PortfolioTracker

Application de suivi de portefeuilles financiers

Rapport Technique

Module : II.1102 — Programmation Java Avancée

Auteur : Adam Houri

Date : Janvier 2026

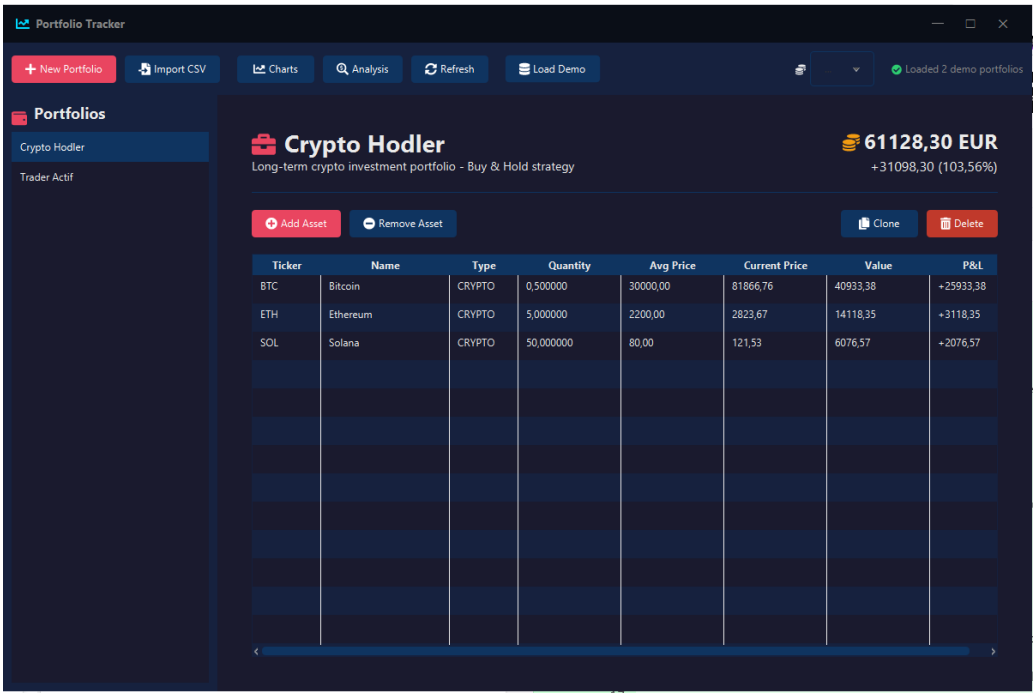

Table des matières

1 Introduction

PortfolioTracker est une application desktop **JavaFX** permettant le suivi de portefeuilles financiers (cryptomonnaies et actions). Elle offre une visualisation en temps réel des positions avec graphiques, import CSV, et fonctionnalités d'analyse avancées.

Aspect	Choix technique
Architecture	MVC + Services (Pattern Singleton)
Interface	JavaFX 21 (FXML + CSS)
Persistance	JSON local (Gson)
APIs	Binance, Yahoo Finance, ExchangeRate
Sécurité	Chiffrement XOR (pédagogique)

TABLE 1 – Synthèse des choix techniques

2 Fonctionnalités implémentées

2.1 Fonctionnalités principales

Fonctionnalités livrées

- **Gestion multi-portfolios** : création, suppression, clonage, changement de devise
- **Gestion d'assets** : ajout/suppression, transactions BUY/SELL/REWARD/CONVERT
- **Graphiques** : évolution valeur (1W/1M/3M/1Y), allocation pie chart, Compare All
- **Import CSV Coinbase** : parsing automatique des transactions
- **Multi-devises** : EUR, USD, GBP, CHF, JPY

2.2 Fonctionnalités avancées

- **Events sur graphiques** : marqueurs crash/hack/décision personnelle
- **Analysis** : Profit vs Loss Days, Best/Worst Day (30 jours)
- **Whale Alerts** : transactions crypto > $1M des dernières 24h
- **Chiffrement local** : activation via passphrase au démarrage

