Challenge Datalake

Groupe : Amayas Bariz , Fayçal Boubekri, Hadil Neji

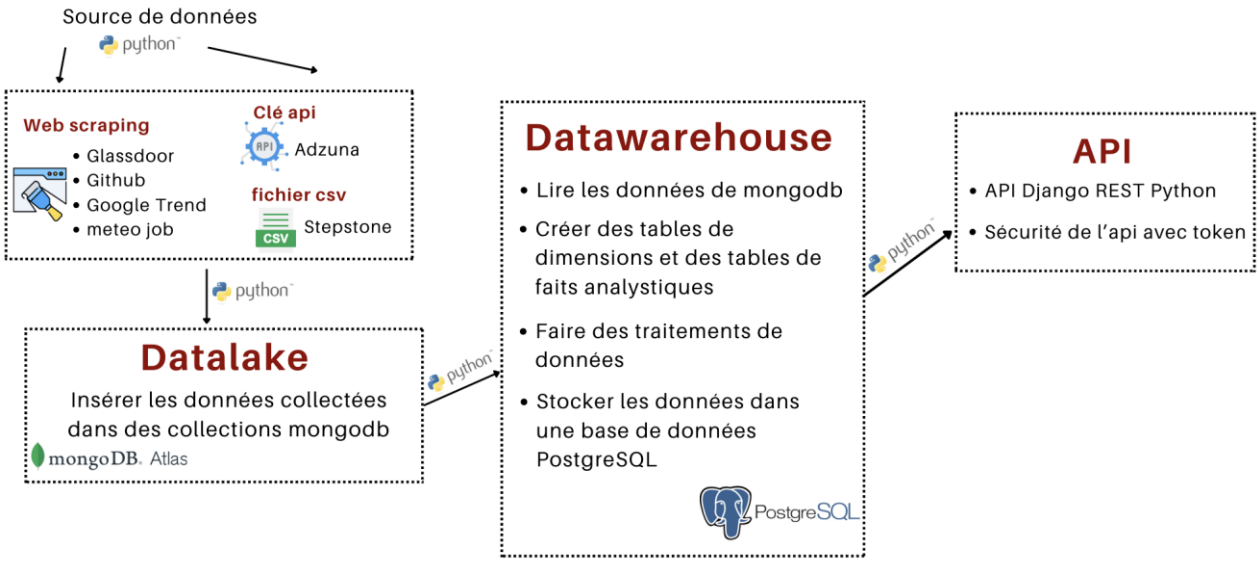
1- Problématique

Comment l'analyse combinée des offres d'emploi, des salaires, des tendances technologiques, des projets open source et de la réputation des entreprises peut-elle aider à mieux s'orienter, recruter, se former ou innover dans un secteur tech en constante mutation — que l'on soit entreprise, recruteur, étudiant ou chercheur ?

L'impact sur les profils visés :

- **Entreprises** : décision RH & positionnement,
- **Étudiants / jeunes diplômés** : choix de carrière, technologies à apprendre,
- **Chercheurs / formateurs** : veille technologique,
- **Candidats** : choix des entreprises, négociation salariale, localisation `

2- Architecture



3- Scraping et fonctionnement du projet

Le module de scraping est conçu pour collecter des données sur les offres d'emploi, les tendances technologiques et les projets GitHub populaires à partir de plusieurs sources en ligne. Les données brutes collectées sont ensuite stockées dans **MongoDB** pour construire un datalake.

Sources de données

- **Glassdoor** : Collecte des offres d'emploi pour différents pays et catégories.
- **Adzuna** : Collecte des offres d'emploi multi-pays.

- **GitHub Trending** : Collecte des projets populaires sur GitHub.
- **Meteojob** : Collecte des offres d'emploi en France.
- **Google Trends** : Collecte des tendances de recherche Google.
- **Stepstone** : Collecte des offres d'emploi à partir du fichier `stepstone_jobs.csv`.

