Rapport Technique — PortfolioTracker

Auteur : Adam Houri
Date : Janvier 2026
Module : II.1102 — Programmation Java Avancée
Version : 1.0

Table des matières

1. Introduction

PortfolioTracker est une application desktop JavaFX conçue pour le suivi de portefeuilles d'actifs financiers (cryptomonnaies et actions). Elle offre une solution complète permettant aux investisseurs de visualiser, analyser et gérer leurs positions en temps réel.

1.1 Contexte du projet

Ce projet a été réalisé dans le cadre du module **II.1102 — Programmation Java Avancée**. L'objectif était de mettre en pratique les concepts avancés de programmation Java : architecture MVC, programmation concurrente, persistance locale, et intégration d'APIs externes.

1.2 Points clés de l'implémentation

Aspect	Choix technique
Architecture	MVC stricte avec couche Service
Interface	JavaFX (FXML + CSS)
Persistance	JSON local (Gson)
Concurrence	Tasks JavaFX asynchrones
Cache	Double niveau (mémoire + disque)
Sécurité	Chiffrement XOR (pédagogique)

1.3 Sources du document

- Code source du projet (packages `model`, `controller`, `service`, `api`, `util`)
- Vues JavaFX (fichiers FXML et CSS)
- Diagrammes UML fournis (classes + séquences)

2. Périmètre & Fonctionnalités livrées

2.1 Fonctionnalités principales

Fonctionnalité	Description	Statut
Gestion multi-portfolios	Création, suppression, clonage, changement de devise	✅ Livré
Gestion d'assets	Ajout/suppression, transactions (BUY/SELL/REWARD/CONVERT)	✅ Livré

Fonctionnalité	Description	Statut
Graphiques	Évolution valeur (1W/1M/3M/1Y), allocation, compare all	✅ Livré
Import CSV Coinbase	Parsing automatique des transactions	✅ Livré
Persistance JSON	Sauvegarde automatique locale	✅ Livré
Multi-devises	Conversion EUR/USD/GBP/CHF/JPY	✅ Livré

2.2 Fonctionnalités avancées

Fonctionnalité	Description	Statut
Events sur graphiques	Marqueurs crash/hack/décision	✅ Livré
Analysis	Profit vs Loss days, Best/Worst day	✅ Livré
Whale Alerts	Transactions > $1M (API + fallback mock)	✅ Livré
Chiffrement local	Activation via passphrase (XOR)	✅ Livré