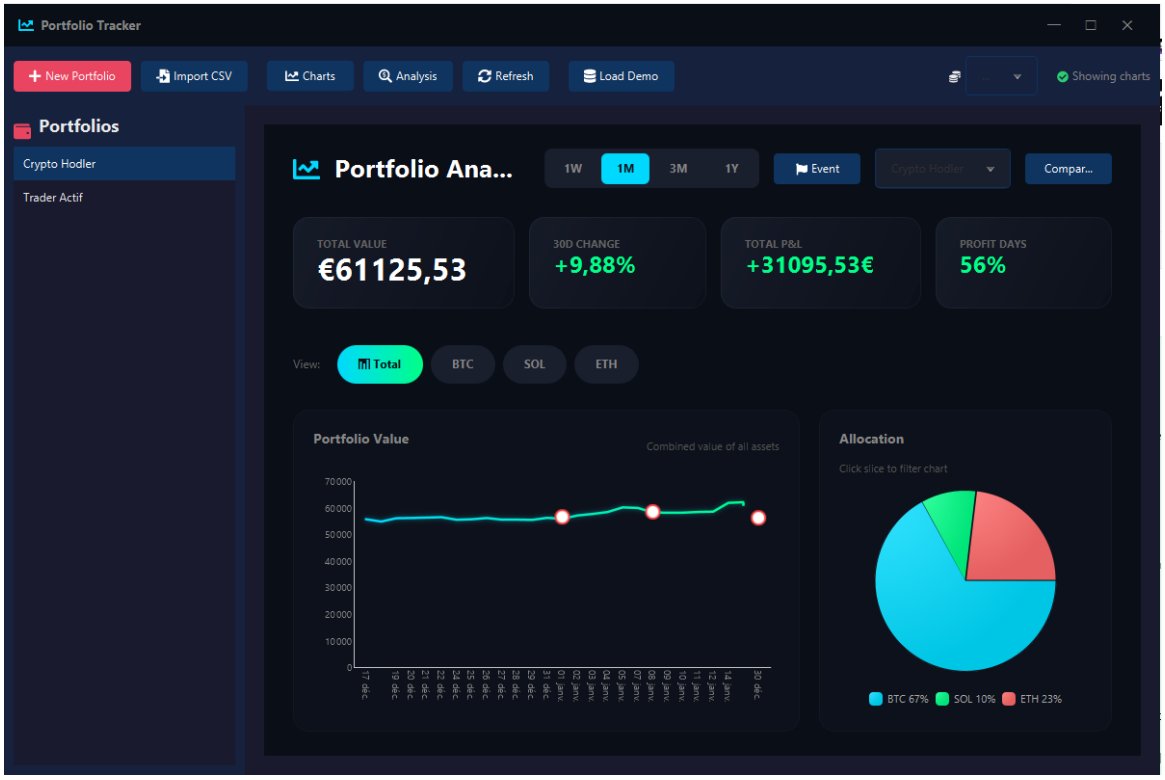

FIGURE 1 – Vue Charts : évolution de la valeur et allocation

3 Architecture logicielle

Le projet suit une architecture **MVC (Model-View-Controller)** avec une couche Service indépendante.

Package	Responsabilité
`com.portfoliotracker.model`	Entités métier (Portfolio, Asset, Transaction)
`com.portfoliotracker.controller`	Orchestration UI, gestion événements
`com.portfoliotracker.service`	Logique métier (Singleton)
`com.portfoliotracker.api`	Clients HTTP externes

TABLE 2 – Organisation des packages

Services implémentés : PortfolioService, MarketDataService, PersistenceService, EncryptionService, CacheService, EventService, AnalysisService, DemoService.

