OptInvest

Rapport de fonctionnalités et guide de lancement

Rabah ACHOUR - Yanis Neddaf - Ahcene Dahoumane

2025-12-27

Table des matières

1 Présentation

OptInvest est une application d'analyse et de simulation de stratégies d'investissement sur les marchés financiers, avec :

- une **API backend** (FastAPI) exposant des endpoints de simulation et de prédiction ;
- une **interface frontend** (Streamlit) pour piloter les simulations, visualiser les résultats, et comparer des stratégies.

1.1 Objectif

Le cœur du projet est de comparer des stratégies d'investissement (DCA vs Lump Sum), d'évaluer des métriques financières (CAGR, volatilité, Sharpe, etc.), de proposer une optimisation Markowitz, et de produire des éléments d'analyse prédictive via une régression OLS.

2 Architecture et composants

2.1 Organisation du code (dossier `app/`)

Le dépôt est structuré autour de trois blocs principaux :

- `app/api/` : API FastAPI (entrée `app/api/main.py`)
- `app/core/` : logique métier (simulation, optimisation, prédiction, data)
- `app/client/` : UI Streamlit (entrée `app/client/main.py`)

2.2 Backend (FastAPI)

Le backend est défini dans `app/api/main.py` et expose :

- `GET /health` (health check)
- `POST /api/simuler` (simulation historique)
- `POST /api/predire` (prédiction + analyses statistiques)

2.3 Frontend (Streamlit)

Le frontend est lancé via `streamlit run app/client/main.py`. Il pilote :

- la saisie utilisateur (choix des actifs, durée, apports, fréquence, frais, durée de prédiction),
- les appels API de simulation et de prédiction,
- les visualisations (Plotly) et tableaux de résultats.

3 Fonctionnalités métier

Cette section reprend les fonctionnalités implémentées et documentées dans le dépôt.

3.1 Simulation de portefeuille (historique)

Endpoint : `POST /api/simuler/`

La simulation permet :

- l'analyse historique de portefeuilles sur actifs réels (Yahoo Finance via `yfinance`) ;
- différentes fréquences d'investissement : *mensuelle, trimestrielle, semestrielle, annuelle* ;
- la prise en compte de **frais de gestion annuels** ;
- le calcul de métriques : **CAGR**, **rendement total**, **volatilité annualisée**, **ratio de Sharpe** ;
- une comparaison (optionnelle côté UI) avec un indice de référence (`ACIM` côté client).

Actifs autorisés (actuels) : AAPL, MSFT, AGGH, TLT, VWCE.DE, SXR8.DE.

3.2 Comparaison de stratégies d'investissement

OptInvest compare notamment :

— **DCA** (Dollar-Cost Averaging) : investissement progressif à fréquence choisie ;
— **Lump Sum** : investissement en une seule fois au début.

Côté UI, l'application peut simuler plusieurs fréquences DCA et ajouter une stratégie *lump sum* afin de comparer les trajectoires.

3.3 Optimisation de portefeuille (Markowitz)

La simulation intègre une optimisation au sens de Markowitz :

— génération d'une **frontière efficiente** (série de portefeuilles en fonction de l'aversion au risque) ;
— identification d'un **portefeuille tangent** (maximisant le ratio de Sharpe) ;
— optimisation quadratique basée sur `scipy.optimize.minimize` ;
— contraintes : somme des poids = 1, poids ≥ 0 (pas de vente à découvert).

3.4 Modélisation prédictive (régression OLS)

Endpoint : `POST /api/predire/`

La partie prédictive repose sur une régression linéaire OLS (via `statsmodels`) et fournit :

— prédictions futures (courbe valeur vs montant investi et timeline) ;
— analyses de qualité (R^2, RMSE relatif, etc.) ;
— diagnostic des résidus (écart-type, Durbin-Watson, distribution vs loi normale) ;
— analyse des coefficients (p-values, intervalles de confiance) ;
— détection d'overfitting via split train/validation (80/20).

3.5 Visualisations et tableau de bord

Via `app/client/plot_manager.py`, l'interface affiche :

— les métriques clés (CAGR, rendement, volatilité, Sharpe, cash investi) ;
— courbes : valeur portefeuille vs cash investi ; portefeuille vs indice ;
— rendements mensuels (bar chart) ;
— comparaison multi-stratégies (historique vs prédictions) ;
— diagnostics statistiques des résidus.

4 Guide de lancement (Local + Docker)

4.1 Prérequis

— Python 3.11+ (le Dockerfile utilise Python 3.11.14-slim)
— (recommandé) `uv` (gestion de dépendances)
— Docker + Docker Compose (pour la méthode containerisée)

4.2 Option A — Lancement en local (uv)

1) Créer et activer l'environnement virtuel

```
pip install uv
uv venv
source .venv/bin/activate
```

2) Installer les dépendances

```
uv add -r requirements.txt
# si uv add pose probleme :
uv pip install -r requirements.txt
```

3) Lancer l'API (terminal 1)

```
uv run --active python app/api/main.py
```

4) Lancer le client Streamlit (terminal 2)

```
uv run --active streamlit run app/client/main.py
```

5) Ouvrir l'interface

```
http://localhost:8501
```

4.3 Option B — Lancement avec Docker Compose (recommandé)

Le dépôt fournit un `Dockerfile` multi-target (`server` et `client`) et un `docker-compose.yaml`.

1) Démarrer

```
docker-compose up
```

2) Accéder à l'UI

```
http://localhost:8501
```

4.4 Configuration de l'URL API côté client

Le client lit la variable d'environnement `API_URL`. Par défaut (local), `API_URL` pointe sur `http://localhost:8000`. En Docker Compose, `API_URL=http://optinvest_server:8000` est injecté au conteneur client.

5 Références (fichiers du dépôt)

- README : description, installation, fonctionnalités, endpoints.
- `app/api/main.py` : entrée FastAPI, inclusion des routeurs.
- `app/api/routers/simulation.py` : endpoint simulation.
- `app/api/routers/prediction.py` : endpoint prédiction.
- `app/client/main.py` : entrée Streamlit.
- `app/client/constants.py` : actifs autorisés, fréquences, URLs API.
- `Dockerfile` et `docker-compose.yaml` : déploiement containerisé.
- `CONCEPTS.md` : détail mathématique (DCA, Lump Sum, Markowitz, OLS).