Issaev Daniil, Cavalcante Donovan

TradeBase DD

Application de simulation de trading.

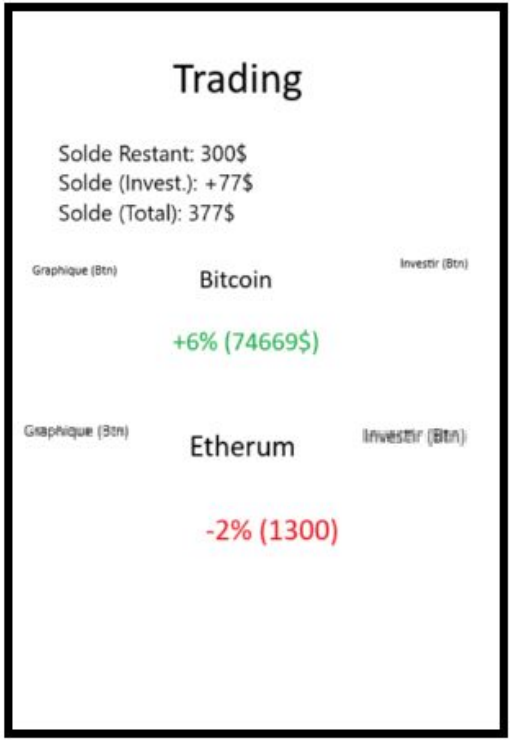
Sommaire

- Mise en place et concept
- Pourquoi ce projet ?
- Utilisation prévue/cible
- Structure du code
- Source de données et Base de données
- Problèmes rencontrés et solutions
- Explication d’une partie du code
- Démonstration
- Etendue des fonctionnalités
- Retour utilisateurs (camarades)
- Axes d’améliorations
- Bilan personnel

Concept

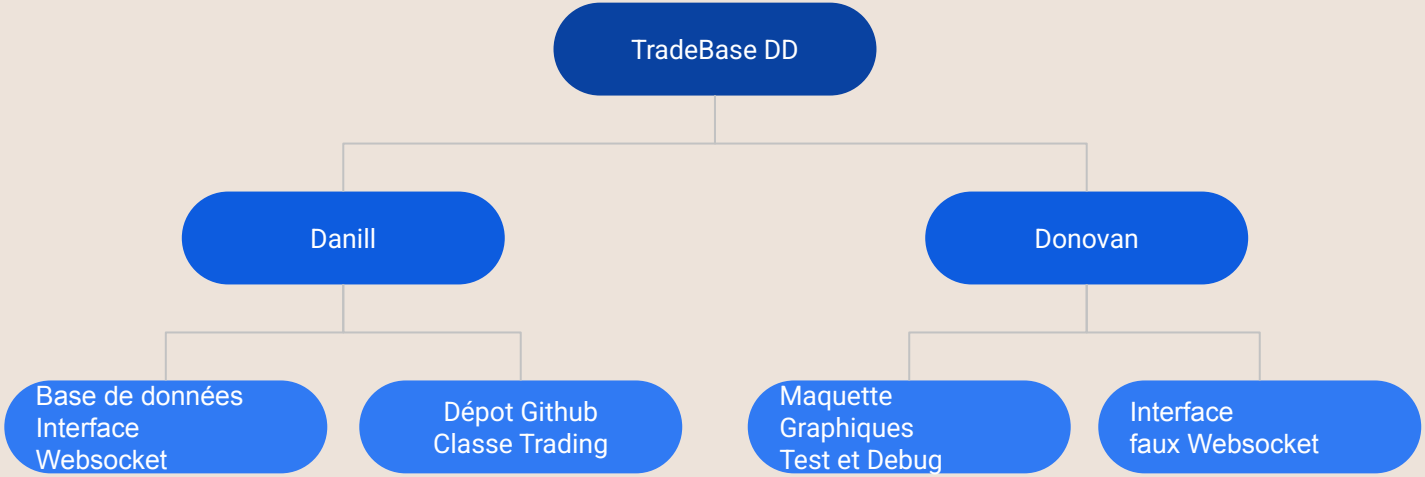
- **Gain** fictifs
- **Tableau de bord** simple
- **Graphiques**
- Mise à jour des **valeurs** en **temps réels**
- **Ajout** de **cryptomonnaies** selon le **choix de l'utilisateur**
- Gestion de **comptes**
- Récupération **dynamique** des **logos**

Maquette



Mise en place

- Répartition des tâches
- Création d'un dépot Github pour synchroniser nos travaux
- Mise en place d'une base de donnée MySQL
- Choix de Python et Tkinter pour l'interface graphique (cadre du programme)
- Planification des tâches à réaliser avec un cahier des charges


