Rapport d'Épreuve E5 : Surveillance et Résolution d'Incidents (C20-C21)

Projet : Bitcoin Analyzer

Candidat : Rida Boualam

Date : Juillet 2025

Certification Visée : RNCP37827 - Développeur en Intelligence Artificielle

Table des Matières

---

1. Introduction : De la Fonctionnalité à la Fiabilité

1.1. Contexte de l'Épreuve E5

Après avoir développé une application fonctionnelle intégrant un service d'IA, le dernier défi consiste à garantir sa fiabilité et sa maintenabilité en conditions opérationnelles. Une application qui fonctionne sur le poste d'un développeur n'est pas une application de production. Il est impératif de pouvoir surveiller son comportement, diagnostiquer les problèmes rapidement et résoudre les incidents de manière structurée.

Cette dernière épreuve se concentre sur deux compétences fondamentales du cycle de vie d'un logiciel :

- **C20 :** Surveiller une application d'intelligence artificielle.
- **C21 :** Résoudre les incidents techniques.

1.2. L'Approche MLOps : Vers une Application Opérationnelle

Cette épreuve incarne la boucle de rétroaction ("feedback loop") de la philosophie MLOps. La journalisation (logging) fournit les données nécessaires pour comprendre ce qui se passe en temps réel, et la résolution d'incidents est le processus qui utilise ces données pour améliorer la robustesse du système. Ce rapport documente la mise en place de ces deux pratiques essentielles pour le projet "Bitcoin Analyzer".

2. C20 : Surveillance de l'Application et Journalisation (Logging)

2.1. Stratégie de Journalisation Multi-Couches

Pour obtenir une visibilité complète sur le système, une stratégie de journalisation a été implémentée à chaque niveau de l'architecture découplée.

- **Frontend (Django) :** Le fichier `viewer/views.py` utilise le module `logging` de Python pour tracer les événements liés à l'utilisateur :
  - Réception d'une requête HTTP (avec l'IP du client).
  - Début de chaque appel HTTP sortant vers l'API FastAPI.
  - Succès ou échec de ces appels.
  - Toute erreur de communication, avec sa trace complète (`exc_info=True`).
- **Backend (FastAPI) :** De même, le fichier `api/app.py` logue toutes les actions internes du service :
  - Réception d'une requête de l'application Django.
  - Interactions avec la base de données (lecture des prix, des actualités).
  - Début et fin de l'appel critique vers le service externe d'IA (Google Gemini).
  - Toute erreur interne au service.

Cette approche permet une traçabilité de bout en bout : il est possible de suivre le parcours complet d'une requête, depuis le clic de l'utilisateur jusqu'à la réponse de l'IA, en corrélant les logs des deux services.

2.2. Preuve de Monitoring : Analyse de Logs pour la Traçabilité de Bout en Bout

L'extrait de log ci-dessous, tiré du fichier `docs/exemples_de_logs.txt`, illustre parfaitement cette traçabilité.

Logs de la console Django (Frontend) :

```
Requête reçue pour le tableau de bord depuis l'IP : 127.0.0.1
Début de l'appel API vers : http://127.0.0.1:8001/latest-news?limit=5
Succès : 3 actualités récupérées.
Début de l'appel API vers : http://127.0.0.1:8001/price-history?limit=24
Succès : 24 points d'historique récupérés.
Début de l'appel API vers : http://127.0.0.1:8001/price-analysis
Succès : Analyse de l'IA récupérée.
[09/Jul/2025 10:23:46] "GET / HTTP/1.1" 200 4697
```

Logs de la console FastAPI (Backend) corrélés dans le temps :

```
2025-07-09 10:23:39,049 - INFO - API - Requête reçue pour les dernières nouvelles
(limite=5).
INFO: 127.0.0.1:61444 - "GET /latest-news?limit=5 HTTP/1.1" 200 OK
2025-07-09 10:23:39,071 - INFO - API - Requête reçue pour l'historique des prix
(limite=24).
INFO: 127.0.0.1:61445 - "GET /price-history?limit=24 HTTP/1.1" 200 OK
2025-07-09 10:23:39,098 - INFO - API - Requête reçue pour l'analyse de prix
(limite=24).
2025-07-09 10:23:39,103 - INFO - API - Appel au service d'analyse IA (Gemini)...
```

