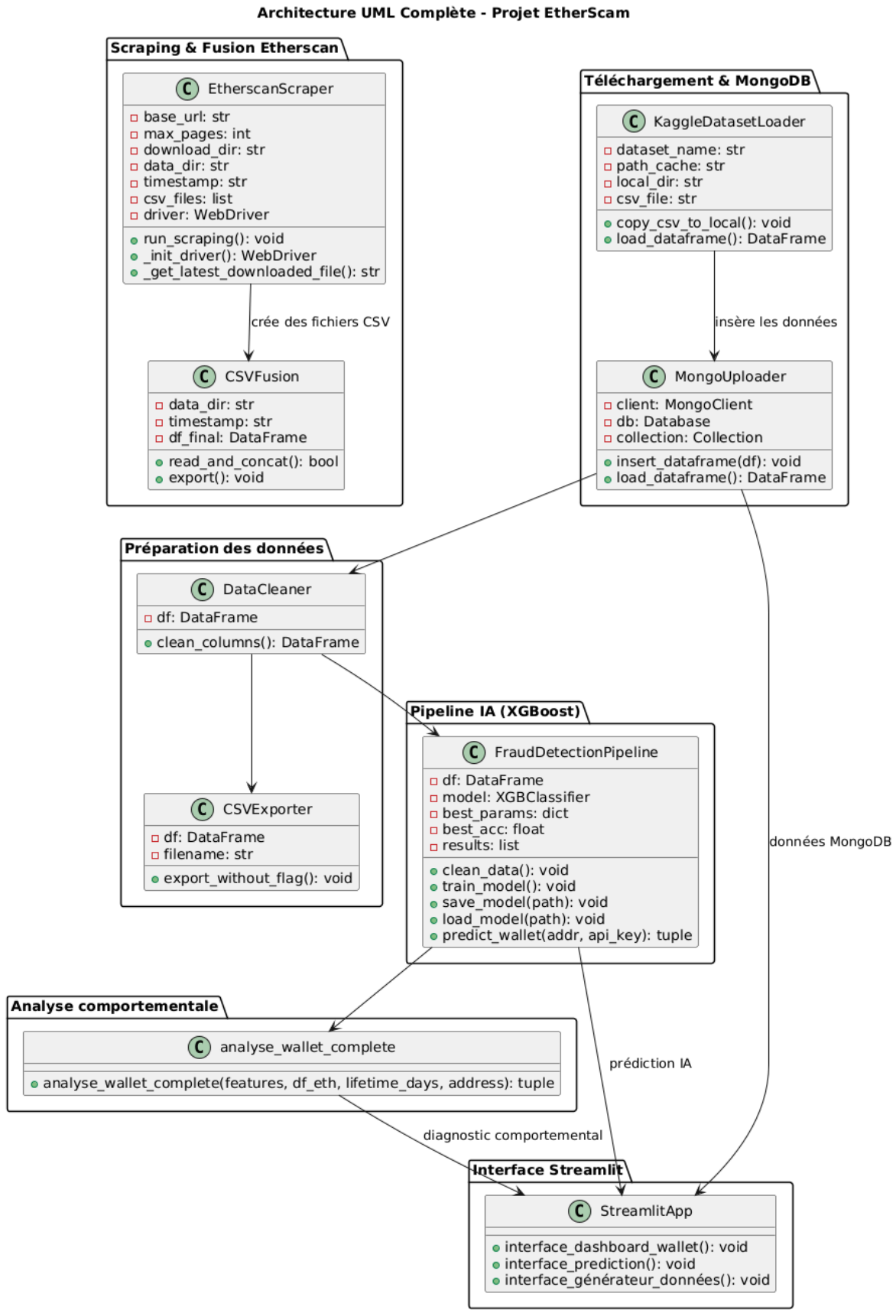

Lien de notre application : [EtherScam]

Explication de l'architecture du code (Diagramme simplifié)

Le projet est construit de manière modulaire avec une architecture orientée objet, articulée autour des étapes clés suivantes :

1. Scraping des transactions depuis Etherscan

- EtherscanScraper : utilise Selenium pour parcourir les pages d'Etherscan et télécharger les transactions en CSV. Chaque fichier est sauvegardé localement.

