EFREI - Master Data IA

PARIS PANTHÉON-ASSAS UNIVERSITÉ

Mr.Cyril VIMARD

Rapport Du Projet

Challenge Datalake

Réalisé la semaine du 30/06/2025

De Ethan Tomaso, Elliot Fesquet, Julien Trémont-Raimi &

Antoine Vandeplanque

I. Introduction du Projet

1.1 Contexte et objectifs

Face à la pénurie chronique de profils numériques en Europe, la Commission européenne vient de voter un plan baptisé **"TalentInsight"**.

Son ambition :
cartographier, en quasi-temps réel, l'offre et la demande de compétences Tech (salaires, stacks, niveau d'expérience, popularité des langages), afin :

1 / d'ajuster les politiques de formation et de reconversion

2/ de guider les start-ups dans leurs stratégies de recrutement

3/ de fournir aux chercheurs un jeu de données public et transparent.

Or, les informations sont dispersées : sites d'emploi, réseaux de codeurs, tendances de recherche, enquêtes annuelles…

1.2 Vision d'ensemble de la solution

- Recueillir auprés de 6 sources de données différentes des informations sur le marché du tech en Europe
- Construire une base de donnée Warehouse pour normaliser, nettoyer les données
- Proposer une API django qui mets à disposition les sources de données, une par une mais aussi liée entre elle sur des données uniformisées
- Documentation du code source sur github

1.3 Technologies utilisées

-Python Vscode/Pycharm (IDE)
-Framework Django
-Mysql & Datagrip (IDE)
-Spark (par le biais de la lib python pyspark)
-selenium (driver d'emulation d'explorateur web pour scrapper)
-multitude de librairies pour servir les scripts en tout genre

II. Architecture Technique

2.1 Architecture Medallion (Bronze, Silver, Gold)

https://github.com/julientremont/Challenge_DL_and_DI

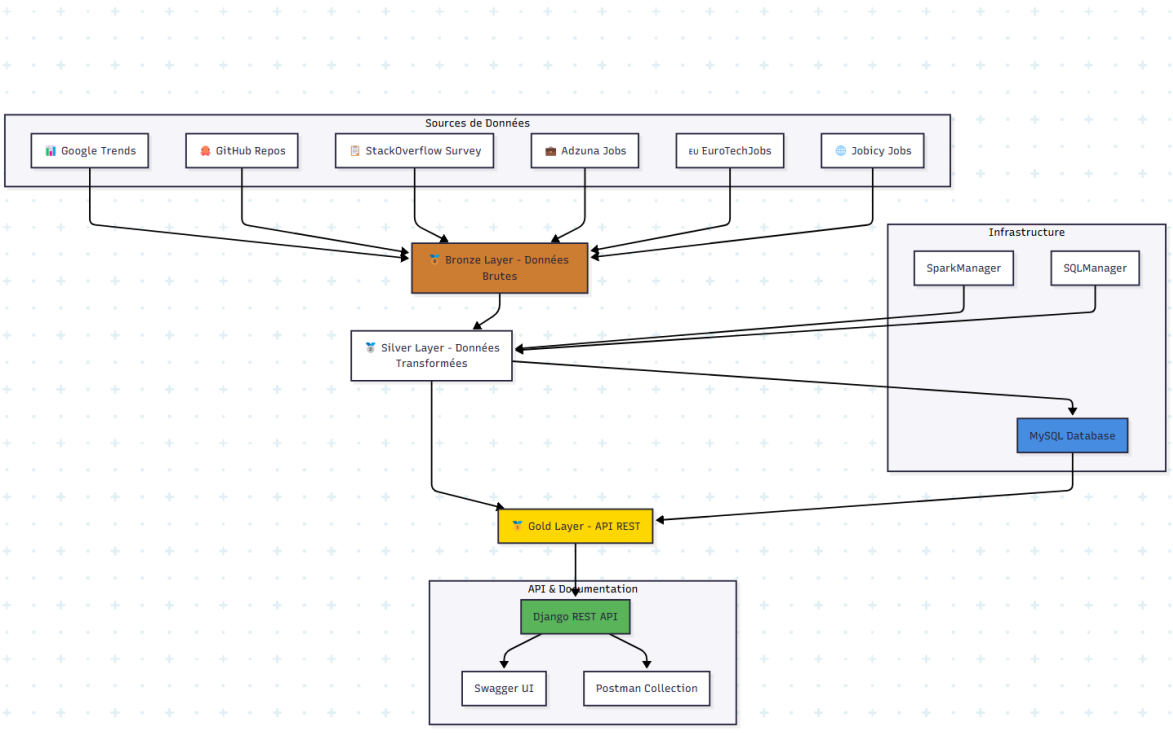
Le projet servira un schéma d'architecture médaillon, qui se compose de 3 parties distinctes :

