# TP - Retrieval Augmented Generation (RAG)

## Consignes générales

- Une bonne qualité de code est requise.
- Créer un repository **GitHub** pour le projet et m'ajouter en tant que code reviewer (**imad.enpc@gmail.com**).
- Groupes de 5 étudiants
- Tout le code source, y compris les fonctionnalités, doit être placé dans un répertoire `src/`.
- Le code doit être écrit en **Python (`.py`)** : **pas de notebooks** dans le rendu.
- Favoriser au maximum la **programmation orientée objet (POO)** en utilisant des classes.
- Fournir un fichier `cli.py` (hors du dossier `src/`) permettant d'exécuter le code depuis le terminal via une ligne de commande.
- Aucune variable ne doit être hardcodée : utiliser un **fichier de configuration** (hors du dossier `src/`).
- Inclure un fichier `requirements.txt` listant toutes les dépendances (hors du dossier `src/`).
- Utiliser le framework **LangChain (** https://python.langchain.com/docs/tutorials/rag/ **)** pour implémenter les fonctionnalités lorsque cela est pertinent.
- **Rédiger un rapport** résumant votre travail, justifiant les choix techniques effectués et expliquant les hypothèses prises lors de la conception du système RAG.

---

## Questions

### Q1 : Mise en place d'un système d'indexation des documents

Créer une classe pour l'indexation des documents en respectant les consignes suivantes :

1. **Choix des technologies** :
    - Utiliser un **vector store** (ex. **ChromaDB**).
    - Sélectionner un **modèle d'embeddings** de votre choix via **Hugging Face**.
2. **Pipeline d'indexation** :
    - **Loading** : Charger les documents avec un **data loader**.

    - **Splitting** : Diviser les documents en **petits chunks**.
        - **Idéalement**, les métadonnées doivent être conservées et stockées.
        - Privilégier un **découpage optimisé pour le format Markdown**.
    - **Embedding** : Calculer l'**embedding** de chaque chunk en utilisant le modèle sélectionné.
    - **Storage** : Stocker les embeddings dans un **vector store**.

---

## Q2 : Recherche documentaire dans la base vectorielle

- Sélectionner un ensemble de **3 ou 4 fichiers PDF** (les documents doivent avoir un sujet commun, ex: texte de loi, article de recherche IA ) de votre choix et les placer dans un répertoire `data/` du repo.
- Appliquer le système d'indexation à ces documents
- Développer une fonction permettant d'**interroger la base vectorielle** à partir d'une requête utilisateur (**query**).
- Cette fonction doit renvoyer :
    - Une **liste des documents** les plus pertinents en réponse à la requête.
    - Les **scores d'affinité** associés à chaque document.
- Tester la fonction avec plusieurs requêtes pertinentes dont les réponses sont contenues dans les PDF fournis.

---

## Q3 : Système de question-réponse basé sur un LLM

- Mettre en place un **système de question-réponse** en utilisant un **Large Language Model (LLM)** open-source de votre choix.
- Exploiter les **paragraphes récupérés de la base vectorielle** pour synthétiser les informations et formuler une réponse adéquate.
- Créer un **template de prompt** contenant tous les éléments nécessaires pour optimiser la réponse du LLM en utilisant le **contexte** extrait de la base de connaissances.

---

## Q4 : Évaluation du système RAG/LLM

- Mettre en place un mécanisme permettant **d'évaluer la pertinence** des réponses générées par le système RAG/LLM.

---

## Q5 ( bonus ) : Construction de chatbot

- Mettre en place un chatbot sur la base du système Question/Réponse que vous avez construit ( la complexité réside and l'inclusion de l’historique de la discussion dans le prompt )