2. Fusion et nettoyage des CSV

- CSVFusion : fusionne tous les CSV téléchargés en un seul fichier de transactions sans doublons.

3. Chargement du dataset Kaggle et stockage NoSQL

- KaggleDatasetLoader : télécharge le dataset de fraude depuis KaggleHub.
- MongoUploader : insère les données dans MongoDB.

4. Nettoyage des colonnes non utiles

- DataCleaner : supprime les colonnes redondantes ou peu informatives.

5. Pipeline IA de détection de fraude

- FraudDetectionPipeline :
    - Nettoie les données (conversion, suppression de doublons, gestion des NaN).
    - Sépare les données en X_train, X_test, y_train, y_test.
    - Entraîne un modèle **XGBoostClassifier** en explorant manuellement une grille d'hyperparamètres via product() et tqdm.
    - Sauvegarde/charge le modèle.
    - Effectue des prédictions sur une nouvelle adresse via l'API Etherscan.

6. Analyse comportementale avancée (heuristique)

- analyse_wallet_complete() :

    - Analyse le comportement temporel, les ratios d’activité, la fragmentation, les patterns de wallet frauduleux (burner, flooder, scam-like...).

    - Génère un score de danger.

7. Application Web (Streamlit)

- Affiche l'interface utilisateur interactive, les visualisations (matrices de confusion, heatmaps, jauge de précision), et permet à l’utilisateur de tester une adresse ou uploader un fichier CSV personnalisé.

Description et explication des algorithmes de détection d'anomalies

1. Modèle de classification supervisée : XGBoostClassifier

Le projet utilise **XGBoost**, un algorithme de boosting très performant pour la classification binaire (scam ou non scam).

- **Entrée du modèle** : un ensemble de 17 features numériques calculées à partir des transactions Ethereum :
    - Moyennes temporelles (Avg min between sent/received)
    - Volume envoyé/reçu
    - Nombre de transactions et d'adresses uniques
    - Solde final et cumul
- **Label** : colonne FLAG du dataset Kaggle
- **Entraînement** :
    - GridSearch manuel sur : n_estimators, max_depth, learning_rate, subsample, colsample_bytree
    - Optimisation selon la métrique accuracy (95.6% atteinte)
    - Séparation en échantillons d'entraînement/test avec stratify=y

2. Critères de détection du wallet (prédiction)

- Lorsqu'un utilisateur entre une adresse Ethereum, l'application :
    - Récupère les transactions via l'API Etherscan
    - Extrait les features
    - Passe les features dans le modèle XGBoost préalablement entraîné
    - Retourne une prédiction 0 (non suspect) ou 1 (suspect) + la probabilité associée

3. Complément d'analyse comportementale (non supervisée)

- Analyse heuristique à base de règles :
    - **Temporalité** (durée de vie < 15 jours)

- **Tx nulle**, **balance à zéro**, **fragmentation**, **concentration horaire**, etc.
- Attribution de **profils types** : Burner, Flooder, Collector, Scam-like, Dormant
- Score de danger de 0 à 10

Questions

1. Comment avez-vous structuré votre code afin de faciliter la maintenance et les futures améliorations ?

Le projet est structuré selon une architecture orientée objet, modulaire et maintenable. Chaque classe a une responsabilité unique (principe SOLID), ce qui facilite la lisibilité, les tests unitaires et les évolutions. Le code est organisé par modules thématiques : scraping, fusion, nettoyage, base de données, modèle de machine learning, interface web.
Le diagramme UML fourni illustre clairement les dépendances entre les classes, tout en garantissant une faible couplage. Les paramètres de configuration (chemins, clés API, hyperparamètres) sont externalisés, ce qui rend le projet facilement adaptable à d'autres cas d'usage.

2. Quels ont été les principaux défis techniques ou conceptuels que vous avez rencontrés lors du développement de ce projet ?

Nous avons rencontré plusieurs défis techniques majeurs.
Le premier a concerné le scraping de données depuis Etherscan. Le site est dynamique et changeant, ce qui nécessitait une implémentation robuste avec Selenium, des délais intelligents, une navigation multipage et une détection du dernier fichier téléchargé.