- Le **bronze**, la partie qui scrap, importe et ingère les données en format parquet en local, son rôle est semblable à celui d'un livreur
- le **silver**, la partie qui nettoie, centralise et normalise les données, elle envoie les données sur le data warehouse mysql, son rôle est semblable à celui d'un manufacturer
- le **gold**, consistant au data warehouse, qui doit être construit de sorte à rendre accessible les données à l'api, son rôle est semblable à celui d'un marchand

Finalement, le gold renverra sur l'api django, qui fait guise de vitrine de donnée, ou plusieurs endpoints seront constitué afin de donner accès aux clients à toute sortes de données

2.2 Schéma d'infrastructure

Nous avons décidé d'utiliser un mermaid pour illustrer le projet, l'illustration et la documentation technique de l'architecture en arbre et autre justification se trouveront dans le **README** du projet



III. Sources de Données

3.1 Google Trends

3.1.1 Collecte automatisée

Le système utilise la bibliothèque `pytrends` pour récupérer les données Google Trends via leur API publique. Deux modes de collecte sont implémentés :

Collecte historique : Extraction des données sur une plage temporelle définie, avec traitement mensuel pour optimiser les requêtes. Le processus segmente automatiquement les périodes longues en tranches mensuelles afin de respecter les limitations de l'API.

Collecte quotidienne : Récupération des données de la veille, permettant une actualisation régulière du dataset. Ce mode est configuré pour s'exécuter automatiquement chaque jour.

Les requêtes intègrent des mécanismes de gestion d'erreurs et de temporisation aléatoire (1-60 secondes) pour éviter la saturation de l'API. En cas d'échec, le système applique des pauses progressives avant de relancer les requêtes.

3.1.2 Structure des données

Couche Bronze : Les données brutes sont stockées au format Parquet avec partitionnement temporel (année/mois/jour). Chaque enregistrement contient :

- `keyword` : Terme de recherche technologique
- `country` : Code pays (AT, BE, CH, DE, ES, GB, IT, NL, PL, FR)
- `date` : Date de la mesure
- `search_frequency` : Fréquence de recherche (0-100)
- `isPartial` : Indicateur de données partielles

Couche Silver : Transformation et nettoyage via PySpark avec :

- Suppression des lignes avec plus de 70% de valeurs nulles
- Élimination des doublons
- Standardisation des noms de colonnes
- Enrichissement avec le nom complet du pays
- Stockage en base MySQL pour requêtes optimisées

La structure finale en base comprend un index composite sur (country_code, date, keyword) pour améliorer les performances de recherche.

3.1.3 Fréquence et volume

Couverture géographique : 10 pays européens principaux **Périmètre technologique** : 21 technologies (langages de programmation, frameworks web) **Volume quotidien** : ~210 points de données par jour (21 technologies × 10 pays) **Rétention** : Historique complet depuis juin 2025 avec actualisation quotidienne

Le pipeline traite environ 6 300 requêtes par mois, générant un dataset de plusieurs milliers d'enregistrements. L'architecture modulaire permet d'étendre facilement le périmètre géographique ou technologique selon les besoins d'analyse.

3.2 StackOverflow Developer Survey

3.2.1 Données annuelles des développeurs

StackOverflow publie chaque année un csv conséquent avec une quantité considérable de questions orienté sur le sujet de l'informatique, sur cette url : https://survey.stackoverflow.co/, le script de scrapping est simple : on précise les années qu'on veut tirer et on itère depuis l'url spécifié avec modulation de l'année, ainsi on utilisera ce genre d'instruction :

```
return f"{self.base_url}stack-overflow-developer-survey-{year}.zip"
```

pour illustrer la donnée, on fait ensuite un mapping des seuls colonnes importantes car il y en a plus d'une centaine, et on stoque en parquet.

