur peut filtrer les actifs affichés via des boutons de sélection, notamment choisir ou non d'afficher certains placements.

Architecture du projet



L'architecture du projet est divisée en deux parties :

Le **frontend** (première image) est une application **React.js**, organisée avec des dossiers pour les composants, le contexte et les pages. Elle inclut une gestion des routes privées, une API pour la récupération des données et des styles en CSS.

Le **backend** (seconde image) est une application **Scala avec Akka**, structurée en modules : actors pour la gestion concurrente, db pour l'accès à la base de données, models pour les entités, services pour la logique métier et streams pour le traitement des flux de données en temps réel.

Déploiement

L'objectif de ce projet était de conteneuriser un frontend en React, un backend en Scala et

une base de données MySQL, puis de déployer l'ensemble sur Minikube en utilisant Docker et Kubernetes.

Nous avons réussi à créer des images Docker distinctes pour chaque composant de l'application :

Frontend React : Conteneurisé avec une image optimisée pour être servie via Nginx.

Backend Scala : Conteneurisé avec les dépendances nécessaires et exposé sur un port défini.

Base de données MySQL : Utilisation d'une image officielle de MySQL avec des configurations adaptées aux besoins du projet.

Déploiement sur Minikube

Une fois les images construites, nous les avons chargées dans Minikube.

Les manifestes Kubernetes ont été rédigés pour déployer chaque service, notamment les Deployment et Service nécessaires à leur interconnexion.

backend(Docker) :

```
cytech@student-laptop:~/akkaEnd$ sudo docker run scala
[info] welcome to sbt 1.10.10 (Ubuntu Java 11.0.26)
[info] loading project definition from /app/project
[info] loading settings for project app from build.sbt...
[info] set current project to app (in build file:/app/)
[info] running Main
17:23:51.245 [financial-app-akka.actor.default-dispatcher-5] INFO akka.event.slf4j.Slf4jLogger - Slf4jLogger started
Server online at http:///127.0.0.1:8080
```

backend(Minikube) :

```
scala-app-98c4f8ccb-l2rq9          1/1     Running             0               12s
```

base de données :

```
cytech@student-laptop:~/akkaEnd$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS                                                  NAMES
65e26646e5c5   mysql:8.0  "docker-entrypoint.s…"   10 minutes ago   Up 10 minutes   0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp   mysqlakka
```

frontend (Docker):

```
mon-app-react                 latest    54cecf163187   About a minute ago   50.4MB
```

frontend(Minikube)

```
mon-app-react-8d64d774c-nk657      1/1     Running             0               3s
```

```
cytech@student-laptop:~/akkaFront$ minikube service react-app-service --url
http://192.168.49.2:32574
```

Problème rencontrés

Gestion des Données et Base de Données

Liaison complexe entre **Scala et MySQL** via JDBC, notamment sur la gestion des connexions et transactions.

Optimisation des requêtes SQL pour éviter les ralentissements lors des mises à jour en temps réel.

Problèmes d'intégration des données historiques et en temps réel sans surcharge du système.

API et Flux de Données

Difficultés à récupérer des données fiables et à jour depuis **CoinGecko, Binance et Alpha Vantage**.

Synchronisation des flux d'Akka Streams avec les mises à jour en base de données.

Gestion des erreurs et des latences API sans perturber l'affichage.

Organisation et Travail en Équipe

Code difficile à structurer proprement à **six développeurs** : conflits Git fréquents, logique parfois redondante.

Coordination compliquée entre le backend en Scala et le frontend React, notamment sur le format des données échangées.

Problèmes d'intégration entre la partie calcul financier (NAV, Sharpe Ratio, EMA) et l'affichage graphique.

2.3. Conteneurisation et Déploiement

Malgré la réussite de la conteneurisation et du chargement des images dans **Minikube**, l'intégration complète reste instable.

Problèmes d'interconnexion entre les services rencontrés, faisant perdre du temps.