PDS

**Phase 1 : Analyse et Conception**

**1. Analyse des besoins**

L’analyse des besoins a été réalisée en collaboration avec l’université pour comprendre les exigences spécifiques, notamment :

* Identification des **questions fréquentes** posées par les étudiants et le personnel.
* Collecte des **documents à inclure** dans la base de connaissances :
  + Règlements universitaires.
  + Programmes de formation.
  + Documents officiels et politiques internes.
  + Fichiers PDF, JSON, et texte.

**2. Choix de la stack technologique**

**Technologies retenues :**

1. **Modèle de langage** :
   * **Choix : Modèle de langage open-source** (e.g., **Llama** ou **mxbai-embed-large**).
   * **Justification** :
     + Adaptabilité aux données locales.
     + Réduction des coûts liés aux API tierces (comme OpenAI).
     + Transparence et flexibilité dans l'entraînement des modèles si nécessaire.
2. **Stockage des données** :
   * **Format des données** :
     + Texte brut (chunks).
     + Embeddings pour les comparaisons vectorielles.
   * **Base de données** :
     + Vector Store (type **FAISS** ou **Chroma**) pour stocker et rechercher des embeddings efficacement.
3. **Traitement des requêtes** :
   * Calcul de similarité : **torch.cosine\_similarity** pour comparer les embeddings.
   * LLM (Large Language Model) pour générer une réponse enrichie après recherche.
4. **Architecture modulaire** :
   * Les composants sont découplés pour garantir une maintenance facile et une évolutivité du système.

**3. Conception de l'architecture**

L’architecture est conçue pour effectuer une **recherche sémantique** en utilisant des embeddings de texte et un modèle de langage pour générer des réponses.

**Description de l'architecture**

* **Entrée** :  
  L'utilisateur pose une question en langage naturel.
* **Prétraitement des données** :
  + Les documents (PDF, JSON, texte) sont divisés en **chunks de texte**.
  + Ces chunks sont stockés dans une base de données **textuelle** et convertis en embeddings pour alimenter un **vector store**.
* **Traitement de la requête** :
  + La question est transformée en **embedding** via un modèle de langage.
  + Une **recherche de similarité** est effectuée entre l'embedding de la question et les embeddings du vector store.
* **Raffinement de la réponse** :
  + Les résultats les plus pertinents sont classés (ranking).
  + Un **LLM** (modèle de langage) est utilisé pour reformuler la réponse de manière naturelle.
* **Sortie** :  
  L’utilisateur reçoit une réponse claire et précise.

**4. Diagramme de l'architecture**

Le diagramme suivant illustre l'architecture conçue :

**5. Diagrammes UML**

**a) Diagramme de cas d'utilisation**

**Acteurs :**

1. Utilisateur (étudiant/personnel universitaire)
2. Système (chatbot)

**Cas d'utilisation principaux :**

* Poser une question.
* Rechercher des informations dans la base de connaissances.
* Obtenir une réponse formulée par le chatbot.

**b) Diagramme de séquence**

**Étapes du processus** :

1. L’utilisateur pose une question.
2. La question est transformée en **embedding**.
3. Le système effectue une **recherche de similarité** dans la base d’embeddings.
4. Les résultats sont classés.
5. Le LLM formule une réponse.
6. Le système retourne la réponse à l’utilisateur.

**c) Diagramme de classes**

* **Classes principales** :
  + DocumentProcessor : Responsable du traitement des documents.
  + Embedder : Génère les embeddings des textes et des requêtes.
  + SimilaritySearch : Effectue la recherche de similarité.
  + LLM : Gère la génération de réponses.
  + Database : Stocke les chunks de texte et les embeddings.

**Conclusion**

La phase d’analyse et de conception a permis de définir une **architecture modulaire**, d’identifier les technologies adaptées, et de préparer les **diagrammes UML** pour une compréhension claire du système. Le système conçu repose sur des principes modernes de **recherche sémantique** et d’intelligence artificielle pour répondre efficacement aux besoins de l’université.