Rapport Technique Détaillé

Webmapping des Bassins de Production au Cameroun

Projet 5GI

Janvier 2026

Table des matières

Chapitre 1

Introduction

Ce document détaille la réalisation technique d'un Système d'Information Géographique (SIG) Web permettant la visualisation des données de production (Agriculture, Élevage, Pêche) au Cameroun.

L'objectif était de livrer un prototype fonctionnel en 4 jours, capable d'afficher les 58 départements avec des données interactives. Ce rapport insiste particulièrement sur les manipulations de données (ETL) et les configurations serveur réalisées sous environnement Linux (Ubuntu 24.04).

Chapitre 2

Architecture Technique

Nous avons déployé une architecture 3-Tiers entièrement Open Source :

1. **SGBD Spatial :** PostgreSQL 18 + PostGIS.
2. **Backend :** Python 3.12 + Flask (API REST).
3. **Frontend :** HTML5 + Leaflet.js.

Chapitre 3

Méthodologie de Réalisation Détaillée

Cette section décrit le flux de travail (workflow) technique complet, depuis l'acquisition des données brutes jusqu'à leur exposition via le moteur de base de données.

3.1 Étape 1 : Acquisition et Préparation des Données Géographiques

La qualité d'un SIG repose avant tout sur la précision de ses fonds de carte.

- **Source de données :** Nous avons sélectionné la base GADM (Global Administrative Areas - `https://gadm.org`), reconnue pour sa précision topologique et sa couverture mondiale standardisée.
- **Sélection du niveau administratif :** Le fichier téléchargé (`gadm41_CMR_shp.zip`) contient plusieurs couches. Nous avons extrait le fichier `gadm41_CMR_2.shp`.
  - *Niveau 0 :* Frontières nationales.
  - *Niveau 1 :* Régions (10 entités).
  - **Niveau 2 :** Départements (58 entités) - C'est l'échelle retenue pour permettre une analyse fine des bassins de production.
- **Système de Coordonnées (CRS) :** Les données sont natives en WGS84 (EPSG :4326), le standard pour le webmapping (coordonnées en latitude/longitude degrés décimaux).

3.2 Étape 2 : Simulation des Données Métiers (Script Python)

L'absence de données statistiques structurées en Open Data pour l'année 2025 nous a contraints à simuler un jeu de données. Pour garantir la crédibilité du prototype, nous avons développé un générateur de données en Python (`generer_donnees.py`).

Logique algorithmique mise en œuvre :

1. **Dictionnaire de structure :** Mapping statique des 58 départements classés par région.
2. **Règles Agro-écologiques :** Le script applique des profils de production différenciés :
   - *Zone Septentrionale :* Coton, Sorgho, Élevage bovin intensif.
   - *Zone Forestière/Sud :* Cacao, Banane, Pêche maritime, Élevage porcin.

— *Hauts-Plateaux de l'Ouest :* Café, Maïs, Vivres frais.

3. **Aléatoire contrôlé :** Des variations de +/- 20% sont appliquées aux volumes pour éviter l'uniformité.

```
# Lancement du script dans l'environnement virtuel
python3 generer_donnees.py

# Sortie : G n ration du fichier CSV structur  avec s parateur point
   -virgule
# Fichier : donnees_agricoles_completes.csv (58 lignes, 16 colonnes)
```

Listing 3.1 – Exécution du script de génération

3.3 Étape 3 : Installation et Configuration Avancée du SGBD

Le déploiement a été effectué sur une distribution Linux **Ubuntu 24.04 LTS**. Cette version récente installe par défaut plusieurs versions de PostgreSQL, nécessitant une gestion fine des clusters.

3.3.1 Installation des paquets

Nous avons installé le moteur de base de données et son extension spatiale :

```
sudo apt update
# postgresql-contrib : fournit des extensions additionnelles
# postgis : transforme PostgreSQL en base de donn es spatiale
sudo apt install postgresql postgresql-contrib postgis
```

3.3.2 Résolution des conflits de versions (Ports)

Une étape critique a été l'identification du port d'écoute. Ubuntu 24.04 peut exécuter les versions 16 et 18 simultanément.

```
pg_lsclusters
# R sultat observ  :
# Ver Cluster Port Status Owner
# 16  main    5432 down   postgres
# 18  main    5433 online postgres  <-- Cluster actif cible
```

3.3.3 Initialisation de la Base Spatiale

Nous avons créé une base dédiée et activé l'extension PostGIS, ce qui ajoute les types de données géométriques (`GEOMETRY`, `GEOGRAPHY`) et plus de 1000 fonctions spatiales.

```
# Connexion au cluster sp cifique (Port 5433)
sudo -u postgres psql -p 5433
```

```sql
-- Cr ation du conteneur de donn es
CREATE DATABASE cameroun_db;

-- Connexion    la base
\c cameroun_db

-- Activation du moteur spatial (CRUCIAL)
CREATE EXTENSION postgis;

-- S curisation de l'utilisateur pour l'acc s API
ALTER USER postgres WITH PASSWORD '123456789';
```

3.4 Étape 4 : Processus ETL (Extract, Transform, Load)

Cette phase consiste à peupler la base de données avec nos deux sources hétérogènes.

