Interface GUI Offres d'emploi

Contexte

Mohamed travaille sur le pipeline d'ingestion des offres d'emploi (`offre-ingestion`). Ce document définit l'interface entre son service et le GUI pour l'affichage des offres.

Analyse de la base SQLite Silver (offers.db)

Structure actuelle (13 tables)

```sql
-- Table principale
CREATE TABLE offers (
    id VARCHAR(50) PRIMARY KEY,
    intitule TEXT,
    description TEXT,
    dateCreation VARCHAR(50),
    dateActualisation VARCHAR(50),
    romeCode VARCHAR(10),
    romeLibelle VARCHAR(255),
    appellationlibelle VARCHAR(255),
    typeContrat VARCHAR(10),
    typeContratLibelle VARCHAR(100),
    natureContrat VARCHAR(100),
    experienceExige VARCHAR(10),
    experienceLibelle VARCHAR(100),
    dureeTravailLibelle TEXT,
    dureeTravailLibelleConverti VARCHAR(100),
    alternance VARCHAR(10),
    nombrePostes INTEGER,
    accessibleTH VARCHAR(10),
    qualificationCode VARCHAR(10),
    qualificationLibelle VARCHAR(100),
    codeNAF VARCHAR(20),
    secteurActivite VARCHAR(20),
    secteurActiviteLibelle VARCHAR(255),
    trancheEffectifEtab VARCHAR(100),
    offresManqueCandidats VARCHAR(10),
    entrepriseAdaptee VARCHAR(10),
    employeurHandiEngage VARCHAR(10)
);

-- Tables satellites (reliées par offer_id)
CREATE TABLE offers_lieu_travail (offer_id, libelle, latitude, longitude, codePostal, commun
CREATE TABLE offers_entreprise (offer_id, nom, entrepriseAdaptee);
CREATE TABLE offers_salaire (offer_id, libelle, commentaire, complement1, complement2);
```

```
CREATE TABLE offers_competences (offer_id, code, libelle, exigence);  -- E=Exigé, S=Souhaité
CREATE TABLE offers_qualites_professionnelles (offer_id, libelle, description);
CREATE TABLE offers_formations (offer_id, codeFormation, domaineLibelle, niveauLibelle, commentaire);
CREATE TABLE offers_permis (offer_id, libelle, exigence);
CREATE TABLE offers_langues (offer_id, libelle, exigence);
CREATE TABLE offers_contact (offer_id, nom, coordonnees1-3, courriel, telephone, urlRecruteur);
CREATE TABLE offers_origine (offer_id, origine, urlOrigine, partenaires);
CREATE TABLE offers_contexte_travail_horaires (offer_id, horaire);
CREATE TABLE offers_salaire_complements (offer_id, code, libelle);
```

Statistiques sample (150 offres)

Type de contrat	Nombre
CDI	61
CDD	34
Intérim	55

Options d'architecture

Option 1 : API REST exposée par `offre-ingestion` Recommandé

```
[offre-ingestion] → PostgreSQL (table offers) → API REST → [gui]
     Mohamed                                          GUI
```

Avantages : - Séparation des responsabilités : Mohamed gère l'ingestion + stockage, GUI consomme - Source unique de vérité : une seule base PostgreSQL partagée - Évolutif : Mohamed peut changer son pipeline sans casser l'UI - Standard industrie : architecture microservices classique

Responsabilités Mohamed : 1. Migrer les données SQLite → PostgreSQL (même schéma) 2. Exposer des endpoints REST : - `GET /offers` - liste paginée - `GET /offers/{id}` - détail d'une offre - `GET /offers?search=...&location=...&contract=...` - recherche filtrée

Responsabilités GUI : 1. Appeler l'API `offre-ingestion` 2. Afficher les résultats dans l'UI

Option 2 : Base partagée directe (moins propre)

```
[offre-ingestion] → PostgreSQL ← [gui]
     Mohamed          (shared)     GUI
```

Avantages : - Plus rapide à implémenter - Pas d'API à maintenir

Inconvénients : - Couplage fort entre services - Si Mohamed change le schéma, le code GUI casse - Pas de contrôle d'accès / rate limiting

Implémentation Django (modèle proxy read-only) :

```python
class JobOffer(models.Model):
    """Read-only model pointing to Mohamed's offers table."""
    class Meta:
        managed = False  # Django won't create/migrate this table
        db_table = 'offers'

    intitule = models.TextField()
    description = models.TextField()
    rome_code = models.CharField(max_length=10, db_column='romeCode')
    # ...
```

Champs nécessaires pour l'UI

Champ UI	Source table	Colonne
Titre	`offers`	`intitule`
Entreprise	`offers_entreprise`	`nom`
Lieu	`offers_lieu_travail`	`libelle`, `codePostal`
Type contrat	`offers`	`typeContratLibelle`
Salaire	`offers_salaire`	`libelle`
Compétences	`offers_competences`	`libelle`, `exigence`
Formation	`offers_formations`	`niveauLibelle`, `domaineLibelle`
Description	`offers`	`description`
Date publication	`offers`	`dateCreation`
Secteur	`offers`	`secteurActiviteLibelle`

Décision à prendre

Question pour Mohamed : Quelle option préfères-tu ?

1. **Option 1 (API)** : Tu exposes une API REST depuis `offre-ingestion`, je l'appelle depuis le GUI
2. **Option 2 (Base partagée)** : Tu écris dans PostgreSQL, je lis avec un modèle Django `managed=False`

Compromis pragmatique possible : - Court terme : Option 2 (base partagée) pour aller vite - Moyen terme : Migration vers Option 1 (API) pour découplage propre