Projet API – `EcoTrack` : API FastAPI

20 novembre 2025

Résumé

Ce document décrit le cahier des charges pour le projet **EcoTrack** : un service API complet développé avec `FastAPI`. Le projet met en application l'ensemble des notions vues en cours : routage, validation Pydantic, authentification JWT, rôle-based access control, ORM (SQLAlchemy), tests, ingestion de données externes, et bonnes pratiques d'ingénierie.

1 Objectifs pédagogiques

— Concevoir et implémenter une API REST maintenable et testée avec `FastAPI`.

— Implémenter l'authentification par JWT et la gestion des rôles (`user`, `admin`).

— Mettre en place un script d'ingestion de données externes.

— Utiliser un ORM (SQLAlchemy) pour modeller la base de données et gérer les migrations.

— Exposer des endpoints de consultation, de recherche filtrée, et de statistiques.

2 Thématique choisie

Suivi d'indicateurs environnementaux locaux : qualité de l'air, émissions CO2 estimées, consommation énergétique et collecte des déchets. Le service vise à agréger des séries temporelles d'indicateurs par zone (ville / code postal / zone géographique) et à fournir des endpoints pour analyses et visualisations.

3 Sources de données (exemples)

Chaque groupe intègre au moins **deux** sources externes. Exemples :

— API OpenAQ (qualité de l'air)

— Open-Meteo (météo historique / mesures)

— Datasets ADEME ou data.gouv (consommation / déchets)

— Fichiers CSV open-data (import via endpoint d'upload)

Pour chaque source, documenter URL, format, fréquence, et limitations (quotas, qualité des timestamps, coordonnées manquantes...).

4 Fonctionnalités attendues

Authentification et sécurité

— Inscription publique et endpoint de connexion (génération de JWT).

— Rôles : `user` (lecture), `admin` (CRUD complet).

— Protection des routes par dépendances FastAPI et scopes Pydantic.

Gestion des utilisateurs

CRUD complet pour les utilisateurs (admin) : liste paginée, activation/désactivation, modification des rôles.

Entités principales (exemples)

Modéliser au moins **une** entité indicateur, par exemple :

— Indicateur : id, source, type, value, unit, timestamp, zoneId, metadata

— Zone : `id, name, geom (optionnel), postal code`

— Source : métadonnées d'origine

Chaque entité doit disposer de routes CRUD, avec validation et gestion d'erreurs.

Recherche, filtres et pagination

Endpoints supportant :

— filtrage par plage de dates, par zone, par type d'indicateur

— pagination (`skip, limit`)

— tri et recherche textuelle si pertinent

Statistiques et agrégations

Exemples d'endpoints :

— `/stats/air/averages?from=yyyy-mm-ddto=yyyy-mm-dd&zone=...`

— `/stats/co2/trend?zone=...&period=monthly`

Les résultats doivent être prêts à être consommés par un frontend (JSON structuré : labels + series).

Tests et qualité

— Tests d'endpoint (tests d'intégration) pour vérifier le fonctionnement réel des routes : création d'utilisateur, login, création d'indicator, récupération filtrée, statistiques.

Front-end

Le front-end doit permettre d'utiliser l'API de manière simple et fluide. Il doit inclure :

— Consommation des endpoints (affichage + actions).

— Une page principale pour visualiser et filtrer les données.

— Un formulaire pour créer ou modifier un élément.

— Une petite gestion des erreurs (messages si l'API renvoie une erreur).

— Réalisation en HTML/CSS/JS ou framework (React/Vue).

— Interface claire, dynamique et intuitive.

5 Livrables

1. Dépôt Git (GitHub/GitLab) comprenant le code API, scripts, tests, et README.
2. Documentation sur la data (liste des sources + justification)
3. Script d'initialisation pour remplir la BDD avec jeu de test.
4. Mini dashboard (optionnel) consommant l'API (ex : une page HTML + JS).