ProBTP

- L’architecture globale de la solution que vous allez mettre en place

1. Utilisateur → Question posée via UI → API Backend (Administrateur aussi ?)

2. API Backend → Recherche dans la base vectorielle (retrieval)

3. Base vectorielle → Fournit le contexte au LLM (chunks)

4. LLM local → Génération de la réponse enrichie

5. Réponse → Retour à l’utilisateur via UI

Frontend :

- Framework : Angular (il y a une librairie en react qui est pas mal pour les chatbot “react-chatbot-kit”)
- Communication avec le backend :(Axios est plus adaptée ?)

Backend API :

- Langage : Python avec FastAPI.
- Package : Langchain pour le RAG.

Base vectorielle :

- **Langage** : Python.
- **Packages et librairies** : LangChain pour organiser les chunks dans le format requis pour le LLM, `nltk` ou `spaCy` pour nettoyer et découper les données en chunks adaptés (par exemple, 500 mots).

LLM :

- LLM local sur les serveurs de l’université.
- LongChain pour la gestion des prompt et intégration avec le model

Flux d'interaction :

Utilisateur → Interface Utilisateur : (Question)

Interface Utilisateur → API Backend : (La question est transmise au backend via une API REST)

API Backend → Base Vectorielle : (Le backend envoie la requête à la base vectorielle pour rechercher les chunks les plus pertinents en fonction de la question utilisateur.)

Base Vectorielle → API Backend : (La base vectorielle renvoie les chunks de contexte (exemple : un extrait de documentation sur les tableaux COBOL)

API Backend → Modèle LLM Local : (Le backend transmet la question utilisateur enrichie par les chunks contextuels au LLM local.)

Modèle LLM Local → API Backend : (Le LLM génère une réponse enrichie)

API Backend → Interface Utilisateur : (La réponse est transmise à l’interface utilisateur pour être affichée.)

Technologie :

Interface Utilisateur (UI)

- Framework web: React
- Communication backend : **API REST**

Backend/API :

- Language Python framework FastAPI

Base Donnees ;

- BD ; MongoDB/Pinecome (je sais que c'est pas la plus adapter il ya FAISS ou même Pinecone pour le cloud mais avec le temps reduit on perdra du temps a se familiariser avec)
- Pipeline : Extraction des donnes en texte Brut, Nettoyer (PYPDF)
- Base Vectorielle : Langchain (embedding ??? , Chunk ???)
  chunks : par nombre de token / par paragraphe
  embedding : Dense Retriever de LangChain recherche par similarité (distance euclidienne)

- Les contraintes liées aux sources de données (format, structuration, …) pour améliorer l'efficacité de la solution
- Les étapes en terme de réalisation (high level)

Sprint 1 : 23 janvier - 31 Janvier
Sprint 2 : 6 février - 14 février
Sprint 3 : 20 février - 28 février
Sprint 4 : 6 mars - 14 mars
Sprint 5 : 19 mars - 24 mars

Sprint 1 : Initialisation et préparation

Dates : 23 janvier - 31 Janvier

- **Objectifs :**
    - Préparer l'environnement de travail.
    - Configurer les outils nécessaires au projet.

- Installer et configurer l'environnement backend (Python, FastAPI, LangChain).
- Configurer les bases de données (MongoDB/Pinecome) et créer une structure de base pour les collections/tables.
- Créer un dépôt GitHub

Sprint 2 : Ingestion et structuration des données

Dates : 6 février - 14 février

- **Objectifs :**
    - Créer des pipelines ETL pour extraire les données de différents formats.
    - Diviser les données en chunks adaptés à l'indexation vectorielle.

- Développer un script Python pour extraire et nettoyer les données de fichiers PDF.
- Implémenter une méthode de segmentation des données en chunks à partir des fichiers extraits.
- Développer un script pour extraire les données des fichiers HTML et PPT.
- Convertir les chunks en embeddings à l'aide d'un modèle Intégrer le pipeline de traitement des données (PDF, HTML, PPT) dans une architecture cohérente.
- Configurer et déployer une base vectorielle
- Vérifier que l'indexation des embeddings fonctionne correctement.

Sprint 3 : Recherche et intégration du LLM

Dates : 20 février - 28 Fevrier

- **Objectifs :**
    - Mettre en place la base vectorielle.
    - Intégrer le LLM local pour la génération de réponses.
- Développer un module backend pour effectuer des recherches dans la base vectorielle (retrieval).
- Intégrer un modèle LLM local dans le backend via LangChain ou une API directe.
- Intégrer le concept de mémoire pour LLM
- Tester les réponses générées par le LLM avec les données récupérées depuis la base vectorielle.
- Début de la création du Front End

Sprint 4 : Gestion avancée et intégration front end

Dates : 6 mars - 14 mars

- **Objectifs :**
    - Développer une interface utilisateur simple.
    - Ajouter des fonctionnalités avancées : mémoire des conversations, gestion des tokens,
- Intégrer la gestion de l'historique des conversations dans l'UI.
- Ajouter un module backend pour la gestion des tokens afin de limiter les coûts d'appel au LLM.
- Connecter l'interface utilisateur au backend pour gérer les requêtes utilisateurs et afficher les réponses.
- Effectuer des tests utilisateur pour valider l'ergonomie et le fonctionnement de l'application.
- Ajout de certaine fonctionnalité sup (tel que vider la mémoire, suppression de conversation)

Sprint 5 : Finalisation et tests

Dates : 19 mars - 24 mars

- **Objectifs :**
    - Tester l'ensemble du système (fonctionnalités, performance, robustesse).
    - Finaliser la documentation et préparer la démonstration.

