Université Paris Nanterre

Licence MIASHS deuxième année option MIAGE

Rapport de projet informatique S3

BlockchainIA: Blockchain Décentralisée avec Minage PoW et IA

Projet réalisé du 28 Octobre 2025 au 29 Décembre 2025

Lien GitHub: `https://github.com/prianpeter/BlockchainIA`

Membres du groupe

PETER CANISIOUS LINTON Prian 44005196
JUDEKEN Anistanroy 44004438

Remerciements

— Nous tenons à remercier chaleureusement nos professeurs d'informatique, M. Delbot et Mme ALAYA, pour leur accompagnement tout au long de ce projet. Leurs conseils éclairés et la manière dont ils nous ont guidés ont été d'une aide précieuse, tant pour la conception que pour la résolution des difficultés rencontrées.
— Nous souhaitons également reconnaître l'apport des outils d'intelligence artificielle qui nous ont apporté un soutien pertinent lors de certaines phases techniques.

Table des matières

1 Introduction

1.1 Qu'est-ce qu'une blockchain ?

Une blockchain est une technologie de stockage d'informations qui a révolutionné notre façon de concevoir les systèmes numériques depuis l'apparition du Bitcoin en 2008. Il s'agit d'un registre numérique distribué où les données sont répliquées sur des milliers d'ordinateurs à travers le monde.

La particularité de cette technologie, c'est qu'elle fonctionne sans autorité centrale. Contrairement aux systèmes traditionnels où une banque, une entreprise ou un gouvernement contrôle les données, la blockchain repose sur un réseau décentralisé où chaque participant possède une copie identique du registre. Cette architecture garantit la transparence et l'immutabilité des données.

1.2 Comment ça fonctionne ?

Les données sont organisées en "blocs". Chaque bloc regroupe un ensemble de transactions avec leurs détails : qui envoie quoi, à qui, et à quel moment. Une fois validé, un bloc est ajouté à la chaîne existante et ne peut plus être modifié.

Ce qui rend le système particulièrement sécurisé, c'est que chaque bloc contient :

— Les transactions du bloc
— Un horodatage précis
— Une empreinte numérique unique (hash), comme une signature cryptographique
— L'empreinte du bloc précédent, créant ainsi une chaîne ininterrompue

Pour modifier une transaction passée, il faudrait recalculer tous les blocs suivants sur la majorité du réseau, ce qui demande une puissance de calcul colossale et est donc pratiquement impossible.

1.3 Le rôle des mineurs

Les "mineurs" sont des participants qui mettent leur puissance de calcul à contribution pour valider et ajouter de nouveaux blocs. Ils doivent résoudre des problèmes mathématiques complexes (c'est le Proof of Work), et le premier à trouver la solution peut ajouter son bloc à la chaîne.

En échange de ce travail de validation, les mineurs reçoivent une récompense en cryptomonnaie. C'est ce mécanisme qui sécurise l'ensemble du réseau sans nécessiter d'autorité centrale, tout en incitant les participants à contribuer honnêtement au système.

1.4 Notre projet : BlockchainIA

Dans le cadre de ce projet, nous avons développé notre propre blockchain complète et fonctionnelle. Notre système intègre les mécanismes fondamentaux de cette technologie :

— **Proof of Work (PoW)** : Un algorithme de consensus pour valider les blocs de manière sécurisée
— **Transactions sécurisées** : Transferts entre portefeuilles avec signatures cryptographiques
— **Smart Contracts** : Contrats automatiques gérant les frais et récompenses de minage

— **Minage optimisé en C++** : Un module compilé pour maximiser les performances
— **Intelligence Artificielle** : Analyse automatique de la blockchain avec Ollama
— **Interface web** : Visualisation en temps réel de la blockchain

2 Environnement de travail

Pour ce projet, nous avons utilisé :
— Python 3.11 comme langage principal
— Visual Studio Code comme environnement de développement intégré
— C++ pour l'optimisation du minage
— Visual Studio Build Tools pour compiler les modules C++
— Environnement virtuel Python créé avec `python -m venv venv`
— Bibliothèques installées via `pip install -r requirements.txt`
— Ollama pour l'analyse IA de la blockchain
L'exécution du projet nécessite :

1. Activer l'environnement virtuel (`venv\Scripts\activate` sur Windows)
2. Compiler le module C++ : `python setup.py build_ext -inplace`
3. Lancer le script principal `main.py`
4. Accéder à l'interface via `http://localhost:5003`

3 Description du projet et objectifs

3.1 Fichiers principaux

Le projet vise à développer une blockchain décentralisée complète avec un système de Proof of Work optimisé, des transactions sécurisées, des smart contracts, et une analyse IA avancée. La structure du projet comprend plusieurs modules principaux :
— `main.py` : Point d'entrée de l'application (menu terminal et serveur Flask)
— `blockchain/blockchain.py` : Gestion de la chaîne de blocs, consensus et portefeuilles
— `blockchain/block.py` : Classe Block avec calcul de hash et minage
— `blockchain/transaction.py` : Système de transactions avec signatures
— `blockchain/fees_contract.py` : Smart contracts pour frais de minage
— `core/mining.py` : Algorithmes de minage automatique et IA
— `core/routes.py` : Interface web Flask (API REST et pages HTML)
— `mine_module.cpp` : Module C++ pour optimisation du minage PoW
— `blockchain_data.json` : Sauvegarde persistante de la blockchain
— `blockchain.db` : Base de données SQLite pour analyse
— `templates/` : Pages HTML (blocks, transactions, miners, analyse IA)

3.2 Structure du projet

Le projet est organisé comme suit :

```
blockchainIA/
 main.py              # Point d'entrée principal
 mine_pow.cpp         # Code source C++ du module de minage
 mine_module.pyi      # Interface Python pour typage
 mine_module.cp311-win_amd64.pyd # Module C++ compilé
 setup.py             # Script de compilation C++
 requirements.txt     # Dépendances Python
 blockchain_data.json # Sauvegarde de la blockchain
 blockchain.db        # Base de données SQLite
 ai/                  # Module IA
  | __init__.py
  | generator.py     # Génération transactions avec Ollama
 blockchain/          # Module blockchain
  | __init__.py
  | blockchain.py    # Classe Blockchain principale
  | block.py         # Classe Block
  | transaction.py   # Classe Transaction
  | fees_contract.py # Smart Contracts
 core/                # Logique métier
  | __init__.py
  | mining.py        # Algorithmes de minage
  | routes.py        # Routes Flask (API + pages)
  | utils.py         # Fonctions utilitaires
  | db.py            # Gestion base SQLite
 templates/           # Pages HTML
  | base.html        # Template de base (parent)
  | home.html        # Tableau de bord
  | blocks_list.html # Liste des blocs
  | block_detail.html # Détail d'un bloc
  | transactions_list.html # Liste transactions
  | transaction_detail.html # Détail transaction
  | miners_simple.html # Statistiques mineurs
  | adresse.html     # Historique portefeuille
  | audit.html       # Audit de portefeuille
  | blockchain_analysis.html # Analyse IA
  | security.html    # Sécurité IA
  | download.html    # Téléchargement projet
  | error.html       # Page d'erreur
 venv/                # Environnement virtuel Python
```

4 Bibliothèques, Outils et technologies

Voici les outils que nous avons utilisés, expliqués simplement :

— **Flask** : Crée les pages web que vous voyez dans votre navigateur
— **hashlib** : Crée les signatures numériques (empreintes) qui protègent chaque bloc
— **C++** : Un langage très rapide qui accélère le minage de 10 à 50 fois
— **time, datetime** : Enregistre l'heure exacte de chaque transaction
— **json** : Sauvegarde la blockchain dans des fichiers lisibles
— **os** : Interagit avec votre ordinateur (fichiers, dossiers)
— **random** : Génère des adresses de portefeuilles aléatoires
— **signal** : Permet d'arrêter proprement le programme avec CTRL+C
— **io** : Crée des fichiers PDF et CSV pour les exports
— **Ollama (Llama 3.2)** : L'intelligence artificielle qui analyse la blockchain et détecte les problèmes
— **threading, Queue** : Fait tourner plusieurs tâches en même temps (minage, sauvegarde, IA)
— **SQLAlchemy** : Gère la base de données pour les analyses avancées
— **requests** : Permet aux ordinateurs de se parler entre eux sur le réseau
— **pybind11** : Fait le lien entre le code Python et le code C++
— **reportlab** : Crée des rapports PDF pour les audits de portefeuilles

5 Travail réalisé

Le travail s'est divisé en deux contributions majeures :

Prian : Blockchain Core et Optimisation C++

- **Architecture de base :** J'ai développé les classes `Block`, `Transaction`, et `Blockchain` avec calcul de hash SHA-256 (empreinte cryptographique unique) et validation de chaîne.
- **Système de portefeuilles :** J'ai créé un système de gestion des adresses (format 0x...), avec validation automatique des soldes et historique complet par portefeuille.
- **Module C++ :** J'ai optimisé le Proof of Work (système de validation des blocs) avec pybind11, multipliant les performances par 10 à 50.
- **Minage automatique :** J'ai implémenté un thread en arrière-plan avec Queue pour gérer les transactions, générer des données IA et diffuser les blocs sur le réseau.

Anistanroy : Smart Contracts, Interface Web et IA

- **Smart Contracts :** J'ai conçu un système modulaire avec `mining_fee_contract` pour distribuer automatiquement les récompenses aux mineurs (ceux qui valident les blocs).
- **Persistance et consensus :** J'ai mis en place la sauvegarde automatique en JSON et l'algorithme "chaîne la plus longue" pour synchroniser les différents nœuds du réseau.
- **Interface Flask :** J'ai développé plus de 15 pages web avec dashboard, exploration détaillée de blocs/transactions et exports de données CSV.
- **IA Ollama :** J'ai intégré l'intelligence artificielle pour l'analyse conversationnelle, la génération automatique de transactions et la détection d'anomalies de sécurité.

5.1 Contribution Individuelle

Prian : Blockchain Core et Optimisation C++

Étape 1 : Blockchain de Base

Requête : *"Crée moi une blockchain simple en Python avec des blocs et des transactions"*

Réponse : voir annexe C.1.1

J'ai commencé par développer les fondations de la blockchain. L'IA m'a généré une structure minimaliste avec les fonctions essentielles : `calculate_hash()` pour créer l'empreinte SHA-256 de chaque bloc, `add_block()` pour ajouter un bloc à la chaîne, et `is_valid_chain()` pour vérifier l'intégrité. Chaque bloc stocke son index, un timestamp (horodatage), les transactions, et le hash du bloc précédent. Cette base m'a permis de construire les fondations nécessaires aux développements suivants.

Étape 2 : Système de Portefeuilles

Requête : *"Ajoute un système de portefeuilles avec validation des soldes"*

Code : voir annexe C.1.2

J'ai ensuite développé le système de portefeuilles. J'ai implémenté les fonctions `get_balance()` pour récupérer le solde d'une adresse, `create_transaction()` qui vérifie automatiquement la disponibilité des fonds avant chaque transfert, et `get_wallet_history()` pour l'historique complet des transactions. Les adresses utilisent le format hexadécimal 0x... (16 caractères). Le système refuse automatiquement toute transaction si le solde est insuffisant, garantissant ainsi l'intégrité financière de la blockchain. J'ai également mis en place un ledger global (registre centralisé) qui conserve la trace de toutes les opérations.

Étape 3 : Minage Python Basique
Requête : *"Comment créer un algorithme de Proof of Work ?"*
Code : voir annexe C.1.3

J'ai développé un algorithme de Proof of Work (PoW), qui est le mécanisme de sécurité central d'une blockchain. Le principe : pour ajouter un bloc, il faut résoudre un problème mathématique difficile. Concrètement, on cherche un nombre appelé "nonce" qui, combiné aux données du bloc, produit un hash (empreinte numérique) commençant par un certain nombre de zéros. Plus il y a de zéros requis, plus c'est difficile. Le processus est itératif : on teste des millions de nombres jusqu'à trouver le bon. Cette version Python était lente (plusieurs secondes par bloc) mais fonctionnelle, posant les bases pour les optimisations futures.

Étape 4 : Module C++ Ultra-Rapide
Requête : *"Crée un module C++ pour miner plus vite"*
Code : voir annexe C.1.4

J'ai créé le module `mine_pow.cpp` en C++, un langage beaucoup plus rapide que Python pour les calculs intensifs. Le C++ permet des optimisations "bas-niveau" : manipulation directe des bits en mémoire, ce qui accélère considérablement les calculs de hash. L'intégration avec Python se fait grâce à pybind11, une bibliothèque qui fait le pont entre les deux langages. Après compilation, on obtient un fichier .pyd que Python peut utiliser directement comme s'il s'agissait de code Python. Résultat : les performances sont multipliées par 10 à 50, rendant le minage quasi-instantané pour des difficultés moyennes. C'est comme passer d'une calculatrice à un supercalculateur.

Étape 5 : Minage Automatique avec Threading
Requête : *"Minage automatique en arrière-plan avec IA"*
Code : voir annexe C.1.5

J'ai finalisé le système avec le minage automatique via `auto_miner()`, qui s'exécute en arrière-plan grâce à `threading.Thread` (mécanisme Python pour exécuter du code en parallèle sans bloquer le programme principal). J'utilise une `Queue` (file d'attente thread-safe) pour gérer les transactions en attente de validation. Le système lance 4 threads simultanés : génération de transactions IA (`prefill_ai_queue()`), sauvegarde automatique (`auto_save_thread()`), minage continu (`auto_miner()`), et menu interactif (`run_menu()`). La fonction `generate_ai_transactions()` communique avec Ollama pour créer des transactions réalistes, et `broadcast_block()` envoie les nouveaux

blocs aux autres nœuds du réseau via requêtes HTTP. Le système est ainsi devenu complètement autonome.

Anistanroy : Smart Contracts, Interface Web et IA

Étape 1 : Smart Contracts
Requête : *"Implémente des smart contracts pour les frais de minage"*
Code : voir annexe C.1.6

J'ai développé le système de smart contracts (contrats intelligents auto-exécutables). J'ai créé une classe `SmartContract` avec une méthode `execute()` pour déclencher la logique du contrat. Le contrat principal `mining_fee_contract` calcule automatiquement 5€ de base plus la somme des frais de transaction, puis attribue la récompense au mineur gagnant. J'ai intégré une gestion d'erreurs avec try-except pour garantir la stabilité du système en cas de dysfonctionnement d'un contrat.

Étape 2 : Persistance JSON
Requête : *"Ajoute sauvegarde/chargement automatique"*
Code : voir annexe C.1.7

J'ai ensuite implémenté la persistance des données. La fonction `save_data()` sérialise l'intégralité de la blockchain au format JSON dans le fichier `blockchain_data.json` après chaque modification. Pour cela, j'ai développé `to_dict()` qui convertit tous les objets Python (blocs, transactions) en dictionnaires sérialisables. À l'inverse, `load_data()` recharge les données au démarrage via `from_dict()` qui reconstruit les objets. Ce mécanisme assure la persistence des données entre les redémarrages de l'application.