Le second défi a été la gestion d'un dataset déséquilibré pour l'apprentissage supervisé. Pour y remédier, nous avons appliqué une séparation stratifiée des données et une optimisation manuelle des hyperparamètres du modèle XGBoost à l'aide d'un grid search basé sur itertools.product et tqdm.

Enfin, la fusion des données issues de différentes sources (Etherscan, Kaggle) a nécessité une normalisation rigoureuse des colonnes et des formats, tout en garantissant l'absence de doublons et de valeurs aberrantes.

3. Comment avez-vous documenté votre code et les processus pour faciliter la compréhension par d'autres développeurs ou utilisateurs ?

Nous avons adopté une approche de documentation rigoureuse, tant à l'intérieur du code que dans des fichiers externes. Des commentaires ciblés sont également intégrés pour expliciter les parties critiques ou techniques du code. Par ailleurs, un fichier de documentation au format Markdown a été rédigé incluant :

- une présentation du projet,
- l'arborescence du code,
- les instructions d'installation et d'exécution,
- une explication de chaque module et des cas d'usage.

Enfin, un diagramme UML permet de visualiser l'architecture complète du projet, et l'organisation en modules thématiques avec des noms explicites (scraping, fusion, nettoyage, IA, interface) facilite l'entrée dans le code pour tout nouveau développeur.

4. Quel type de base de données ou de structure de stockage avez-vous choisi pour vos données et pourquoi ? Quelles autres solutions pourriez-vous proposer ?

Nous avons choisi MongoDB comme base de données, car elle permet de stocker des documents au format JSON, ce qui est parfaitement adapté aux transactions Ethereum, souvent semi-structurées et hétérogènes. Ce choix permet une grande flexibilité, une insertion rapide des documents et une compatibilité native avec les bibliothèques Python telles que PyMongo.

En termes d'alternatives, une base relationnelle comme PostgreSQL avec le type JSONB aurait pu être envisagée pour combiner flexibilité et puissance de requêtage. Pour un usage entièrement cloud et temps réel, Firebase Firestore aurait aussi pu être une option. Toutefois, MongoDB reste la solution la plus adaptée dans notre cas, notamment pour sa simplicité d'utilisation et sa bonne intégration avec les workflows Python/Streamlit.

5. Comment avez-vous assuré que vos visualisations communiquent efficacement les informations clés aux parties prenantes ?

Les visualisations sont intégrées dans une interface web développée avec Streamlit. Cette interface affiche les résultats de manière claire et hiérarchisée, permettant une prise de décision rapide.

L'interface comprend notamment des matrices de confusion, des jauges de probabilité claires pour les utilisateurs non techniques, ainsi qu'un système de score de risque synthétique (de 0 à 10) basé sur des critères comportementaux

L'objectif a été de proposer des visualisations synthétiques, explicites et interactives, accessibles à tout utilisateur, quel que soit son niveau technique. L'accent a été mis sur la lisibilité, la rapidité de compréhension et la transparence des diagnostics.

Conclusion

La double approche permet d'allier **modèle IA performant** et **lecture humaine interprétable**.

Pour conclure, l'architecture du projet EtherScam repose sur une approche **hybride**, combinant à la fois un **modèle de machine learning supervisé**, très performant grâce à XGBoost, et une **analyse heuristique** basée sur des règles comportementales.

Ce choix permet d'obtenir un **double niveau de diagnostic** :

- D'un côté, une **prédiction automatique précise**, avec plus de 95 % de bonnes classifications sur les données historiques.

- De l'autre, une **interprétation compréhensible** pour l'utilisateur, basée sur des profils types comme les wallets "burner", "dormant" ou "scam-like".

Cette complémentarité entre IA et analyse humaine permet de **renforcer la fiabilité globale** du système, tout en **conservant une grande transparence** dans les résultats.

Enfin, la plateforme Streamlit vient offrir une **interface intuitive et interactive**, rendant l'outil accessible à tous, même sans compétences techniques.

L'architecture du projet a été pensée pour faciliter son extension : ajout de nouveaux modèles IA, intégration d'autres blockchains (BSC, Polygon), ou adaptation à d'autres types de fraudes sont rendus possibles sans refonte majeure du code.