3.2.2 Structure des données

Couche Bronze : on aura donc un format du type :

- `country` : Nom complet du pays
- `age`: age du participant
- `developer_type`: nature du métier du participant
- `education_level`: niveau du diplôme du participant
- `salary_usd`: salaire en dollars américain du participant
- `*_worked`: technologies que le participant utilise activement
- `collected_at`: Période de mesure (format YYYY-MM)
- `main_branch`: situation du participant (actif ou étudiant ou autre)

Couche Silver :

on compacte les données en les normalisant, supprimant les aberrations et autres opérations , afin d'extraits de nouvelles colonnes :

- `salary_range`: fourchette de salaire
- `primary_language`: langage utilisé principalement dans le contexte du métier
- `data_quality_score`: qualité global partant de 100 baissant avec les compromis de normalisation
- `education_normalized`: catégorise en catégorie reconnaissable les diplômes

3.3 Adzuna Jobs

3.3.1 Collecte automatisée

L'intégration Adzuna exploite l'API REST officielle pour extraire deux types de données complémentaires sur le marché de l'emploi IT en Europe :

Données historiques de salaires : Récupération mensuelle des moyennes salariales via l'endpoint `/history` avec filtrage sur la catégorie "it-jobs". Le processus interroge séquentiellement chaque pays pour constituer un historique complet.

Distribution salariale : Extraction de l'histogramme des salaires via l'endpoint `/histogram`, fournissant la répartition des offres d'emploi par tranche salariale. Ces données snapshots sont collectées mensuellement pour suivre l'évolution du marché.

Le système intègre une gestion robuste des erreurs avec vérification du statut HTTP, validation du format JSON et traitement spécifique des exceptions API (pays non supportés, limites de taux). Un délai de 3 secondes entre les requêtes prévient la surcharge des serveurs Adzuna.

3.3.2 Structure des données

Couche Bronze : Organisation en deux flux distincts avec partitionnement temporel (année/mois) :

Flux Salaires :

- `keyword` : Catégorie d'emploi ("it-jobs")
- `country` : Nom complet du pays
- `date` : Période de mesure (format YYYY-MM)
- `average_salary` : Salaire moyen en devise locale

Flux Distribution :

- `keyword` : Catégorie d'emploi
- `country` : Nom complet du pays
- `salary_range` : Tranche salariale
- `job_count` : Nombre d'offres dans cette tranche
- `date` : Date de collecte

Couche Silver : Deux tables MySQL optimisées avec nettoyage avancé (seuil de nullité à 70%) et standardisation :

- `salary_it_jobs` : Moyennes salariales mensuelles par pays
- `it_jobs_salary` : Distribution détaillée des offres par tranche salariale

3.3.3 Fréquence et volume

Couverture géographique : 10 pays européens (Autriche, Belgique, Suisse, Allemagne, Espagne, France, Royaume-Uni, Italie, Pays-Bas, Pologne) **Secteur ciblé** : Emplois informatiques uniquement **Actualisation** : Mensuelle pour optimiser l'usage des quotas API **Volume traité** : ~240 points de données par collecte (10 pays × 2 endpoints × historique variable)

Les données historiques remontent selon la disponibilité de l'API Adzuna, généralement sur 12-24 mois. La collecte mensuelle permet un suivi régulier des tendances salariales tout en respectant les contraintes de l'API gratuite (1000 appels/mois).

3.4 EuroTech Jobs

3.4.1 Données entreprises

Scrapping d'EuroTech Jobs en ciblant essentiellement les technologies suivantes:

```
tech_keywords = [
            'python', 'javascript', 'java', 'typescript', 'c++', 'c#',
'php', 'ruby', 'go', 'rust',
            'kotlin', 'swift', 'scala', 'dart', 'matlab', 'perl', 'lua',
'haskell', 'clojure',
      ....
```

Chaque page listant les offres est parcourue, avec parsing des titres, entreprises, localisations et technologies détectées.