Étape 3 : Consensus Distribué
Requête : *"Synchronisation entre nœuds avec consensus"*
Code : voir annexe C.1.7

Pour le réseau distribué, j'ai implémenté `resolve_conflicts()` qui applique l'algorithme de consensus "chaîne la plus longue" (principe selon lequel la blockchain valide la plus longue fait autorité). Cette fonction récupère les chaînes des autres nœuds du réseau via requêtes HTTP, puis `is_valid_chain()` vérifie l'intégrité du chaînage des hash et la cohérence des transactions. Si une chaîne plus longue et valide est détectée, elle remplace automatiquement la chaîne locale. La communication entre nœuds s'effectue via des requêtes HTTP REST.

Étape 4 : Interface Flask de Base
Requête : *"Crée interface Flask pour visualiser la blockchain"*
Code : voir annexe C.1.8

J'ai développé l'interface web Flask avec Jinja2 et Tailwind CSS : `base.html` (template parent), `home.html` (dashboard), `blocks_list.html`, `transactions_list.html` (tableaux avec export CSV), `error.html`. Routes principales : `@app.route('/')`, `/blocks`, `/transactions`.

Étape 5 : Pages Détaillées et Fonctionnalités Avancées

Requête : *"Ajoute pages de détails pour blocs et transactions"*
Code : voir annexes 1 et 2

J'ai ajouté les pages spécialisées : `block_detail.html`, `transaction_detail.html`, `miners.html` (statistiques), `adresse.html` (historique portefeuille), `audit.html` (analyse avec graphiques et PDF), `download.html` (exports - voir annexe 6). Fonction principale : `export_to_csv()` pour générer les fichiers CSV.

Étape 6 : Intégration Ollama et Pages IA
Requête : *"Intègre Ollama pour analyse automatique"*
Code : `core/routes.py`, `templates/blockchain_analysis.html`

J'ai intégré l'IA Ollama : `blockchain_analysis.html` (chat conversationnel) et `security.html` (détection d'anomalies). L'analyse des comportements suspects (richesse >30%, mineurs >40%, montants répétitifs) est effectuée directement dans les routes Flask. Interface AJAX pour communication avec Ollama.

Prian et Anistanroy : Intégration et Tests Réseau

Code et tests : voir annexes 3 (synchro), 4 (mineurs), 7 (auto-miner) et 8 (ZeroTier)

Après avoir développé nos parties respectives, nous avons procédé à l'intégration complète du système. Pour tester le fonctionnement distribué de la blockchain, nous avons utilisé ZeroTier, un VPN (Virtual Private Network) qui crée un réseau local virtuel. Après installation et connexion au même réseau ZeroTier, chacun dispose d'une adresse IP locale (192.168.x.x) accessible à distance, simulant ainsi un véritable réseau local.

Cette configuration nous a permis de tester les fonctionnalités réseau :

- **Enregistrement de pairs :** Via le menu terminal, l'option `5. Enregistrer un Pair` permet d'ajouter l'adresse d'un nœud distant au réseau
- **Synchronisation :** `resolve_conflicts()` récupère les blockchains des pairs et applique l'algorithme de consensus
- **Diffusion :** `broadcast_block()` envoie automatiquement chaque nouveau bloc miné à tous les pairs enregistrés

Nous avons également intégré le module C++ avec le code Python. Le module `mine_pow.cpp` est compilé en `mine_module.cp311-win_amd64.pyd` via la commande `python setup.py build_ext -inplace`, puis importé automatiquement au démarrage, assurant ainsi des performances optimales pour le minage.

Les tests via ZeroTier ont validé le bon fonctionnement du consensus distribué, de la synchronisation automatique et de la résolution de conflits entre nœuds distants.

5.2 Rôle de l'Intelligence Artificielle dans le développement

Les outils d'IA (ChatGPT, DeepSeek, GitHub Copilot) ont accéléré notre développement en générant du code initial (comme détaillé dans la section 5.1), en expliquant des concepts complexes (Proof of Work, consensus distribué, cryptographie SHA-256) et en résolvant les erreurs techniques. Pour corriger les bugs rencontrés, l'IA a notamment créé `is_valid_chain()` pour détecter les incohérences de hachage,

`resolve_conflicts()` pour gérer les divergences entre nœuds, et des mécanismes de gestion des erreurs HTTP (`timeout=`, `try-except requests.exceptions`). Cette assistance nous a permis de nous concentrer sur l'architecture globale et l'intégration des composants.

5.3 Fonctionnement de l'IA intégrée : Ollama (Llama 3.2)

Nous avons intégré Ollama (modèle Llama 3.2) directement dans la blockchain pour l'analyse intelligente (voir annexes 5 interface analyse et C.2.1 code génération). L'exécution locale garantit confidentialité et rapidité sans dépendance externe.

5.3.1 Génération automatique de transactions

La fonction `generate_ai_transactions()` (voir annexe C.2.1) génère des transactions réalistes pour alimenter la blockchain. Le système sélectionne aléatoirement des portefeuilles via `random.choice()`, détermine des montants avec `random.uniform(100, 3000)` et Ollama génère les paires expéditeur-destinataire via l'API `chat()` du module ollama. Implémentation : `ai/generator.py`.

5.3.2 Analyse conversationnelle

L'interface `blockchain_analysis.html` (voir annexe C.2.2) permet de dialoguer avec Ollama pour obtenir des analyses de la blockchain : statistiques ("Combien de blocs ?"), analyses avancées ("Distribution de richesse ?"), et recommandations ("Améliorer la sécurité ?"). La communication avec Ollama s'effectue via l'API `chat()` du module Python ollama. Page accessible : `/blockchain-analysis`.

5.3.3 Détection d'anomalies

La page `/security` (voir annexe C.2.4) analyse automatiquement la blockchain pour détecter les comportements suspects. Le système évalue :

- Concentration de richesse : portefeuilles détenant >30% du capital total
- Dominance de mineurs : mineurs ayant produit >40% des blocs
- Patterns suspects : montants identiques répétés (wash trading), transactions à haute fréquence
- Transferts anormaux : montants représentant >80% du solde d'un portefeuille

Chaque anomalie reçoit un niveau de gravité (high/medium/low). Implémentation : `core/routes.py`, fonction `view_security()`.

5.3.4 Implémentation technique

L'architecture repose sur le module Python `ollama` qui effectue des requêtes vers `http://localhost:11434/api/generate`. La fonction `chat()` du module ollama gère la communication avec le serveur Ollama local. Les données de la blockchain sont formatées en contexte textuel avant d'être envoyées, et les réponses sont affichées directement dans l'interface web via AJAX.

6 Difficultés rencontrées

6.1 Compilation du module C++

La compilation du module C++ a représenté un défi majeur, notamment en raison des différences entre plateformes. Sous Windows, Visual Studio Build Tools nécessitait une configuration précise des chemins d'accès et des variables d'environnement. Nous avons dû installer manuellement les outils de compilation, configurer les chemins système (variables PATH), et résoudre plusieurs erreurs où le compilateur ne trouvait pas certaines bibliothèques nécessaires. Le module C++ étant obligatoire pour le minage, l'application refuse de démarrer si la compilation échoue, ce qui nous a forcés à maîtriser le processus de compilation avec pybind11.

6.2 Optimisation du Proof of Work

L'équilibre entre rapidité et sécurité du Proof of Work a nécessité de nombreux ajustements. Une difficulté trop faible rendait le système vulnérable aux attaques, tandis qu'une difficulté trop élevée ralentissait excessivement le minage. Nous avons dû effectuer des tests approfondis pour trouver le paramétrage optimal.

6.3 Synchronisation réseau

La gestion de la synchronisation entre nœuds distants via ZeroTier a révélé plusieurs problèmes imprévus :

- **Timeouts réseau :** Les requêtes HTTP entre nœuds échouaient parfois en raison de latences variables
- **Ordre des blocs :** Les blocs arrivaient parfois dans le désordre, nécessitant un mécanisme de réorganisation
- **Conflits simultanés :** Deux nœuds minant simultanément créaient des branches divergentes, testant ainsi notre algorithme de consensus
- **Perte de connexion :** La gestion de la reconnexion automatique et de la resynchronisation complète a dû être implémentée

6.4 Collaboration à distance

La collaboration à distance a exigé une communication rigoureuse via Discord pour coordonner nos efforts et éviter les conflits dans le code. L'utilisation de Git pour le versioning et la gestion des branches a été essentielle.

6.5 Intégration d'Ollama

L'intégration d'Ollama pour la génération de transactions et l'analyse de la blockchain a nécessité des ajustements manuels :

- Amélioration des questions posées à l'IA pour obtenir des réponses pertinentes et bien organisées
- Gestion des délais d'attente quand l'IA prend du temps à répondre aux questions complexes
- Lecture et interprétation des réponses de l'IA qui n'étaient pas toujours parfaitement formatées

— Trouver le bon compromis entre la précision des analyses et la rapidité des réponses

7 Bilan

7.1 Conclusion

Ce projet a été une expérience enrichissante qui nous a permis de découvrir le monde fascinant de la blockchain et des cryptomonnaies tout en développant de nombreuses compétences techniques.

Grâce à ce travail, nous avons beaucoup progressé en Python, découvert les bases de la programmation C++ pour rendre les calculs plus rapides, appris comment sécuriser des données avec le hachage SHA-256, et compris comment plusieurs ordinateurs peuvent se mettre d'accord sur une même version de la blockchain. Nous avons également appris à travailler en équipe à distance en utilisant Discord pour communiquer et Git pour organiser notre code, et nous avons surmonté ensemble les nombreux défis techniques rencontrés.

Fiers du résultat obtenu et de tout ce que nous avons appris, nous sommes motivés à continuer notre exploration de ce domaine passionnant.

7.2 Perspectives

— **Connexion directe via Internet** : Connecter les ordinateurs directement sans réseau local
— **Proof of Stake** : Remplacer le système actuel par un système moins gourmand en électricité
— **Smart Contracts avancés** : Créer un langage pour des contrats plus complexes
— **IA prédictive** : Anticiper les comportements suspects avec l'apprentissage automatique
— **Application mobile** : Consulter et gérer la blockchain depuis un smartphone

8 Bibliographie

— Documentation officielle de Python 3.11 et Flask
— Satoshi Nakamoto - Bitcoin : A Peer-to-Peer Electronic Cash System (2008)
— Cours de cryptographie (SHA-256, signatures numériques)

9 Webographie

— ChatGPT et Gemini : Suggestions de code et explications techniques
— GitHub Copilot : Génération de code et résolution des erreurs
— Ollama Documentation : Intégration IA
— Visual Studio Build Tools : Compilation C++
— Tutoriels C++ pour Python (pybind11, Boost.Python)

A Cahier des charges

A.1 Objectifs principaux

- Créer une blockchain décentralisée complète avec Proof of Work
- Implémenter un système de transactions sécurisées avec signatures
- Développer une interface web pour visualiser et analyser la blockchain
- Intégrer l'IA pour analyse automatique et génération de données

A.2 Fonctionnalités attendues

A.2.1 Système Blockchain

- Création et validation de blocs avec calcul de hash SHA-256
- Algorithme de consensus (chaîne la plus longue)
- Gestion de portefeuilles avec soldes et historiques
- Synchronisation entre nœuds distribués

A.2.2 Minage et Performance

- Module C++ pour minage ultra-rapide (mine_module)
- Proof of Work avec difficulté ajustable
- Minage automatique en arrière-plan
- Récompenses de bloc et frais de transaction via smart contracts

A.2.3 Interface web

- Affichage des blocs, transactions, et statistiques
- Exploration détaillée (détails de bloc/transaction)
- Export CSV pour analyse externe
- Analyse IA conversationnelle avec Ollama
- Audit de portefeuille complet

A.3 Contraintes techniques

- Langage : Python 3.11
- Bibliothèques :
  - Flask (interface web)
  - hashlib (cryptographie)
  - sqlite3 (persistance)
  - threading/Queue (concurrence)
- Module C++ : mine_module pour optimisation du minage
- Compilation : Visual Studio Build Tools (Windows) ou gcc (Linux)
- Outils externes :
  - Ollama (Llama 3.2) pour analyse IA
  - GitHub Copilot pour l'assistance au codage

A.4 Livrables

A.4.1 Scripts fonctionnels

- `main.py` : Point d'entrée (menu + serveur Flask)

— `blockchain/*.py` : Modules blockchain
— `core/*.py` : Logique métier (minage, routes, utils, db)
— `mine_pow.cpp` : Module C++ optimisé
— `templates/*.html` : Interface web complète

A.4.2 Documentation

— Manuel utilisateur (installation, compilation, utilisation)
— Rapport détaillé (architecture, contributions, défis)

A.5 Calendrier

— Phase 1 : Développement de la blockchain de base (blocs, transactions)
— Phase 2 : Implémentation du minage et module C++
— Phase 3 : Création de l'interface web Flask
— Phase 4 : Intégration Ollama et analyse IA
— Phase 5 : Tests, optimisation et synchronisation réseau

B Manuel utilisateur

B.1 Installation et lancement

1. Créer l'environnement virtuel :

```
python -m venv venv
```

2. Activer l'environnement virtuel :

```
# Windows
venv\Scripts\activate
# Linux/Mac
source venv/bin/activate
```

3. Se placer dans le répertoire du projet :

```
cd d:\blockchainIA
```

4. Installer les dépendances Python :

```
pip install -r requirements.txt
```

5. Compiler le module C++ :

```
python setup.py build_ext --inplace
```

6. Lancer l'application :

```
python main.py
```

7. Accès à l'interface web :
   — Ouvrir `http://localhost:5003` dans un navigateur

B.2 Menu terminal

L'application propose un menu interactif avec les options suivantes :
— `1. Miner manuellement` : Mine un nombre spécifique de blocs
— `2. Afficher la blockchain` : Visualise tous les blocs et transactions
— `3. Afficher les transactions échouées` : Liste les transactions rejetées
— `4. Consulter un portefeuille` : Affiche solde et historique
— `5. Enregistrer un Pair` : Ajoute un nœud au réseau
— `6. Synchroniser la Chaîne` : Lance le consensus avec les pairs
— `0. Quitter` : Arrête l'application

B.3 Interface web - Pages disponibles

— `/` : Tableau de bord avec statistiques globales
— `/blocks` : Liste complète des blocs
— `/block/<index>` : Détails d'un bloc spécifique
— `/transactions` : Liste de toutes les transactions
— `/transaction/<id>` : Détails d'une transaction
— `/miners` : Statistiques des mineurs
— `/adresse/<wallet>` : Historique d'un portefeuille
— `/audit` : Audit complet d'un portefeuille
— `/blockchain-analysis` : Analyse IA de la blockchain
— `/security` : Sécurité et détection d'anomalies
— `/download` : Télécharger le projet