2.3 Aperçu de l'application

Figure 1 — Vue principale : Portfolio avec assets et P&L

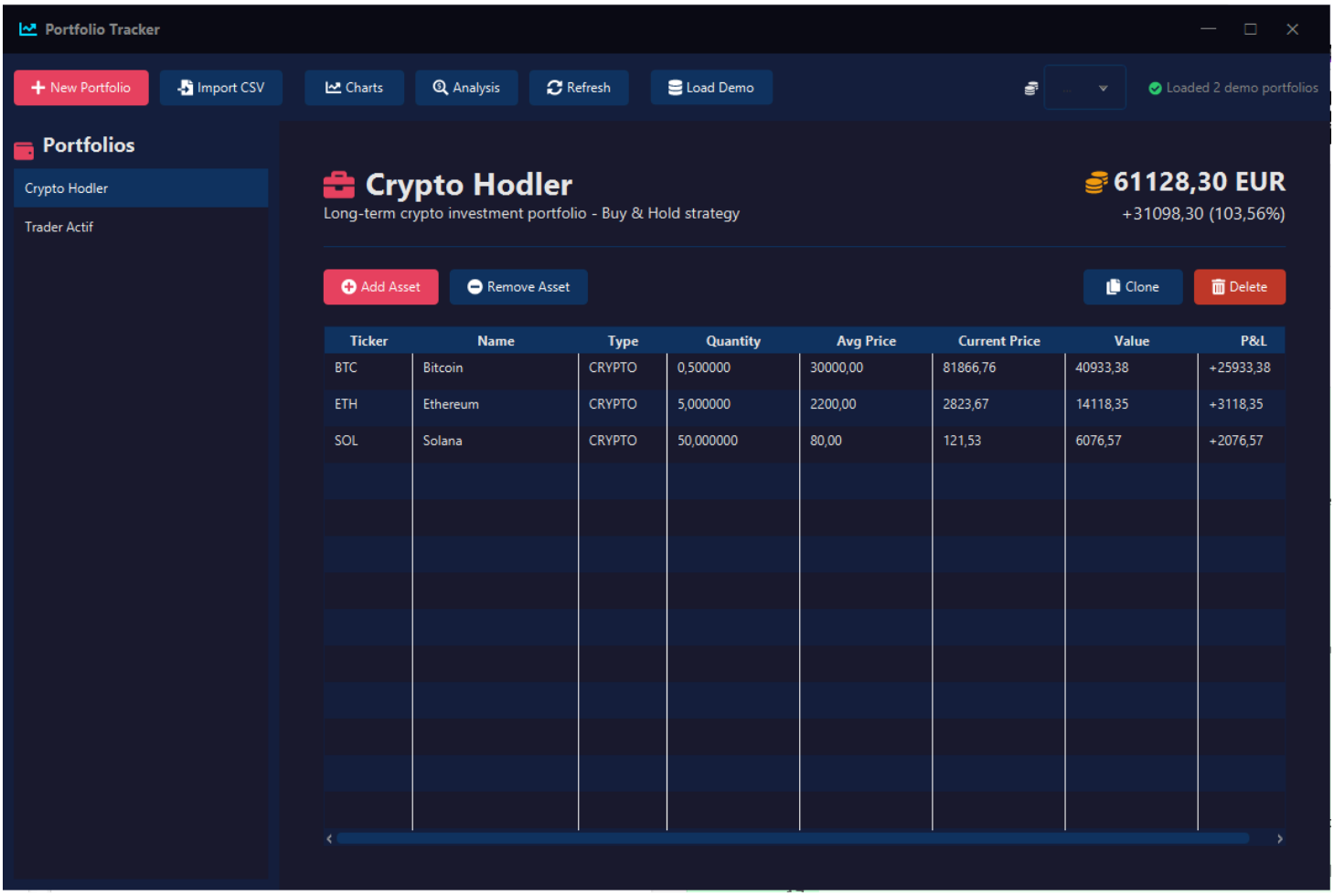


Figure 2 — Vue Charts : Évolution de la valeur et allocation

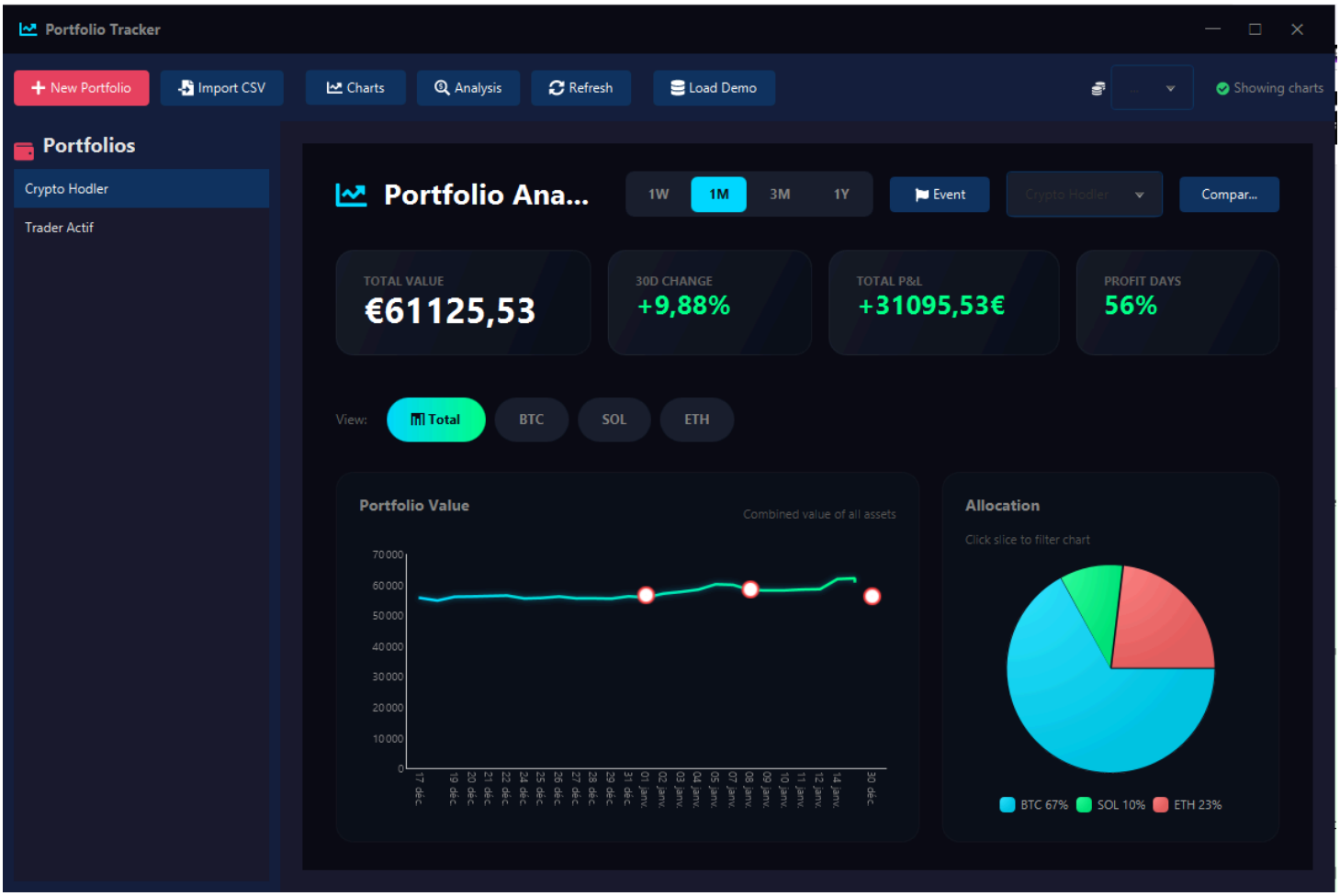
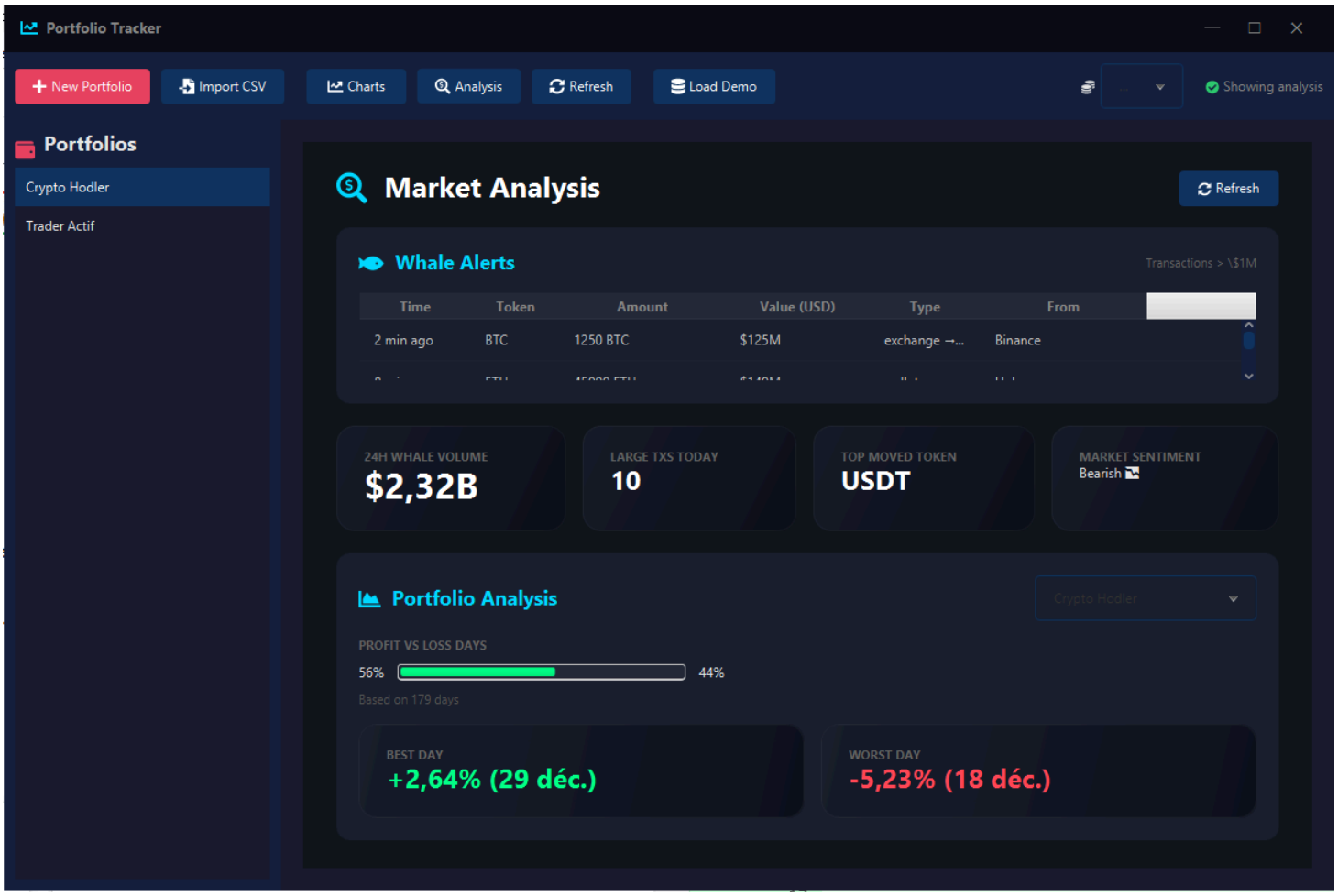


Figure 3 — Vue Analysis : Whale Alerts et statistiques



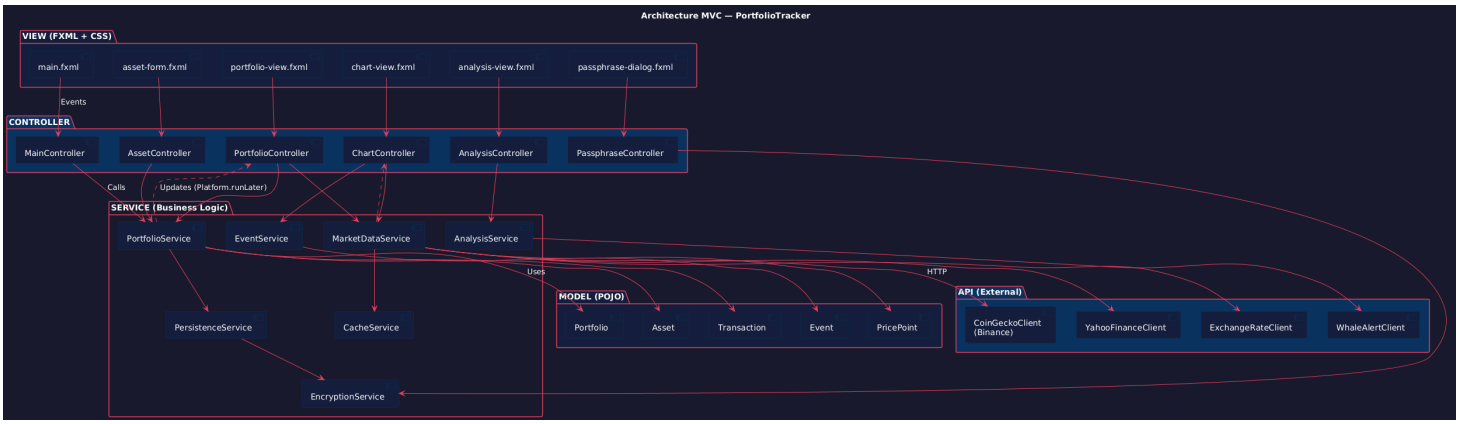
2.4 Éléments hors périmètre (assumés)

- **Sécurité forte** : le chiffrement XOR est volontairement simplifié (objectif pédagogique).
- **Historique transactionnel exact** : les courbes représentent la valeur à quantité actuelle, pas un historique "réel" des achats/ventes.
- **Disponibilité API** : dépendance à des endpoints externes non contractuels.

3. Architecture logicielle

Le projet suit une architecture **MVC (Model-View-Controller)** structurée en couches. L'objectif est de **séparer la présentation**, la **logique métier** et l'**accès aux données**.

Figure — Diagramme d'architecture MVC



3.1 Découpage en couches

```
┌─────────────────────────────────────────────────────────────┐
│                      VIEW (FXML/CSS)                        │
│  main.fxml │ portfolio-view.fxml │ chart-view.fxml │ ...    │
└──────────────────────────┬──────────────────────────────────┘
                           │ Événements UI
                           ▼
┌─────────────────────────────────────────────────────────────┐
│                      CONTROLLER                             │
│  MainController │ PortfolioController │ ChartController     │
│  AnalysisController │ AssetController │ PassphraseController│
└──────────────────────────┬──────────────────────────────────┘
                           │ Appels métier
                           ▼
┌─────────────────────────────────────────────────────────────┐
│                    SERVICE LAYER                            │
│  PortfolioService │ MarketDataService │ PersistenceService  │
│  EncryptionService │ CacheService │ EventService           │
│  AnalysisService │ DemoService                              │
└──────────────────────────┬──────────────────────────────────┘
                           │ Données
                           ▼
┌─────────────────────────────────────────────────────────────┐
│                    MODEL + API LAYER                        │
│  Portfolio │ Asset │ Transaction │ Event │ PricePoint       │
│  CoinGeckoClient │ YahooFinanceClient │ ExchangeRateClient  │
└─────────────────────────────────────────────────────────────┘
```

3.2 Mapping packages → responsabilités

Package	Responsabilité
`com.portfoliotracker.model`	Entités métier (POJO)
`com.portfoliotracker.model.enums`	Énumérations (AssetType, TransactionType)
`com.portfoliotracker.controller`	Orchestration UI + appels services
`com.portfoliotracker.service`	Logique métier + persistance + cache
`com.portfoliotracker.api`	Clients HTTP externes
`com.portfoliotracker.util`	Utilitaires (adapters Gson)