Modules de scraping

- `glassdor_scraping.py` : Scrape les données d'emploi de Glassdoor.
- `azuna_scaping.py` : Scrape les données d'emploi d'Adzuna.
- `github_trending_scraping.py` : Scrape les projets GitHub populaires.
- `meteojob_scraping.py` : Scrape les données d'emploi de Meteojob.
- `google_trend.py` : Scrape les tendances Google.
- 'Stepstone' : Les données sont directement chargées à partir du fichier `stepstone_jobs.csv`.

Remarque :

- Tous les scrapings ont été effectués en utilisant le **multithreading** afin d'accélérer le processus.
- Le **scraping est incrémental** : les scripts évitent de récupérer les données déjà collectées précédemment, ce qui permet de réduire la charge et d'améliorer l'efficacité.

Fonctionnement

Chaque module de scraping utilise Selenium , beautiful soup ou des requêtes HTTP pour collecter les données. Les données sont ensuite sauvegardées dans des fichiers CSV intermédiaires dans le dossier `output/`. Les données sont chargées dans MongoDB de manière incrémentale à partir du fichier les fichier csv .

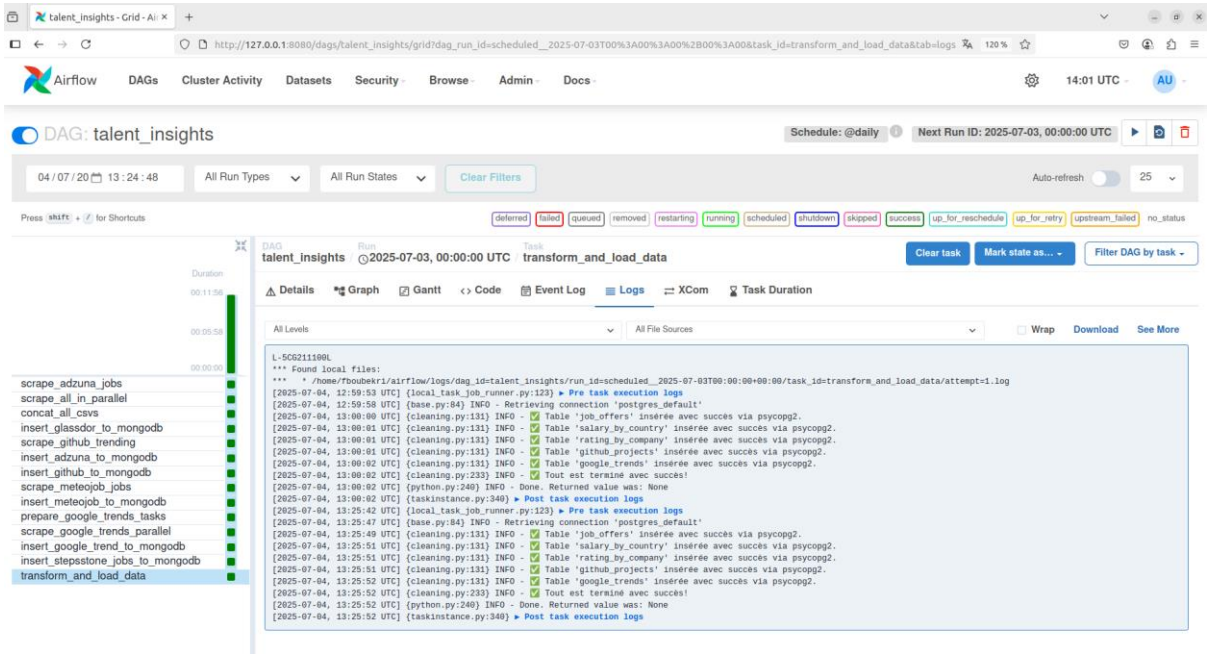
Utilisation dans le DAG

Le DAG `multi_url_scraping_glassdoor` orchestre les tâches de scraping, de nettoyage, et de stockage des données dans MongoDB et PostgreSQL.

