MBA 2 - Python - Projet : Exposition de données sous la forme d'une API

Rakib SHEIKH
Document v1 (*Le sujet est amené à évoluer*)

---

https://www.kaggle.com/datasets/computingvictor/transactions-fraud-datasets/data?select=transactions_data.csv

Énoncé générale :

Votre manager directe vous a envoyé un mail urgent de la part de la direction exécutif, au moment de son départ en vacances. Elle vous confie donc la tâche de déployer une application complète, permettant de d'exposer les données de transactions bancaires à travers une API, à destination de l'équipe en charge de l'application support métier pour les commerciales bancaires.

Modalités des groupes :

- Groupe jusqu'à 4 personnes maximum, les demandes de groupes de 5 seront ignorés d'avances.

Date limite : 28 décembre 2025 à 6h00 du matin

Règles pénalités de retard :

- Chaque heure de retard entamé est due en son intégralité : Le groupe en retard sera facturé 5 points de sa note finale à chaque heure de retard entamé.

✉ | Mail de votre manager directe

Bonjour,

Je sais que je te préviens très tardivement par rapport à mon départ en congé longue durée mais l'information de mes supérieurs hierarchique vient de tomber.

Nous devons concevoir une application, fait maison permettant d'exposer les données de production auprès de l'équipe technique en charge du développement de la nouvelle application web de la gestion du portefeuille de chaque agence de la boite.

Comme tu as pu le comprendre, cette application utilisera les données de production, elle devra donc être conçus pour être tourné dans des environnements de production, donc utilisation obligatoire de github, avec toute la CI/CD qui suit.

Afin d'épauler l'équipe CI/CD et en connaissance de ton niveau actuel, je te demande juste que l'application puisse être sous la forme de paquet python, avec l'ensemble des tests unitaire rédigés. L'équipe de la CI/CD vont s'occuper du reste du travail.

Bien évidement, l'ensemble des routes API reste à ta charge du développement, que tu trouvera les specs en pièce jointe.

Le conducteur du train (je te rappel qu'on est sous framework SAFe v6, section Team and Technical Agility Discipline si t'avais oublié) nous à demandé à ce que le projet soit finalisé avant le sprint 1 générale de la boite, c'est à dire **avant le 28 décembre 2025** et qu'elle doit être rendu sous la forme d'un Pull-Request.

On a de la chance car cette fois-ci, l'équipe de production nous a mis à disposition des données fictives permettant de reproduire l'environnement de production que nous n'avons pas accès.

Encore merci par avance de ton implication dans le projet, je ferais en sorte qu'elle soit valorisé lors de notre prochain entretien annuel !

Bien à toi, Le manager direct.

Technical Specifications

For portfolio asset retrieval management.

Please note this document is highly confidential and should not be shared outside the technical team, and to any person not related to the project !

Spécifications fonctionnelles et techniques

- Projet : Banking Transactions API
- Version : 1.0
- Date limite : 28 décembre 2025
- Framework : FastAPI
- Langage : Python 3.12+
- Livrable : package Python + tests unitaires + PR GitHub

---

1. Objectif général

Développer une API REST complète pour exposer et manipuler les données de transactions bancaires issue des données fournie par l'équipe de production

L'API doit :

- Permettre la consultation, recherche et filtrage des transactions
- Fournir des statistiques agrégées et analytiques
- Intégrer des endpoints liés à la fraude et à la surveillance
- Être documentée (numpydoc + swagger en option) et testé (unit et features)
- Être packagée sous forme de module Python installable

---

2. Organisation des endpoints (20 routes totales)

Catégorie	Description	Routes
Transactions	Consultation, filtrage, recherche, suppression	1–8
Statistiques	Agrégations globales et par critères	9–12
Fraude	Analyse et détection	13–15
Clients	Exploration des portefeuilles clients	16–18
Administration	Métadonnées & supervision du service	19–20

---

3. Détail des routes

1. GET /api/transactions

Description : Liste paginée des transactions Paramètres : `page`, `limit`, `type`, `isFraud`, `min_amount`, `max_amount` Réponse :

```json
{ "page": 1, "transactions": [ { "id": "tx_0001", "amount": 500.0, "type": "CASH_OUT" } ] }
```

---

2. GET /api/transactions/{id}

Description : Détails d'une transaction par son identifiant

---

3. POST /api/transactions/search

Description : Recherche multicritère (POST avec corps JSON) Exemple de requête :

```json
{ "type": "TRANSFER", "isFraud": 1, "amount_range": [1000, 5000] }
```

---

4. GET /api/transactions/types
Description : Liste des types de transactions disponibles (valeurs uniques de `type`)

---

5. GET /api/transactions/recent
Description : Renvoie les N dernières transactions du dataset (paramètre `n`, défaut=10)

---

6. DELETE /api/transactions/{id}
Description : Supprime une transaction fictive (utilisée uniquement en mode test)

---

7. GET /api/transactions/by-customer/{customer_id}
Description : Listes des transactions associées à un client (origine)

---

8. GET /api/transactions/to-customer/{customer_id}
Description : Liste des transactions reçues par un client (destination)

---

9. GET /api/stats/overview
Description : Statistiques globales du dataset Réponse :

```json
{
  "total_transactions": 6362620,
  "fraud_rate": 0.00129,
  "avg_amount": 178642.15,
  "most_common_type": "CASH_OUT"
}
```