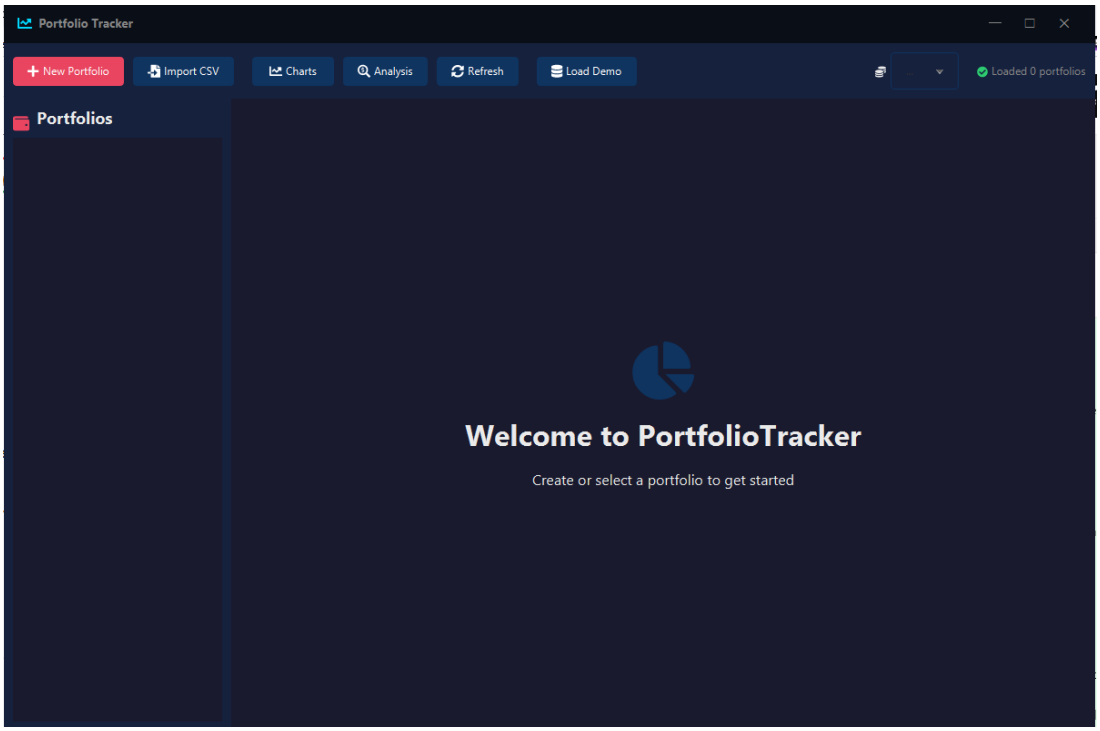

FIGURE 2 – Page principale de l'application

4 Modèle de données

4.1 Portfolio

Conteneur principal avec : `id`, `name`, `description`, `currency`, `createdAt`, et une liste d'`Asset`.

4.2 Asset

Actif financier avec : `ticker`, `name`, `type` (CRYPTO/STOCK), et liste de `Transaction`.
Méthodes de calcul : `getTotalQuantity()`, `getAverageBuyPrice()`, `getTotalInvested()`

4.3 Transaction

Opération avec : `type` (BUY/SELL/REWARD/CONVERT), `quantity`, `pricePerUnit`, `date`, `fees`.

5 Services clés

5.1 Pattern Singleton

Listing 1 – Implémentation Singleton

```
public static PortfolioService getInstance() {
    if (instance == null) {
        instance = new PortfolioService();
    }
    return instance;
}
```

5.2 Cache à deux niveaux

Le `MarketDataService` implémente un cache optimisé :

— **Niveau 1 — Mémoire** : `ConcurrentHashMap` avec TTL de 60 secondes
— **Niveau 2 — Disque** : fichiers JSON persistants pour les prix USD

6 Interface utilisateur

Écran	Fichier FXML	Description
Main	main.fxml	Toolbar + navigation
Portfolio	portfolio-view.fxml	Liste des assets + P&L
Charts	chart-view.fxml	Courbes + allocation
Analysis	analysis-view.fxml	Whale Alerts + statistiques