B.4 Export de données

L'application permet d'exporter les données sous plusieurs formats :
— `/export/blocks.csv` : Export CSV de tous les blocs
— `/export/transactions.csv` : Export CSV des transactions
— `/export/transactions-for-address.csv?address=<wallet>` : Transactions d'un portefeuille

B.5 Utilisation de l'IA (Ollama)

Pour utiliser l'analyse IA, Ollama doit être installé et en fonctionnement :

```
# Installer Ollama (ollama.ai)
# Lancer le modele Llama 3.2
ollama run llama3.2
```

L'IA permet :
— Poser des questions sur la blockchain via le bouton chat
— Analyser automatiquement les tendances et anomalies
— Générer des transactions réalistes pour tests

B.6 Arrêt du système

— Dans le menu terminal : Tapez `0`
— Ou utilisez `Ctrl+C` pour arrêter l'application et le serveur Flask

C Annexes

C.1 Évolution du projet

C.1.1 Création d'une blockchain simple avec blocs et transactions

```
import hashlib
import time

class Block:
    def __init__(self, index, transactions, previous_hash):
        self.index = index
        self.timestamp = time.time()
        self.transactions = transactions
        self.previous_hash = previous_hash
        self.hash = self.calculate_hash()

    def calculate_hash(self):
        block_string = f"{self.index}{self.timestamp}{self.
           transactions}{self.previous_hash}"
        return hashlib.sha256(block_string.encode()).hexdigest
           ()

class Blockchain:
    def __init__(self):
        self.chain = [self.create_genesis_block()]

    def create_genesis_block(self):
        return Block(0, [], "0")

    def add_block(self, transactions):
        previous_block = self.chain[-1]
        new_block = Block(len(self.chain), transactions,
           previous_block.hash)
        self.chain.append(new_block)
        return new_block

    def is_valid_chain(self):
        for i in range(1, len(self.chain)):
            current = self.chain[i]
            previous = self.chain[i-1]
            if current.hash != current.calculate_hash():
                return False
            if current.previous_hash != previous.hash:
                return False
        return True
```

C.1.2 Ajout du système de portefeuilles avec validation des soldes

```
class Blockchain:
    def __init__(self):
        self.chain = [self.create_genesis_block()]
```

```
        self.wallets = {}  # Format: {address: balance}

    def get_balance(self, address):
        """Calcule le solde actuel d'une adresse"""
        if address not in self.wallets:
            self.wallets[address] = 0.0
        return self.wallets[address]

    def create_transaction(self, sender, receiver, amount):
        """Cree une transaction avec verification du solde"""
        if self.get_balance(sender) < amount:
            print(f"Erreur: Solde insuffisant pour {sender}")
            return None

        # Deduire du sender
        self.wallets[sender] -= amount
        # Crediter le receiver
        if receiver not in self.wallets:
            self.wallets[receiver] = 0.0
        self.wallets[receiver] += amount

        return {'sender': sender, 'receiver': receiver, 'amount
           ': amount}

    def get_wallet_history(self, address):
        """Retourne l'historique complet d'un portefeuille"""
        history = []
        for block in self.chain:
            for tx in block.transactions:
                if tx['sender'] == address or tx['receiver'] ==
                    address:
                    history.append((block.index, tx))
        return history
```

C.1.3 Algorithme de Proof of Work basique en Python

```
import hashlib
import time

class Block:
    def __init__(self, index, transactions, previous_hash,
       difficulty=4):
        self.index = index
        self.timestamp = time.time()
        self.transactions = transactions
        self.previous_hash = previous_hash
        self.difficulty = difficulty
        self.nonce = 0
        self.hash = None

    def calculate_hash(self):
        """Calcule le hash du bloc avec le nonce actuel"""
```

```
        block_string = f"{self.index}{self.timestamp}{self.
            transactions}{self.previous_hash}{self.nonce}"
        return hashlib.sha256(block_string.encode()).hexdigest
            ()

    def mine_block(self):
        """Algorithme␣Proof␣of␣Work:␣trouve␣un␣hash␣avec␣N␣
            zeros␣au␣debut"""
        target = '0' * self.difficulty
        start_time = time.time()

        while True:
            self.hash = self.calculate_hash()

            # Verifie si le hash commence par le nombre requis
                de zeros
            if self.hash[:self.difficulty] == target:
                elapsed = time.time() - start_time
                print(f"Bloc␣{self.index}␣mine␣en␣{elapsed:.2f}
                    s␣avec␣nonce={self.nonce}")
                return self.hash

            self.nonce += 1

            # Affiche progression tous les 100000 essais
            if self.nonce % 100000 == 0:
                print(f"Tentative␣{self.nonce}...")
```

C.1.4 Module C++ ultra-rapide pour minage optimisé

```
// mine_pow.cpp
#include <pybind11/pybind11.h>
#include <openssl/sha.h>
#include <string>
#include <sstream>
#include <iomanip>

namespace py = pybind11;

std::string sha256(const std::string& input) {
    unsigned char hash[SHA256_DIGEST_LENGTH];
    SHA256((unsigned char*)input.c_str(), input.length(), hash)
        ;

    std::stringstream ss;
    for(int i = 0; i < SHA256_DIGEST_LENGTH; i++) {
        ss << std::hex << std::setw(2) << std::setfill('0')
            << (int)hash[i];
    }
    return ss.str();
}
```

```
py::tuple mine_pow_cpp(const std::string& block_data,
                       int difficulty,
                       long long max_nonce) {
    std::string target(difficulty, '0');

    for(long long nonce = 0; nonce < max_nonce; nonce++) {
        std::string data = block_data + std::to_string(nonce);
        std::string hash = sha256(data);

        if(hash.substr(0, difficulty) == target) {
            return py::make_tuple(nonce, hash);
        }
    }

    return py::make_tuple(-1, "");
}

PYBIND11_MODULE(mine_module, m) {
    m.def("mine_pow", &mine_pow_cpp, "Mine␣PoW␣optimise␣en␣C++"
       );
}
```

Compilation :

```
python setup.py build_ext --inplace
```

C.1.5 Système de minage automatique avec génération IA de transactions

```
import threading
from queue import Queue
import time

def auto_miner(blockchain, ai_queue, stop_event):
    """Thread␣de␣minage␣automatique"""
    while not stop_event.is_set():
        if not ai_queue.empty():
            transactions = []
            # Recuperer jusqu'a 5 transactions
            for _ in range(min(5, ai_queue.qsize())):
                tx = ai_queue.get()
                transactions.append(tx)

            if transactions:
                print(f"Minage␣de␣{len(transactions)}␣
                   transactions...")
                new_block = blockchain.add_block(transactions)
                new_block.mine_block()

                # Diffuser aux pairs
                broadcast_block(new_block, peers)

                # Sauvegarder
```

```
                blockchain.save_data()

        time.sleep(2)

def prefill_ai_queue(ai_queue, blockchain, stop_event):
    """Thread de generation de transactions IA"""
    from ai.generator import generate_ai_transactions

    while not stop_event.is_set():
        if ai_queue.qsize() < 10:
            txs = generate_ai_transactions(blockchain, count=5)
            for tx in txs:
                ai_queue.put(tx)
        time.sleep(5)

# Demarrage des threads
stop_event = threading.Event()
ai_queue = Queue()

miner_thread = threading.Thread(target=auto_miner,
                                args=(bc, ai_queue, stop_event)
                                   )
ai_thread = threading.Thread(target=prefill_ai_queue,
                             args=(ai_queue, bc, stop_event))

miner_thread.start()
ai_thread.start()
```

C.1.6 Implémentation des smart contracts pour gestion automatique des frais

```
class SmartContract:
    def __init__(self, name, logic):
        self.name = name
        self.logic = logic

    def execute(self, blockchain, block):
        """Execute la logique du contrat"""
        try:
            return self.logic(blockchain, block)
        except Exception as e:
            print(f"Erreur dans le contrat {self.name}: {e}")
            return False

def mining_fee_contract(blockchain, block):
    """Contrat de recompense de minage"""
    base_reward = 5.0  # Euros
    fee_per_tx = 0.5   # Euros par transaction

    # Calculer la recompense totale
    total_fees = len(block.transactions) * fee_per_tx
    total_reward = base_reward + total_fees
```

```
    # Crediter le mineur
    miner_address = block.miner
    if miner_address not in blockchain.wallets:
        blockchain.wallets[miner_address] = 0.0
    blockchain.wallets[miner_address] += total_reward

    print(f"Mineur␣{miner_address}␣recompense:␣{total_reward}␣
       EUR")
    return True

# Enregistrer le contrat
contracts = {
    'mining_fee': SmartContract('mining_fee',
       mining_fee_contract)
}
```

C.1.7 Système complet avec sauvegarde JSON et consensus distribué

```
import json
import requests

class Blockchain:
    def save_data(self, filename='blockchain_data.json'):
        """Sauvegarde␣la␣blockchain␣en␣JSON"""
        data = {
            'chain': [block.to_dict() for block in self.chain],
            'wallets': self.wallets
        }
        with open(filename, 'w') as f:
            json.dump(data, f, indent=2)

    def load_data(self, filename='blockchain_data.json'):
        """Charge␣la␣blockchain␣depuis␣JSON"""
        try:
            with open(filename, 'r') as f:
                data = json.load(f)
            self.chain = [Block.from_dict(b) for b in data['
               chain']]
            self.wallets = data['wallets']
        except FileNotFoundError:
            print("Fichier␣non␣trouve,␣creation␣d'une␣nouvelle␣
               blockchain")

    def resolve_conflicts(self, peers):
        """Algorithme␣de␣consensus:␣chaine␣la␣plus␣longue"""
        longest_chain = None
        max_length = len(self.chain)

        for peer in peers:
            try:
                response = requests.get(f'http://{peer}/chain',
                    timeout=5)
```

```
                if response.status_code == 200:
                    data = response.json()
                    length = data['length']
                    chain = data['chain']

                    if length > max_length and self.
                       is_valid_chain(chain):
                        max_length = length
                        longest_chain = chain
            except:
                continue

        if longest_chain:
            self.chain = [Block.from_dict(b) for b in
               longest_chain]
            return True
        return False
```

C.1.8 Création d'une page Flask pour afficher les blocs de la blockchain

```
from flask import Flask, render_template, jsonify

app = Flask(__name__)

@app.route('/')
def home():
    """Dashboard␣principal"""
    stats = {
        'total_blocks': len(bc.chain),
        'total_wallets': len(bc.wallets),
        'total_transactions': sum(len(b.transactions) for b in
           bc.chain)
    }
    return render_template('home.html', stats=stats)

@app.route('/blocks')
def blocks_list():
    """Liste␣de␣tous␣les␣blocs"""
    blocks = []
    for block in bc.chain:
        blocks.append({
            'index': block.index,
            'hash': block.hash,
            'previous_hash': block.previous_hash,
            'timestamp': block.timestamp,
            'transactions': len(block.transactions),
            'miner': getattr(block, 'miner', 'Genesis')
        })
    return render_template('blocks_list.html', blocks=blocks)

@app.route('/block/<int:index>')
def block_detail(index):
```

```
    """Detail␣d'un␣bloc␣specifique"""
    if index >= len(bc.chain):
        return render_template('error.html', message='Bloc␣non␣
           trouve'), 404

    block = bc.chain[index]
    return render_template('block_detail.html', block=block)

@app.route('/chain', methods=['GET'])
def full_chain():
    """API␣JSON␣de␣la␣blockchain␣complete"""
    return jsonify({
        'chain': bc.to_json_chain(),
        'length': len(bc.chain)
    }), 200
```

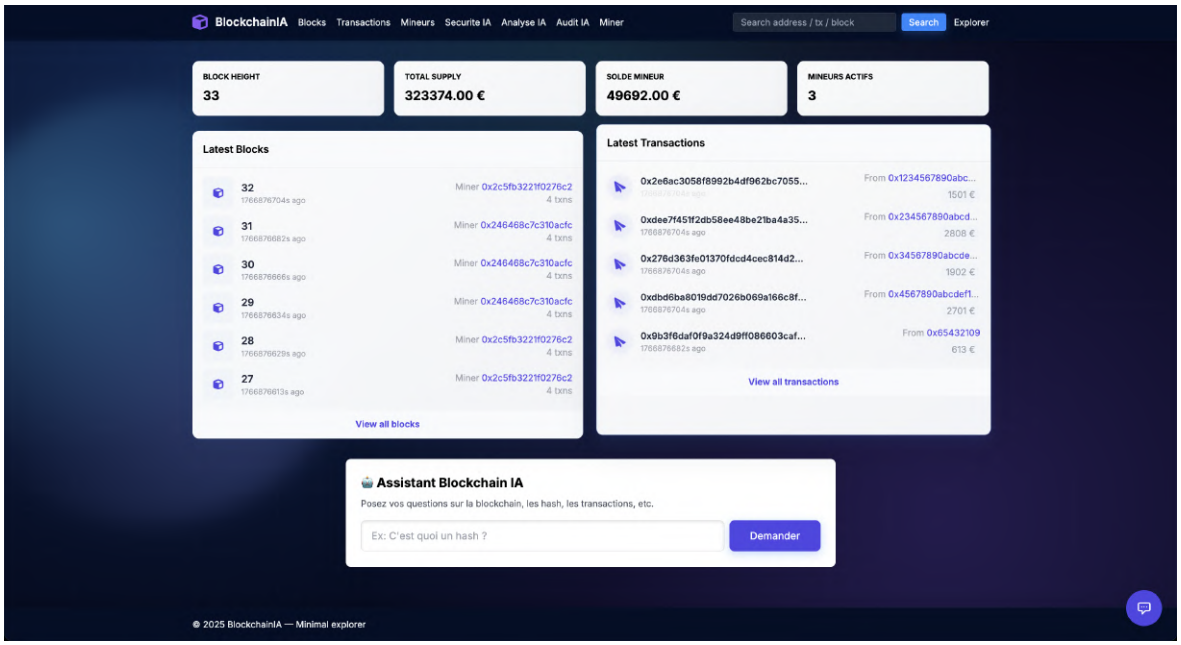

FIGURE 1 – Dashboard principal : statistiques globales de la blockchain

C.2 Intégration de l'IA

C.2.1 Fonction generate_ai_transactions() : génération automatique par l'IA

```
# ai/generator.py
import ollama
import random

def generate_ai_transactions(blockchain, count=5):
    """Genere␣des␣transactions␣realistes␣avec␣Ollama"""
    wallets = list(blockchain.wallets.keys())
    if len(wallets) < 2:
        return []
```

