Présentation technique du projet (format Questions / Réponses)

Ce doc a pour objectif d'expliquer nos choix techniques réalisés dans le cadre du projet, sous forme de questions-réponses, afin d'anticiper d'éventuelles interrogations concernant l'utilisation des outils et l'architecture du code. (c'est plus interactif :))

Il est important de préciser que de l'intelligence artificielle a bien été utilisée dans notre projet, mais uniquement comme **outil d'aide** : aide à la structuration du code, rechercher des alternatives techniques et résolvé nos pb. L'architecture, l'organisation du code et les choix méthodologiques restent entièrement maîtrisés !

Architecture générale du projet

Pourquoi une organisation du projet en 5 dossiers (01 à 05) ?

Nous avons choisis d'organiser notre projet en cinq dossiers numérotés afin de représenter clairement l'ensemble des notions vues en cours. Cette organisation nous a faciliter pour la lecture et la compréhension de notre code

- **Dossier 01 – Collecte des données** : contient tous les scripts de web scraping.
- **Dossier 02 – Stockage et structuration** : gère la base de données et le schéma relationnel.
- **Dossier 03 – Calculs et transformations** : cœur du projet, où sont effectués les calculs et agrégations (le +important
- **Dossier 04 – API REST** : expose les données via une interface web.
- **Dossier 05 – Visualisations et dashboards** : permet l'exploration de nos résultats.

Choix du langage

Pourquoi Python plutôt qu'un autre langage ?

Python a été choisi pour plusieurs raisons :

1. **Cohérence avec le cours** : Python est le langage que nous avons étudié.
2. **Richesse de l'écosystème** :
    - requests et BeautifulSoup pour le scraping,
    - pandas pour la manipulation de données,
    - plotly pour les visualisations,
    - Flask pour la création d'API.
3. **Simplicité d'utilisation** : la syntaxe est lisible et accessible.

4. **Intégration avec les bases de données** : les bibliothèques comme mysql-connector sont fiables et bien documentées.

Collecte des données

Pourquoi utiliser des classes plutôt que des fonctions simples ?

Des classes (par ex DoctolibScraper) on été utilisées pour encapsuler à la fois l'état et le comportement.

Une classe permet nous a permis de:

- de conserver l'état entre plusieurs appels (session HTTP, délais, compteurs),
- d'éviter de passer de nombreux paramètres aux fonctions,
- de faciliter les extensions futures grâce à l'héritage.

Pourquoi utiliser requests plutôt que Selenium ?

A la base nous utilisions Selenium mais il était trop lent ettrop complexe pour notre cas d'usage. (on a mis longtemps avant de récupéré nos données nécéssaires de praticiens pour notre premiere analyse)

requests a été privilégié car :

- son API est simple et efficace,
- les données de Doctolib sont accessibles via des requêtes API JSON,
- l'exécution de JavaScript complexe n'était pas nécessaire.

Stockage et base de données

Pourquoi MySQL plutôt que SQLite ?

MySQL a été choisi pour plusieurs raisons :

1. **Performance** : optimisé pour les lectures fréquentes.
2. **Compatibilité** : très bien supporté dans l'écosystème Python.
3. **Index géographiques** : efficace pour nos index sur latitude et longitude.

Pourquoi un schéma relationnel plutôt qu'une base NoSQL ?

Un modèle relationnel etait + adapté car nos données sont liées entre elles :

- praticiens ↔ communes,
- communes ↔ régions,
- statistiques ↔ régions et dates.

Ce modèle évite la duplication des données et garantit une meilleure cohérence. Une base NoSQL aurait été plus pertinente pour des données non structurées, ce qui n'est pas notre cas.

Pourquoi ces index spécifiques ?

Des index ont été créés sur les colonnes les + sollicitées :

- region pour les filtres régionaux,
- code_postal pour les recherches géographiques,
- index composite (latitude, longitude) pour les calculs de distance,
- specialite pour les filtres par spécialité médicale. (pour peute etre plus tard si on veut dévélopper notre projet)

Pourquoi la table communes séparée de la table praticiens ?