TABLE 3 – Écrans de l'application

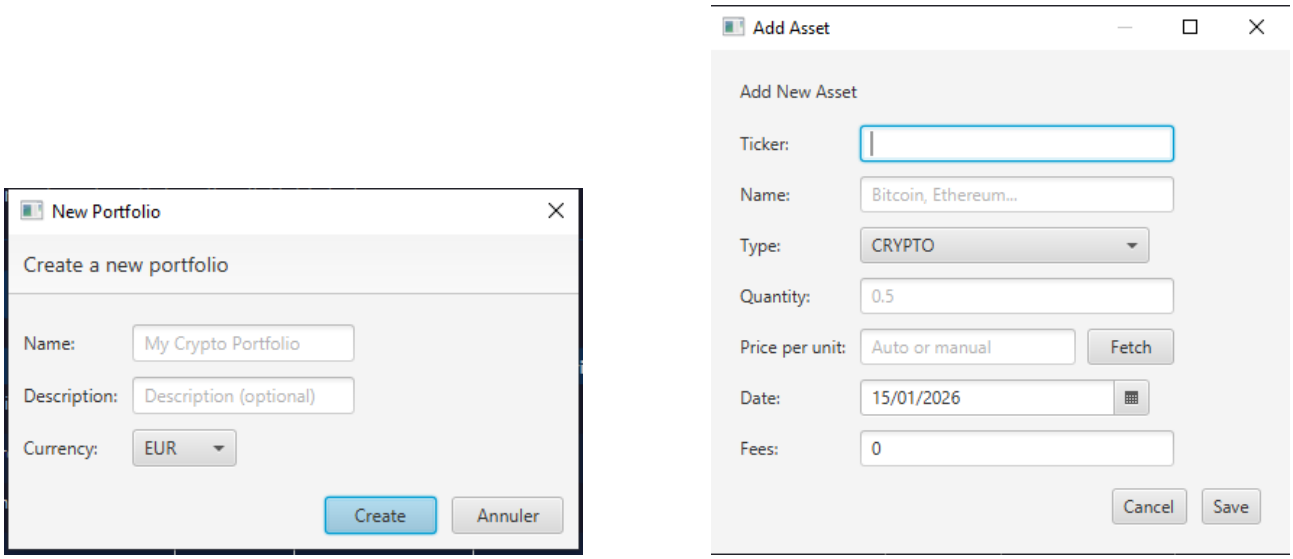

FIGURE 3 – Formulaires d'ajout : Portfolio (gauche) et Asset (droite)