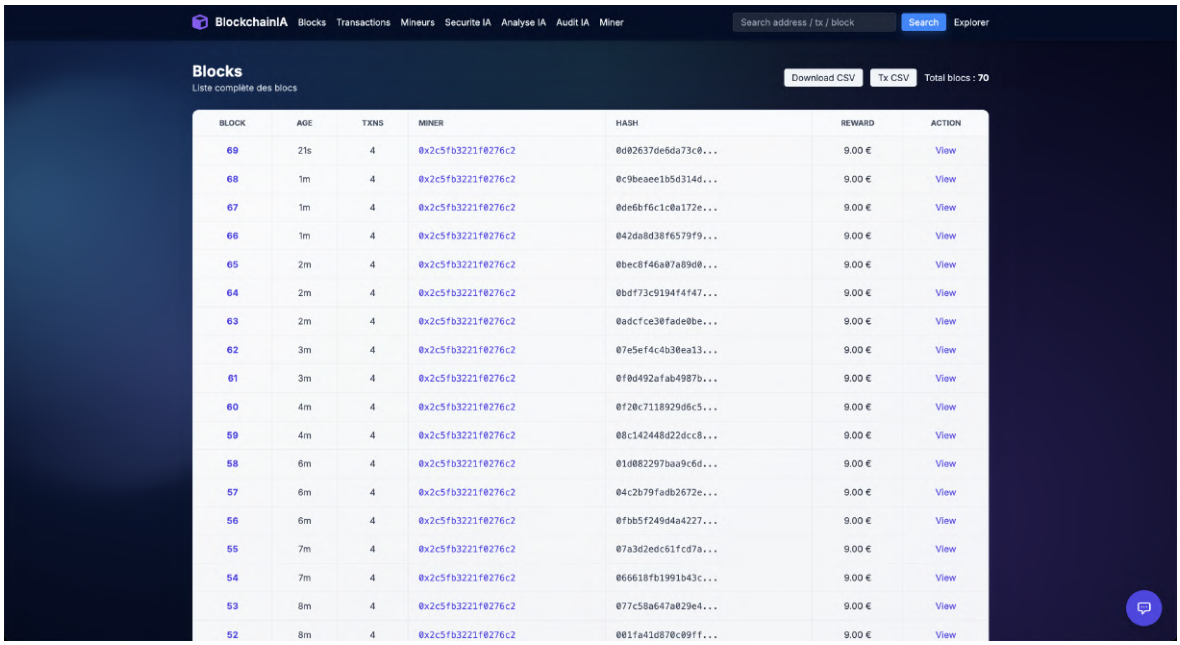

FIGURE 2 – Page d'exploration : liste des blocs avec détails des transactions

FIGURE 3 – Synchronisation réseau : connexion à un pair via ZeroTier

```
    # Selectionner aleatoirement des portefeuilles
    selected = random.sample(wallets, min(count * 2, len(
        wallets)))

    prompt = f"""Generate␣{count}␣realistic␣blockchain␣
        transactions.
Format:␣sender|receiver␣(one␣per␣line)
Available␣wallets:␣{',␣'.join(selected[:10])}
"""

    try:
        response = ollama.chat(
            model='llama3.2',
            messages=[{'role': 'user', 'content': prompt}]
```

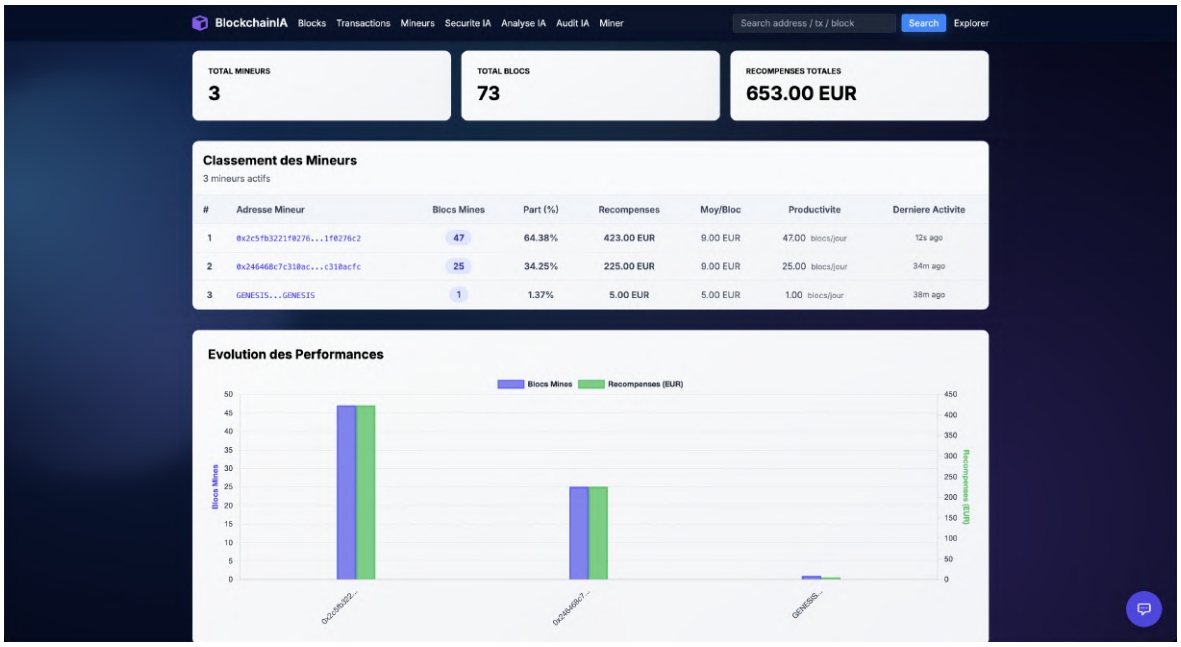

FIGURE 4 – Statistiques des mineurs : classement et apparition d'un nouveau mineur

```
        )

        lines = response['message']['content'].strip().split('\
            n')
        transactions = []

        for line in lines[:count]:
            if '|' in line:
                parts = line.split('|')
                sender = parts[0].strip()
                receiver = parts[1].strip()
                amount = random.uniform(100, 3000)

                if sender in wallets and receiver in wallets:
                    from blockchain.transaction import
                        Transaction
                    tx = Transaction(sender, receiver, amount)
                    transactions.append(tx)

        return transactions
    except Exception as e:
        print(f"Erreur generation IA: {e}")
        return []
```

C.2.2 Route /analyze-blockchain : dialogue conversationnel avec Ollama

```
# core/routes.py
@app.route('/analyze-blockchain', methods=['POST'])
def analyze_blockchain():
```

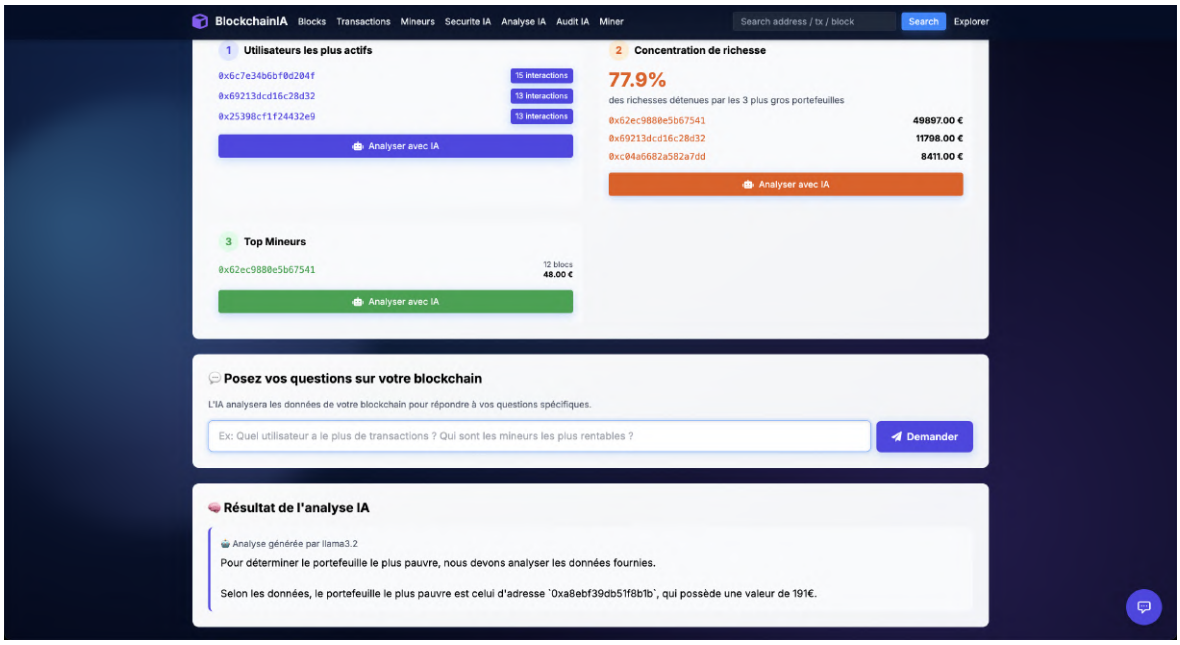

FIGURE 5 – Analyse IA avec Ollama : question posée et réponse générée

```
"""Route␣pour␣analyser␣la␣blockchain␣avec␣donnees␣completes
    """
data = request.get_json()
question = data.get('question', '').strip()

if not question:
    return jsonify({'error': 'Question␣vide'}), 400

# Collecter statistiques
total_blocks = len(bc.chain)
total_wallets = len(bc.wallets)
total_txs = sum(len(block.transactions) for block in bc.
    chain)

# Top utilisateurs actifs
activity_counter = {}
for block in bc.chain:
    for tx in block.transactions:
        sender = getattr(tx, 'sender', None)
        receiver = getattr(tx, 'receiver', None)
        if sender:
            activity_counter[sender] = activity_counter.get
                (sender, 0) + 1
        if receiver:
            activity_counter[receiver] = activity_counter.
                get(receiver, 0) + 1

top_users = sorted(activity_counter.items(),
                   key=lambda x: x[1], reverse=True)[:5]
```

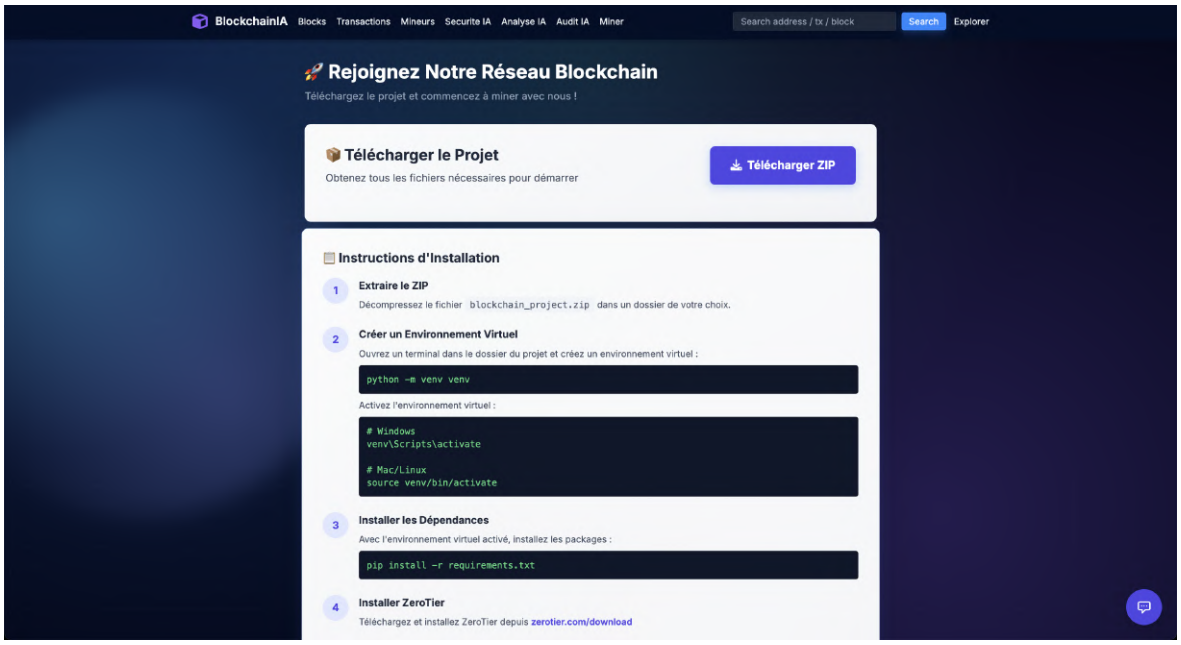

FIGURE 6 – Page de téléchargement du projet

```
    # Preparer contexte pour Ollama
    context = f"""Analyste␣blockchain␣expert.
Donnees:␣{total_blocks}␣blocs,␣{total_wallets}␣portefeuilles,␣{
   total_txs}␣TX
Question:␣{question}"""

    # Appel Ollama
    import requests
    response = requests.post('http://localhost:11434/api/
       generate',
        json={'model': 'llama3.2', 'prompt': context, 'stream':
            False},
        timeout=120)

    if response.status_code == 200:
        result = response.json()
        return jsonify({'answer': result.get('response', 'Pas␣
           de␣reponse')})

    return jsonify({'error': 'Ollama␣non␣disponible'}), 503
```

C.2.3 Préparation du contexte blockchain pour l'IA

```
def prepare_blockchain_context(blockchain):
    """Prepare␣un␣resume␣de␣la␣blockchain␣pour␣l'IA"""

    # Statistiques globales
    total_blocks = len(blockchain.chain)
    total_wallets = len(blockchain.wallets)
```