3.4.1 Injection des vecteurs spatiaux (Shapefiles)

Nous avons utilisé `shp2pgsql`, l'outil natif de PostGIS, pour transformer les fichiers binaires `.shp` en SQL.

```
cd data/
# Param tres utilis s :
# -I : Cr er un Index Spatial (GiST) pour acc l rer les requ tes futures
# -s 4326 : Forcer le syst me de coordonn es WGS84
# departements : Nom de la table cible
shp2pgsql -I -s 4326 gadm41_CMR_2.shp departements | sudo -u postgres psql -p 5433 -d cameroun_db
```

Listing 3.2 – Importation avec indexation spatiale

3.4.2 Chargement des données tabulaires (CSV)

L'importation du fichier CSV a nécessité une élévation de privilèges. PostgreSQL, tournant sous l'utilisateur système `postgres`, n'a pas les droits de lecture dans les répertoires utilisateurs (`/home/user/`).

Contournement appliqué : Transit par le répertoire temporaire `/tmp`.

```
# Copie vers un r pertoire universellement accessible
cp donnees_agricoles_completes.csv /tmp/
# Attribution des droits de lecture universels (rw-rw-rw-)
chmod 777 /tmp/donnees_agricoles_completes.csv
```

Ensuite, nous avons structuré la table de réception et chargé les données :

```sql
-- Cr ation de la table attributaire
CREATE TABLE production_agri (
    id SERIAL PRIMARY KEY,
    nom_dept VARCHAR(100), -- Cl  de jointure future
    nom_region VARCHAR(100),
    agri_1 VARCHAR(100), vol_agri_1 INTEGER,
    agri_2 VARCHAR(100), vol_agri_2 INTEGER,
    agri_3 VARCHAR(100), vol_agri_3 INTEGER,
    elev_1 VARCHAR(100), vol_elev_1 INTEGER,
```

```
    -- ... (tronqué pour lisibilité)
    peche_type VARCHAR(100), vol_peche INTEGER
);

-- Importation en masse (Bulk Load)
\copy production_agri FROM '/tmp/donnees_agricoles_completes.csv'
   DELIMITER ';' CSV HEADER;
```

3.5 Étape 5 : Modélisation (Jointure Spatiale)

Plutôt que de modifier la table géographique `departements`, nous avons opté pour la création d'une **Vue SQL**. Cela permet de séparer la géométrie (qui change rarement) des données métiers (qui changent chaque année).

```
CREATE VIEW vue_globale AS
SELECT
    d.geom,             -- La forme géométrique (PostGIS)
    d.name_2 as nom,    -- Le nom normalisé
    p.*                 -- Les données de production associées
FROM departements d
JOIN production_agri p ON d.name_2 = p.nom_dept;
```

Listing 3.3 – Création de la vue applicative

Cette vue devient l'interface unique interrogée par notre API Backend.

Chapitre 4

Analyse Comparative : Alternative GeoServer

Le cahier des charges suggérait l'utilisation possible de GeoServer. Bien que nous ayons choisi une approche "Code-First" avec Python Flask, cette section détaille comment une architecture basée sur GeoServer aurait été implémentée et configure pour une mise en production industrielle.

4.1 Architecture théorique avec GeoServer

Dans ce scénario, GeoServer agit comme un *Middleware* standardisé OGC (Open Geospatial Consortium). Il remplace notre script Python `app.py`.

1. **Couche de Données :** Reste identique (PostgreSQL/PostGIS).
2. **Serveur Cartographique (GeoServer) :**
   — Se connecte à la base de données via JDBC.
   — Transforme les données spatiales en flux web standardisés (WMS pour les images, WFS pour les vecteurs).
3. **Client (Leaflet) :** Consomme directement les flux WFS.