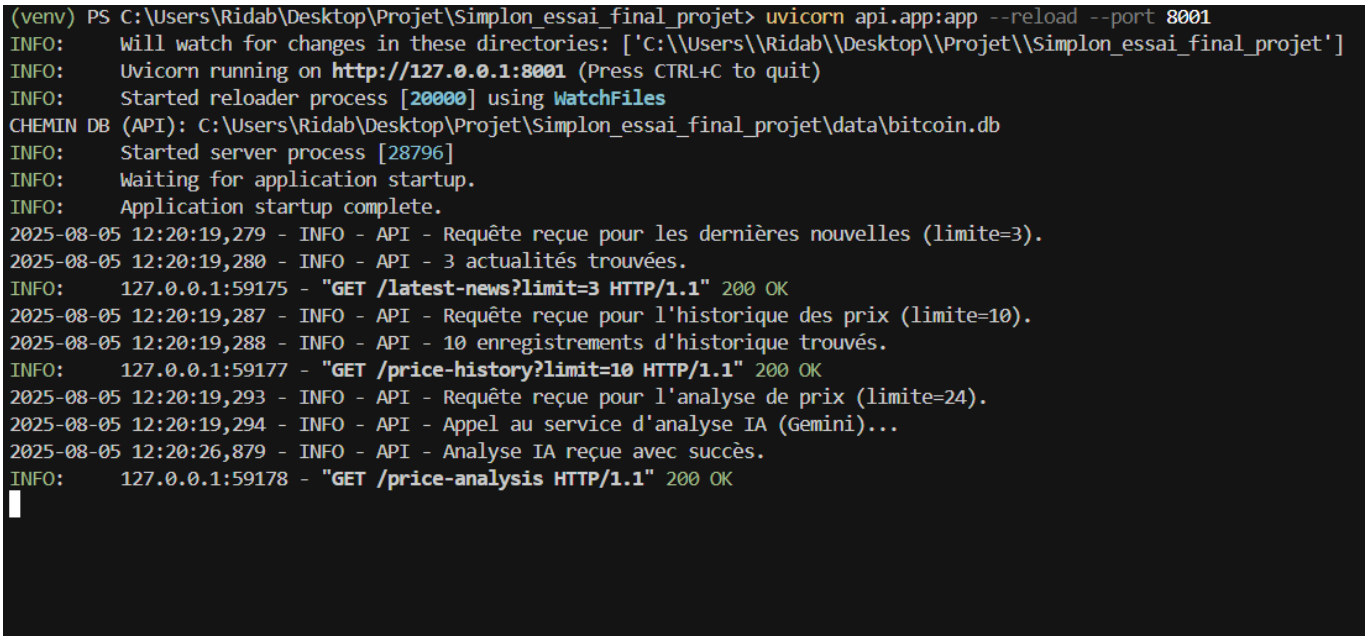
```
2025-07-09 10:23:46,068 - INFO - API - Analyse IA reçue avec succès.
INFO: 127.0.0.1:61446 - "GET /price-analysis HTTP/1.1" 200 OK
```

Analyse de ces logs :

- On peut clairement corréler la requête de l'utilisateur (log Django) aux trois appels reçus par l'API (logs FastAPI).
- On peut mesurer des métriques de performance critiques. Par exemple, l'appel à l'API Gemini a commencé à 10:23:39 et s'est terminé à 10:23:46, indiquant une latence d'environ 7 secondes. Cette information est essentielle pour le monitoring de la performance.

2.3. Visualisation des Logs en Temps Réel

En complément de l'analyse des fichiers, la surveillance en temps réel via la sortie de la console est un outil de débogage indispensable au quotidien.



3. C21 : Résolution d'Incidents Techniques

Un développeur compétent n'est pas celui qui n'a jamais de bugs, mais celui qui sait les résoudre de manière méthodique.