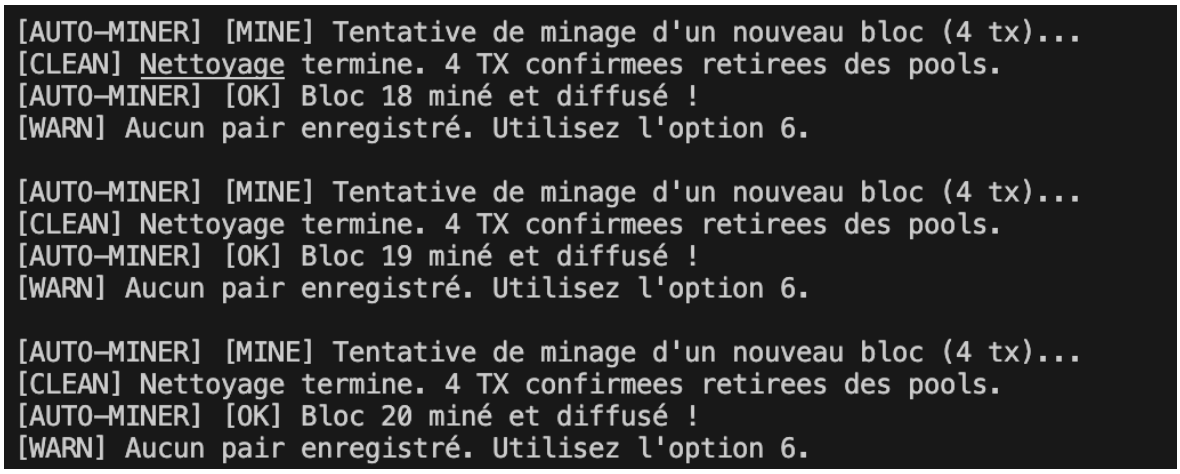

FIGURE 7 – Auto-mineur activé : processus de minage automatique en action

```
    total_txs = sum(len(b.transactions) for b in blockchain.
       chain)
    total_wealth = sum(blockchain.wallets.values())

    # Top portefeuilles
    sorted_wallets = sorted(blockchain.wallets.items(),
                            key=lambda x: x[1], reverse=True)
    richest = sorted_wallets[:5]
    poorest = sorted_wallets[-5:] if len(sorted_wallets) > 5
       else sorted_wallets

    # Statistiques mineurs
    miner_stats = {}
    for block in blockchain.chain[1:]:
        miner = getattr(block, 'miner', None)
        if miner:
            if miner not in miner_stats:
                miner_stats[miner] = {'blocks': 0, 'rewards':
                   0}
            miner_stats[miner]['blocks'] += 1
            miner_stats[miner]['rewards'] += len(block.
               transactions) * 0.5

    top_miners = sorted(miner_stats.items(),
                        key=lambda x: x[1]['blocks'], reverse=
                           True)[:5]

    # Formater le contexte
    context = f"""BLOCKCHAIN␣STATS:
-␣Blocs:␣{total_blocks}
-␣Portefeuilles:␣{total_wallets}
-␣Transactions:␣{total_txs}
-␣Richesse␣totale:␣{total_wealth:.2f}␣EUR

TOP␣5␣RICHES:\n"""
```

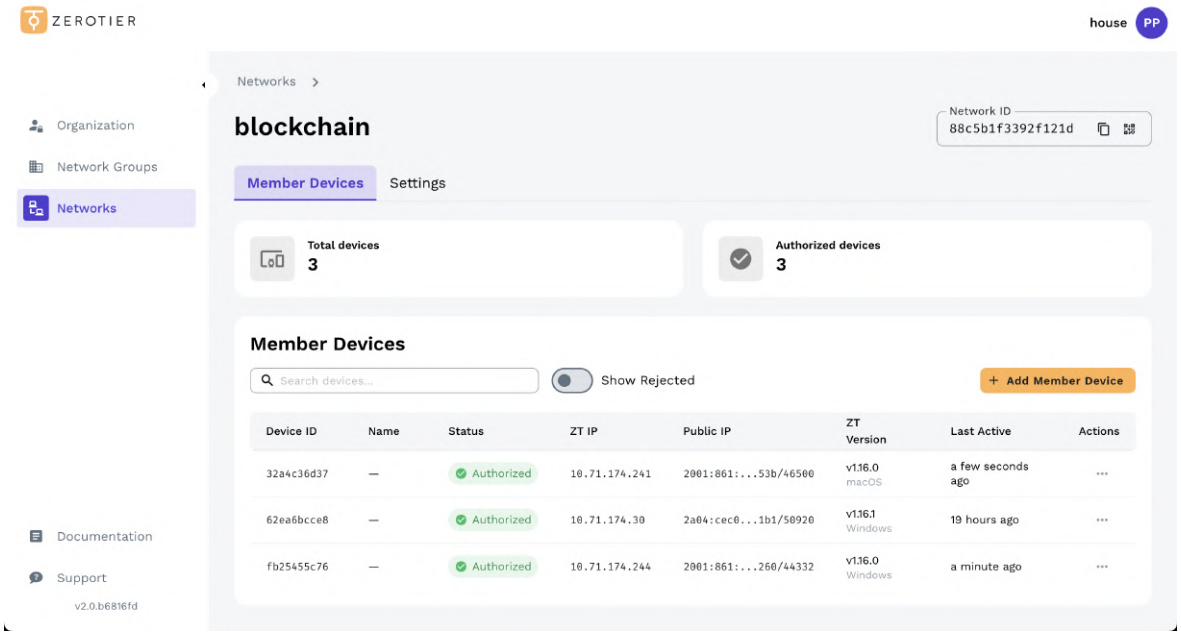

FIGURE 8 – Test ZeroTier : configuration du réseau VPN pour blockchain distribuée

```
    for i, (wallet, balance) in enumerate(richest, 1):
        context += f"{i}. {wallet}: {balance:.2f} EUR\n"

    return context
```

C.2.4 Détection automatique de failles et vulnérabilités

```
# core/routes.py
@app.route('/security')
def view_security():
    """Page de securite avec detection d'anomalies"""
    anomalies = []

    # 1. Concentration de richesse
    total_wealth = sum(bc.wallets.values())
    sorted_wallets = sorted(bc.wallets.items(),
                            key=lambda x: x[1], reverse=True)

    for wallet, balance in sorted_wallets:
        percentage = (balance / total_wealth * 100) if
           total_wealth > 0 else 0
        if percentage > 30:
            anomalies.append({
                'type': 'wealth_concentration',
                'severity': 'high',
                'message': f'{wallet} detient {percentage:.1f}%
                   de la richesse',
                'wallet': wallet
```

```
            })

    # 2. Dominance de mineurs
    total_blocks = len(bc.chain) - 1
    miner_stats = {}
    for block in bc.chain[1:]:
        miner = getattr(block, 'miner', None)
        if miner:
            miner_stats[miner] = miner_stats.get(miner, 0) + 1

    for miner, count in miner_stats.items():
        percentage = (count / total_blocks * 100) if
           total_blocks > 0 else 0
        if percentage > 40:
            anomalies.append({
                'type': 'miner_dominance',
                'severity': 'high',
                'message': f'{miner} a mine {percentage:.1f}%
                   des blocs',
                'wallet': miner
            })

    # 3. Patterns suspects (montants identiques)
    amount_counter = {}
    for block in bc.chain:
        for tx in block.transactions:
            amt = getattr(tx, 'amount', 0)
            amount_counter[amt] = amount_counter.get(amt, 0) +
               1

    for amount, count in amount_counter.items():
        if count > 5:
            anomalies.append({
                'type': 'suspicious_pattern',
                'severity': 'medium',
                'message': f'Montant {amount:.2f} EUR repete {
                   count} fois'
            })

    return render_template('security.html', anomalies=anomalies
       )
```

C.2.5 Analyse de concentration des richesses

```
def check_wealth_distribution(blockchain, threshold=30):
    """Verifie si la richesse est trop concentree"""
    total_wealth = sum(blockchain.wallets.values())

    if total_wealth == 0:
        return {'status': 'ok', 'message': 'Pas de richesse'}

    sorted_wallets = sorted(blockchain.wallets.items(),
```

```
                            key=lambda x: x[1], reverse=True)

    # Analyser top 3
    top_3_wealth = sum(w[1] for w in sorted_wallets[:3])
    concentration = (top_3_wealth / total_wealth * 100)

    if concentration > threshold:
        return {
            'status': 'warning',
            'severity': 'high' if concentration > 50 else '
               medium',
            'concentration': concentration,
            'message': f'Top␣3␣portefeuilles␣detiennent␣{
               concentration:.1f}%',
            'top_wallets': sorted_wallets[:3]
        }

    return {
        'status': 'ok',
        'concentration': concentration,
        'message': f'Distribution␣equilibree␣({concentration:.1
           f}%)'
    }

# Exemple d'utilisation
result = check_wealth_distribution(bc, threshold=30)
if result['status'] == 'warning':
    print(f"ALERTE:␣{result['message']}")
    print("Top␣portefeuilles:")
    for wallet, balance in result['top_wallets']:
        print(f"␣␣{wallet}:␣{balance:.2f}␣EUR")
```

C.2.6 Détection de monopoles de mineurs (risque attaque 51%)

```
def analyze_miner_dominance(blockchain, threshold=40):
    """Detecte␣si␣un␣mineur␣domine␣le␣reseau␣(risque␣attaque␣
       51%)"""
    total_blocks = len(blockchain.chain) - 1  # Exclure genesis

    if total_blocks == 0:
        return {'status': 'ok', 'message': 'Pas␣de␣blocs␣mines'
           }

    # Compter blocs par mineur
    miner_stats = {}
    for block in blockchain.chain[1:]:
        miner = getattr(block, 'miner', None)
        if miner:
            if miner not in miner_stats:
                miner_stats[miner] = {'blocks': 0, 'percentage'
                   : 0}
            miner_stats[miner]['blocks'] += 1
```

```
    # Calculer pourcentages
    for miner, stats in miner_stats.items():
        stats['percentage'] = (stats['blocks'] / total_blocks *
            100)

    # Chercher dominance
    dominant_miners = []
    for miner, stats in miner_stats.items():
        if stats['percentage'] > threshold:
            dominant_miners.append({
                'miner': miner,
                'blocks': stats['blocks'],
                'percentage': stats['percentage'],
                'severity': 'critical' if stats['percentage'] >
                    51 else 'high'
            })

    if dominant_miners:
        return {
            'status': 'warning',
            'dominant_miners': dominant_miners,
            'message': f'{len(dominant_miners)} mineur(s)
               dominant(s) detecte(s)'
        }

    return {'status': 'ok', 'message': 'Minage bien distribue'}

# Utilisation
result = analyze_miner_dominance(bc, threshold=40)
if result['status'] == 'warning':
    for miner_info in result['dominant_miners']:
        print(f"ALERTE {miner_info['severity'].upper()}: "
              f"{miner_info['miner']} controle {miner_info['
                 percentage']:.1f}%")
```

C.2.7 Identification de comportements suspects (wash trading)

```
def find_suspicious_patterns(blockchain):
    """Detecte les patterns suspects (montants repetitifs,
       haute frequence)"""
    anomalies = []

    # 1. Montants identiques repetes (possible wash trading)
    amount_counter = {}
    amount_details = {}

    for block in blockchain.chain:
        for tx in block.transactions:
            amt = round(getattr(tx, 'amount', 0), 2)
            sender = getattr(tx, 'sender', None)
            receiver = getattr(tx, 'receiver', None)
```

```
            if amt not in amount_counter:
                amount_counter[amt] = 0
                amount_details[amt] = []

            amount_counter[amt] += 1
            amount_details[amt].append((sender, receiver))

    for amount, count in amount_counter.items():
        if count > 5:  # Meme montant plus de 5 fois
            anomalies.append({
                'type': 'repeated_amount',
                'severity': 'medium',
                'amount': amount,
                'count': count,
                'message': f'Montant {amount:.2f} EUR repete {
                    count} fois',
                'details': amount_details[amount][:3]  # Top 3
                    exemples
            })

    # 2. Transactions circulaires (A->B, B->A)
    circular = []
    for block in blockchain.chain:
        for tx in block.transactions:
            sender = getattr(tx, 'sender', None)
            receiver = getattr(tx, 'receiver', None)

            # Chercher transaction inverse
            for block2 in blockchain.chain:
                for tx2 in block2.transactions:
                    if (getattr(tx2, 'sender', None) ==
                        receiver and
                        getattr(tx2, 'receiver', None) ==
                            sender):
                        circular.append((sender, receiver))

    if circular:
        anomalies.append({
            'type': 'circular_transactions',
            'severity': 'high',
            'count': len(circular),
            'message': f'{len(circular)} transaction(s)
                circulaire(s) detectee(s)'
        })

    return anomalies
```

C.2.8 Détection de transferts circulaires (A→B→A)

```
def detect_repeated_transfers(blockchain, window=10):
    """Detecte les transferts repetes entre memes adresses"""
```

```
    # Construire historique des paires
    transfer_pairs = {}

    for block in blockchain.chain:
        for tx in block.transactions:
            sender = getattr(tx, 'sender', None)
            receiver = getattr(tx, 'receiver', None)
            amount = getattr(tx, 'amount', 0)
            timestamp = getattr(tx, 'timestamp', 0)

            if sender and receiver:
                pair = tuple(sorted([sender, receiver]))

                if pair not in transfer_pairs:
                    transfer_pairs[pair] = []

                transfer_pairs[pair].append({
                    'sender': sender,
                    'receiver': receiver,
                    'amount': amount,
                    'timestamp': timestamp
                })

    # Analyser patterns
    suspicious = []
    for pair, transfers in transfer_pairs.items():
        if len(transfers) >= window:
            # Verifier si alterne (A->B, B->A, A->B...)
            alternating = True
            for i in range(1, len(transfers)):
                if transfers[i]['sender'] == transfers[i-1]['
                   sender']:
                    alternating = False
                    break

            if alternating:
                total_amount = sum(t['amount'] for t in
                   transfers)
                suspicious.append({
                    'pair': pair,
                    'count': len(transfers),
                    'total_amount': total_amount,
                    'severity': 'high',
                    'message': f'{len(transfers)} transferts
                       alternants entre '
                               f'{pair[0][:10]}... et {pair
                                  [1][:10]}...'
                })

    return suspicious
```

C.2.9 Évaluation du niveau de gravité des anomalies

```
def assess_severity(anomaly_type, metrics):
    """Evalue le niveau de gravite d'une anomalie detectee"""

    severity_rules = {
        'wealth_concentration': {
            'critical': lambda m: m['percentage'] > 70,
            'high': lambda m: m['percentage'] > 50,
            'medium': lambda m: m['percentage'] > 30,
            'low': lambda m: True
        },
        'miner_dominance': {
            'critical': lambda m: m['percentage'] > 51,  #
               Attaque 51%
            'high': lambda m: m['percentage'] > 40,
            'medium': lambda m: m['percentage'] > 25,
            'low': lambda m: True
        },
        'suspicious_pattern': {
            'high': lambda m: m['count'] > 20,
            'medium': lambda m: m['count'] > 10,
            'low': lambda m: True
        },
        'circular_transactions': {
            'critical': lambda m: m['total_amount'] > 100000,
            'high': lambda m: m['count'] > 10,
            'medium': lambda m: m['count'] > 5,
            'low': lambda m: True
        }
    }

    if anomaly_type not in severity_rules:
        return 'unknown'

    rules = severity_rules[anomaly_type]

    for severity in ['critical', 'high', 'medium', 'low']:
        if severity in rules and rules[severity](metrics):
            return severity

    return 'low'

# Exemple d'utilisation
anomalies = find_suspicious_patterns(bc)
for anomaly in anomalies:
    severity = assess_severity(anomaly['type'], anomaly)
    print(f"[{severity.upper()}] {anomaly['message']}")
```

