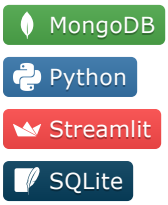
🚇 Paris 2055 - Système de Supervision des Transports



📋 Description

Projet académique réalisé dans le cadre du **BUT Science des Données (3ème année)** à l'Université de Poitiers. Ce projet explore la migration de données d'une base SQL vers MongoDB et analyse les performances des transports publics parisiens dans un contexte futuriste (2055).

L'objectif principal est de comparer les approches **relationnelle (SQL)** et **documentaire (NoSQL)** pour gérer et analyser un système de transport urbain complexe incluant des données de trafic, pollution, incidents et horaires.

✨ Fonctionnalités

🔄 Migration SQL → MongoDB

- Migration complète d'une base SQLite vers MongoDB
- Dénormalisation intelligente des données
- Création de documents imbriqués pour optimiser les requêtes NoSQL
- Gestion des relations géographiques (GeoJSON)

📊 Analyses Avancées (14 requêtes)

- **Trafic** : Retards moyens, taux de ponctualité, incidents
- **Environnement** : Émissions CO2, nuisances sonores, température
- **Performance** : Analyse par ligne, véhicule et chauffeur
- **Urbanisme** : Distribution des arrêts par quartier
- **Corrélations** : Trafic/pollution, retards sans incidents

📈 Dashboard Interactif (Streamlit)

- **KPI en temps réel** : Lignes actives, incidents, CO2 moyen
- **Graphiques dynamiques** :
  - Retards par ligne (barres)
  - Répartition des véhicules (camembert)
  - Évolution CO2 (chronologique)
  - Types d'incidents (barres)
- **Cartographie interactive** :
  - Visualisation des arrêts avec MarkerCluster

    - Carte choroplèthe de pollution par quartier
    - Filtrage par ligne de transport
- **Comparateur SQL/NoSQL** : Validation côte-à-côte des résultats

🛠 Technologies

- **Base de données** : SQLite, MongoDB
- **Outils MongoDB** : MongoDB Compass (interface graphique)
- **Langages** : Python 3.x
- **Visualisation** : Streamlit, Plotly, Folium
- **Traitement** : Pandas, PyMongo
- **Cartographie** : GeoJSON, Folium.plugins

🗀 Structure du Projet

```
Paris2055_MongoDB_migration_SQL/
├── csv/                          # Dossier des résultats d'analyses
│   ├── A_sql.csv - N_sql.csv   # Résultats SQL (14 requêtes)
│   └── A_nosql.csv - N_nosql.csv # Résultats NoSQL (14 requêtes)
├── partie_1_req_sql.py           # Requêtes SQL (14 analyses)
├── partie_2_migration.py         # Script de migration SQL → MongoDB
├── partie_3_req_nosql.py         # Requêtes NoSQL équivalentes
├── partie_4_dashboard.py         # Dashboard Streamlit
├── Paris2055.sqlite              # Base source (non fournie)
├── .gitignore                    # Fichiers à exclure du versioning
└── README.md                     # Documentation du projet
```

🚀 Installation

Prérequis

- Python 3.8+
- MongoDB Community Server
- MongoDB Compass (optionnel, interface graphique)
- SQLite3

Installation des dépendances

```
pip install pymongo pandas streamlit plotly folium streamlit-folium
```

Configuration MongoDB

1. Démarrer le serveur MongoDB local :

```
mongod --dbpath /chemin/vers/data
```

2. Vérifier la connexion sur `mongodb://localhost:27017/`

3. **[Optionnel]** Utiliser MongoDB Compass pour visualiser les données :

    - Ouvrir MongoDB Compass
    - Se connecter à `mongodb://localhost:27017`
    - Explorer les collections `Paris2055`
    - Visualiser les documents, créer des requêtes graphiquement

🕮 Utilisation

1 Exécution des requêtes SQL

```
python partie_1_req_sql.py
```

Génère les fichiers `A_sql.csv` à `N_sql.csv`

2 Migration vers MongoDB

```
python partie_2_migration.py
```

Crée la base `Paris2055` avec 5 collections :

- `Reseau` (lignes, arrêts, véhicules)
- `TraficEvents` (incidents, retards)
- `Quartiers` (géométries GeoJSON)
- `Mesures` (capteurs environnementaux)
- `Horaires` (passages, passagers)

3 Requêtes NoSQL

```
python partie_3_req_nosql.py
```

Génère les fichiers `A_nosql.csv` à `N_nosql.csv`

4 Lancement du Dashboard

```
streamlit run partie_4_dashboard.py
```

Accès via `http://localhost:8501`

📊 Exemples de Requêtes

SQL (Relationnel)

```sql
-- Moyenne des retards par ligne
SELECT Ligne.nom_ligne, AVG(Trafic.retard_minutes) AS retard_moyen
FROM Trafic
LEFT JOIN Ligne ON Trafic.id_ligne = Ligne.id_ligne
GROUP BY Ligne.nom_ligne
ORDER BY retard_moyen DESC;
```

MongoDB (Documentaire)

```javascript
# Moyenne des retards par ligne
db.TraficEvents.aggregate([
    {"$group": {
        "_id": "$id_ligne",
        "retard_moyen": {"$avg": "$retard_minutes"}
    }},
    {"$lookup": {
        "from": "Reseau",
        "localField": "_id",
        "foreignField": "_id",
        "as": "info_ligne"
    }},
    {"$sort": {"retard_moyen": -1}}
])
```

🎯 Objectifs Pédagogiques

- ☑ Comprendre les différences SQL/NoSQL
- ☑ Maîtriser l'agrégation pipeline MongoDB
- ☑ Optimiser les structures de données documentaires
- ☑ Visualiser des données géospatiales
- ☑ Créer des dashboards interactifs

👥 Auteures

- **Jade Le Brouster**
- **Emmanuelle Orain**
- **Lola Dixneuf**

Formation : BUT Science des Données 3 - Université de Poitiers
Année : 2025-2026

📄 Licence

⭐ N'hésitez pas à mettre une étoile si ce projet vous a été utile !

Paris2055_MongoDB_migration_SQL

🚇 Système de supervision des transports Paris 2055 | Migration SQL→MongoDB | 14 requêtes comparatives | Dashboard Streamlit avec cartographie interactive | Analyse trafic, pollution & incidents | BUT Science des Données