Récupération par batch quotidienne

Grâce à la configuration du DAG talent_insights, dont le paramètre schedule_interval est défini à @daily, le scraping et la récupération des données s'effectuent automatiquement chaque jour. Cette exécution quotidienne permet de collecter les

données par batch de manière régulière, assurant ainsi une mise à jour continue et fiable des informations issues des différentes sources ciblées.



Configuration

- ****Dossier de sortie**** : Les fichiers CSV intermédiaires sont stockés dans `/talent_insights/output`.
- ****Fichier final**** : Les données consolidées sont sauvegardées dans `/home/fboubekri/talent_insights/output/all_jobs.csv`.

Exécution

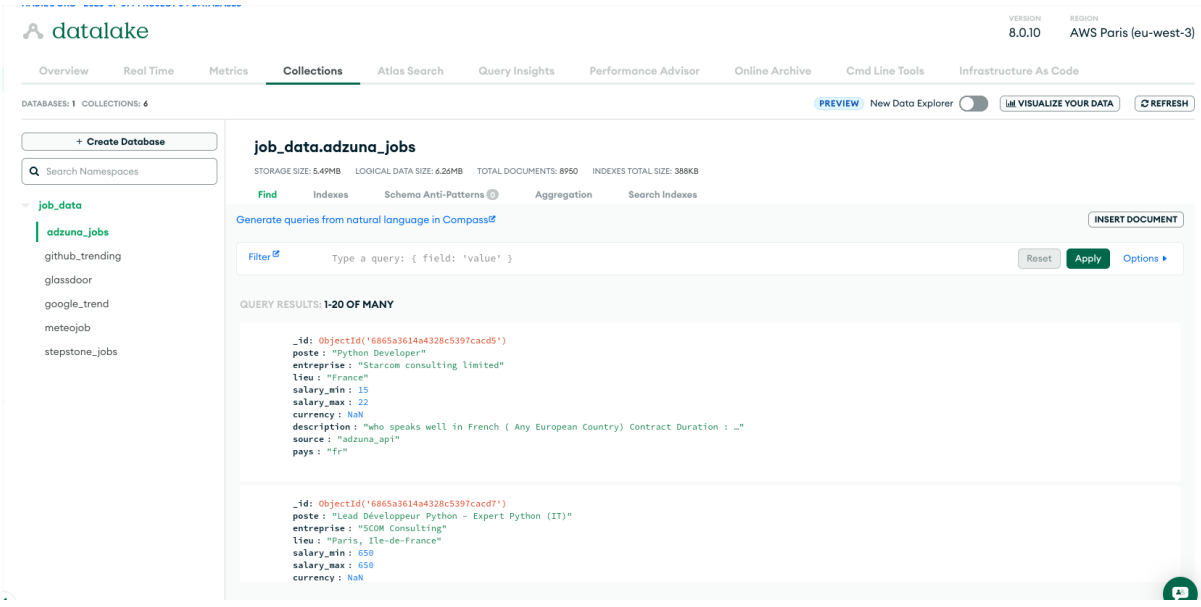
Le DAG est configuré pour s'exécuter quotidiennement (`@daily`) et orchestre toutes les tâches de manière séquentielle ou parallèle selon les dépendances définies. Les données Stepstone sont chargées de manière incrémentale pour éviter de recharger les anciennes données déjà insérées.

4- Les sources de données

Source	Données (captures d'écran des collections mongoDB atlas)	Techno et méthode

Adzuna	`_id: ObjectId('6865a3614a4328c5397cacd9')`
	`poste : "Ingénieur Backend Python - Fastapi H/F"`
	`entreprise : "Savane Consulting"`
	`lieu : "Betton, Rennes"`
	`salary_min : 40000`
	`salary_max : 50000`
	`currency : NaN`
	`description : "Ingénieur Backend Python / API Gateway - CDI - Rennes (35)`
	`source : "adzuna_api"`
	`pays : "fr"`
		Clé Api Requests