3.1. Méthodologie de Résolution d'Incidents

Pour chaque incident rencontré, j'ai appliqué un processus structuré en 5 étapes, documenté dans `suivi_projet.md` :

1. **Contexte :** Description de ce que j'essayais de faire.
2. **Symptôme :** Description du problème observé (message d'erreur, comportement inattendu).
3. **Diagnostic :** Analyse des logs et du code pour identifier la cause racine.
4. **Résolution :** Description des actions correctives apportées au code.
5. **Leçon Apprise :** Ce que l'incident m'a appris et comment éviter qu'il se reproduise.

3.2. Incident 1 : Conflit de Ports en Architecture Microservices

- **Contexte :** Premier lancement simultané de l'application Django (frontend) et de l'API FastAPI (backend).
- **Symptôme :** L'application Django affichait une erreur `requests.exceptions.ConnectionError` et ne parvenait pas à contacter le backend.
- **Diagnostic :** Les deux serveurs tentaient d'utiliser le port 8000 par défaut, créant une collision. Seul le premier service lancé pouvait s'attribuer le port, l'autre échouait à démarrer.
- **Résolution :** Lancement explicite du serveur FastAPI sur un port différent (`uvicorn api.app:app --port 8001`) et mise à jour de la constante `API_BASE_URL` dans le code Django.
- **Leçon Apprise :** La nécessité de définir et de documenter explicitement la topologie réseau (ports, adresses) dans une architecture multi-services pour éviter les conflits.

3.3. Incident 2 : Erreur d'Environnement dans la CI/CD

- **Contexte :** Premier déploiement du workflow d'intégration continue sur GitHub Actions.
- **Symptôme :** Le workflow échouait systématiquement à l'étape des tests, alors que tous les tests passaient en local. Le log d'erreur indiquait `ModuleNotFoundError: No module named 'httpx'`.
- **Diagnostic :** L'erreur prouvait que l'environnement de la CI était différent de mon environnement local. Le paquet `httpx` (une dépendance du `TestClient` de FastAPI) avait été installé manuellement sur ma machine mais n'avait pas été ajouté au fichier `requirements.txt`.
- **Résolution :** Ajout de la dépendance manquante (`httpx`) au fichier `requirements.txt` et commit de la modification. Le workflow s'est ensuite exécuté avec succès.
- **Leçon Apprise :** Cet incident démontre la valeur fondamentale de l'intégration continue. Elle agit comme un garde-fou qui garantit la reproductibilité des environnements et force une gestion rigoureuse des dépendances, évitant ainsi le classique "ça marche sur ma machine".

3.4. Incident 3 : Pivot Stratégique face à un Échec de Scraping

- **Contexte :** Le script de scraping des actualités (`extraction_news.py`) échouait systématiquement à collecter des données de sa cible initiale.
- **Symptôme :** Les logs montraient une erreur HTTP 403 Forbidden, indiquant un refus d'accès.
- **Diagnostic :** L'analyse du site cible a révélé la mise en place de protections anti-bot avancées (type Cloudflare) nécessitant l'exécution de JavaScript, ce qu'une simple requête `requests` ne peut pas faire. S'obstiner sur cette cible aurait nécessité des techniques de contournement complexes et peu fiables.
- **Résolution :** Au lieu d'une correction technique mineure, une décision stratégique a été prise : changer de source de données pour une cible plus accessible, `news.bitcoin.com`. Le script a été entièrement réécrit pour utiliser `undetected-chromedriver`, simuler un vrai navigateur, attendre le chargement du contenu, et utiliser de nouveaux sélecteurs CSS pour extraire les informations.
- **Leçon Apprise :** La flexibilité est une compétence clé. Parfois, la meilleure solution n'est pas de s'acharner sur un problème technique, mais de pivoter vers une approche plus fiable pour garantir la continuité et la robustesse du service.

4. Conclusion de l'Épreuve E5

Cette épreuve finalise le cycle de vie du projet en démontrant la capacité à transformer une application fonctionnelle en un service fiable et maintenable. La mise en place d'une journalisation complète (C20) a fourni les outils nécessaires pour observer et diagnostiquer le système. L'analyse et la résolution documentée de plusieurs incidents techniques concrets (C21) ont prouvé une approche méthodique face aux défis inévitables du développement logiciel.

Le projet "Bitcoin Analyzer" est désormais doté des mécanismes de surveillance et de la robustesse acquise par la résolution d'incidents, le préparant ainsi à des conditions d'exploitation réelles.