Cette séparation nous a permis d'éviter la duplication de nos données démographiques car plusieurs praticiens peuvent appartenir à une même commune, dont nos info (population, région..) ne sont stockées qu'1 seule fois.

Calculs et agrégations (partie importante du projet)

Pourquoi une classe AggregationRunner plutôt que des fonctions ?

La classe AggregationRunner permet d'encapsuler la co à notre bdd et de la réutiliser pour plusieurs requêtes. Ca évite d'ouvrir une nvl c à chaque appel et améliore les perf

Pourquoi une détection automatique du nom de table ?

Une méthode _detect_table_name a été mise en place pour identifier auto si la table s'appelle neurologue ou praticiens. (car on avait crée une table practiciens et neurologue et on ne se souvenait plus de laqiem était bonne donc on a fait une détection auto)

Pourquoi utiliser pandas plutôt que du SQL pur pour les calculs ?

Car pandas est particulièrement adapté aux calculs complexes :

- boucles,
- normalisations,
- calculs multi-étapes (comme l'ICA).

Pourquoi utiliser la formule de Haversine pour les distances ?

La formule de Haversine est la méthode standard pour calculer la distance entre deux points sur une sphère. Contrairement à une distance euclidienne, elle tient compte de la courbure de la Terre.

Pourquoi une normalisation robuste basée sur les percentiles ?(vue en analyse de données)

Une normalisation basée sur les percentiles 1 et 99 a été utilisée afin de limiter l'influence des valeurs aberrantes.

Pourquoi calculer notre ICA en Python plutôt qu'en SQL ?

Le calcul de notre ICA nécessite :

- des calculs de distance,
- des comptages conditionnels,
- des normalisations,
- des pondérations configurables,
- une classification finale.

Pourquoi des paramètres configurables w_geo et w_demo ?

Les poids représentent des choix méthodologiques. Les rendre configurables permet :

- d'effectuer des analyses de sensibilité,
- de tester différents scénarios sans modifier le code.

API REST

Pourquoi Flask plutôt que Django ou FastAPI ?

Flask a été choisi pour sa simplicité et sa légèreté.

- Django est trop lourd pour une API simple.
- FastAPI est performant mais plus complexe à prendre en main.

Pourquoi utiliser des blueprints ?

Les routes sont organisées par domaine fonctionnel :

- neurologue : praticiens,
- accessibilite : calculs géographiques,
- indicateurs : indicateurs statistiques.

Dashboard et visualisations

Pourquoi Streamlit ?

Streamlit permet de créer rapidement un dashboard interactif avec très peu de code, ce qui accélère fortement le développement.

Pourquoi Plotly plutôt que Matplotlib ou Seaborn ?

Plotly génère des visualisations interactives par défaut :

- zoom,
- survol,
- exploration dynamique.

Pourquoi charger les données depuis un CSV en priorité ?

Le CSV joue le rôle de cache :

- lecture plus rapide qu'une requête SQL,
- amélioration significative de l'expérience utilisateur.

Pourquoi tester plusieurs chemins pour le fichier CSV ?

Selon le point de lancement du dashboard, les chemins relatifs peuvent varier. Tester plusieurs chemins garantit le bon fonctionnement du dashboard dans tous les cas. (sur chaque ordi)

Gestion des erreurs

Pourquoi utiliser de nombreux blocs try-except ?

Les blocs try-except permettent de gérer les erreurs de manière contrôlée :

- éviter les crashs,
- afficher des messages clairs,
- proposer des solutions alternatives.

Pourquoi fournir des valeurs par défaut pour les configurations ?

Les valeurs par défaut permettent a notre projet de fonctionner même sans fichier .env, ce qui facilite les tests, le développement et la prise en main.

Conclusion

Nos choix techniques réalisés privilégient la **simplicité**, la **maintenabilité** et la **robustesse**. L'architecture modulaire permet de faire évoluer notre projet facilement et d'ajt de nouvelles fonctionnalités

Chaque décision a été prise en tenant compte des contraintes du projet, des performances attendues et du future.