glassdoor	_id: ObjectId('6866860ce1d3f3cf6be89333') poste : "Senior UX Group Expert F/H" lieu : "Lyon" entreprise : "Framatome" salaire : "60 k € - 75 k € (fourni par l'employeur)" note : "4,0" description : "Vous aurez l'occasion de travailler sur des projets	selenium
meteojob	_id: ObjectId('6865a3763bc37da41da8507f') date : "2025-07-02T23:20:03.182196" position : "Data Analyst - (H/F) - En alternance" company : "OpenClassrooms" location : "Clichy (92)" contract : "Alternance / Apprentissage- 0 Year" salary : "900 € - 1 600 € par mois"	BeautifulSoup
Google trends	_id: ObjectId('6865a391b68808f6241b23ba') date : "2025-04-07" keyword : "Python" country : "PL" timeframe : "today 3-m" popularity : 97	Pytrends Request BeautifulSoup
Github	_id: ObjectId('6865a3703c68314065aeffe8') author : "microsoft" name : "generative-ai-for-beginners" description : "21 Lessons, Get Started Building with Generative AI language : "Jupyter Notebook" stars : 89825 stars_period : "1,880" url : "https://github.com/microsoft/generative-ai-for-beginners"	BeautifulSoup Requests
Kaggle (stepstone jobs)	_id: ObjectId('6865a2cb693480af12db8b01') Job Title : "Data Engineer / Data Scientist (m/w/d) Remote / NRW" Company Name : "PAKi Logistics GmbH" Location : "Ennepetal" Contract Type : "Permanent contract" Job Type : "Full Time, Home office possible" Published Days : "Published: 1 day ago" Salary Range : "43.000€ - 64.000€/year (full-time estimate)" Verified Status : "-1"	Fichier CSV



Connexion entre Airflow et PostgreSQL

Pour connecter Airflow à PostgreSQL, nous avons utilisé le `PostgresHook` fourni par le package Airflow. La connexion a été configurée via l'interface Airflow dans la section "Admin > Connexions". Voici les étapes principales :

1. Création d'une connexion PostgreSQL dans l'interface Airflow avec les paramètres nécessaires (hôte, port, utilisateur, mot de passe, base de données).

2. Utilisation du `PostgresHook` dans les scripts Python pour exécuter des requêtes SQL et insérer les données transformées dans PostgreSQL.

5- Traitement des données

Le but est de centraliser, nettoyer, harmoniser et structurer des données issues de plusieurs sources d'offres d'emploi tech, puis pour les exporter dans des tables analytiques sous PostgreSQL.

Types de traitements effectués :

- Centralisation et fusion multi-sources : récupère les données brutes depuis MongoDB
- Nettoyage et harmonisation : Renomme les colonnes pour avoir des noms standards
- Ajoute des colonnes manquantes
- Génère des identifiants uniques et ajoute des métadonnées (source, date..)
- Normalisation des salaires : convertit tous les montants dans une même unité annuelle et une même devise, extrait salaire min/max, gère les différents formats (mensuel, horaire, annuel, etc).
- Détection du type de contrat : infère automatiquement si le poste est un CDI, CDD, stage, alternance, etc. à partir du titre
- Nettoyage et standardisation des titres de poste : enlève les mentions inutiles, formate les titres, les mappe vers une liste de métiers standard
- Extraction automatique des compétences : détecte la présence de technologies et compétences (Python, Docker, etc) dans la description de poste

Structuration analytique :

- Fusionne toutes les offres dans une table centrale job_offers au format unifié.
- Crée des tables de synthèse analytiques à partir de la table centrale :

salary_by_country : statistiques de salaires par pays et métier.
rating_by_company : notes des entreprises par poste .
github_projects : tendances des technos open-source.
google_trends : popularité des mots-clés métiers tech.

