Développement d’une API version PRO

Sommaire

Introduction

Bienvenue dans ce projet de groupe qui va vous plonger dans un vrai environnement de développement backend/API moderne. Vous allez créer une application complète avec FastAPI (backend) et Streamlit (frontend), tout en travaillant avec des outils et des pratiques professionnelles : Docker, OAuth2, JWT, Prometheus, Grafana, gestion des logs et des tests automatisés.

Tout le projet se fera en mode agilité : chaque groupe aura un équipier élu Scrum Master (SM) et un Product Owner (PO). Vous devrez avancer par itérations, pousser votre code sur GitHub régulièrement, maintenir un README à jour, et réaliser des "sprints" de 1/2 à 1 journée.

Objectifs pédagogiques

A l'issue de ce projet, vous serez capables de :

- Concevoir une API REST modulaire, rapide et sécurisée
- Créer un frontend d'interaction simple avec Streamlit
- Mettre en place une authentification OAuth2 avec gestion de tokens
- Protéger une application via Docker/Docker-compose
- Monitorer une API avec Prometheus et Grafana
- Travailler en équipe avec une vraie logique agile
- Déployer un projet structurant pour votre portfolio

L’idée est que chaque groupe développe un template qu’il pourra réutiliser plus tard (comme pour la certification)

Agilité

Vous devrez travailler en agilité et faire des daily etc... Utilisez des outils d’avancement de votre travail (trello, jira...)

Vision d'ensemble de la progression

Étape	Objectif pédagogique
0 → 4	Environnement / Structuration
5 → 7	Base de données / Organisation
8 → 11	API simple + Frontend Streamlit
12 → 15	Authentification complète (OAuth2 + tokens)
16	Rotation de tokens sécurisée
17 → 18	Logs d'activité + Monitoring
19	Version finale du projet (Release v0.2)

Ceux qui vont au bout seront prêts pour des missions backend/API modernes. Vous aurez vécu un vrai projet d'entreprise en conditions semi-réelles. Vous aurez manipulé de vraies problématiques de production.

SAS IA - FastAPI

Environnement et structuration

Mise en place des groupes

Constitution des équipes de travail. Chaque équipe désigne un Scrum Master et un Product Owner. L'organisation sera faite en agilité avec des rôles clairs et des mini-sprints.

Résumé :

- Créer les groupes
- Définir les rôles (SM / PO)

Installation de l'environnement

Chaque groupe crée un environnement Python isolé, un fichier requirements.txt, un README.md initial et un fichier .env pour stocker les variables sensibles.

Résumé :

- Venv activé
- Fichiers de base créés : requirements.txt, .env, README.md

Création d'une API "Hello World"

Démarrer avec un premier serveur FastAPI fonctionnel affichant simplement un message "Hello World".

Résumé :

- API FastAPI démarrée avec une route de base

Architecture modulaire

Structuration du projet avec un main.py et création des premiers dossiers users/ et db/. Mise en place des premiers fichiers routes.py pour chaque module.

Résumé :

- Dossier users/ avec routes.py
- Dossier db/ avec routes.py
- Import propre dans main.py

Version 0 : Initialisation sur GitHub

Première mise à jour du README.md avec explication de l'installation et capture du /docs. Premier push GitHub avec tag version v0.

Résumé :

- Documentation initiale prête
- Premier commit/push avec tag v0

Base de données et organisation

Conception des modèles

Création du Modèle Conceptuel de Données (MCD), du Modèle Logique de Données (MLD), et du Modèle Physique de Données (MPD). Création des modèles Pydantic (FastAPI) et SQLAlchemy (ORM) pour gérer les utilisateurs, en anticipant les champs pour les rôles et les refresh tokens.

Résumé :

- Schémas MCD/MLD/MPD créés
- Modèles Pydantic et SQLAlchemy préparés

Gestion des connexions à la base de données

Utilisation des add_event_handler de FastAPI pour gérer les connexions et déconnexions à la base de données avec des logs informatifs via Loguru.

Résumé :

- Fonctions connect/disconnect présentes
- Log des événements via Loguru

Premiers tests sur SQLite

Connexion réelle à une base de données SQLite. Implémentation des premières routes login et logout dans users/routes.py avec retour de codes HTTP clairs (201, 400, 401, 403, 404).

Résumé :

- Connexion à SQLite fonctionnelle
- Routes de login/logout créées
- Codes HTTP utilisés correctement

API simple + Frontend Streamlit

Séparation Frontend (Streamlit) / Backend (API)

À partir de cette étape, le projet sera scindé en deux parties :

- api/ pour toute la partie FastAPI
- frontend/ pour l'application Streamlit

Mise en place d'une interface de connexion simple dans Streamlit, dans un dossier pages/0_login.py

Les utilisateurs pourront créer un compte (s'ils n'en ont pas) ou se connecter via un formulaire validant :

- que l'email est valide,
- que les deux mots de passe correspondent,
- que le mot de passe est suffisamment sécurisé.