C.2.10 Communication HTTP avec l'API Ollama

```
import requests
import json

def query_ollama_api(prompt, model='llama3.2', temperature=0.7,
    max_tokens=800):
    """Envoie une requete a l'API Ollama et retourne la reponse
       """

    url = 'http://localhost:11434/api/generate'

    payload = {
        'model': model,
        'prompt': prompt,
        'stream': False,
        'options': {
            'temperature': temperature,
            'num_predict': max_tokens
        }
    }

    try:
        response = requests.post(
            url,
            json=payload,
            headers={'Content-Type': 'application/json'},
            timeout=120
        )

        if response.status_code == 200:
            data = response.json()
            return {
                'success': True,
                'response': data.get('response', ''),
                'model': data.get('model', model),
                'done': data.get('done', False)
            }
        else:
            return {
                'success': False,
                'error': f'HTTP {response.status_code}',
                'message': 'Ollama API non disponible'
            }

    except requests.exceptions.Timeout:
        return {'success': False, 'error': 'timeout',
                'message': 'Delai depasse (120s)'}

    except requests.exceptions.ConnectionError:
        return {'success': False, 'error': 'connection',
                'message': 'Impossible de contacter Ollama. '
                           'Verifiez qu\'Ollama est demarre.'}
```

```
    except Exception as e:
        return {'success': False, 'error': 'unknown', 'message'
            : str(e)}
```

C.3 Exemple d'exécution du projet

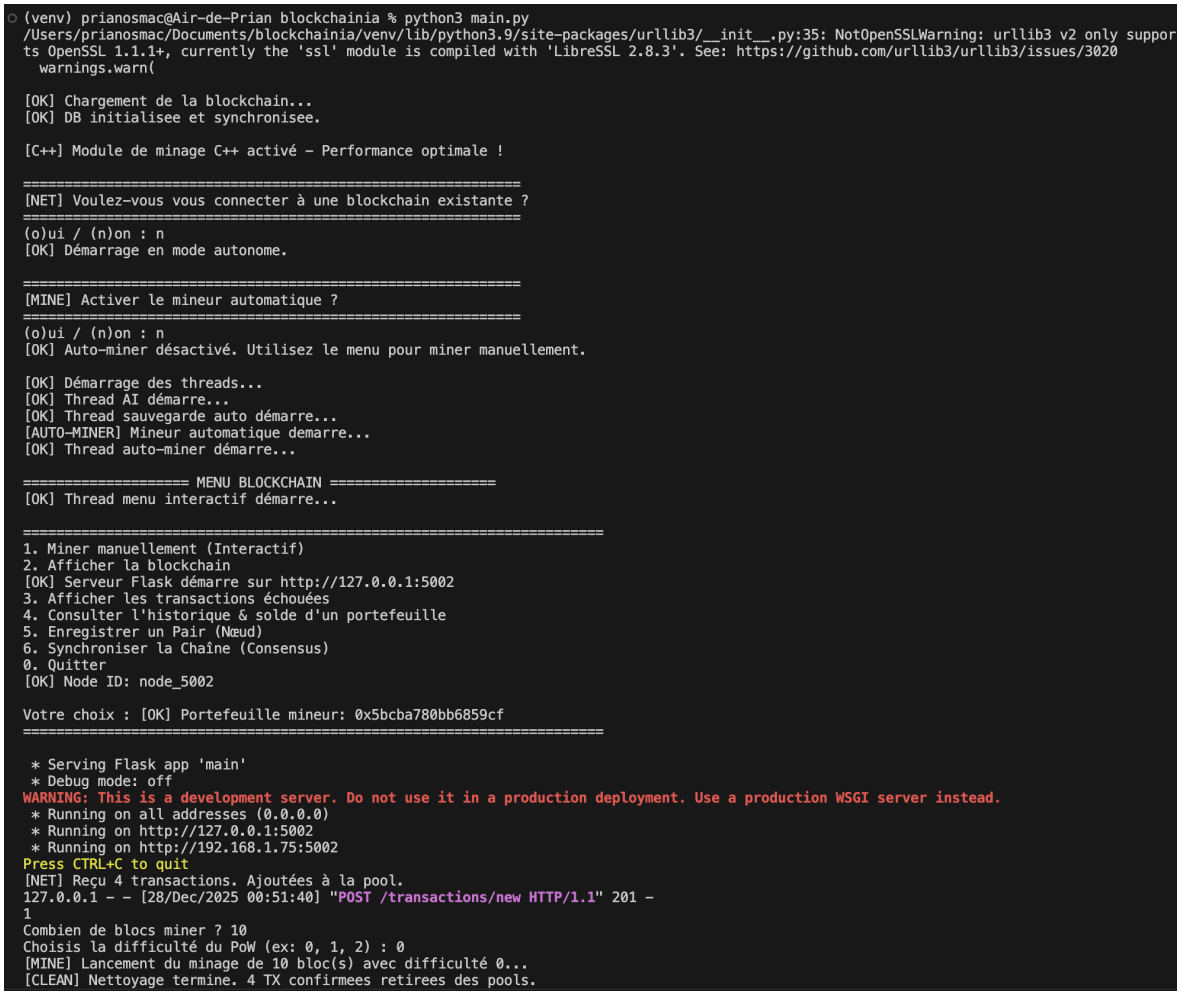

FIGURE 9 – Menu interactif au démarrage de l'application

Listing 1 – Option 5 : Enregistrer un Pair

```
5. Enregistrer un Pair
Adresse du pair    ajouter (ex: 192.168.1.10:5003) : 192.168.1.11:5003
Pair ajout  avec succ s !
```

Listing 2 – Option 6 : Synchroniser la Chaîne

```
6. Synchroniser la Cha ne
Synchronisation en cours avec les pairs...
Cha ne mise    jour : la cha ne distante  tait  plus longue et valide.
```

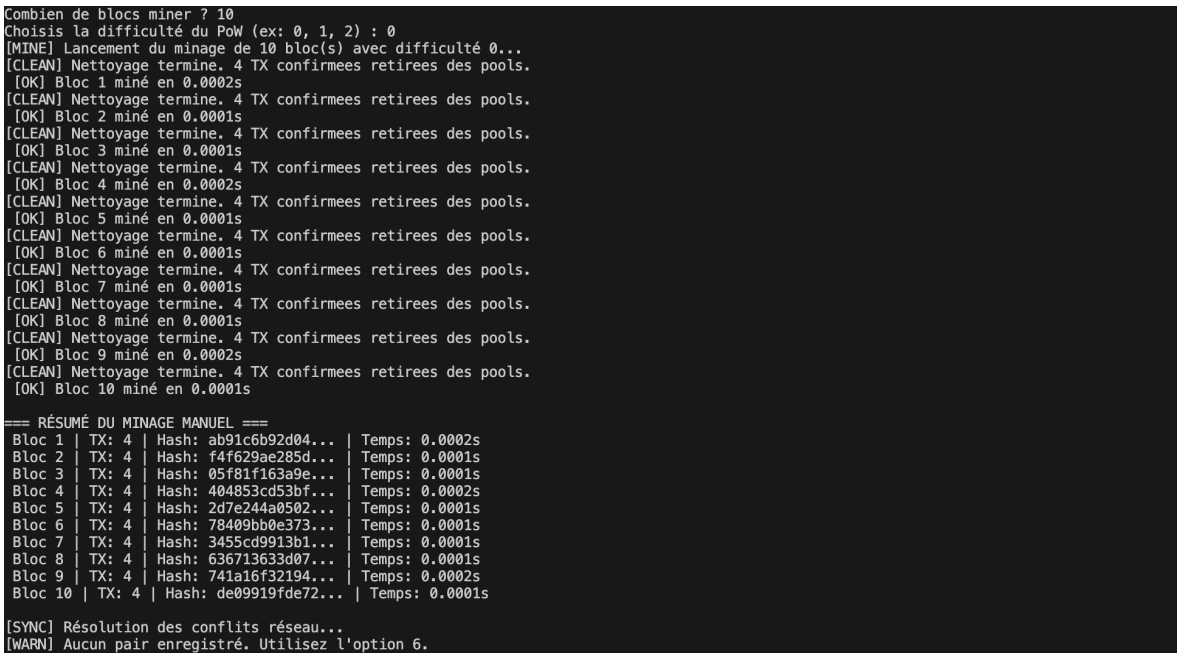

FIGURE 10 – Minage d'un bloc avec calcul du nonce et hash

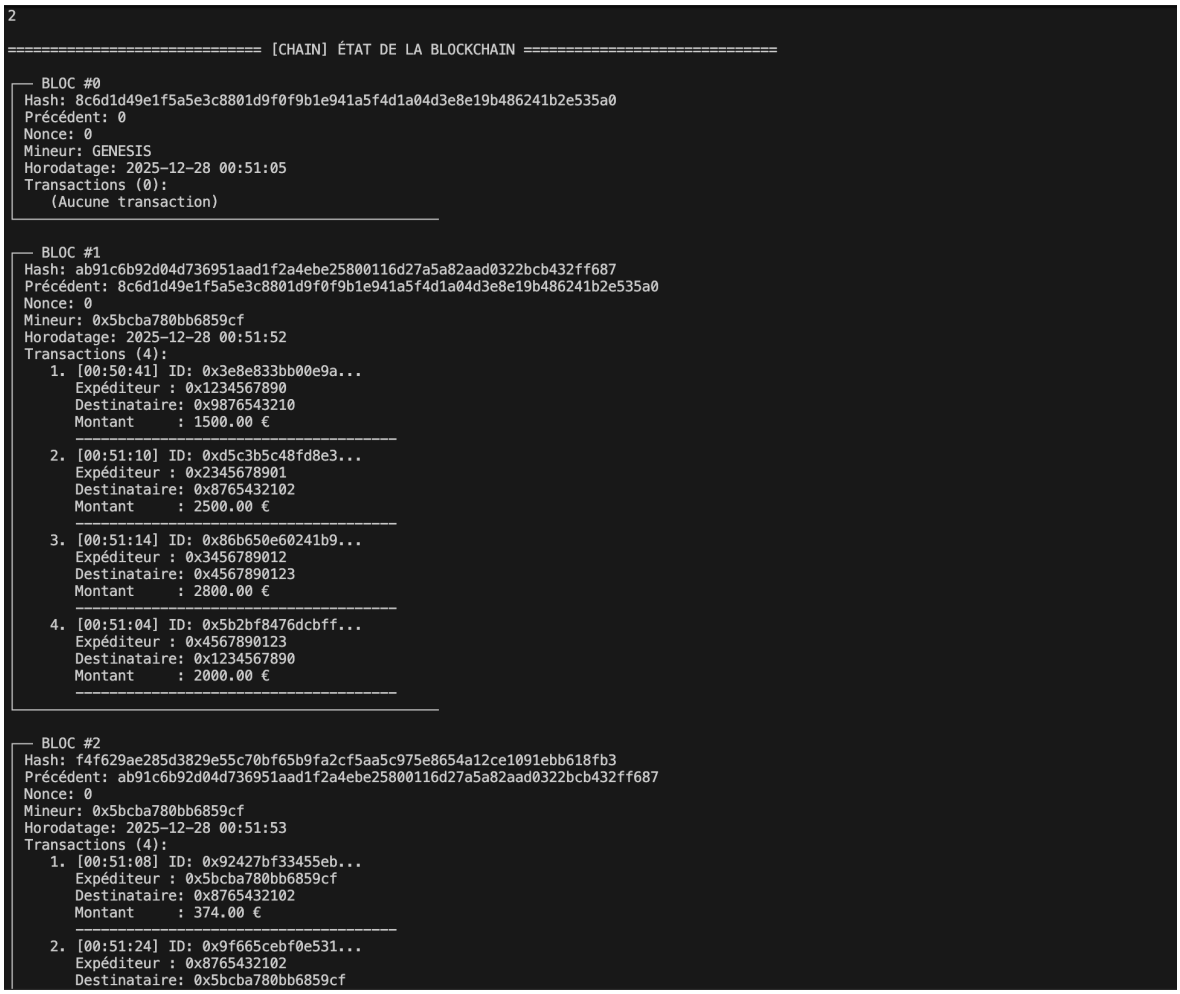

FIGURE 11 – Visualisation complète de la chaîne de blocs avec transactions

```
4
Entrez l'ID du portefeuille (ex: 0x...) : 0x3456789012

================================================================================
 SOLDE : 5417.00 €  |  WALLET : 0x3456789012
================================================================================

--- HISTORIQUE DES TRANSACTIONS ---
 0x86b650e60241b90f6d0983878e2b3771271cae0aa57e5838cb1381b27f365ae2 : 0x3456789012 -> 0x4567890123 | 2800.00 € | 28/12/2025 00:51:14
 0x4b3be9a07f16e4bad10a084d5d9331ea4316520887fb53ce3c4bc928b02c4288 : 0x3456789012 -> 0x2345678901 | 234.00 € | 28/12/2025 00:51:15
 0x98bbf356a0d39d4425e6e1dd52e0c28ad8d0fd299c7ba86a9e1e36fa342f678d : 0x8765432102 -> 0x3456789012 | 1064.00 € | 28/12/2025 00:51:01
 0xcced2c4375aff568579ad52bf6c2155467de7f6acbc3b96abbede0855b9c2b7d : 0x4567890123 -> 0x3456789012 | 1006.00 € | 28/12/2025 00:50:58
 0x8b2349c97bd31291fc608d191632f949a522e5af89c817e45d34026ab2b1f65d : 0x9876543210 -> 0x3456789012 | 1149.00 € | 28/12/2025 00:51:45
 0xfb8e59ded314ad9550350db689840d32ac689a95ff1d2fe5ad753c97bbacb902 : 0x3456789012 -> 0x2345678901 | 1099.00 € | 28/12/2025 00:51:33
 0x7241ec2199781da801d5e918e88f27ca1aa34aae7ea2902fdec4cafdf6267e1c : 0x2345678901 -> 0x3456789012 | 568.00 € | 28/12/2025 00:51:05
 0x6b852aa3cdeec2ecba57c349eca09087ec0405aa85879caa7fbda13ae07892fa : 0x4567890123 -> 0x3456789012 | 1195.00 € | 28/12/2025 00:51:50
 0xe81f84f0ca05e50dd246e4c6fa51be84b2c13d9ed3dc365b469599d73ab1fd95 : 0x2345678901 -> 0x3456789012 | 580.00 € | 28/12/2025 00:51:42
 0x6d79bd863a54e6632c0c05f38eecd60e38c22eb62bae8e4d86228629dc2cd227 : 0x3456789012 -> 0x5bcba780bb6859cf | 1008.00 € | 28/12/2025 00:52:01

--- FRAIS ET RÉCOMPENSES ---
 [00:51:14] Frais : -1.00 € (PAYÉ)
 [00:51:15] Frais : -1.00 € (PAYÉ)
 [00:51:33] Frais : -1.00 € (PAYÉ)
 [00:52:01] Frais : -1.00 € (PAYÉ)

--- TRANSACTIONS ÉCHOUÉES (POOL) ---
 (aucune transaction échouée)
```