- Crée des tables de dimensions

--Les tables de dimensions:

--Table d_date :

CREATE TABLE d_date (date_key DATE PRIMARY KEY, day INTEGER, month INTEGER, quarter INTEGER, year INTEGER, day_week INTEGER);

--Table d_skill :

CREATE TABLE d_skill (tech_label TEXT, skill_group TEXT, id_skill INTEGER PRIMARY KEY);

--Table d_country :

CREATE TABLE d_country (country TEXT, id_country INTEGER, iso2 TEXT, country_name TEXT PRIMARY KEY, monnaie_iso3 TEXT);

--Table d_source :

CREATE TABLE d_source (source_name TEXT PRIMARY KEY, id_source INTEGER);

--Les tables de faits :

--Table job_offers :

CREATE TABLE job_offers (id UUID PRIMARY KEY, job_title TEXT, normalized_title TEXT, company TEXT, location TEXT, source TEXT, country TEXT, date DATE, description TEXT, skills TEXT, salary_min NUMERIC, salary_max NUMERIC, currency TEXT, contract_type TEXT, CONSTRAINT fk_job_offers_country FOREIGN KEY (country) REFERENCES d_country(country_name), CONSTRAINT fk_job_offers_date FOREIGN KEY (date) REFERENCES d_date(date_key), CONSTRAINT fk_job_offers_source FOREIGN KEY (source) REFERENCES d_source(source_name));

--Table salary_by_country :

CREATE TABLE salary_by_country (id UUID PRIMARY KEY, country_name TEXT, normalized_title TEXT, salary_min NUMERIC, salary_max NUMERIC,

salary_median NUMERIC, currency TEXT, date DATE, CONSTRAINT fk_salary_country FOREIGN KEY (country_name) REFERENCES d_country(country_name), CONSTRAINT fk_salary_date FOREIGN KEY (date) REFERENCES d_date(date_key));

--Table rating_by_company :

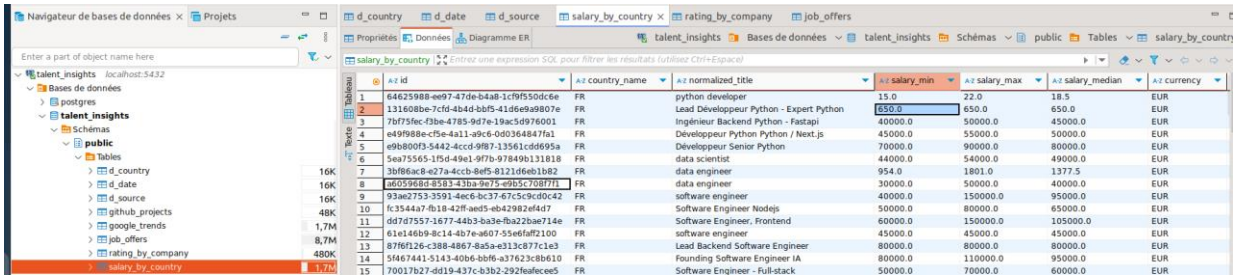
CREATE TABLE google_trends (id UUID PRIMARY KEY, date DATE, keyword TEXT, country_name TEXT, timeframe TEXT, popularity INTEGER, CONSTRAINT fk_trends_country FOREIGN KEY (country_name) REFERENCES d_country(country_name));

--Table github_projects :

CREATE TABLE github_projects (id UUID PRIMARY KEY, author TEXT, name TEXT, description TEXT, language TEXT, stars INTEGER, stars_period INTEGER, url TEXT, date DATE, CONSTRAINT fk_github_projects_date FOREIGN KEY (date) REFERENCES d_date(date_key));

--Table google_trends :

CREATE TABLE google_trends (id UUID PRIMARY KEY, date DATE, keyword TEXT, country_name TEXT, timeframe TEXT, popularity INTEGER, CONSTRAINT fk_trends_country FOREIGN KEY (country_name) REFERENCES d_country(country_name), data date;



6- API

Afin de valoriser les données traitées et de les rendre exploitables par des outils externes (dashboard, frontend, applications métier), une API REST a été développée à l'aide du framework **Django REST Framework**.