4.2 Procédure de Configuration Détaillée

4.2.1 1. Connexion à la Source de Données (Data Store)

Dans l'interface d'administration de GeoServer (`http://localhost:8080/geoserver`) :
— Navigation vers : **Entrepôts → Ajouter un nouvel entrepôt**.
— Type : **PostGIS - PostGIS Database**.
— Paramètres de connexion :
  — `host: localhost`
  — `port: 5433` (Important : correspondre au cluster Ubuntu)
  — `database: cameroun_db`
  — `user: postgres`

4.2.2 2. Publication de la Couche (Layer Publishing)

Une fois la base connectée, GeoServer détecte la vue `vue_globale`.

— Action : **Publier**.
— Définition du SRS (Système de Référence Spatiale) : Forcer le **EPSG :4326** (WGS84) pour assurer la compatibilité avec Leaflet.
— Calcul des emprises (Bounding Boxes) : Clic sur "Calculer depuis les données".

4.2.3 3. Consommation WFS côté Client

L'intégration dans Leaflet change radicalement. Au lieu d'appeler notre API personnalisée, nous appelons l'URL standardisée WFS.

Listing 4.1 – Appel WFS GeoServer dans Leaflet

```
// URL WFS standard avec param tres de filtre
const geoServerUrl = "http://localhost:8080/geoserver/wfs";
const params = {
    service: 'WFS',
    version: '1.1.0',
    request: 'GetFeature',
    typeName: 'cameroun:vue_globale', // Nom de la couche publi e
    outputFormat: 'application/json', // Demande explicite de GeoJSON
    srsName: 'EPSG:4326'
};

// Construction de l'URL compl te
const url = geoServerUrl + L.Util.getParamString(params, geoServerUrl);

// R cup ration via Fetch API
fetch(url)
    .then(response => response.json())
    .then(data => {
        // Le traitement des donn es reste identique (onEachFeature, style
    });
```

4.3 Justification du choix final (Python Flask)

Nous n'avons pas retenu l'option GeoServer pour ce prototype pour deux raisons techniques majeures :

1. **Gestion des CORS :** Pour que le navigateur accepte les requêtes WFS locales, il faut modifier manuellement le fichier `web.xml` dans le conteneur Tomcat de GeoServer, une opération complexe et propice aux erreurs dans un temps imparti court. Avec Flask, une simple ligne `CORS(app)` suffit.
2. **Personnalisation du JSON :** Notre API Python nous permet de formater le JSON exactement comme nous le voulons (structure "Top 3 cultures"). GeoServer renvoie un JSON standard "plat" qui aurait nécessité plus de traitement JavaScript côté client pour reconstruire les hiérarchies de données.

Chapitre 5

Développement de l'Application Web

5.1 Backend (API Python)

Nous avons développé un script `app.py` utilisant Flask. Sa fonction principale est d'exécuter une requête SQL qui transforme directement les données en format **GeoJSON** grâce à PostGIS.

```
SELECT json_build_object(
    'type', 'FeatureCollection',
    'features', json_agg(json_build_object(
        'type', 'Feature',
        'geometry', ST_AsGeoJSON(geom)::json, -- Conversion
    g om trique
        'properties', json_build_object(
            'nom', nom,
            'vol_agri', vol_agri_1
            -- ... autres attributs
        )
    ))
) FROM vue_globale;
```

Listing 5.1 – Requête SQL intégrée dans Python

5.2 Frontend (Interface Leaflet)

L'interface `index.html` récupère ce GeoJSON via `fetch()` et applique :

- **Carte Choroplèthe :** Une fonction JavaScript colore les départements selon le volume de production (Vert foncé pour $> 20\,000$ tonnes).
- **Recherche :** Une barre de recherche avec autocomplétion qui zoome dynamiquement sur le département choisi via `map.fitBounds()`.
- **Filtres :** Un menu déroulant permettant de changer la couche affichée (Agriculture, Élevage ou Pêche) sans recharger la page.

Chapitre 6

Problèmes Résolus et Configuration Serveur

L'environnement Linux a nécessité des ajustements de sécurité spécifiques pour PostgreSQL.

6.1 Erreur d'authentification "Peer/Ident"

Par défaut, PostgreSQL refusait la connexion par mot de passe (*FATAL : password authentication failed*). Nous avons dû modifier le fichier de configuration `pg_hba.conf`.

Commande utilisée pour la correction automatique :

```
# Forcer la m thode 'trust' pour les connexions locales
sudo sed -i '/127.0.0.1\/32/s/scram-sha-256/trust/' /etc/postgresql/18/
   main/pg_hba.conf

# Red marrage du service
sudo systemctl restart postgresql
```

Chapitre 7

Conclusion

Le projet a abouti à une application fonctionnelle respectant toutes les exigences : visualisation multi-thématique, détails par bassins et interface de recherche. L'utilisation de PostGIS pour le traitement des données assure une robustesse et une évolutivité pour l'intégration future de données réelles.