---

10. GET /api/stats/amount-distribution
Description : Histogramme du montant des transactions (en classes de valeurs) Réponse :

```json
{
  "bins": ["0-100", "100-500", "500-1000", "1000-5000"],
  "counts": [10000, 53000, 42000, 21000]
}
```

---

11. GET /api/stats/by-type
Description : Montant total et nombre moyen de transactions par type Réponse :

```json
[
  {"type": "PAYMENT", "count": 1250000, "avg_amount": 350.21},
  {"type": "TRANSFER", "count": 400000, "avg_amount": 13000.58}
]
```

---

12. GET /api/stats/daily
Description : Moyenne et volume des transactions par jour (`step`)

---

13. GET /api/fraud/summary

Description : Vue d'ensemble de la fraude Réponse :

```json
{ "total_frauds": 8213, "flagged": 16, "precision": 0.95, "recall": 0.88 }
```

---

14. GET /api/fraud/by-type

Description : Répartition du taux de fraude par type de transaction

---

15. POST /api/fraud/predict

Description : Endpoint de scoring pour prédire si une transaction donnée est frauduleuse Corps JSON attendu :

```json
{ "type": "TRANSFER", "amount": 3500.0, "oldbalanceOrg": 15000.0, "newbalanceOrig": 11500.0 }
```

Réponse :

```json
{ "isFraud": true, "probability": 0.89 }
```

---

16. GET /api/customers

Description : Liste paginée des clients (extraits de `nameOrig`)

---

17. GET /api/customers/{customer_id}

Description : Profil client synthétique (nombre de transactions, solde moyen, fraude impliquée, etc.) Réponse :

```json
{
  "id": "C1231006815",
  "transactions_count": 58,
  "avg_amount": 205.33,
  "fraudulent": false
}
```

---

18. GET /api/customers/top

Description : Top clients classés par volume total de transactions Paramètre : `n` (défaut=10)

---

19. GET /api/system/health

Description : Vérifie l'état de santé de l'API (ping, latence, chargement du dataset) Réponse :

```json
{ "status": "ok", "uptime": "2h 15min", "dataset_loaded": true }
```

---

20. GET /api/system/metadata

Description : Informations sur la version du service et la date de dernière mise à jour Réponse :

```json
{ "version": "1.0.0", "last_update": "2025-12-20T22:00:00Z" }
```

---

4. Services internes prévus

Service	Rôle
`transactions_service.py`	Lecture, pagination, filtrage, recherche multi-critères
`stats_service.py`	Calcul des agrégations et distributions

Service	Rôle
`fraud_detection_service.py`	Calcul de taux de fraude, scoring simplifié
`customer_service.py`	Agrégation par client
`system_service.py`	Diagnostic du service et métadonnées

---

5. Tests unitaires attendus

Domaine	Tests recommandés
Routes	1 test par endpoint (≥20 tests)
Services	tests sur les fonctions de stats et de fraude
Validations	vérification du format des entrées JSON
Performance	latence max < 500ms pour 100 transactions filtrées
Couverture cible	≥ 85 %

---

6. Préparation CI/CD & packaging

1. Lint (`flake8`)
2. Typage (`mypy`)
3. Tests (`pytest --cov`)
4. Build du paquet (`poetry build` ou `python -m build`)

Barème de notation

Attention, donnée à titre indicatif, le barème final peut être ajusté en conséquence

Notation des routes implémentés sous FastAPI

- Notation sur 10 points.

Critère de d'évaluation :

- Route fonctionnel, sans erreurs avec gestion des erreurs courantes.
- Chaque route est évalué sur 1 point. La note totale est de 20pts, ramené à 10 sur le barème finale.

Respect de la qualitée du code (Norme PEP8)

- Notation sur 2 points.

Critère de d'évaluation :

- Aucun erreur généré par flake8. La note est soit 0, soit 2.

Respect de l'utilisation du Typing sur l'ensemble des variables

- Notation sur 2 points.

Critère d'évaluation :

- L'ensemble des variables sont typés, y compris les routes.
- La tolérance attribuant la moitiés des points est de 80% de variables typés.

Respect de la conformité de la mise sous paquet python

- Notation sur 2 points.

Critère d'évaluation:

- La phase de mise sous paquet ne génère aucune erreur.
- Le lancement de l'application ne génère aucune erreur, ni de warning en phase de production.
- L'ensemble des fichiers requises pour la mise sous paquet est présente.
- Documentation complète du code source, avec le style de documentation numpy.

Respect des tests unitaire et de features

Notation sur 4 points.

- 2 Points pour les tests unitaires via PyTest
- 2 Points pour les tests features via unittest

Critère de notation :

- Points maximales si la converture du teste est de 100%
- 3 Points si la converture des tests est au moins 95%
- 2 Points si la converture des tests est au moins 85%
- 1 Point si la converture de tests est au moins 80%
- 0 Point si la converture de de tests est inférieur à 80% ou absente.

Jusqu'à 4 Points bonus accordés si la note finale est supérieur ou égale à 14 pour la réalisation des actions suivantes :

- Application Web pour tester les routes via Swagger. (Inclus dans le même projet)
- Application Web métier pour tester les routes via Streamlit (Projet externe, doit être séparé du projet)
- Mise en conformité pipeline CI/CD fonctionnel (Github Actions fonctionnel)
- Mise à disposition de l'API sous la forme de container Docker (Doit être inclus dans le même)