TEST TECHNIQUE - AI SISTERS

Objectif

Le but de ce test est d'évaluer la capacité à implémenter une solution backend et frontend intégrant :

1. La génération d'un quiz basé sur deux LLMs (Ollama en local et OpenAI API).
2. Une pipeline RAG (Retrieval-Augmented Generation) exploitant des données issues de Wikipedia pour enrichir les prompts des modèles.
3. Une interface utilisateur permettant de saisir des paramètres pour générer le quiz.

Étapes à réaliser

1. Implémentation de la génération de quiz avec 2 LLMs

- Connectez **Ollama** (modèle local) et **l'API OpenAI** (GPT).
- Ajoutez une logique dans le backend pour générer un JSON contenant 5 questions liées à l'**Euro** (le championnat de football) de l'année sélectionnée par l'utilisateur.
  - Le JSON généré doit correspondre au format de `mockQuestions.json`.
- Les modèles doivent être appelés avec un **prompt structuré** pour générer ces questions.
- Gérer les erreurs en cas de réponse incorrecte des modèles ou d'indisponibilité de l'un des services.

2. Pipeline RAG pour enrichir le quiz

- Implémentez une pipeline RAG (Retrieval-Augmented Generation) pour récupérer des informations depuis **Wikipedia** sur l'année sélectionnée (par exemple, événements marquants de l'Euro cette année-là).
- Enrichissez les prompts envoyés aux LLMs avec les données récupérées pour des réponses plus précises et contextualisées.
  - Exemple de prompt enrichi :

```
{data_from_wikipedia}
Prompt here
```

- Implémentez un cache pour stocker les résultats RAG afin d'éviter des requêtes répétées (optionnel mais recommandé).

3. Modification du front-end

- Ajoutez un **champ d'entrée (input)** permettant à l'utilisateur de sélectionner une année pour le quiz.
- Ajoutez une option pour inclure ou non les données RAG dans la génération du quiz.

Critères d'évaluation

1. **Frontend/Backend**
    - Fonctionnalité pour sélectionner l’année et inclure/exclure RAG.
    - Fonctionnement correct de la génération de quiz avec les deux LLMs.
    - Intégration et utilisation de la pipeline RAG.
    - Gestion des erreurs et validation JSON Schema.
2. **Prompting**
    - Structure et clé de chaque partie du prompt.
    - Utilisation de variables pour inclure dynamiquement les données RAG et l'année sélectionnée par l'utilisateur.
3. **Code**
    - Propreté et lisibilité.
    - Structure modulaire et réutilisable.
4. **Optionnel**
    - Implémentation du cache pour les données RAG.
    - Faire un json schema pour l'output d'openai.