Côté API FastAPI, vous devez mettre en place :

- Route de création d'utilisateur,
- Route de login,
- Route de déconnexion (sans authentification encore à ce stade), avec stockage hashé des mots de passe dans la base SQLite.

Résumé :

- Création des dossiers api/ et frontend/
- Formulaire Streamlit simple de création/connexion utilisateur
- Stockage hashé du mot de passe dans la base

Mise en place des tests avec Pytest

Début de la couverture de tests avec pytest :

- Tests unitaires pour la connexion et déconnexion à la base de données,
- Tests de création, connexion et suppression d'un utilisateur.

Les données sensibles doivent être cryptées dans une seconde base dédiée, avec une clé différente par utilisateur.

Résumé :

- Première batterie de tests fonctionnels
- Mise en place du cryptage des données sensibles utilisateurs

10. Mise à jour GitHub : Release v0.1

Mise à jour du README.md avec la documentation d'installation, des captures d'écran de /docs, et push GitHub en taggant la nouvelle version : v0.1.

Résumé :

- Documentation actualisée
- Version v0.1 poussée sur GitHub

11. Enrichissement du profil utilisateur

Ajout de champs supplémentaires pour les utilisateurs :

- Pseudo,
- Email,
- Mot de passe (possibilité de modification),
- Petite biographie,
- Rôle (exemple : membre ou admin).

Mise à jour de l’interface Streamlit pour permettre :

- Modifier son profil,
- Supprimer son compte,
- Pour un admin : voir la liste de tous les pseudos via une nouvelle page pages/administration.

Résumé :

- Formulaire utilisateur enrichi dans Streamlit
- Nouvelles routes API pour patch et delete utilisateur
- Page d’administration de liste des utilisateurs

Authentification OAuth2 + Tokens

Refactorisation avant sécurité

Un nettoyage complet du projet est effectué : relecture du code, suppression des éléments inutiles, amélioration de la structure. Cela garantit une base saine avant d'intégrer des mécaniques de sécurité complexes.

Résumé :

- Refacto général du projet
- Validation de la base technique

Mise en place de l'authentification simple

Début de l'intégration d'OAuth2 avec FastAPI : création des tokens JWT simples au moment de la connexion. Les rôles (user ou admin) sont encore gérés manuellement pour simplifier l'étape.

Résumé :

- Implémentation d'OAuth2PasswordBearer
- Génération d'Access Token simple
- Authentification basique opérationnelle

Mise à jour GitHub : Release v0.2

Ajout de tous les nouveaux fichiers : security.py, auth routes, gestion des tokens, et Dockerisation initiale. Push GitHub en version v0.2.

Résumé :

- Sécurisation de l'API
- Fichiers Dockerfile et docker-compose.yaml ajoutés
- Version v0.2 poussée sur GitHub

Introduction aux Scopes et Permissions

Mise en place des scopes pour contrôler les permissions d'accès selon les rôles utilisateur (ex: accès au dashboard admin réservé). Mise à jour du système de vérification des tokens JWT.

Résumé :

- Définition de scopes utilisateurs
- Vérification d'accès sur les routes via scopes
- Premiers tests d'accès restreints

Rotation sécurisée des tokens

Mise en place de la rotation de Refresh Tokens

Pour renforcer la sécurité, le projet évolue en mettant en place un système de rotation sécurisée des refresh tokens :

- À chaque génération d'un nouveau access token, un nouveau refresh token est émis.
- Les anciens refresh tokens deviennent invalides.
- Il est nécessaire de créer un modèle SQLAlchemy pour stocker les refresh tokens côté serveur.

Résumé :

- Refresh tokens persistés en base
- Invalidation automatique des anciens tokens
- Rotation sécurisée fonctionnelle

Logs d'activité + Monitoring

Mise en place de Loguru pour le suivi des utilisateurs

Mise en place d'un système de logs avec Loguru pour tracer toutes les activités importantes :

- Connexions
- Déconnexions
- Échecs d'authentification

Les logs devront être clairs, horodatés, et exploitables.

Résumé :

- Système de log opérationnel
- Activités critiques tracées

Intégration Prometheus et Grafana pour le monitoring

Déploiement d'instances Prometheus et Grafana via Docker Compose. Exposition d'une route /health dans FastAPI pour permettre à Prometheus de collecter des métriques. Option bonus : ajouter des alertes (ex : tentative de connexion échouée).

Résumé :

- Monitoring Prometheus installé
- Dashboard Grafana accessible
- (Bonus) Alertes configurées

Version finale du projet (Release v0.2)

Finalisation et documentation complète

Finalisation de toute la documentation du projet : installation, déploiement, utilisation. Push de la version finale sur GitHub avec tag de release v0.2. Vérification finale de la clarté du README, de la qualité du code, et des fichiers Docker.

Résumé :

- README et documentations finalisés
- Version v0.2 officielle publiée
- Projet complet et prêt à être présenté