3.3 Flux principal (haut niveau)

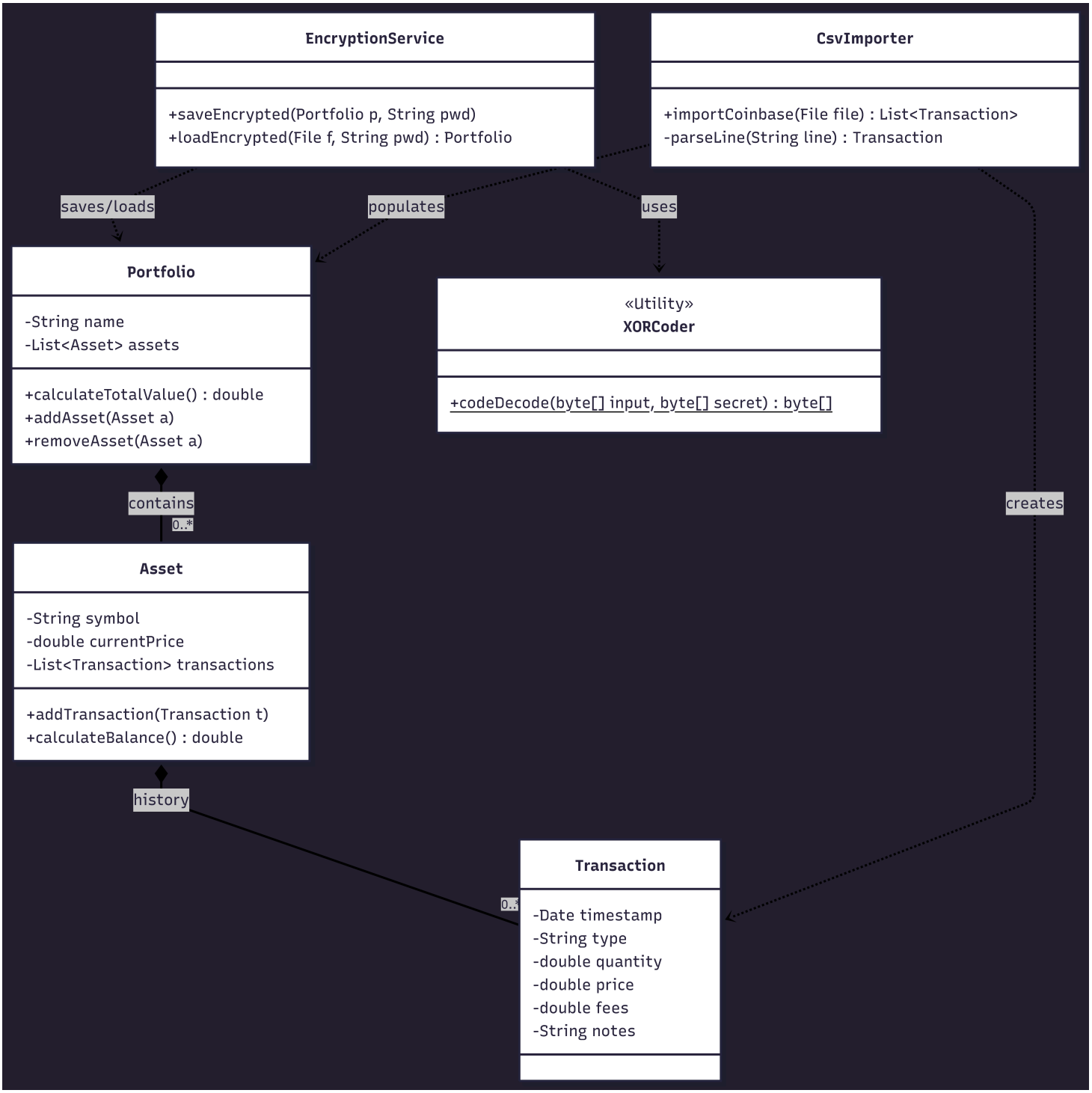
1. L'utilisateur interagit avec une **View** (FXML).
2. Le **Controller** déclenche des actions (ex. ajout d'asset, import CSV).
3. Les **Services** exécutent la logique métier, appellent la persistance ou les APIs.
4. Le **Controller** met à jour l'UI avec les résultats.

3.4 Raisons du choix architectural

- **Découplage** : limiter le couplage entre UI et logique métier.
- **Réutilisabilité** : réutiliser la logique (ex. calculs) sans dépendre d'une vue.
- **Évolutivité** : préparer un éventuel changement d'UI (vers le Web par exemple).
- **Testabilité** : tester la logique métier en isolation.

4. Modèle de données

Figure 4 — Diagramme UML des classes



4.1 Portfolio

Attribut	Type	Description
`id`	`String`	Identifiant unique (UUID)
`name`	`String`	Nom du portfolio
`description`	`String`	Description optionnelle

Attribut	Type	Description
`currency`	`String`	Devise de référence (EUR, USD, etc.)
`createdAt`	`LocalDateTime`	Date de création
`assets`	`List<Asset>`	Liste des actifs

Méthodes clés :

- `addAsset(Asset)` / `removeAsset(String)` — Gestion des actifs
- `getAssetByTicker(String)` — Recherche par ticker
- `clone()` — Deep copy pour simulation

```
public Portfolio clone() {
    Portfolio cloned = new Portfolio();
    cloned.setName(this.name + " (Copy)");
    cloned.setDescription(this.description);
    cloned.setCurrency(this.currency);

    for (Asset asset : this.assets) {
        Asset clonedAsset = new Asset(asset.getTicker(), asset.getName(), asset.getType());
        for (Transaction t : asset.getTransactions()) {
            Transaction clonedTransaction = new Transaction(
                t.getType(), t.getQuantity(), t.getPricePerUnit(),
                t.getDate(), t.getFees(), t.getNotes()
            );
            clonedAsset.addTransaction(clonedTransaction);
        }
        cloned.addAsset(clonedAsset);
    }
    return cloned;
}
```

4.2 Asset

Attribut	Type	Description
`id`	`String`	Identifiant unique
`ticker`	`String`	Symbole (BTC, ETH, AAPL...)
`name`	`String`	Nom complet

Attribut	Type	Description
`type`	`AssetType`	CRYPTO ou STOCK
`transactions`	`List<Transaction>`	Historique des transactions

Méthodes clés de calcul financier :

```
public double getTotalQuantity() {
    double total = 0;
    for (Transaction t : transactions) {
        if (t.getType() == TransactionType.BUY || t.getType() == TransactionType.REWARD) {
            total += t.getQuantity();
        } else if (t.getType() == TransactionType.SELL) {
            total -= t.getQuantity();
        }
    }
    return total;
}

public double getAverageBuyPrice() {
    double totalCost = 0;
    double totalQuantity = 0;
    for (Transaction t : transactions) {
        if (t.getType() == TransactionType.BUY) {
            totalCost += t.getQuantity() * t.getPricePerUnit();
            totalQuantity += t.getQuantity();
        }
    }
    if (totalQuantity == 0) return 0;
    return totalCost / totalQuantity;
}

public double getTotalInvested() {
    double total = 0;
    for (Transaction t : transactions) {
        if (t.getType() == TransactionType.BUY) {
            total += t.getTotalCost();
        }
    }
    return total;
}
```

4.3 Transaction

Attribut	Type	Description
`id`	`String`	Identifiant unique
`type`	`TransactionType`	BUY, SELL, REWARD, CONVERT
`quantity`	`double`	Quantité échangée
`pricePerUnit`	`double`	Prix unitaire
`date`	`LocalDateTime`	Date de la transaction
`fees`	`double`	Frais de transaction
`notes`	`String`	Notes optionnelles

4.4 Event

Attribut	Type	Description
`id`	`String`	Identifiant unique
`name`	`String`	Nom de l'événement
`date`	`LocalDateTime`	Date de l'événement
`portfolioId`	`String`	ID du portfolio (null si global)

4.5 PricePoint

Structure simple pour l'historique des prix :

- `timestamp` : `LocalDateTime`
- `price` : `double`

5. Services (logique métier)

Tous les services suivent le **pattern Singleton** pour garantir un point d'accès unique et centralisé.

