**Projet de système centralisé de recommandation d’offres d’emploi**

Dans un marché du travail en pleine mutation, en particulier dans le secteur de la data et du numérique, trouver des opportunités professionnelles pertinentes est un défi de taille, tant pour les chercheurs d’emploi que pour les recruteurs. Les plateformes de recherche d’emploi traditionnelles dispersent souvent les offres d’emploi sur plusieurs sources, ce qui rend difficile pour les candidats de découvrir des opportunités qui correspondent à leurs compétences et à leurs objectifs de carrière.

Ce projet vise à résoudre ce problème en créant un système centralisé d’affichage des offres d’emploi et en mettant en place un système de recommandation intelligent. Cette approche simplifie non seulement le processus de recherche d’emploi pour les utilisateurs, mais améliore également le processus de correspondance entre les profils des candidats et les offres d’emploi disponibles.

# Objectif du projet

L’objectif de ce projet est de décrire toutes les étapes, de l’analyse des besoins à la mise en œuvre technique, en passant par la conception et l’architecture. L’objectif est de présenter une solution innovante et efficace qui établit de nouvelles normes pour la recherche d’emploi dans le domaine des données.

**Project Work Package 1.0 : centraliser automatiquement les informations sur les offres d’emploi**

**Tâche WP1.1 : Ingérer les données collectées dans un système de stockage Instructions détaillées pour la réalisation du projet**

Pour mener à bien ce projet, suivez ces instructions détaillées :

1. **Comprendre le problème :\*\***

Commencer par rechercher et comprendre les enjeux auxquels sont confrontés les chercheurs d’emploi et les recruteurs dans le secteur de la data et du numérique. Identifier les lacunes des plateformes de recherche d’emploi actuelles et l’impact des offres d’emploi fragmentées sur l’expérience utilisateur.

1. **Analyse des besoins** :

Effectuer une analyse approfondie des besoins afin d’identifier les principales exigences d’un système centralisé de recommandation d’offres d’emploi. Tenez compte de facteurs tels que les préférences des utilisateurs, l’adéquation des compétences et les tendances du secteur.

1. **Plan de mise en œuvre technique** :\*\*

Élaborer un plan détaillé pour la mise en œuvre technique du système centralisé. Cela devrait inclure la sélection de technologies, de méthodes de collecte de données et de solutions de stockage appropriées.

1. **Design et architecture** :

Créez une conception et une architecture complètes pour le système centralisé. Tenez compte de l’interface utilisateur, de la structure de la base de données et de l’intégration du système de recommandation intelligent.

1. **Ingestion de données (WP1.1):**

Concentrez-vous sur l’ingestion des données d’offre d’emploi collectées dans un système de stockage. Cela implique la mise en place de processus pour rassembler automatiquement les offres d’emploi provenant de diverses sources et les stocker dans un emplacement centralisé.

1. **Tests et optimisation** :

Mettre en œuvre des procédures de test pour assurer la fonctionnalité et l’efficacité du système centralisé. Optimiser les algorithmes de recommandation pour améliorer le processus de correspondance entre les profils de candidats et les offres d’emploi.

1. **Documentation et présentation** :

Préparer une documentation détaillée décrivant l’ensemble du projet, de l’analyse initiale à la mise en œuvre finale. Créez une présentation convaincante pour mettre en valeur la solution innovante et efficace développée.

1. **Gestion des délais** :

Gérer efficacement le calendrier du projet pour s’assurer que les livrables pour les WP1.0 et WP1.1 sont terminés avant la date limite spécifiée du dimanche 23 février 2025.

En suivant ces instructions détaillées, vous serez en mesure de contribuer au développement d’une solution de pointe qui révolutionne l’expérience de recherche d’emploi dans le secteur des données et du numérique. Bonne chance dans votre projet ! aide moi pour ce projet on va s'y etape par etape. tu me dis la premiere etape je fais je fini avant qu'on continu. je veux qu'on le fasse bien

Oui, tu peux tout à fait terminer d'abord ton application **backend** et **frontend** avant d'intégrer le scraping. Voici un ordre d'étapes que tu pourrais suivre pour assurer une bonne progression du projet :

### 1. **Finaliser l'Application Backend**

* **Modèles et API** : Assure-toi que les modèles (JobOffer, JobSeekerProfile, RecruiterProfile, etc.) sont bien définis et fonctionnent correctement dans ta base de données. Vérifie que les vues de ton API (JobOfferViewSet, JobSeekerProfileViewSet, etc.) sont opérationnelles et que tu peux récupérer, créer, et mettre à jour les offres d'emploi via des requêtes API.
* **Authentification et Permissions** : Mets en place l'authentification pour tes utilisateurs (chercheurs d'emploi et recruteurs). Tu peux utiliser JWT (JSON Web Tokens) pour gérer l'authentification via DRF (Django Rest Framework).
* **Tests Backend** : Assure-toi de tester toutes les fonctionnalités de l'API backend pour vérifier qu'elles répondent aux exigences fonctionnelles.