3.4.2 Informations jobs et salaires

Pour chaque offre, on extrait les champs pertinents (job title, company, location, techs) ainsi que les mentions explicites de salaires ou fourchettes quand elles existent. Les données sont ensuite nettoyées, normalisées et stockées au format Parquet.

3.5 Github Repositories

3.5.1 Données repositories

Le scrapping des repositories GitHub est effectué via l'API publique, en ciblant des projets open-source en fonction de critères précis tels que le langage de programmation, le nombre d'étoiles (stars), les forks, et la dernière date de mise à jour. L'objectif est de recenser les projets les plus populaires et les plus actifs sur la plateforme.

3.5.2 Scrapping et recensement des données open-sources

Le processus de scrapping consiste à parcourir les pages des repositories et à extraire un ensemble d'informations clés : nom du projet, propriétaire (owner), description, langage principal utilisé, type de licence (si disponible), ainsi que des métriques d'engagement telles que le nombre d'étoiles (stars), de forks et d'issues ouvertes. Ces données sont collectées de manière paginée via l'API GitHub, puis nettoyées et normalisées pour être stockées au format Parquet, permettant ainsi une exploitation facile et rapide pour des analyses futures.

3.6 Jobicy

3.6.1 Données entreprises et offres

Le scrapping de Jobicy consiste à extraire les annonces d'emploi dans le secteur tech en ciblant des entreprises et rôles spécifiquement orientés vers le télétravail. On se concentre sur des critères comme les technologies utilisées, les intitulés de postes et les types de contrats proposés (temps plein, freelance, etc.).

3.6.2 Scrapping et recensement des données des offres

Le processus de scrapping permet de récupérer des informations détaillées sur chaque offre : titre du poste, entreprise, localisation (si précisé), technologies recherchées, ainsi que des informations sur le salaire ou les avantages (quand disponibles). Les données sont extraites de manière systématique depuis les pages de Jobicy, nettoyées pour éliminer les doublons et inconsistances, puis stockées sous forme de fichier Parquet pour une analyse future et une utilisation dans des modèles prédictifs.

IV. Automatisation des flux d’ingestion

4.1 POSTMAN

a structure de la collection POSTMAN est organisée en plusieurs sections, chacune dédiée à un type de donnée ou à une fonctionnalité spécifique. Par exemple, l'endpoint de **Google Trends** permet de récupérer des listes de tendances en temps réel, d'analyser des mots-clés selon leur popularité géographique, ou d'extraire des résumés statistiques pour une période donnée. De même, les endpoints liés aux **GitHub Repositories** permettent de filtrer les projets selon des critères spécifiques comme la technologie utilisée (ex. : Python) ou le niveau d'activité, facilitant ainsi l'exploration de données sur des projets à fort potentiel.

POSTMAN offre aussi une interface Swagger UI interactive pour consulter la documentation API en temps réel, ce qui permet de tester facilement les différentes requêtes et d'affiner les paramètres en fonction des besoins. Cela facilite également l'intégration avec d'autres outils et services en s'assurant que les appels API respectent les normes de sécurité et de performance.

Il convient de noter que, pour toute opération nécessitant un accès authentifié, l'utilisateur devra impérativement créer un compte afin d'obtenir une clé API valide. Sans cette clé, les requêtes ne pourront pas être exécutées correctement, et des erreurs d'authentification pourraient survenir. Ainsi, une gestion rigoureuse des accès et des permissions est essentielle pour garantir la sécurité des flux de données et assurer le bon déroulement des automatisations.

4.2 Script main

Le script principal présenté gère l'exécution d'un pipeline de traitement de données à trois niveaux : **bronze**, **silver**, et **gold**. Chaque niveau correspond à une étape distincte de l'ingestion, du traitement et de l'unification des données.

1. **Configuration de l'environnement** :
   Le script commence par configurer les logs et ajouter le répertoire racine du projet au chemin Python pour importer facilement les modules internes. Il utilise la bibliothèque `logging` pour consigner les informations d'exécution, notamment les succès et les erreurs des différentes étapes du pipeline.