5.1 Pattern Singleton — Implémentation

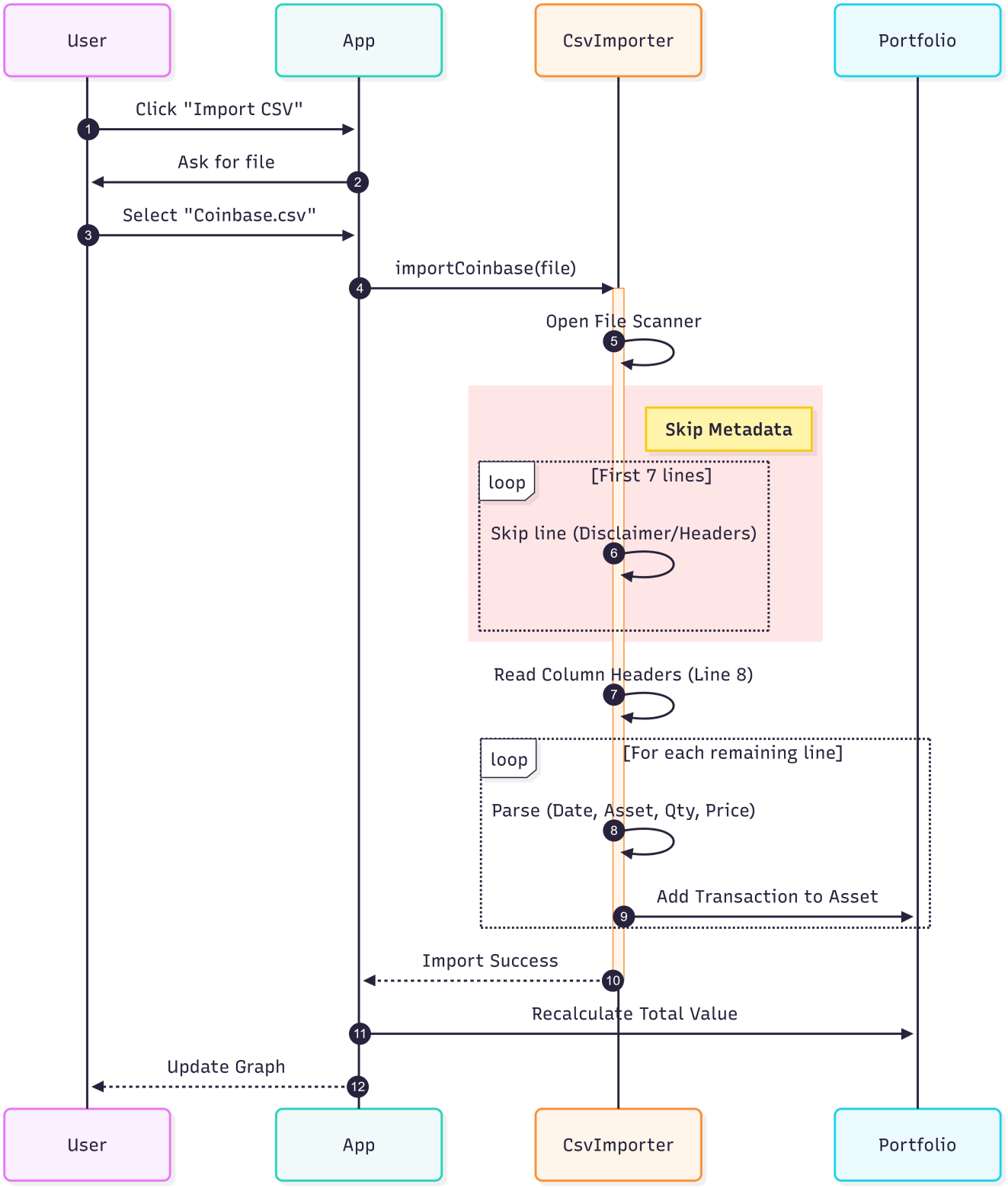
```
public class PortfolioService {
    private static PortfolioService instance;
    private final PersistenceService persistenceService;
    private final MarketDataService marketDataService;

    private PortfolioService() {
        this.persistenceService = PersistenceService.getInstance();
        this.marketDataService = MarketDataService.getInstance();
    }

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

5.2 Vue d'ensemble des services

Service	Singleton	Responsabilités	Dépendances
`PortfolioService`	✅	CRUD portfolios, import CSV	PersistenceService, MarketDataService
`MarketDataService`	✅	Prix, historiques, cache	CoinGeckoClient, YahooClient, CacheService
`PersistenceService`	✅	Sérialisation JSON	EncryptionService
`EncryptionService`	✅	Chiffrement XOR	—
`CacheService`	✅	Cache disque prix	—
`EventService`	✅	Gestion événements	PersistenceService
`AnalysisService`	✅	Calculs analytiques	MarketDataService
`DemoService`	✅	Données de démonstration	—

5.3 Import CSV Coinbase

Figure 5 — Diagramme de séquence : Import CSV



Implémentation clé :

```java
public void importFromCoinbaseCSV(String portfolioId, File csvFile) {
    Portfolio portfolio = getPortfolio(portfolioId);
    if (portfolio == null) return;

    try (CSVReader reader = new CSVReader(new FileReader(csvFile))) {
        List<String[]> lines = reader.readAll();

        int dataStartLine = 8;  // Les données commencent ligne 8 dans Coinbase
        DateTimeFormatter formatter = DateTimeFormatter.ISO_DATE_TIME;

        for (int i = dataStartLine; i < lines.size(); i++) {
            String[] row = lines.get(i);
            if (row.length < 10 || row[0].isEmpty()) continue;

            try {
                String timestamp = row[0];
                String txType = row[1];
                String ticker = row[2];
                double quantity = parseDouble(row[3]);
                double spotPrice = parseDouble(row[5]);
                double fees = parseDouble(row[8]);
                String notes = row.length > 9 ? row[9] : "";

                LocalDateTime date = LocalDateTime.parse(timestamp, formatter);
                TransactionType type = parseTransactionType(txType);

                // Création automatique de l'asset si inexistant
                Asset asset = portfolio.getAssetByTicker(ticker);
                if (asset == null) {
                    asset = new Asset(ticker, ticker, AssetType.CRYPTO);
                    portfolio.addAsset(asset);
                }

                Transaction transaction = new Transaction(type, quantity, spotPrice, date, fees,
                asset.addTransaction(transaction);

            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        updatePortfolio(portfolio);

    } catch (IOException | CsvException e) {
```

```
            e.printStackTrace();
        }
    }
```

5.4 MarketDataService — Stratégie de cache

Le service implémente un **cache à deux niveaux** :

1. **Cache mémoire** (ConcurrentHashMap) — TTL : 60 secondes pour les prix, 5 minutes pour l'historique
2. **Cache disque** (JSON) — Persistant, uniquement pour les prix USD

```java
public double getPrice(String ticker, AssetType type, String currency) {
    String cacheKey = ticker.toUpperCase() + "_" + currency.toUpperCase();

    // Niveau 1 : Cache mémoire
    CachedPrice cached = priceCache.get(cacheKey);
    if (cached != null && !cached.isExpired()) {
        return cached.price;
    }

    // Niveau 2 : Cache disque (USD uniquement)
    LocalDate today = LocalDate.now();
    if (currency.equalsIgnoreCase("USD")) {
        Optional<Double> diskCached = cacheService.getCachedPrice(ticker, today);
        if (diskCached.isPresent()) {
            priceCache.put(cacheKey, new CachedPrice(diskCached.get()));
            return diskCached.get();
        }
    }

    // Appel API si cache miss
    double price = 0;
    if (type == AssetType.CRYPTO) {
        String coinId = TICKER_TO_COINGECKO.getOrDefault(ticker.toUpperCase(), ticker.toLowerCase
        price = coinGeckoClient.getCurrentPrice(coinId, currency);
    } else {
        price = yahooClient.getCurrentPrice(ticker);
        if (price > 0 && !currency.equalsIgnoreCase("USD")) {
            price = exchangeClient.convert(price, "USD", currency);
        }
    }

    // Mise en cache
    if (price > 0) {
        if (currency.equalsIgnoreCase("USD")) {
            cacheService.cachePrice(ticker, today, price);
        }
        priceCache.put(cacheKey, new CachedPrice(price));
    }
    return price;
}
```

6. APIs externes

6.1 Tableau récapitulatif

API	Client	Usage	Endpoint	Limites
Binance	`CoinGeckoClient`	Prix crypto, historique	`/ticker/price` , `/klines`	Rate limit standard
Yahoo Finance	`YahooFinanceClient`	Prix actions	`/v8/finance/chart/{symbol}`	Non-officiel, peut changer
ExchangeRate	`ExchangeRateClient`	Conversion devises	`api.exchangerate-api.com`	Plan gratuit
Whale Alert	`WhaleAlertClient`	Transactions > $1M	`api.whale-alert.io`	API key requise

6.2 Gestion des erreurs et fallbacks

- **Latence/Rate limit** : le cache local limite la fréquence des appels.
- **Indisponibilité** : fallback sur données mockées pour Whale Alerts.
- **Erreurs parsing** : valeurs par défaut (0) en cas d'échec.

6.3 Mapping ticker → ID

Pour les cryptomonnaies, un mapping statique convertit les tickers vers les identifiants API :