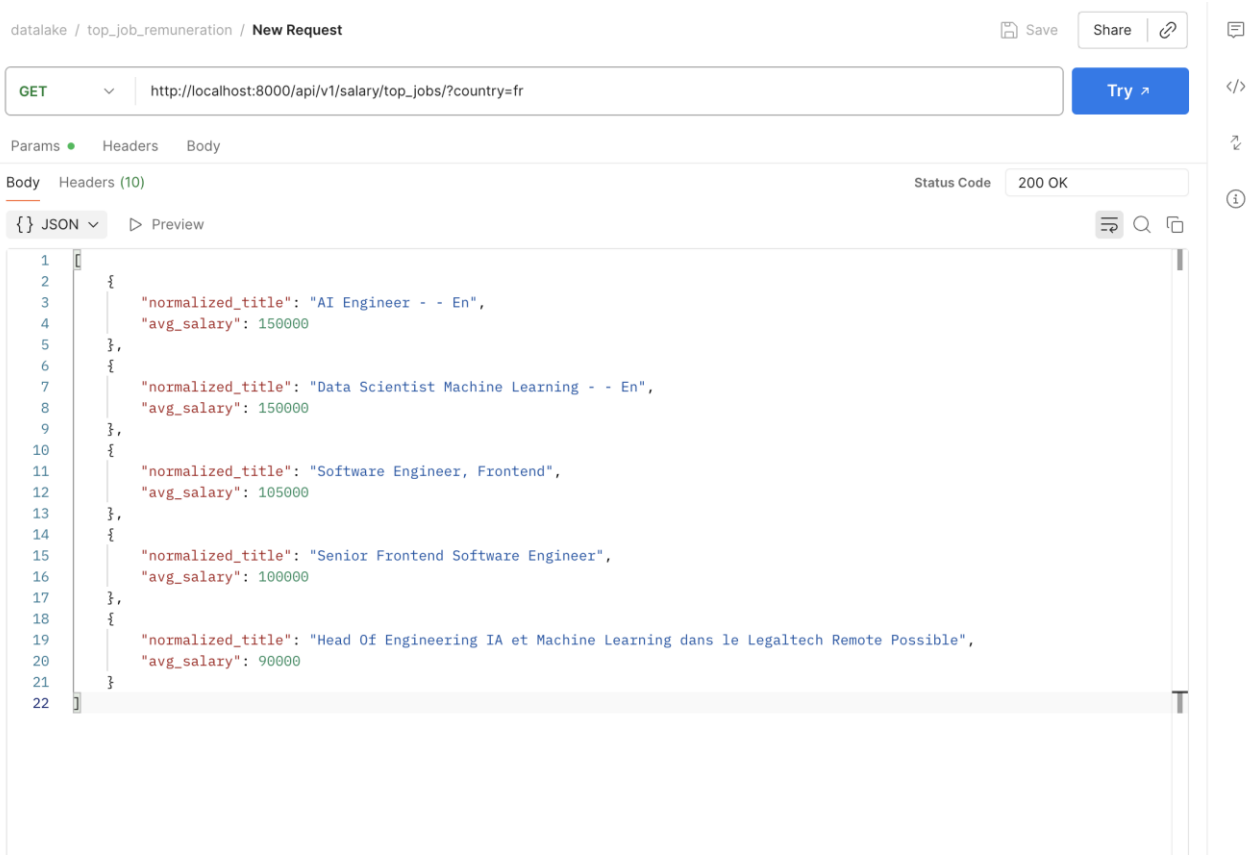
Cette API permet de répondre à un ensemble de **questions métiers clés**, en exposant des **endpoints simples, accessibles en HTTP**, et en lien direct avec les besoins des recruteurs, des candidats, des analystes ou des formateurs.

