Plateforme de moissonnage et d'exposition des données ouvertes - DataQC

Sébastien Lionel Sarr

Thierno Sadou Diallo

2 novembre 2025

Table des matières

1 Introduction

Ce rapport présente la réalisation de **DataQC**, une plateforme web de moissonnage et d'exposition des données ouvertes du Québec, développée dans le cadre du travail pratique du cours *INF37407 – Technologies de l'inforoute*.

L'objectif de ce projet est de créer une application Django complète permettant de moissonner, stocker et exposer les métadonnées provenant du portail Données Québec (`https://www.donneesquebec.ca/`). Ce portail utilise l'API CKAN standard pour la diffusion des métadonnées, ce qui facilite l'intégration.

La plateforme DataQC offre les fonctionnalités suivantes :

— Moissonnage automatique des métadonnées depuis Données Québec
— Stockage des données dans une base de données relationnelle via Django ORM
— Exposition des données via API REST (Django REST Framework)
— Exposition des données via API GraphQL (Graphene-Django)
— Interface d'administration complète pour gérer les données et paramètres
— Page de statistiques avec visualisations interactives

Le développement de cette plateforme a permis de mettre en pratique les concepts appris dans les cours 03 à 07 du module, notamment l'architecture Django MVT, l'ORM Django, le Django REST Framework, et les API GraphQL.

Technologies principales utilisées :

— Django 5.2.7
— Django REST Framework 3.15.2
— Graphene-Django 3.1.5
— Bootstrap 5.1.3
— Chart.js

2 Architecture du système

2.1 Architecture Django MVT

Le projet suit l'architecture Model–View–Template (MVT) de Django, organisé en trois applications principales :

— **coeur**
— **donnees**
— **moissonneur**

2.2 Structure de la base de données

La base est gérée par Django ORM et comprend les modèles : *Organisation*, *JeuDonnees*, *Ressource*, *Categorie*, *SourceDonnees*, *ConfigurationFiltres* et *ConfigurationPlanification*.

2.3 Relations entre modèles

Les relations sont gérées via des `ForeignKey` garantissant l'intégrité référentielle.

3 Implémentation

3.1 Session 1 : Installation et configuration initiale

Mise en place de l'environnement Python virtuel, installation de Django, création du projet et configuration initiale dans `settings.py`.

3.2 Session 2 : Structure et interface de base

Création des trois applications (`coeur`, `donnees`, `moissonneur`), template `base.html` avec Bootstrap, page d'accueil de test.

3.3 Session 3 : Modèles de données et moissonnage

Création des modèles Django, migrations, service de moissonnage via l'API CKAN avec la bibliothèque `requests`, interface utilisateur de moissonnage avec AJAX.

3.4 Session 4 : API REST et GraphQL

Implémentation des ViewSets DRF, endpoints REST, et schéma GraphQL (queries, mutations, statistiques).

3.5 Session 5 : Tests et documentation

Rédaction de tests unitaires pour les modèles et services, configuration Swagger via `drf-yasg`, et collection Postman.

3.6 Session 6 : Interface et statistiques

Amélioration de l'interface admin, ajout des graphiques Chart.js et des statistiques dynamiques.

4 Résultats

4.1 Données moissonnées

— Organisations : 136
— Jeux de données : 2450
— Ressources : 6800

4.2 Fonctionnalités testées

Toutes les fonctionnalités principales ont été validées : moissonnage, API REST/GraphQL, Swagger, statistiques.

4.3 Respect des exigences

Toutes les sections demandées dans le travail pratique ont été couvertes avec succès.

5 Difficultés rencontrées

5.1 Problème GraphQL

Erreur d'importation avec `DjangoFilterConnectionField` remplacée par `graphene.List`.

5.2 Problème Swagger

Conflit de dépendances résolu par réinstallation de `setuptools`.

5.3 Paramètres de moissonnage

Refonte des modèles *SourceDonnees*, *ConfigurationFiltres* et *ConfigurationPlanification*.

6 Conclusion

Ce projet a permis de mettre en pratique les concepts du cours INF37407 :
— Architecture Django MVT
— ORM et migrations
— API REST et GraphQL
— Tests unitaires
— Documentation automatique (Swagger)
La plateforme DataQC est fonctionnelle, modulable et conforme aux exigences. Les améliorations futures incluent un déploiement public, l'authentification JWT et des optimisations de performance.