TradeBase DD
Danill
Donovan
Base de données
Interface
Websocket
Dépot Github
Classe Trading
Maquette
Graphiques
Test et Debug
Interface
faux Websocket

Utilisation prévue & cible

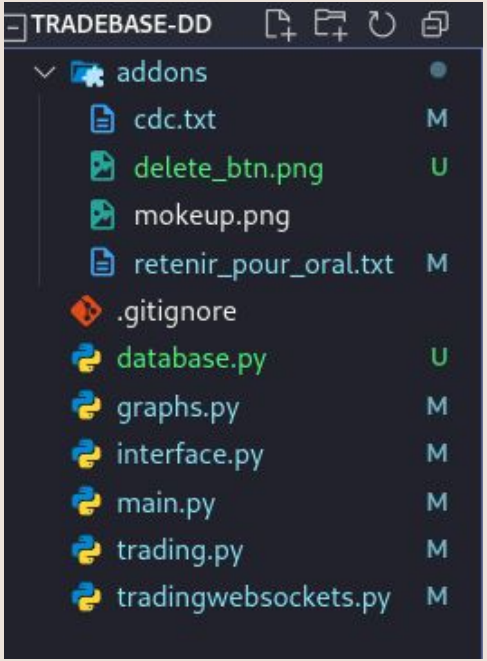
- Interface simple et accessible
- Destiné à des fins éducatives et ludiques
- Pour les traders débutants voir expérimentée pour s’exercer.

Pourquoi ce Projet ?

- Envie de créer une **application interactive**
- Permet de **s'entraîner** au **trading** sans prendre de risques réels
- Aborder et maîtriser différentes technologies : Websocket, API, Tkinter, MySQL, MatPlotLib