### 2. **Finaliser l'Application Frontend**

* **Conception UI/UX** : Crée l'interface utilisateur pour que les chercheurs d'emploi et recruteurs puissent s'inscrire, se connecter, consulter les offres d'emploi, publier des offres, etc.
* **Connexion avec l'API** : Utilise des appels API pour récupérer et afficher les offres d'emploi, et pour permettre aux utilisateurs d'interagir avec le système (postuler, sauvegarder des offres, etc.).
* **Tests Frontend** : Vérifie l'ergonomie et l'interactivité de l'application frontend. Assure-toi que les pages se chargent rapidement et que l'expérience utilisateur est fluide.

### 3. **Mettre en Place le Scraping (Après le Backend et le Frontend)**

Une fois que ton système de gestion des utilisateurs et des offres d'emploi fonctionne bien et est testé, tu peux intégrer le scraping. Tu auras alors une structure stable pour recevoir, enregistrer et afficher les offres d'emploi scrappées.

Le scraping servira à **compléter les offres existantes** et à centraliser les données provenant de plateformes externes. Tu peux configurer le scraping pour qu'il soit exécuté périodiquement ou manuellement via une commande Django, selon tes besoins.

* **Stockage des Offres** : Après avoir scrappé les offres d'emploi, tu pourras les enregistrer directement dans ta base de données (assure-toi de gérer les doublons).
* **Récupération des Offres dans l'API** : Les nouvelles offres ajoutées via le scraping pourront ensuite être récupérées par l'API pour les afficher sur le frontend.

### 4. **Tester l'ensemble du Système**

Une fois que le backend, le frontend et le scraping sont intégrés, tu pourras tester tout le système dans son ensemble :

* Vérifie que les utilisateurs peuvent interagir avec les offres d'emploi de manière fluide.
* Teste le processus de publication d'offres pour les recruteurs.
* Assure-toi que le scraping fonctionne correctement et que les offres d'emploi sont bien centralisées et accessibles via l'API.

### 5. **Déploiement**

Une fois que tout est terminé et testé :

* **Déploiement du backend** : Héberge ton application backend (par exemple sur Heroku, AWS, DigitalOcean).
* **Déploiement du frontend** : Si tu utilises un framework comme React ou Vue.js, tu peux déployer ton frontend sur un service comme Netlify ou Vercel.
* **Cron Jobs ou Celery pour le scraping** : Si tu utilises Celery pour planifier des tâches, configure-le sur ton serveur de production. Sinon, tu peux utiliser un cron job pour exécuter la commande de scraping à intervalles réguliers.

### Résumé des étapes :

1. **Backend :** Finalise les modèles, les vues, l'authentification et l'API.
2. **Frontend :** Conçois l'interface utilisateur et fais le lien avec l'API.
3. **Scraping :** Ajoute le scraping une fois que le backend et frontend sont prêts pour recevoir les données.
4. **Tests :** Assure-toi que tout fonctionne comme prévu.
5. **Déploiement :** Déploie le projet et mets en place des tâches planifiées pour le scraping.

Une fois que tu as fini avec le backend et le frontend, tu seras prêt à ajouter le scraping et à centraliser les offres d'emploi ! 😄

cd offre\_emploi

dir

D:\doc.fr icare\M1 DSP\Ecosysteme Data de lentreprise\tp\_offre emploi\offre\_emploi

cd "D:\doc.fr icare\M1 DSP\Ecosysteme Data de lentreprise\tp\_offre emploi"

core/tests.py

from rest\_framework.test import APIClient

from rest\_framework import status

class JobOfferTests(TestCase):

    def setUp(self):

        self.client = APIClient()

        self.job\_offer\_data = {

            "title": "Data Scientist",

            "company": "Tech Corp",

            "description": "A great opportunity",

            "location": "Paris",

            "salary": "45000"

        }

    def test\_create\_job\_offer(self):

        response = self.client.post('/api/joboffers/', self.job\_offer\_data, format='json')

        self.assertEqual(response.status\_code, status.HTTP\_201\_CREATED)

    def test\_get\_job\_offers(self):

        response = self.client.get('/api/joboffers/')

        self.assertEqual(response.status\_code, status.HTTP\_200\_OK)