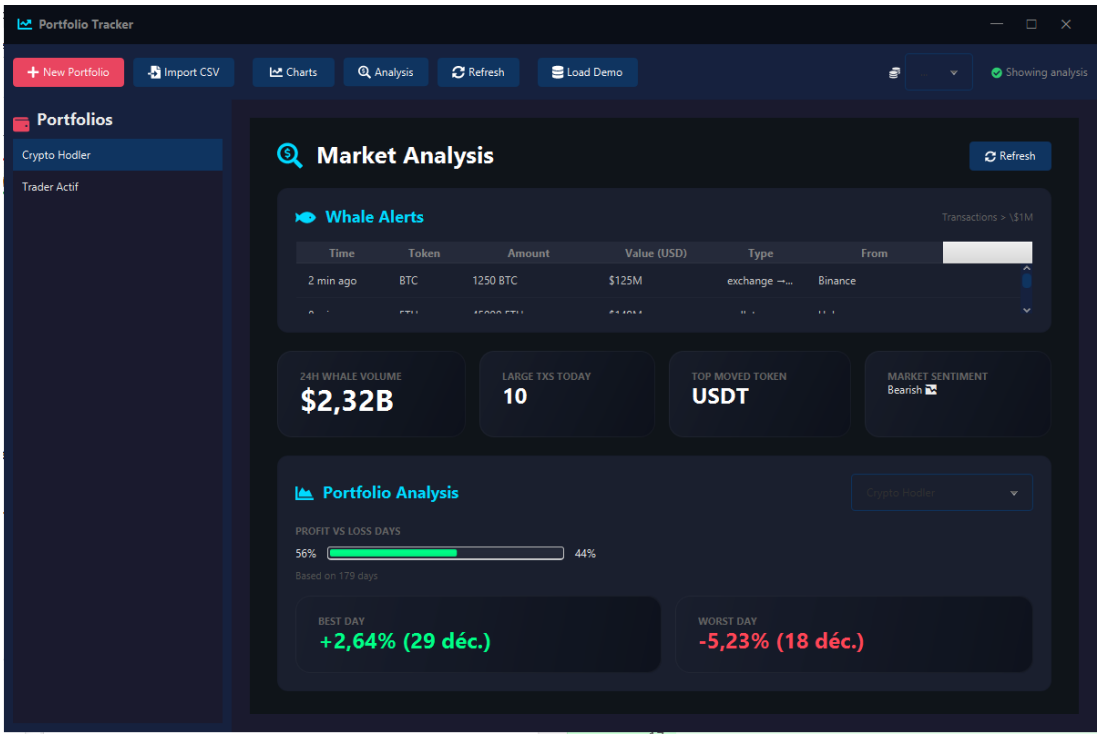

FIGURE 4 – Vue Analysis : Whale Alerts et Portfolio Analysis

7 Concurrence & Performance

Les appels réseau sont exécutés dans des **Tasks JavaFX** pour ne pas bloquer l'interface :

Listing 2 – Chargement asynchrone

```
Task<Map<String, Double>> task = new Task<>() {
    @Override
    protected Map<String, Double> call() {
        // Appels API en arrière-plan
        return prices;
    }
};
task.setOnSucceeded(e -> updateUI());
new Thread(task).start();
```

8 Persistance & Chiffrement

8.1 Structure de stockage

```
data/
 portfolios/     # JSON (ou .enc si chiffré)
 cache/          # Prix historiques
 events/         # Événements utilisateur
```

8.2 Chiffrement XOR

Note pédagogique

Le chiffrement XOR est une implémentation simplifiée à but éducatif. Une application de production nécessiterait AES-256.

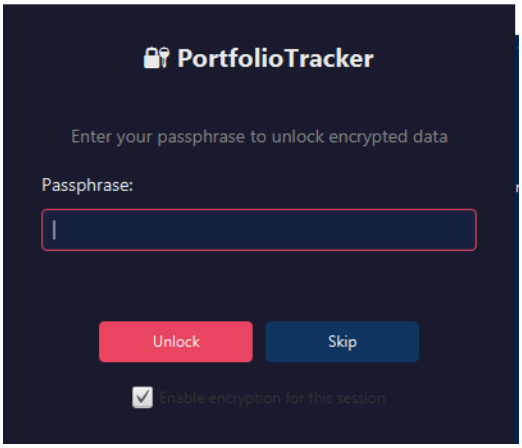

FIGURE 5 – Dialog de passphrase au démarrage

9 Tests unitaires

Classe de test	Nb tests	Objectif
AssetTest	5	Calculs financiers (quantité, prix moyen)
EncryptionServiceTest	6	Round-trip encrypt/decrypt

TABLE 4 – Couverture des tests

```
$ mvn test
Tests run: 11, Failures: 0, Errors: 0
BUILD SUCCESS
```

10 Conclusion

Objectifs atteints

- Gestion multi-portfolios avec visualisation graphique
- Import CSV Coinbase et APIs temps réel
- Fonctionnalités avancées : Analysis, Whale Alerts, Encryption
- Architecture MVC propre, testée et documentée

10.1 Limites et évolutions

Limite actuelle	**Évolution envisagée**
Chiffrement XOR simplifié	Implémenter AES-256
Cache limité à USD	Étendre aux autres devises
Pas de tests UI	Ajouter TestFX

Table 5 – Pistes d'amélioration