```
private static final Map<String, String> TICKER_TO_COINGECKO = new HashMap<>();
static {
    TICKER_TO_COINGECKO.put("BTC", "bitcoin");
    TICKER_TO_COINGECKO.put("ETH", "ethereum");
    TICKER_TO_COINGECKO.put("SOL", "solana");
    TICKER_TO_COINGECKO.put("LTC", "litecoin");
    TICKER_TO_COINGECKO.put("ADA", "cardano");
    TICKER_TO_COINGECKO.put("XRP", "ripple");
    // ...
}
```

7. Interface utilisateur (JavaFX)

7.1 Écrans principaux

Écran	Fichier FXML	Controller	Description
Main	`main.fxml`	`MainController`	Toolbar + navigation
Portfolio	`portfolio-view.fxml`	`PortfolioController`	Liste assets + P&L
Charts	`chart-view.fxml`	`ChartController`	Courbes + allocation
Analysis	`analysis-view.fxml`	`AnalysisController`	Whale Alerts + stats
Passphrase	`passphrase-dialog.fxml`	`PassphraseController`	Chiffrement
Asset Form	`asset-form.fxml`	`AssetController`	Ajout d'asset

7.2 Captures d'écran

Figure 6 — Écran principal

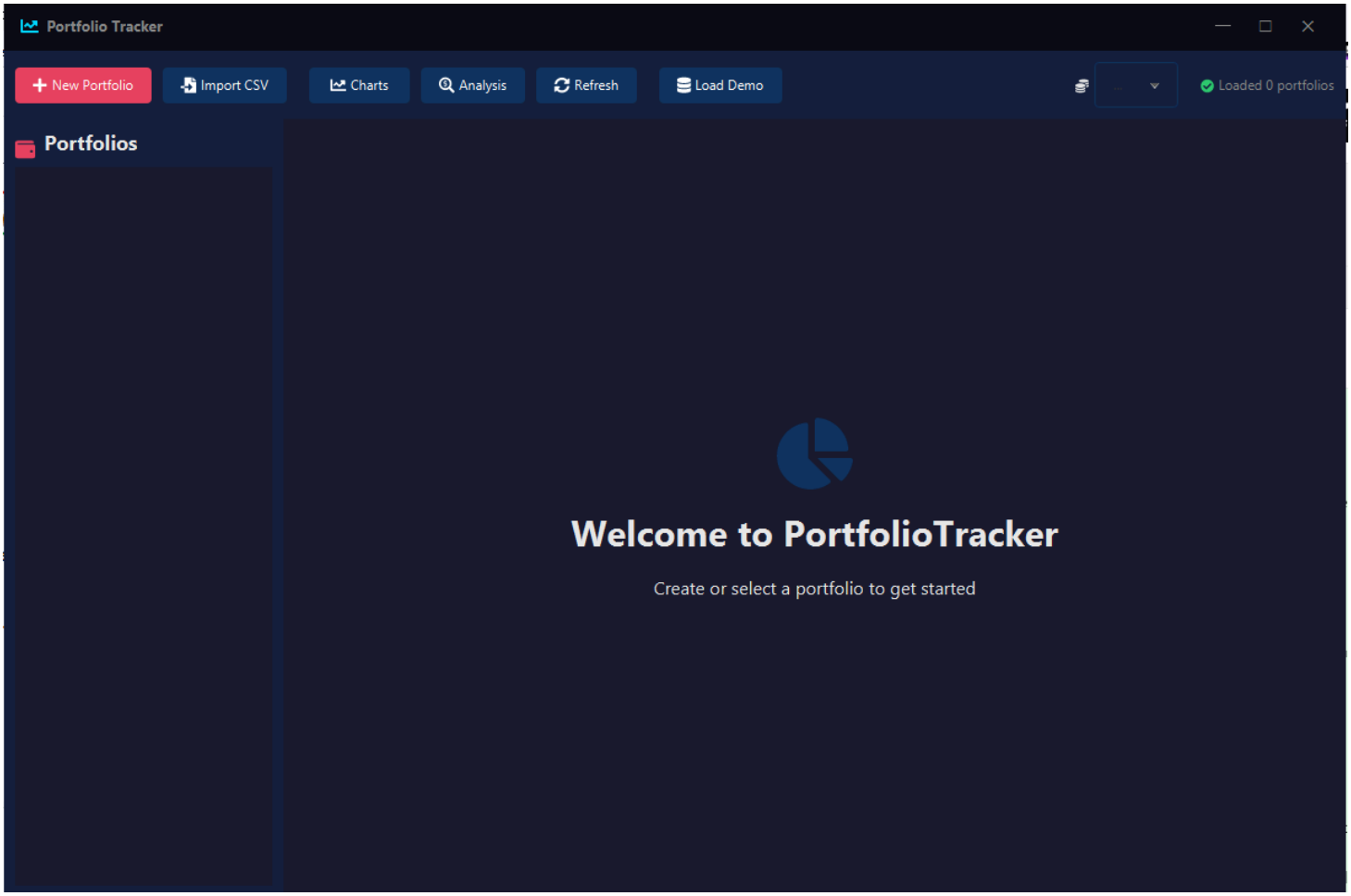


Figure 7 — Ajout d'un portfolio

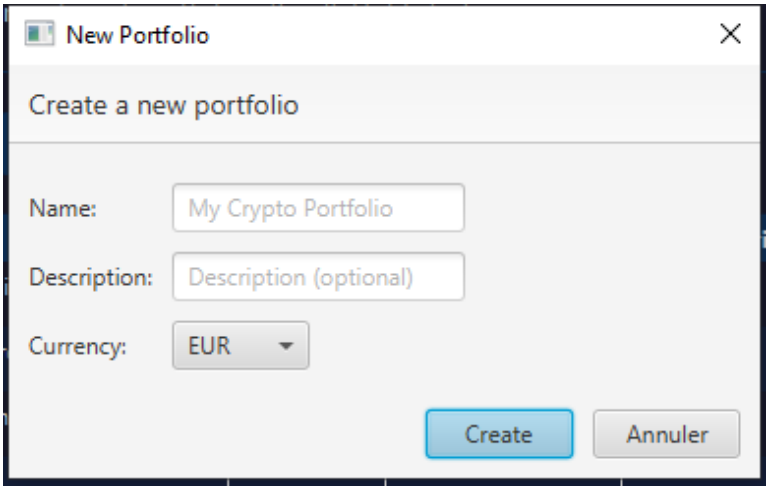


Figure 8 — Ajout d'un asset

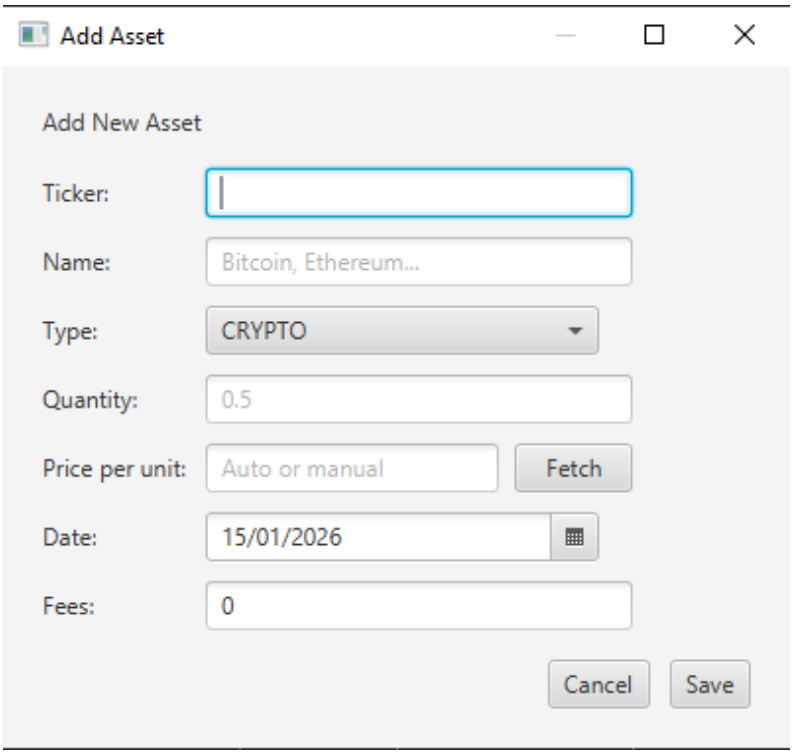
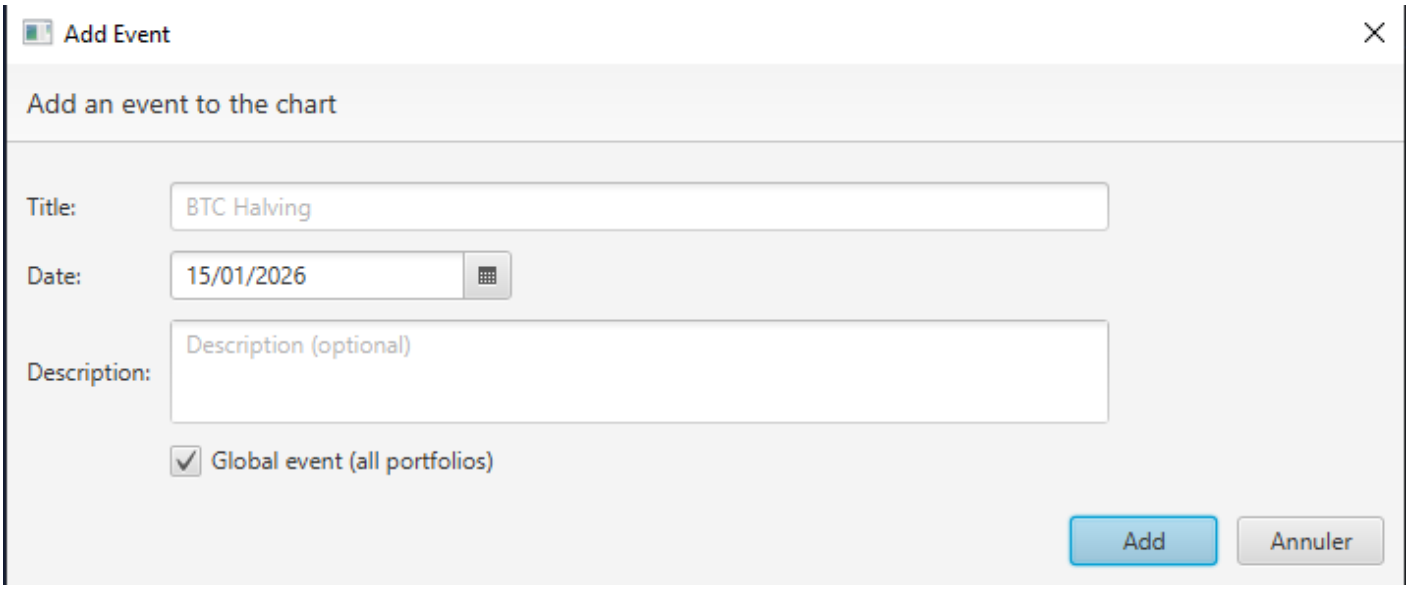


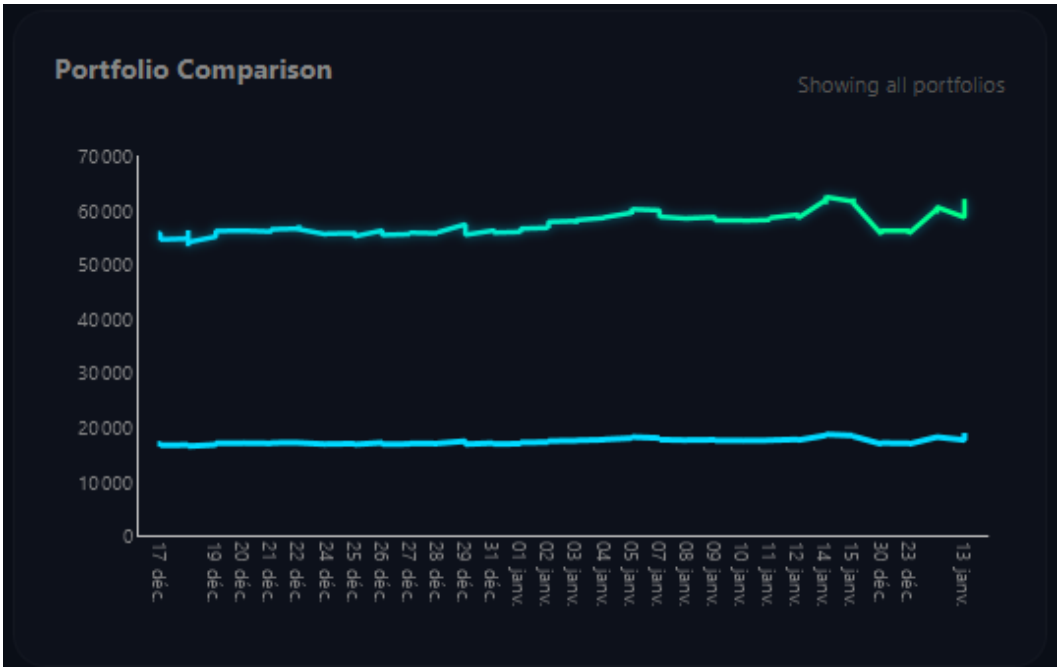
Figure 9 — Création d'un événement



7.3 Comportements UI notables

- **Navigation dynamique** : chargement des vues via le contrôleur principal.
- **Feedback visuel** : couleurs dynamiques pour P&L (vert/rouge).
- **Filtres temporels** : sélection 1W/1M/3M/1Y dans Charts.
- **Compare All** : superposition des courbes multi-portfolios.

Figure 10 — Mode Compare All



8. Concurrence & performance

8.1 Principe

Les appels réseau sont exécutés dans des **Tasks JavaFX** afin de préserver la réactivité de l'interface :

- Les appels I/O tournent en **arrière-plan** (méthode `call()`).
- L'UI est mise à jour **uniquement** dans le callback `setOnSucceeded`.

8.2 Points d'usage

Controller	Usage	Données chargées
`PortfolioController`	`loadPricesAsync()`	Prix actuels des assets
`ChartController`	`loadChartData()`	Historique + séries
`AnalysisController`	`loadWhaleAlerts()`	Transactions whale

8.3 Exemple d'implémentation