2. **Paramétrage des sources de données** :
   Le script utilise `argparse` pour permettre à l'utilisateur de spécifier des options de ligne de commande pour sauter certaines étapes ou pour ne traiter qu'un seul niveau de données. Par exemple, l'utilisateur peut choisir de traiter uniquement la couche **bronze** ou d'omettre certaines sources de données spécifiques (Google Trends, GitHub, StackOverflow, etc.).

3. **Vérification de l'environnement** :
   Avant d'exécuter le pipeline, le script vérifie si le fichier `.env` est présent et si toutes les variables d'environnement nécessaires à la connexion à MySQL sont définies. Si des variables sont manquantes, un avertissement est émis.

4. **Exécution du pipeline** :
   Selon les options définies par l'utilisateur, le script exécute les différentes étapes du pipeline. Chaque couche (bronze, silver, et gold) est responsable d'une phase différente :

   - La **bronze layer** collecte les données brutes à partir de diverses sources externes (Google Trends, GitHub, etc.).

   - La **silver layer** traite ces données, les nettoie et les transforme pour les rendre exploitables.

   - La **gold layer** effectue l'unification et la consolidation des tables de données pour les rendre prêtes à l'analyse.

5. **Gestion des erreurs** :
   Chaque étape est entourée de blocs `try-except` afin de capturer et loguer les erreurs éventuelles. Si une étape échoue, le script continue d'exécuter les autres étapes, mais un message d'erreur est consigné dans les logs.

6. **Temps d'exécution** :
   Enfin, le script mesure le temps total d'exécution du pipeline et consigne ce détail dans les logs.

En résumé, ce script offre un cadre robuste pour l'automatisation du traitement de données dans un pipeline ETL (Extract, Transform, Load), tout en permettant une gestion flexible des sources et des étapes, selon les besoins spécifiques de l'utilisateur.

V. API REST Django

5.1 Architecture API

5.1.1 Design RESTful

L'API suit strictement les principes REST avec une architecture modulaire organisée par sources de données :

Structure des endpoints :

- `/api/trends/` - Google Trends
- `/api/stackoverflow-survey/` - Enquête StackOverflow
- `/api/adzuna-jobs/` - Offres d'emploi Adzuna
- `/api/github-repos/` - Dépôts GitHub
- `/api/eurotechjobs/` - Emplois tech européens
- `/api/jobicy/` - Emplois remote européen
- `/api/analysis/` - Analyses cross-sources (Gold layer)

Méthodes HTTP standardisées :

- GET : Récupération des données (list/detail)
- POST : Création (admin uniquement)
- PUT/PATCH : Modification (admin uniquement)
- DELETE : Suppression (admin uniquement)

5.1.2 Documentation Swagger

Documentation automatique complète via **drf-spectacular** :

Configuration avancée :

python
```
SPECTACULAR_SETTINGS = {
    'TITLE': 'Challenge DL & DI API',
    'VERSION': '1.0.0',
    'COMPONENT_SPLIT_REQUEST': True,
    'SWAGGER_UI_SETTINGS': {
        'deepLinking': True,
        'persistAuthorization': True,
        'displayOperationId': True,
        'docExpansion': 'none',
        'filter': True,
        'tagsSorter': 'alpha',
    }
```

```
}
```

Fonctionnalités clés :

- Interface interactive `/api/docs/`
- Documentation ReDoc `/api/redoc/`
- Schéma OpenAPI `/api/schema/`
- Tags organisés par source de données
- Exemples d'authentification intégrés

5.1.3 Authentification et sécurité

Système d'authentification dual :

1. **Authentification par session** (`/api/auth/login/`)
2. **Authentification JWT** (`/api/auth/jwt/login/`)

Gestion des permissions personnalisée :

```
python
class IsAdminOrReadOnlyForUsers(BasePermission):
    # Admin : accès complet
    # Utilisateurs : lecture seule (list/detail)
    # Anonymes : aucun accès
```

Comptes de test intégrés :

- **Admin** : `admin/admin123` (accès complet)
- **User** : `user/user123` (lecture seule)

Configuration JWT sécurisée :

- Tokens d'accès : 60 minutes
- Tokens de rafraîchissement : 7 jours
- Rotation automatique des tokens