Méthode	Endpoint	Paramètres	Description	Authentification

POST	http://localhost:8000/api/v1/register/	username, password	Crée un utilisateur + retourne les tokens	Non
GET	http://localhost:8000/api/v1/salary/stats/	title, country	Statistiques salaire min / max / médian pour un poste et pays	Oui
GET	http://localhost:8000/api/v1/salary/top_jobs/	country	Top 5 des métiers avec le salaire médian le plus élevé dans un pays	Oui
GET	http://localhost:8000/api/v1/salary/top_countries	title (optionnel)	Classement des pays avec les meilleurs salaires (global ou par métier)	Oui
GET	http://localhost:8000/api/v1/salary/trends/	title	Tendance du salaire médian au fil du temps pour un métier	Oui
GET	http://localhost:8000/api/v1/joboffers/most_demanded/	-	Les 10 métiers les plus demandés	Oui
GET	http://localhost:8000/api/v1/joboffers/by_company/	title	Classement des entreprises qui recrutent le plus pour un poste donné	Oui
GET	http://localhost:8000/api/v1/joboffers/best_salary/	-	Top 10 des offres avec le salaire max le plus élevé	Oui
GET	http://localhost:8000/api/v1/joboffers/salary_by_region/	title	Moyenne des salaires par région pour un métier	Oui
GET	http://localhost:8000/api/v1/github/top/	-	Top 10 projets GitHub les	Oui

			plus étoilés sur la période	
GET	http://localhost:8000/api/v1/github/languages/	-	Nombre de projets GitHub par langage	Oui
GET	http://localhost:8000/api/v1/ratings/top/	-	Entreprises avec une note ≥ 4	Oui
GET	http://localhost:8000/api/v1/ratings/average/	-	Moyenne des notes par entreprise	Oui

Exemple : Top 5 des métiers avec le salaire médian le plus élevé dans un pays :



API: Fonctionnement, étapes et sécurisation :

Objectif:

L'API développée avec Django REST Framework permet d'exposer les données nettoyées et stockées dans PostgreSQL. Elle offre un accès structuré à des analyses métier exploitables par des dashboards, des applications tierces ou des outils d'aide à la décision.

Fonctionnement et architecture

L'API repose sur une architecture modulaire qui s'appuie sur les concepts suivants :

Modèles (models.py) : représentent les tables du Data Warehouse PostgreSQL (job_offers, salary_by_country, google_trends, github_projects, etc.).

Sérialiseurs (serializers.py) : transforment les objets Python (instances de modèle) en JSON pour les réponses HTTP.

Vues (views.py) : contiennent la logique de traitement métier, les filtres et les agrégations.

Routes (urls.py) : définissent les chemins accessibles via des requêtes HTTP sous le préfixe /api/v1/.

Étapes de fonctionnement

Requête HTTP : L'utilisateur effectue une requête GET sur l'un des endpoints.

Contrôle des droits d'accès : La requête est validée par un système d'authentification.

Appel à la vue Django : La vue exécute une logique métier (par exemple : regroupement par pays ou métier).

Interrogation PostgreSQL : Les données sont récupérées ou agrégées via le modèle Django connecté à PostgreSQL.

Réponse JSON : Le résultat est sérialisé et retourné à l'utilisateur au format JSON.

Sécurisation de l'API

L'API est sécurisée grâce à un mécanisme d'authentification par JWT (JSON Web Token). Seuls les utilisateurs connectés peuvent accéder aux endpoints protégés.

Étapes de sécurisation :

Inscription : création d'un compte via /api/v1/register/.

Authentification : récupération d'un token via /api/token/.

Utilisation du token : le token est inclus dans l'en-tête de chaque requête (Authorization: Bearer <token>).

Les routes sensibles nécessitent ce token pour garantir que seules les personnes autorisées accèdent aux données. Certaines routes publiques restent accessibles sans authentification (par exemple : /api/v1/joboffers/).

Degré de sécurité :

Authentification via JWT : garantit que chaque requête est émise par un utilisateur identifié.

Lecture seule sur les endpoints analytiques : aucune modification directe de la base n'est possible depuis l'API.

Données anonymisées : aucune donnée personnelle n'est exposée.

Versioning clair : l'API suit une logique de versioning (/api/v1/) pour assurer la stabilité en cas d'évolution.