```
private void loadPricesAsync() {
    Task<Map<String, Double>> task = new Task<>() {
        @Override
        protected Map<String, Double> call() {
            Map<String, Double> prices = new HashMap<>();
            for (Asset asset : currentPortfolio.getAssets()) {
                double price = marketDataService.getPrice(
                        asset.getTicker(), asset.getType(), currentPortfolio.getCurrency());
                prices.put(asset.getTicker(), price);
            }
            return prices;
        }
    };

    task.setOnSucceeded(e -> {
        priceCache.clear();
        priceCache.putAll(task.getValue());
        refreshTable();
        updateSummary();
    });

    new Thread(task).start();
}
```

8.4 Performance perçue

- ✅ **Pas de blocage** du thread JavaFX Application Thread.
- ✅ **Chargement progressif** des vues avec indicateurs visuels.
- ✅ **Cache efficace** : réduction des appels API répétés.

9. Persistance & chiffrement

9.1 Structure de stockage

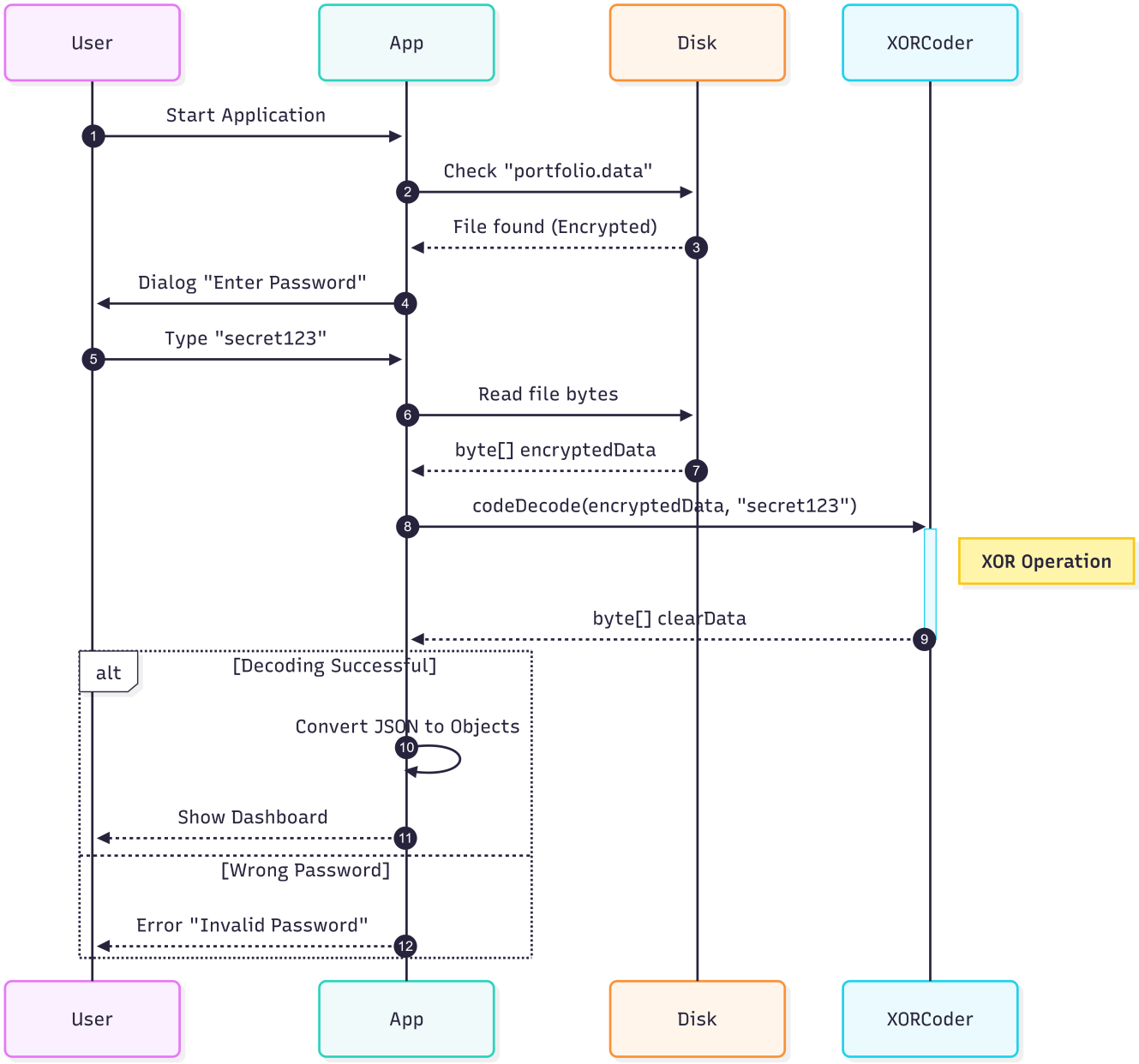
```
data/
├── cache/                    # Cache des prix historiques
│   ├── btc_cache.json
│   ├── eth_cache.json
│   └── ...
├── config/                   # Configuration (réservé)
├── events/
│   └── events.json           # Événements globaux
└── portfolios/
    ├── {uuid}.json           # Portfolio non chiffré
    └── {uuid}.json.enc       # Portfolio chiffré
```

9.2 Sérialisation JSON

- **Bibliothèque** : Gson avec `GsonBuilder().setPrettyPrinting().create()`
- **Sauvegarde automatique** : après chaque modification de portfolio.
- **Adaptateurs** : `LocalDateTimeAdapter` pour les dates.

9.3 Chiffrement XOR

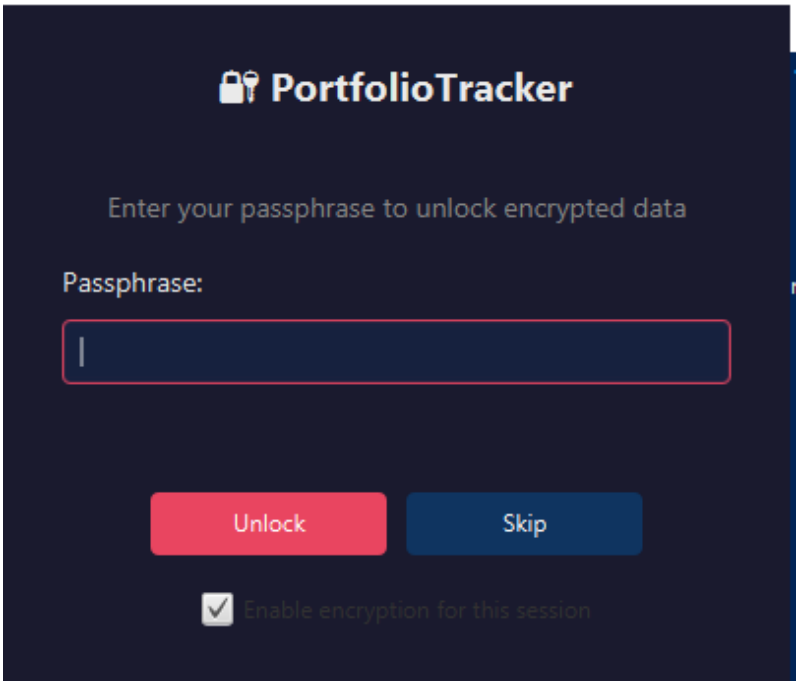
Figure 11 — Diagramme de séquence : Chiffrement



Implémentation :