FIGURE 12 – Affichage du solde et de l'historique complet d'un portefeuille

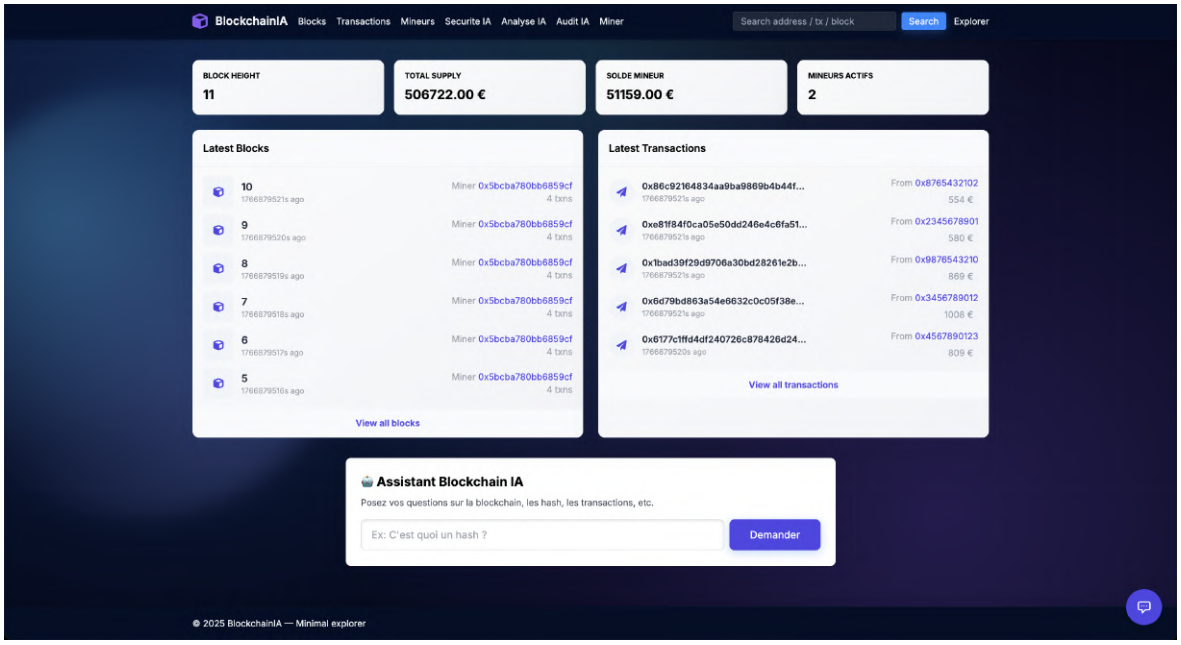

FIGURE 13 – Interface web : Dashboard avec statistiques globales

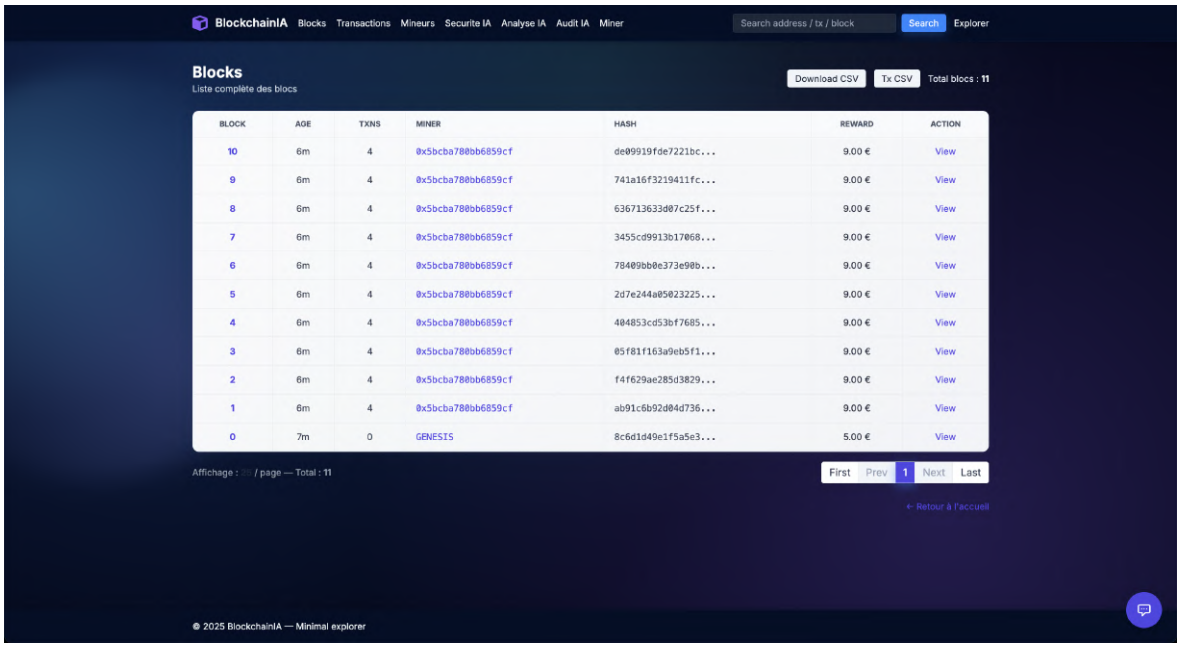

FIGURE 14 – Page d'exploration des blocs avec hash, timestamp et transactions

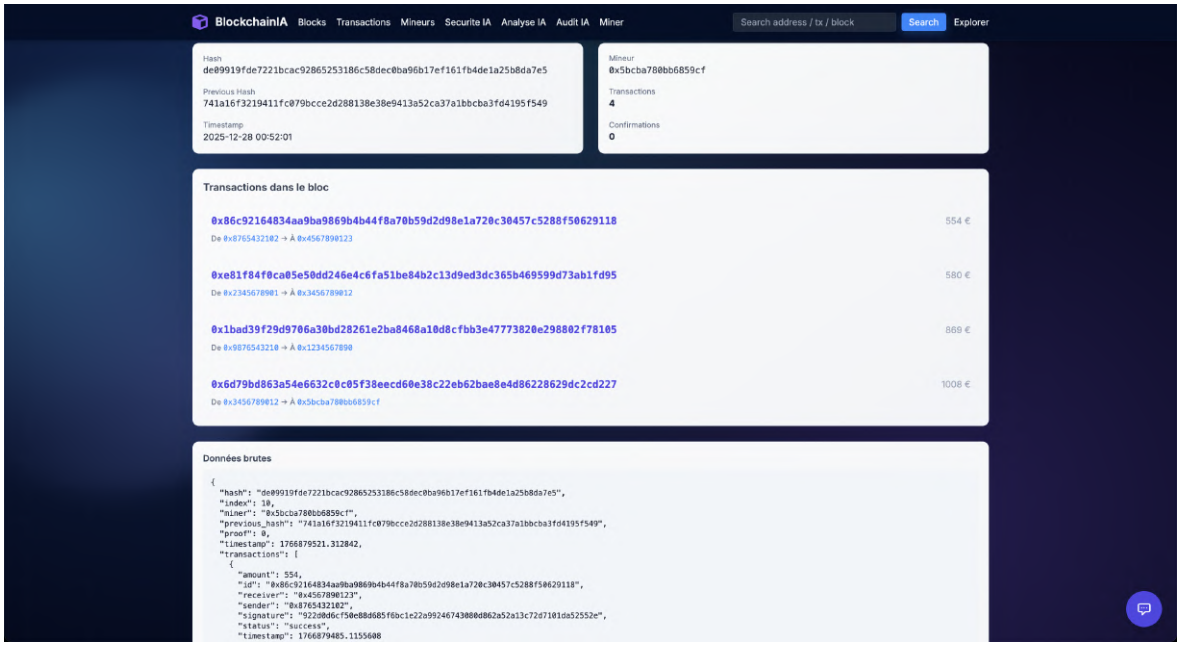

FIGURE 15 – Page de détails d'un bloc avec hash, nonce et liste des transactions

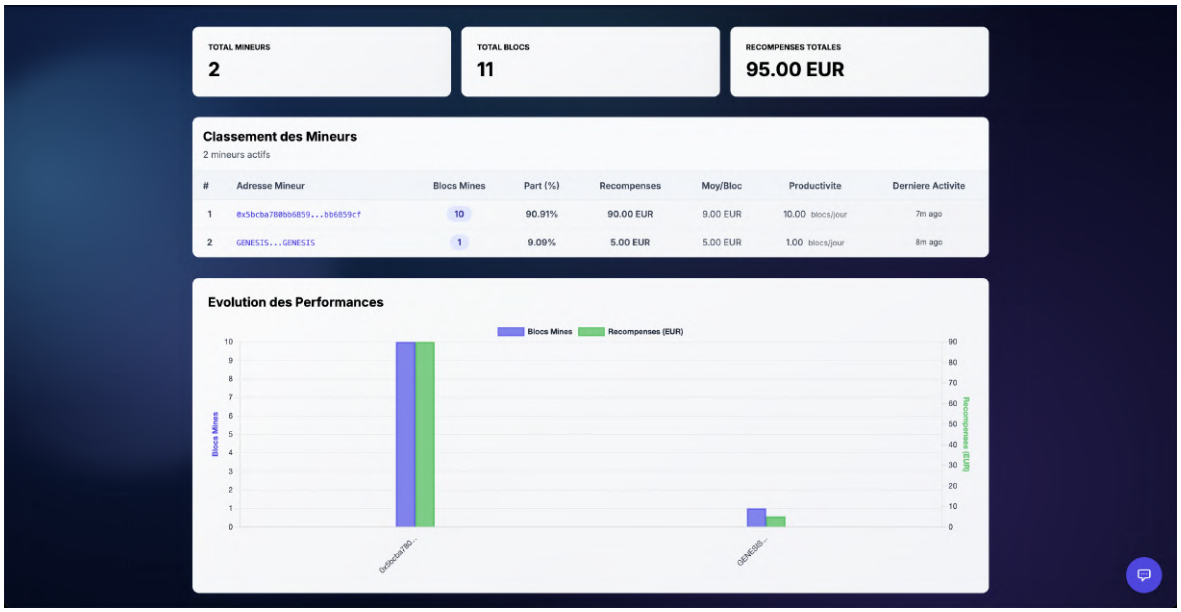

FIGURE 16 – Classement des mineurs avec blocs minés et récompenses gagnées

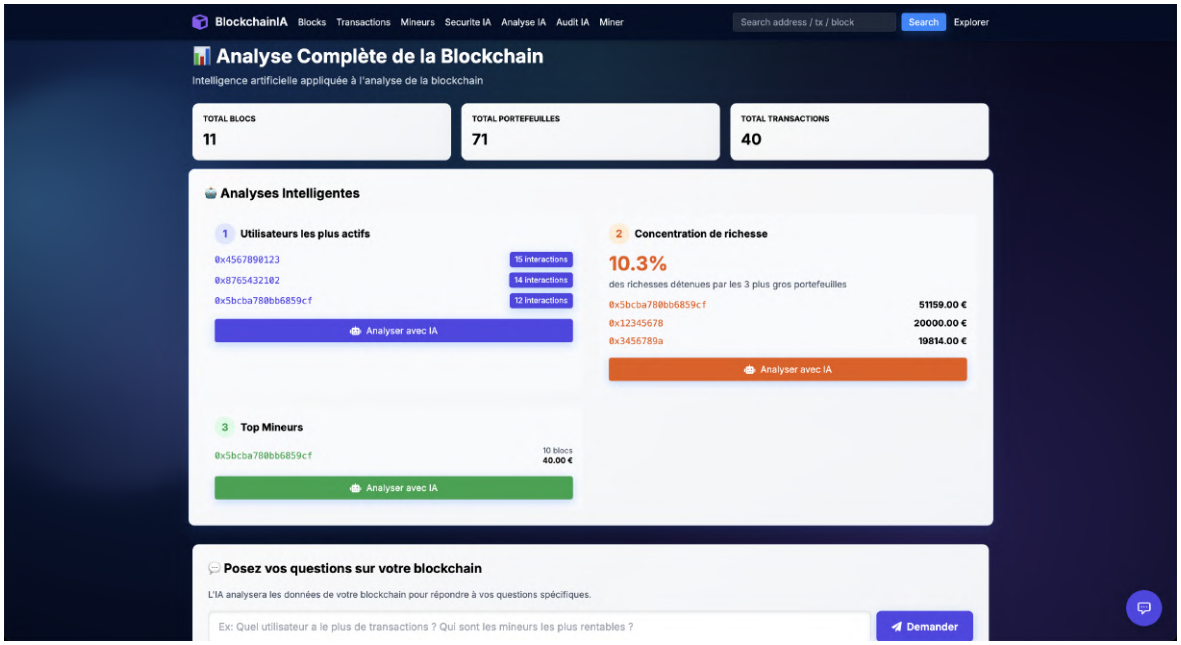

FIGURE 17 – Assistant IA Ollama analysant la blockchain et détectant des anomalies

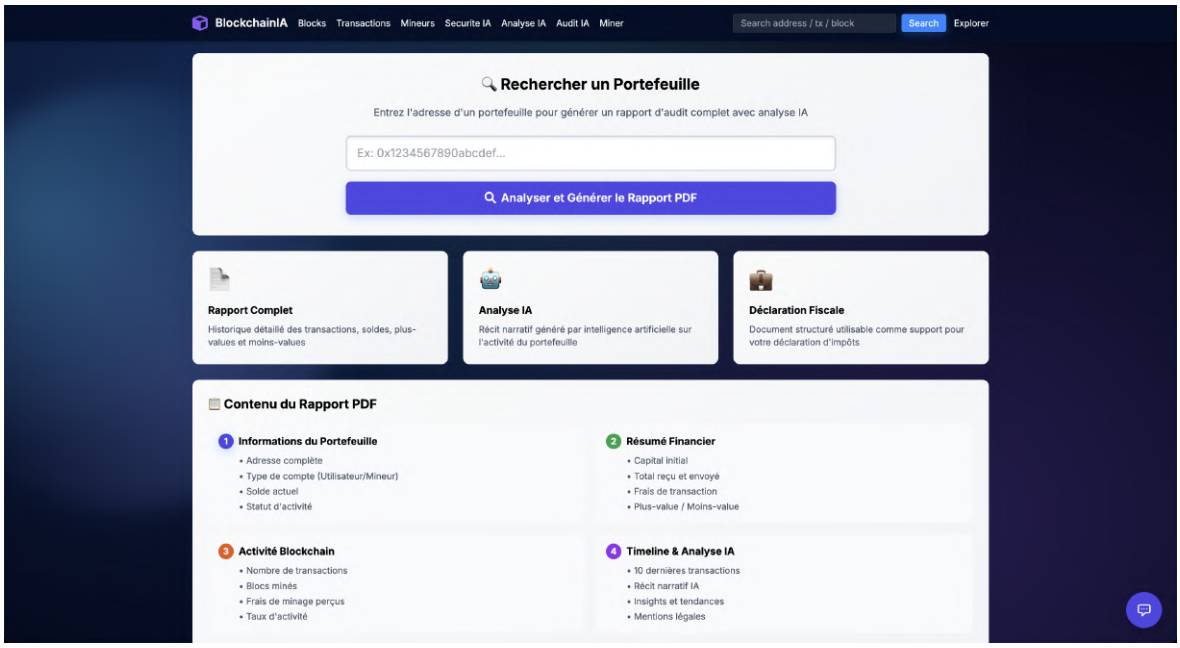

FIGURE 18 – Audit complet d'un portefeuille avec graphiques et statistiques détaillées