5.2 Endpoints par source

5.2.1 Google Trends API

Endpoints disponibles :

- `/api/trends/` - Liste des tendances
- `/api/trends/summary/` - Résumé statistique
- `/api/trends/aggregated/` - Données agrégées
- `/api/trends/time-series/` - Séries temporelles

- `/api/trends/top-keywords/` - Mots-clés populaires

5.2.2 StackOverflow Survey API

Endpoints spécialisés :

- `/api/stackoverflow-survey/` - Données d'enquête
- `/api/stackoverflow-survey/developer-types/` - Types de développeurs
- `/api/stackoverflow-survey/technologies/` - Technologies populaires
- `/api/stackoverflow-survey/salary-analysis/` - Analyses salariales

5.2.3 Adzuna Jobs API

Endpoints métier :

- `/api/adzuna-jobs/` - Offres d'emploi
- `/api/adzuna-jobs/by-location/` - Répartition géographique
- `/api/adzuna-jobs/by-company/` - Analyses par entreprise
- `/api/adzuna-jobs/salary-trends/` - Tendances salariales
- `/api/adzuna-jobs/skills-analysis/` - Analyse des compétences

5.2.4 Jobicy

Endpoints techniques :

- `/api/github-repos/` - Dépôts GitHub
- `/api/github-repos/technology-stats/` - Statistiques technologiques
- `/api/github-repos/activity-analysis/` - Analyse d'activité
- `/api/github-repos/popular/` - Projets populaires

5.3 Fonctionnalités avancées

5.3.1 Filtrage et pagination

Pagination standardisée :

python
'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',

'PAGE_SIZE': 100

Filtres automatiques :

- DjangoFilterBackend (filtres par champs)
- OrderingFilter (tri personnalisé)
- SearchFilter (recherche textuelle)

5.3.2 Analyses temps réel

Architecture Gold Layer :

- Base de données dédiée aux analyses
- Routeur de base personnalisé
- Modèles optimisés pour l'analytique

Endpoints analytiques :

- `/api/analysis/tech-activity/` - Activité technologique
- `/api/analysis/job-market/` - Marché de l'emploi
- `/api/analysis/tech-activity/technology_stats/` - Stats tech
- `/api/analysis/job-market/market_summary/` - Résumé marché

5.3.3 Export de données

Formats supportés :

- JSON (par défaut)
- Réponses paginées avec métadonnées
- Structure standardisée pour tous les endpoints

Gestion des erreurs :

- Codes HTTP appropriés
- Messages d'erreur détaillés
- Validation des données automatique

Performances :

- Optimisation des requêtes ORM
- Mise en cache des analyses
- Pagination intelligente pour les gros volume

VI. Conclusion

Dans le cadre du projet Challenge Data, nous avons conçu une API performante et robuste qui sert d'agrégat pour plusieurs sources de données, allant des plateformes de scraping telles que Jobicy et EuroTechJobs, aux sources de données plus traditionnelles comme StackOverflow et GitHub. L'objectif de cette API est de centraliser et d'unifier des données disparates provenant de multiples canaux, incluant des imports CSV, des requêtes API et des scraping de pages web, afin de fournir une ressource unique et cohérente pour les professionnels de l'industrie.

L'API est structurée en plusieurs couches de traitement de données (bronze, silver, et gold) qui permettent de collecter, transformer et unifier les informations de manière ordonnée. Grâce à cette architecture, elle répond à des besoins complexes d'automatisation des flux d'ingestion tout en offrant une flexibilité maximale pour l'intégration avec d'autres systèmes ou l'exploitation des données par des applications tierces.

Les sources de données variées sont accessibles via une interface simple, enrichie par des mécanismes d'authentification et des contrôles d'intégrité, garantissant ainsi la sécurité et la fiabilité des informations extraites. Cette API s'inscrit donc comme un outil essentiel pour les professionnels recherchant une solution d'agrégation de données à la fois fiable, scalable et facile à intégrer dans leurs processus analytiques et décisionnels.

En résumé, ce projet a permis de répondre à un besoin réel d'automatisation de l'ingestion et de traitement de données tout en fournissant une interface API flexible et orientée métier, prête à être utilisée dans des contextes professionnels exigeants.