```
public class EncryptionService {
    private static EncryptionService instance;
    private String passphrase;
    private boolean enabled;

    public byte[] encrypt(byte[] data, String passphrase) {
        return codeDecode(data, passphrase.getBytes());
    }

    public byte[] decrypt(byte[] data, String passphrase) {
        return codeDecode(data, passphrase.getBytes());
    }

    private byte[] codeDecode(byte[] input, byte[] secret) {
        byte[] output = new byte[input.length];
        if (secret.length == 0) {
            throw new IllegalArgumentException("Empty security key");
        }
        int spos = 0;
        for (int pos = 0; pos < input.length; pos++) {
            output[pos] = (byte) (input[pos] ^ secret[spos]);
            spos++;
            if (spos >= secret.length) {
                spos = 0;
            }
        }
        return output;
    }
}
```

Figure 12 — Dialog de passphrase



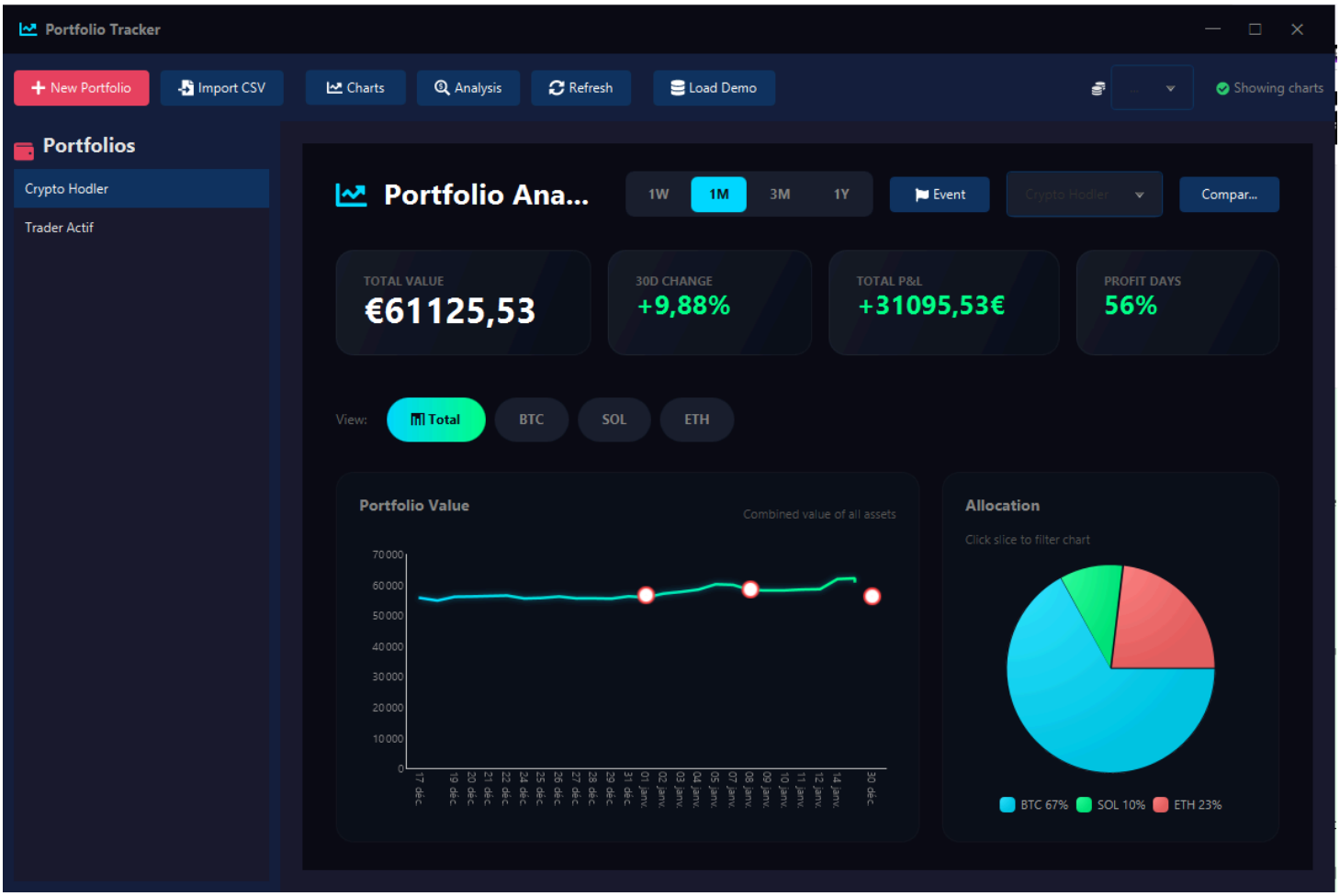
> ⚠ **Note** : Le chiffrement XOR est une implémentation **pédagogique**. Pour une application en production, un algorithme tel que AES-256 serait requis.

10. Analyse & statistiques

10.1 Vue Charts

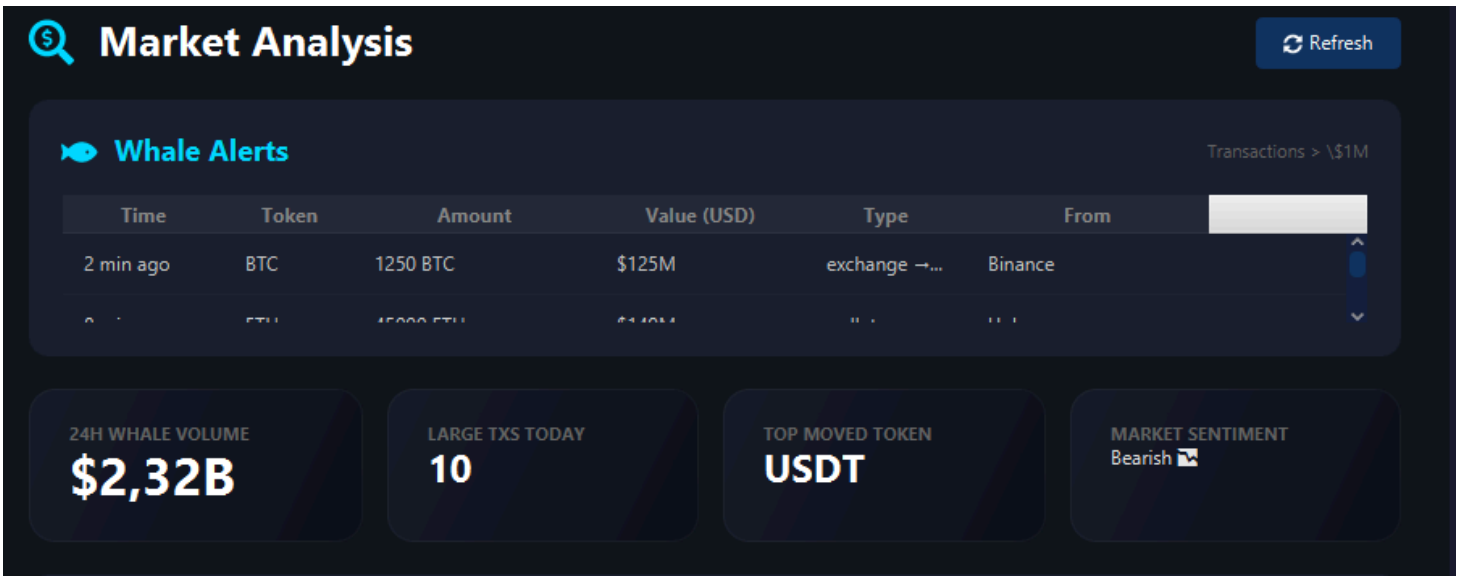
- **Graphique de valeur** : évolution sur 1W / 1M / 3M / 1Y.
- **Sélection d'asset** : visualisation individuelle.
- **Compare All** : superposition multi-portfolios.
- **Allocation** : camembert de répartition.

Figure 13 — Vue Charts complète



10.2 Vue Analysis

Whale Alerts

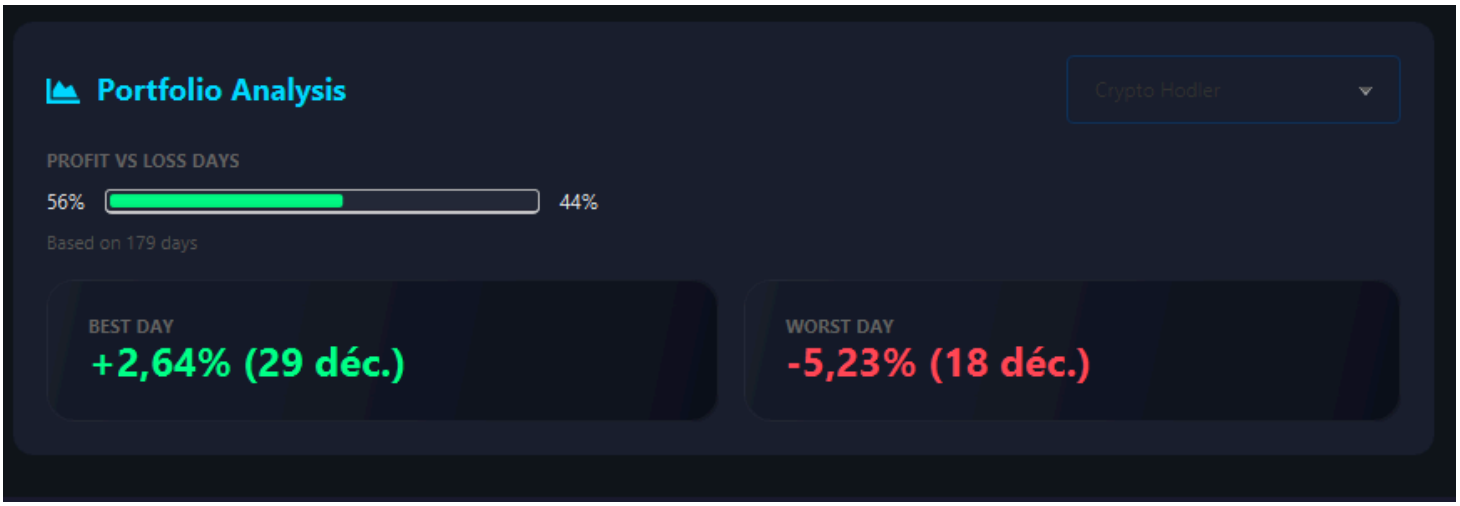


Affichage des transactions crypto supérieures à $1M, avec :

- Volume total des dernières 24h.

- Token le plus actif.

Profit vs Loss Days



Analyse sur 30 jours :

- Ratio jours profitables vs déficitaires.
- Meilleur et pire jour du mois.

10.3 Limite assumée

Les courbes représentent une valeur calculée aux **quantités actuelles**. Ce n'est pas un historique transactionnel complet qui suivrait l'évolution réelle des achats/ventes dans le temps.

11. Tests

11.1 Framework et objectif

- **Framework** : JUnit 5
- **Objectif** : valider les calculs critiques et la stabilité du chiffrement.

11.2 Couverture

Classe de test	Cible	Tests
`AssetTest`	`Asset.java`	Quantités, prix moyen, total investi
`EncryptionServiceTest`	`EncryptionService.java`	Encrypt/decrypt round-trip

11.3 Exemple de test unitaire