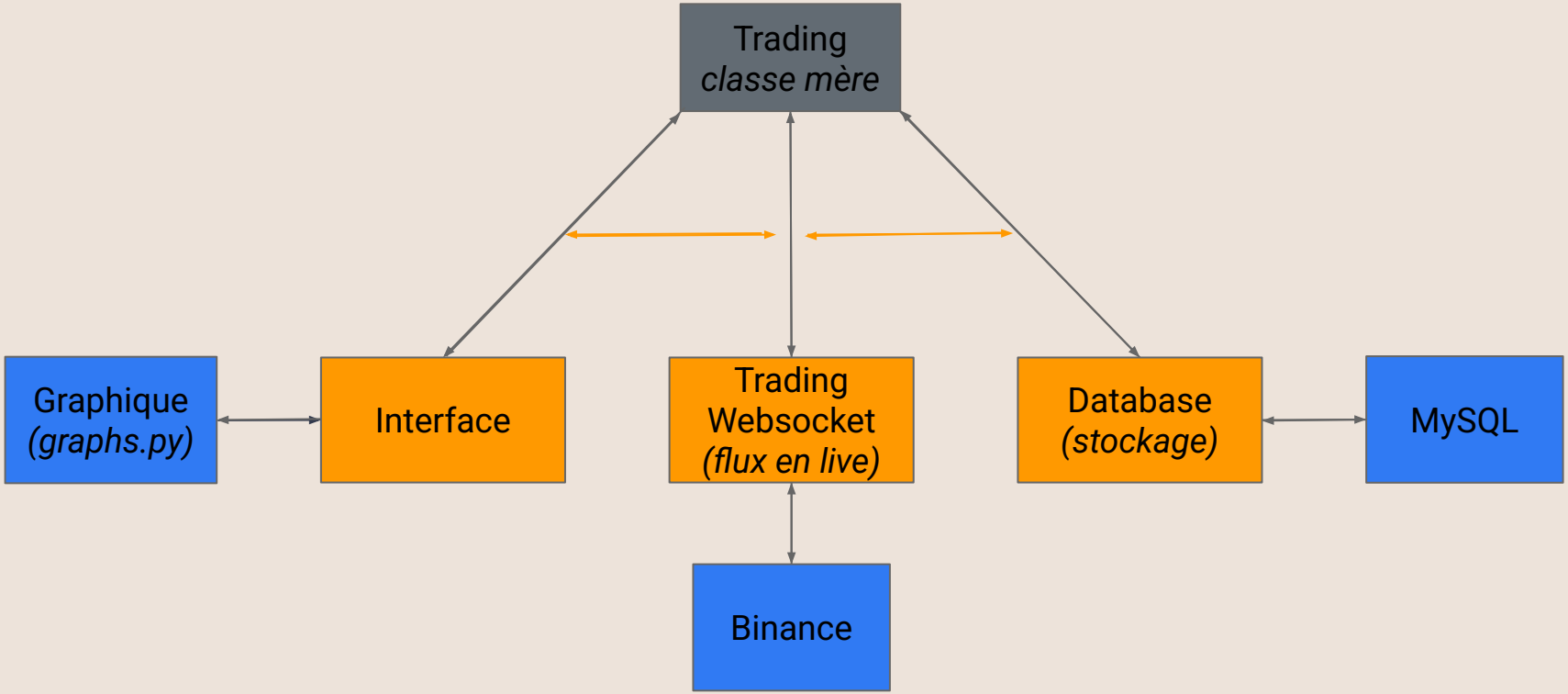
Structure

- **Séparation** des différentes **classes** dans différents **fichiers**
- Choix d'une **POO** (programmation orientée objet) → **Faciliter** les **échanges** de données entre le **frontend** et le **backend**
- Connexion **Websocket Binance**, et requêtes à l'API CoinGecko
- **Mise à jour** en **temps réels** grâce aux différentes **méthodes** inter-classes présentes



Bibliothèques:
tkinter, mysql, matplotlib, pillow, websocket, requests

POO: Structure



Source et traitement des données

- Récupération des 250 cryptos les plus populaires et leurs icônes à l'aide de l'API de CoinGecko et la bibliothèque requests.

- Etablissement d'une connexion websocket Binance, à l'aide de la bibliothèque websocket, sur les 250 cryptos les plus populaires

- Traitement et filtrage des données selon l'utilisateur connecté pour une expérience personnalisée.

Base de données

- Utilisation d'une base de données MySQL à l'aide de XAMPP
- Deux tables:
- users
  → (identifiant, budget,investissements, historique
- cryptos
  → (nom de la crypto, ancien prix)

		cryptoName	oldPrice
☐	Edit Copy Delete	BNBUSDT	27265.27000000
☐	Edit Copy Delete	BTCUSDT	104157.26000000
☐	Edit Copy Delete	DOGEUSDT	0.23000000
☐	Edit Copy Delete	ETHUSDT	2610.06000000
☐	Edit Copy Delete	JLPUSDT	19574.00000000
☐	Edit Copy Delete	POPCATUSDT	44360.08000000
☐	Edit Copy Delete	SOLUSDT	180.06000000
☐	Edit Copy Delete	USDCUSDT	38329.11000000

		id	username	password	budget	listenedCryptos	investments	history
☐	Edit Copy Delete	1	a	1	99368.47	["BTCUSDT", "SOLUSDT"]	[]	[[1, "BTCUSDT", -0.01, 1747136054.8885238], [0, "B...
☐	Edit Copy Delete	2	donovan	&	11.20	["USDCUSDT", "JLPUSDT"]	[]	[[1, "BTCUSDT", 0.08, 1747228240.0814288], [0, "BT...

Problèmes et solutions

- **Limitation** des requêtes API **REST**
  → Passage au **WebSocket** Binance pour données en temps réel