# Tache à faire ce soir avec une connexion internet fluide.

Mettre en place des tests automatisés pour l’application

Les tests permettront de vérifier que les fonctionnalités de ton backend, comme les modèles, les API et les permissions, fonctionnent comme prévu

## Installer les dépendances

* S’assurer d'avoir installé pytest et pytest-django avec

pip install pytest pytest-django

## Configurer les tests dans pytest

* Ensuite, crée un fichier pytest.ini à la racine de ton projet pour configurer pytest

# pytest.ini

[pytest]

DJANGO\_SETTINGS\_MODULE = ton\_projet.settings

Cela permettra à pytest de savoir quel fichier de paramètres utiliser pour les tests.

## Écrire des tests pour tes modèles

Déjà fais avec les test models ; api et permission.

**Étapes pour commencer le frontend :**

1. **Choix de la technologie** :
   * Vue.js, React, ou un simple HTML/CSS avec JavaScript ?
   * Vu que tu veux un système dynamique, **React** serait un bon choix. Tu es OK avec ça ?
2. **Mise en place du projet** :
   * Création d'un projet React (npx create-react-app frontend ou npm create vite@latest frontend --template react).
   * Configuration du routing avec React Router.
   * Liaison avec l'API Django pour afficher les offres d'emploi.
3. **Développement des pages principales** :
   * Page d'accueil avec la liste des offres.
   * Détail d'une offre.
   * Formulaire de création d'offre (si tu veux gérer ça en frontend aussi).
   * Authentification pour les recruteurs (connexion et gestion des offres).

# Evolution front end

Très bien, voici un plan pour avancer dans ces différentes étapes. Commençons par les éléments essentiels et structurons cela :

**1. Ajouter une page de détails pour chaque offre :**

* On peut commencer par la page de détails de l'offre d'emploi. Cette page permettra d'afficher plus d'informations sur chaque offre lorsque l'utilisateur cliquera sur le bouton "Détails".

**Étapes à suivre :**

* Créer une vue job\_offer\_detail qui récupère l'offre d'emploi en fonction de son ID.
* Créer un template job\_offer\_detail.html qui affiche toutes les informations de l'offre (titre, description, lieu, contrat, recruteur, etc.).

**2. Intégrer des formulaires d'inscription et de connexion :**

* Une fois la page de détails en place, tu pourras passer à l'intégration des formulaires d'inscription et de connexion.

**Étapes à suivre :**

* Utiliser les vues génériques de Django pour l'inscription et la connexion (ex. UserCreationForm, AuthenticationForm).
* Créer les templates nécessaires pour afficher ces formulaires (register.html et login.html).
* Ajouter des redirections appropriées après connexion (par exemple, rediriger vers la page d'accueil ou le tableau de bord de l'utilisateur).

**3. Ajouter une interface pour les recruteurs et candidats :**

* Tu pourras créer des pages spécifiques pour les recruteurs et les candidats. Ces pages afficheront des informations spécifiques, comme la liste des candidatures pour les recruteurs et les offres postulées pour les candidats.

**Étapes à suivre :**

* Créer des modèles ou des vues spécifiques pour les recruteurs et les candidats.
* Créer des pages distinctes pour ces utilisateurs où ils pourront gérer leurs offres et candidatures.

**4. Ajouter un bouton "Postuler" :**

* Pour permettre aux candidats de postuler, tu peux ajouter un bouton "Postuler" sur la page de détails de l'offre. Ce bouton redirigera l'utilisateur vers un formulaire où il pourra soumettre sa candidature.

**Étapes à suivre :**

* Créer un modèle de candidature qui inclut des champs comme le nom, l'email, et éventuellement un CV ou message de motivation.
* Créer une vue et un template pour gérer l'envoi de la candidature.

**5. Créer une page d'inscription & connexion :**

* Une fois les formulaires d'inscription et de connexion en place, tu ajouteras les pages correspondantes.
* Assure-toi que l'utilisateur puisse s'inscrire, se connecter, et accéder à ses informations une fois connecté.

**Plan de démarrage :**

Commençons par la page de détails de l'offre. Voici comment procéder :

1. **Crée une vue pour afficher les détails d'une offre d'emploi :** Dans ton fichier views.py, ajoute une vue qui récupère l'offre d'emploi via son ID et affiche ses détails.
2. **Crée un template job\_offer\_detail.html pour afficher les informations :** Ce template doit afficher les informations de l'offre de manière détaillée, avec une mise en page attrayante.