```
@Test
void testGetTotalQuantity_withBuyAndSell() {
    asset.addTransaction(new Transaction(TransactionType.BUY, 2.0, 50000, LocalDateTime.now(), 10
    asset.addTransaction(new Transaction(TransactionType.BUY, 1.5, 48000, LocalDateTime.now(), 5,
    asset.addTransaction(new Transaction(TransactionType.SELL, 0.5, 55000, LocalDateTime.now(), 8

    assertEquals(3.0, asset.getTotalQuantity(), 0.001);
}

@Test
void testGetAverageBuyPrice() {
    asset.addTransaction(new Transaction(TransactionType.BUY, 1.0, 40000, LocalDateTime.now(), 0,
    asset.addTransaction(new Transaction(TransactionType.BUY, 1.0, 50000, LocalDateTime.now(), 0,

    assertEquals(45000, asset.getAverageBuyPrice(), 0.001);
}

@Test
void testGetTotalInvested() {
    asset.addTransaction(new Transaction(TransactionType.BUY, 1.0, 40000, LocalDateTime.now(), 10
    asset.addTransaction(new Transaction(TransactionType.BUY, 0.5, 50000, LocalDateTime.now(), 50

    double expected = (1.0 * 40000 + 100) + (0.5 * 50000 + 50);
    assertEquals(expected, asset.getTotalInvested(), 0.001);
}
```

11.4 Exécution

```
mvn test
```

Résultat attendu :

```
[INFO] Tests run: 11, Failures: 0, Errors: 0, Skipped: 0
[INFO] BUILD SUCCESS
```

11.5 Limites de couverture

Élément	Testé	Raison
Modèles (calculs)	✅	Logique critique
Chiffrement	✅	Stabilité requise
Controllers JavaFX	❌	Nécessiterait TestFX
Appels réseau	❌	Dépendance externe
Validations UI	❌	Couverture manuelle

12. Limites connues

Limite	Impact	Mitigation envisagée
`CoinGeckoClient` utilise Binance	Nom trompeur	Renommer le client
Cache disque limité à USD	Pas de cache multi-devises	Étendre le cache
Encryption XOR non sécurisé	Non exploitable en production	Implémenter AES-256
API Yahoo Finance non officielle	Peut changer sans préavis	Surveiller, prévoir fallback
Charts à quantités actuelles	Approximation historique	Implémenter historique réel

13. Pistes d'évolution

Priorité	Évolution	Effort
P1	Chiffrement AES-256	Moyen
P1	Tests UI automatisés (TestFX)	Moyen
P2	Historique multi-devises en cache	Faible
P2	Export PDF/CSV des rapports	Faible
P3	Notifications prix temps réel	Élevé

Priorité	Évolution	Effort
P3	Version mobile (Android)	Élevé

Annexes

A. Table des figures

Figure	Description
Fig. 1	Vue principale : Portfolio avec assets et P&L
Fig. 2	Vue Charts : Évolution de la valeur et allocation
Fig. 3	Vue Analysis : Whale Alerts et statistiques
Fig. 4	Diagramme d'architecture MVC
Fig. 5	Diagramme UML des classes
Fig. 6	Diagramme de séquence : Import CSV
Fig. 7	Écran principal
Fig. 8	Ajout d'un portfolio
Fig. 9	Ajout d'un asset
Fig. 10	Création d'un événement
Fig. 11	Mode Compare All
Fig. 12	Diagramme de séquence : Chiffrement
Fig. 13	Dialog de passphrase
Fig. 14	Vue Charts complète

B. Arborescence du projet

```
Portfolio-Tracker-/
├── pom.xml                          # Configuration Maven
├── src/
│   ├── main/
│   │   ├── java/com/portfoliotracker/
│   │   │   ├── App.java             # Point d'entrée
│   │   │   ├── api/                 # Clients HTTP
│   │   │   ├── controller/          # Controllers JavaFX
│   │   │   ├── model/               # Entités métier
│   │   │   ├── service/             # Logique métier
│   │   │   └── util/                # Utilitaires
│   │   └── resources/
│   │       ├── css/styles.css       # Styles CSS
│   │       ├── fxml/                # Vues FXML
│   │       └── images/              # Assets graphiques
│   └── test/java/com/portfoliotracker/
│       ├── model/AssetTest.java
│       └── service/EncryptionServiceTest.java
└── data/                            # Données persistantes
    ├── cache/
    ├── events/
    └── portfolios/
```

C. Dépendances Maven principales

```xml
<dependencies>
    <!-- JavaFX -->
    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-controls</artifactId>
        <version>21</version>
    </dependency>

    <!-- JSON -->
    <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.10.1</version>
    </dependency>

    <!-- CSV -->
    <dependency>
        <groupId>com.opencsv</groupId>
        <artifactId>opencsv</artifactId>
        <version>5.8</version>
    </dependency>

    <!-- Tests -->
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter</artifactId>
        <version>5.10.0</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Document généré le : Janvier 2026

Auteur : Adam Houri