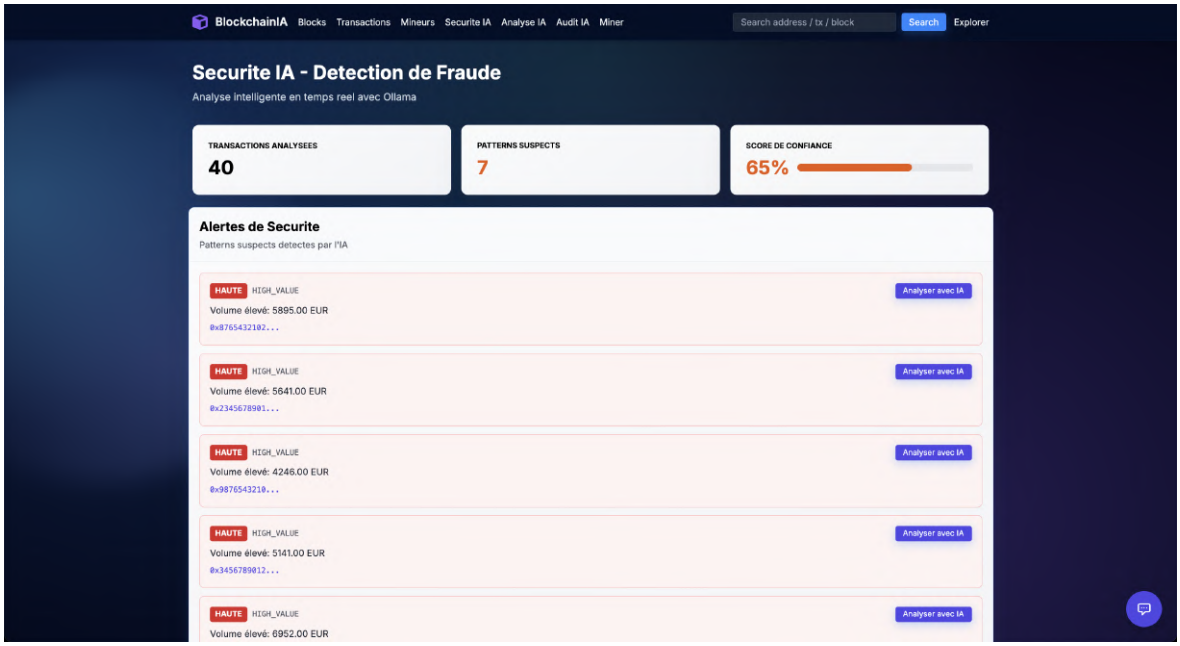

FIGURE 19 – Page de sécurité avec validation de chaîne et consensus entre nœuds

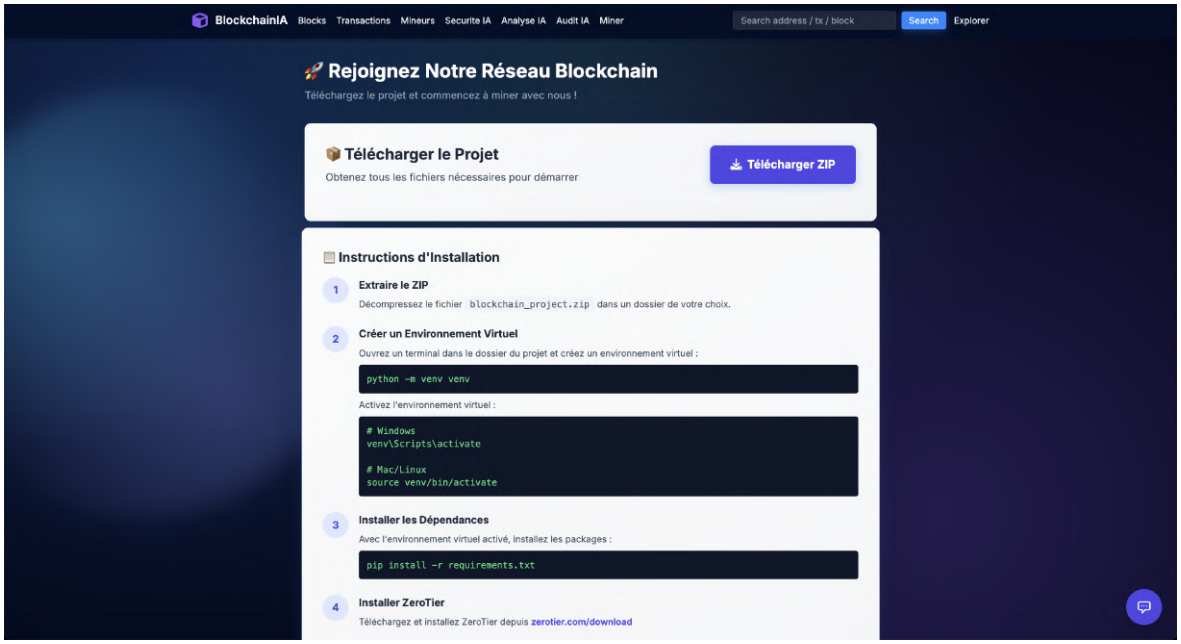

FIGURE 20 – Page de téléchargement du projet : instructions d'installation et réseau

blocks

index	hash	previous_hash	timestamp	miner	tx_count	nonce	reward
10	de09919fde7221bcac92865253186c58dec0ba96b17ef161fb4de1a25b8da7e5	741a16f3219411fc079bcce2d288138e38e9413a52ca37a1bbcba3fd4195f549	1766879521.312842	0x5bcba780bb6859cf	4	0	9.0
9	741a16f3219411fc079bcce2d288138e38e9413a52ca37a1bbcba3fd4195f549	636713633d07c25f2d1664793ec009723fe8b18193777830d641fab8e2de5e3f	1766879520.285291	0x5bcba780bb6859cf	4	0	9.0
8	636713633d07c25f2d1664793ec009723fe8b18193777830d641fab8e2de5e3f	3455cd9913b170685df7d0b3a9855fa53d4dd9f34a98d32eacf0dcfa909916ca	1766879519.2503428	0x5bcba780bb6859cf	4	0	9.0
7	3455cd9913b170685df7d0b3a9855fa53d4dd9f34a98d32eacf0dcfa909916ca	78409bb0e373e90b965e3a8a6789f7e648af78c1b010b48d2d8cce6e3edd8e79	1766879518.2199512	0x5bcba780bb6859cf	4	0	9.0
6	78409bb0e373e90b965e3a8a6789f7e648af78c1b010b48d2d8cce6e3edd8e79	2d7e244a0502322251382f6a8c3f39283660344bf8e9178662bcae1b976fa2972	1766879517.189125	0x5bcba780bb6859cf	4	0	9.0
5	2d7e244a0502322251382f6a8c3f39283660344bf8e9178662bcae1b976fa2972	404853cd53bf7685e3192cb81473993e0cf13477dd19b751730b4e6e50c67838	1766879516.167615	0x5bcba780bb6859cf	4	0	9.0
4	404853cd53bf7685e3192cb81473993e0cf13477dd19b751730b4e6e50c67838	05f81f163a9eb5f116cbee6a922d2410c9a956b70efc26ca854dc45c2dd4aeda	1766879515.1349518	0x5bcba780bb6859cf	4	0	9.0
3	05f81f163a9eb5f116cbee6a922d2410c9a956b70efc26ca854dc45c2dd4aeda	f4f629ae285d3829e55c70bf65b9fa2cf5aa5c975e8654a12ce1091ebb618fb3	1766879514.107111	0x5bcba780bb6859cf	4	0	9.0
2	f4f629ae285d3829e55c70bf65b9fa2cf5aa5c975e8654a12ce1091ebb618fb3	ab91c6b92d04d736951aad1f2a4ebe25800116d27a5a82aad0322bcb432ff687	1766879513.088172	0x5bcba780bb6859cf	4	0	9.0
1	ab91c6b92d04d736951aad1f2a4ebe25800116d27a5a82aad0322bcb432ff687	8c6d1d49e1f5a5e3c8801d9f0f9b1e941a5f4d1a04d3e8e19b486241b2e535a0	1766879512.0442681	0x5bcba780bb6859cf	4	0	9.0
0	8c6d1d49e1f5a5e3c8801d9f0f9b1e941a5f4d1a04d3e8e19b486241b2e535a0	0	1766879465.664952	GENESIS	0	0	5.0

transactions

id	sender	receiver	amount	status	timestamp	block_index
0x6d79bd863a54e6632c0c05f38eecd60e38c22eb62bae8e4d86228629dc2cd227	3456789012	0x5bcba780bb6859cf	1008.0	success	1766879521.1096857	10
0x60c8fc903ec8262bb351da9e013caf5625af461df92d3c49d0e64136602e7312	8765432102	2345678901	1494.0	success	1766879513.6769872	9
0x8ce863503b4279f9d5aa38194a51cf8c3ad637ec2e89dac0d0a293e16f544f14	4567890123	8765432102	887.0	success	1766879512.5820913	9
0x6b852aa3cdeec2ecba57c349eca09087ec0405aa85879caa7fbda13ae07892fa	4567890123	3456789012	1195.0	success	1766879510.8790717	8
0xc14d63e8e7721a172175398d4877eba2b2f26ecebb59397ab47f3ddbba4b97c8	8765432102	4567890123	300.0	success	1766879507.1404817	8
0x8b2349c97bd31291fc608d191632f949a522e5af89c817e45d34026ab2b1f65d	9876543210	3456789012	1149.0	success	1766879505.4007673	5
0x922fedade6336a8217f40e3b07b60fe4ef4ffbd0ca6f77119146a77a81f74fae	4567890123	0x5bcba780bb6859cf	851.0	success	1766879504.5744598	8
0xa4f8163c83ce62a12981b99108516b98c3fcb97f36d2f2beb599f2d63f7b1305	9876543210	1234567890	614.0	success	1766879504.2019708	5
0xe81f84f0ca05e50dd246e4c6fa51be84b2c13d9ed3dc365b469599d73ab1fd95	2345678901	3456789012	580.0	success	1766879502.494917	10
0x1bad39f29d9706a30bd28261e2ba8468a10d8cfbb3e47773820e298802f78105	9876543210	1234567890	869.0	success	1766879501.3156416	10
0x46ebb9c7f0001b64a92bd608c846873b358ae41581d14060afc322db80cd1b06	0x5bcba780bb6859cf	4567890123	844.0	success	1766879498.4915433	7
0x2ad3e52c13aca6006c9d1eb528aebf2d47091f3940213b9ad49d788943ad9bcd	0x5bcba780bb6859cf	9876543210	299.0	success	1766879498.0785272	6
0x00042a7f7ca71be1d18188e65f0e60cb923acf385c087de6f1e3b5fda9022d5b	0x5bcba780bb6859cf	2345678901	239.0	success	1766879496.2026262	4
0xe677eeb5fcaefd99bf378b719c7da87b1ba2da86323e98564d93d374295be6a0	4567890123	9876543210	204.0	success	1766879493.6987324	9
0xfb8e59ded314ad9550350db689840d32ac689a95ff1d2fe5ad753c97bbacb902	3456789012	2345678901	1099.0	success	1766879493.2143066	7
0x6c1e999fc9993a88ade4520b56840500f6db790fa283319eaebccad9ac15d64d	1234567890	4567890123	210.0	success	1766879491.4028096	6
0x599934204135880513 6e79e0d010816e95d276bc8a6fa2bbea828625dda6dc9e	2345678901	0x5bcba780bb6859cf	1148.0	success	1766879489.9987135	3
0x07fd9b97a7f7d65d69fd6dc4031f1646f77125bfee11c8a47c13cd82d6464220	0x5bcba780bb6859cf	8765432102	1143.0	success	1766879488.3334382	4
0x86c92164834aa9ba9869b4b44f8a70b59d2d98e1a720c30457c5288f50629118	8765432102	4567890123	554.0	success	1766879485.1155608	10
0x4f4d60707319d1a2f983a468158aa5f6098d03ce6f549a27776cd6660c66cb33	2345678901	4567890123	198.0	success	1766879484.8838763	6
0x9f665cebf0e5319d906412aaa035616c577307f0d8546551e6a1005cd64e875d	8765432102	0x5bcba780bb6859cf	437.0	success	1766879484.079694	2
0xe5011fa374a3a226657db42dfe37247448affda9ee8d0264ef94ed10fcc96be6	1234567890	8765432102	573.0	success	1766879483.7719731	7
0x4fe788e50c67a47caa4ec4954983b7822b090472e65c0713c20ffc33c671aca8	1234567890	4567890123	1348.0	success	1766879480.7947938	3
0x6177c1ffd4df240726c878426d24febc14c71d0d9ae774dd31af0d42ab6e71cd	4567890123	2345678901	809.0	success	1766879475.7828343	9
0x51db73bd2824cd7cfc65a559781af64089b2dd1054bc28fbef9ae5bed3d3bbd9	9876543210	0x5bcba780bb6859cf	177.0	success	1766879475.698466	3
0x4b3be9a07f16e4bad10a084d5d9331ea4316520887fb53ce3c4bc928b02c4288	3456789012	2345678901	234.0	success	1766879475.4760349	4
0x86b650e60241b90f6d0983878e2b3771271cae0aa57e5838cb1381b27f365ae2	3456789012	4567890123	2800.0	success	1766879474.7063804	1
0xbef039cee4b955e7612c843b41504f042d23af92b8ef1676e9a86b6a984e64e6	9876543210	8765432102	729.0	success	1766879472.6562028	2
0xd5c3b5c48fd8e3fad42af257f731b4cf938a46fec061821900cbbf952ac40ab2	2345678901	8765432102	2500.0	success	1766879470.8199177	1
0x8b708a46e5610d14e1317f199d7d5df0e2a5223442900c60a9525fb3a82c496e	8765432102	1234567890	1440.0	success	1766879470.655287	5
0x238b4b002b44d15c2b0b1fa6c4b83db477726ff22b280d878c488ea6c428796f	0x5bcba780bb6859cf	8765432102	305.0	success	1766879470.39332	6
0x92427bf33455eb3bcc0641535d91c46aa7486eb6b628a779fc854e1c05019393	0x5bcba780bb6859cf	8765432102	374.0	success	1766879468.2971036	2
0x7241ec2199781da801d5e918e88f27ca1aa34aae7ea2902fdec4cafdf6267e1c	2345678901	3456789012	568.0	success	1766879465.3128242	8

FIGURE 21 – Export CSV de blocs et transactions