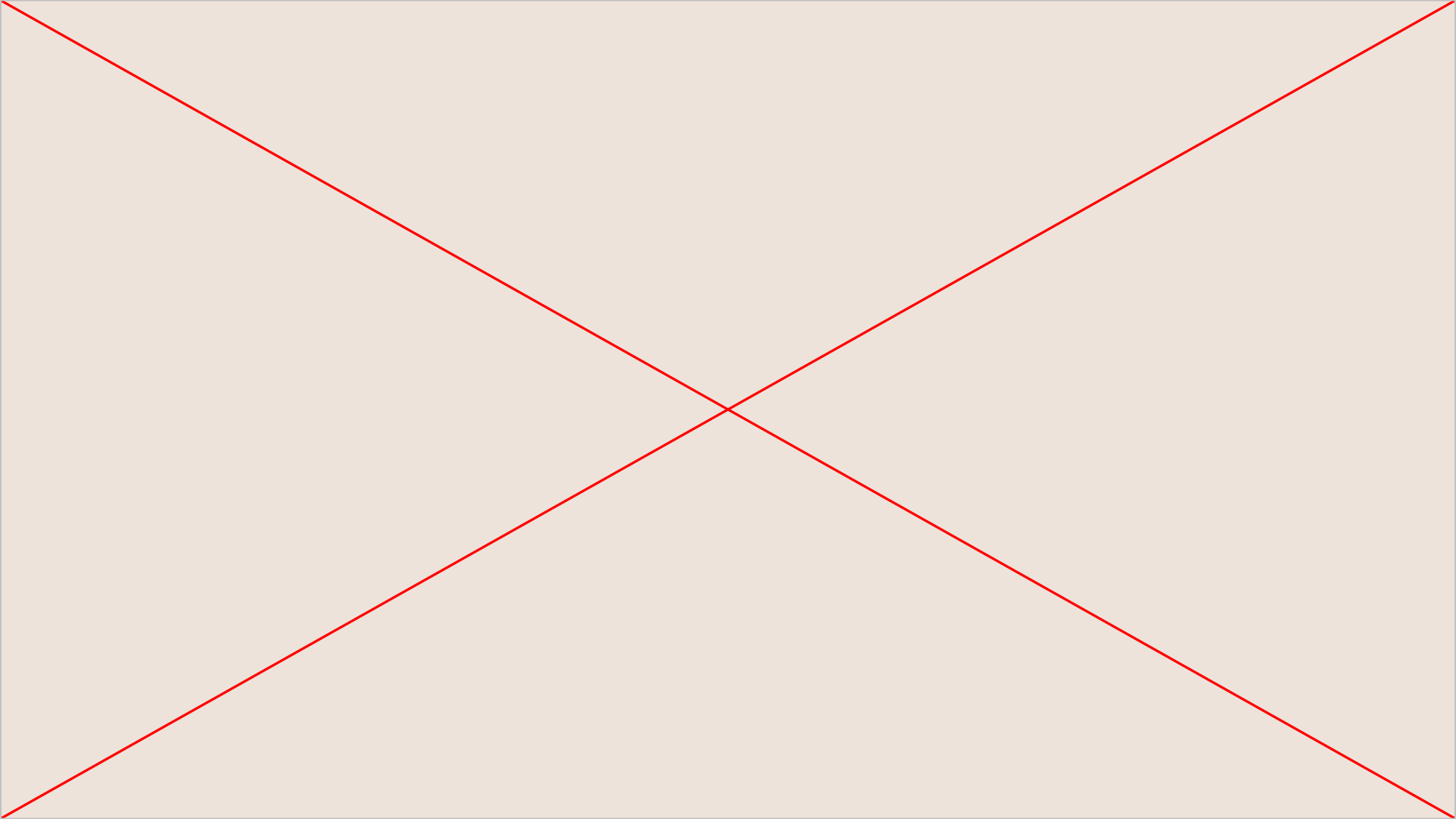
- **Monothreading** Python empêchant interface + WebSocket + BDD simultanés
  → Utilisation de la bibliothèque **threading**

- **Images** Tkinter **détruites** par le **garbage collector**
  → Stockage des images dans une **liste**

Explication du code

```python
async def connect(self):  1 usage
    print("Connexion en cours avec le websocket de Binance.")
    self.parseUrl()
    async with websockets.connect(self.websocket_url, ping_interval=10, ping_timeout=15) as websocket:
        while True:
            response = await websocket.recv()
            data = json.loads(response)
            self.tradingClient.interface.root.after(0, self.tradingClient.interface.updateCrypto, data["s"], data["c"])
```

- Méthode de la classe TradingWebsocket
- self.parseUrl() renvoie l'url entière pour les 250 cryptos
- les pings permettent de garder la connexion active
- boucle pour recevoir les données en continu



Étendue des fonctionnalités

- Gestion de **compte** (connexion/création)
- **Interface** mise en place en fonction des **préférences** de l'utilisateur connecté
- Mise à jour en **temps réels** des données crypto
- **Achat/Vente** fictif de cryptomonnaie
- Ajout **dynamique** de cryptomonnaie (**250** cryptos)
- **Historique** des achats/ventes
- **Graphique** sur différentes périodes

Retour utilisateurs

Points **Positifs**:

- Interface simple
- Belle harmonie de couleurs
- Données en temps réels appréciés.
- Présence d'un historique avec une belle présentation
- Sensation de vrai trading
- Appli intuitive sans notice

Points **Négatifs**:

- Pas de mode jour/nuit
- Pas de sauvegarde du dernier nom d'utilisateur connecté
- Uniquement 250 cryptos disponibles
- Graphiques peu interactifs
- Pas de confirmations avant une vente (erreur possible)

Axes d’améliorations

- Passage à une **bibliothèque graphique** plus **moderne** → (ex : PyQt6, CustomTkinter) pour une interface plus **esthétique** et **ergonomique**.

- Mode **compétitif**
  → **Classement** entre **utilisateurs** en fonction de leurs **performances** de **trading** virtuelles.

- Système de notifications intelligentes
  → **Alertes** personnalisées en cas de fortes hausses ou pertes sur une **crypto** du portefeuille.

- Vente automatique conditionnelle
  → Possibilité pour **l’utilisateur** de définir un **seuil de vente automatique** pour sécuriser ses **gains**.

Bilan personnel

- Découverte et utilisation de nouvelles technologies WebSocket, API, MySQL, threading, etc.
- Meilleure maîtrise de la programmation orientée objet (POO)
- Apprentissage de GitHub pour le travail collaboratif
- Gestion de problèmes techniques complexes et recherche de solutions
- Travail en équipe : répartition des tâches, communication, coordination