

Projet fil rouge – API de gestion de personnages de manga

1. Récapitulatif du projet – Partie par partie

Partie	Ce que vous avez appris et construit
Partie 1 – Exposer ses données	<ul style="list-style-type: none">- Création d'une API REST en FastAPI- Endpoint GET /personnages renvoyant une liste JSON- Ajout de sécurité simple par token- Découverte des headers, CORS, documentation Swagger
Partie 2 – Requête des données	<ul style="list-style-type: none">- Requête de l'API en JavaScript (fetch) et Python (requests)- Récupération paginée de données externes- Transformation : nettoyage, enrichissement, filtrage- Sauvegarde dans un fichier .json local
Partie 3 – API Push	<ul style="list-style-type: none">- Création d'un webhook pour recevoir automatiquement un nouveau personnage- Simulation d'un appel webhook (Make, curl)- Traitement de l'événement et enregistrement dans un fichier- Début de réflexion sur l'architecture évènementielle (publish/subscribe)
Partie 4 – Publier des messages	<ul style="list-style-type: none">- Création d'un endpoint POST recevant des données à enrichir- Requête POST avec JSON depuis un script Python- Réutilisation des données extraites précédemment- Construction d'un flux automatisé extraction → transformation → publication

2. Ce que le projet permet actuellement

L'API et les scripts associés permettent de :

Côté API (FastAPI) :

- Fournir une liste de personnages (GET)
- Recevoir un nouveau personnage (POST /webhook)
- Enrichir un personnage à la volée (POST /traitement)
- Sauvegarder les données dans un fichier local (ou base ultérieurement)
- Gérer un **flux de données entrant et sortant**

Côté client (script Python/JS) :





- Requête des APIs distantes, avec pagination
- Nettoyer, filtrer, enrichir des données
- Enregistrer un jeu de données structuré
- Envoyer des objets un par un à une API pour traitement
- Gérer les erreurs, timeouts et retry patterns

3. Ce qu'il est possible de réaliser à ce stade

- Créer un **mini pipeline ETL complet** en local
- Comprendre **comment fonctionnent les webhooks et les flux push**
- Construire une **architecture modulaire**, avec des **endpoints spécialisés**
- Organiser un **flux automatisé d'entrée et de sortie de données**
- Simuler une intégration avec un outil externe comme **Make, Zapier**, ou une vraie API comme Stripe / GitHub

4. Bonus & prolongements possibles

Fonctionnels :

-  Ajout d'un **système d'identifiants uniques** pour les personnages
-  Intégration d'un système de **badges** ou de **niveaux**
-  Création d'un **dashboard simple** (ex : en HTML + fetch, ou Streamlit)
-  Ajout d'un système de **pagination ou tri côté API**

Sécurité :

- Passer d'un token statique à une **authentification par JWT**
- Implémenter une vérification de signature sur les webhooks reçus (type Stripe)

Stockage :

- Remplacer le fichier JSON par une **base SQLite** ou **MongoDB**
- Ajouter un endpoint GET /personnage/{id} pour retrouver un personnage unique

Tests :

- Intégrer des **tests automatisés** avec pytest
- Vérifier que le système d'enrichissement fonctionne pour toutes les entrées

Déploiement :

- Déployer l'API sur **Render**, **Railway** ou **Vercel (pour le front)**
- Ajouter une **documentation interactive** sur Swagger ou ReDoc

Ce que tu dois me rendre

L'ensemble des exercices fait durant le cours et si tu veux les bonus ici qui peuvent t'ajouter 5 points.

Dépose-moi le lien git sur ce formulaire : <https://forms.gle/Bk8o5zLi8A4B5J5s6>