Version V0 : Prototype fonctionnel basique

Objectif principal :

Créer une structure de base avec des fonctionnalités minimales pour tester le flux utilisateur et administrateur.

Fonctionnalités principales :

1. **Authentification :**
    - Système de connexion/déconnexion avec rôle utilisateur (user) et administrateur (admin).
    - Génération et validation de JWT.
2. **Chatbot simple :**
    - Interface utilisateur pour saisir une question et recevoir une réponse statique ou prédéfinie.
3. **Gestion des utilisateurs (Admin) :**
    - Liste des utilisateurs affichée dans le tableau de bord administrateur.
    - Suppression des utilisateurs.
4. **Structure de backend basique :**
    - Gestion des endpoints REST pour l'authentification et le chat.
    - MongoDB utilisé pour stocker les utilisateurs et les sessions.
5. **Frontend :**
    - Interface utilisateur simple avec React :
        - Page de login.
        - Page de chat pour les utilisateurs.
        - Tableau de bord minimal pour les administrateurs.

Tests :

- Tests unitaires pour les endpoints d'authentification.
- Simulation des appels API avec Postman.

Version V1 : Fonctionnalités avancées de chatbot et gestion dynamique des documents

Objectif principal :

Ajouter des fonctionnalités de base pour le RAG (Retrieval-Augmented Generation) et enrichir la gestion des documents et des sessions.

Fonctionnalités principales :

1. **Préparation des documents :**
    - Les administrateurs peuvent uploader des fichiers PDF.
    - Extraction automatique du texte des fichiers PDF avec PyPDF2 ou pdfminer.
2. **Découpage et stockage des chunks :**
    - Découper les documents en chunks de 500 mots.
    - Stocker ces chunks dans MongoDB.
3. **Recherche contextuelle basique :**
    - Implémenter une recherche naive pour retourner les chunks pertinents (sans base vectorielle).
4. **Gestion des sessions utilisateur :**
    - Création et suppression des sessions utilisateur.
    - Historique des messages stocké dans MongoDB.
5. **Améliorations du frontend :**
    - Interface utilisateur plus intuitive pour les administrateurs :
        - Uploader des documents.
        - Voir les documents et les permissions associées.
    - Interface utilisateur utilisateur avec historique de session.
6. **Sécurité améliorée :**
    - Middleware pour restreindre l'accès aux routes protégées (authentification par rôle).

Tests :

- Tests d'intégration pour la gestion des documents et le découpage.
- Tests des flux utilisateurs et administrateurs.

Version V2 : Fonctionnalités avancées avec recherche vectorielle et personnalisation

Objectif principal :

Intégrer une base vectorielle pour le RAG et permettre une gestion avancée des permissions et des recherches personnalisées.

Fonctionnalités principales :

1. **Recherche vectorielle :**
    - Génération des embeddings des chunks avec OpenAI, Hugging Face, ou une autre bibliothèque NLP.
    - Stockage des embeddings dans une base vectorielle comme Pinecone, Weaviate, ou Milvus.
    - Recherche contextuelle précise en utilisant la similarité cosine.
2. **Gestion des permissions par document :**
    - Les administrateurs peuvent assigner ou révoquer l'accès aux documents pour des utilisateurs spécifiques.
3. **Personnalisation des sessions :**
    - Chaque utilisateur peut choisir les documents ou domaines à inclure dans la recherche pour une session donnée.
4. **Interface utilisateur enrichie :**
    - Tableau de bord utilisateur :
        - Sélection des domaines pour une session.
        - Visualisation des documents disponibles.
    - Tableau de bord administrateur :
        - Gestion avancée des utilisateurs et documents.
        - Analyse des logs d'activité.
5. **Statistiques et logs pour les administrateurs :**
    - Voir les statistiques d'utilisation des sessions et des documents.
    - Historique des modifications (ajout/suppression de documents).
6. **Déploiement complet :**
    - Conteneurisation avec Docker pour le frontend, le backend et MongoDB.
    - CI/CD avec GitHub Actions ou GitLab CI.

Tests :

- Tests E2E (End-to-End) avec Cypress.
- Vérification des performances pour la recherche vectorielle.
- Simulation des scénarios utilisateur pour garantir la stabilité.

Contraintes

1. Contraintes liées aux données :

- **Format des données** : Les données doivent être en texte brut, ce qui nécessite une extraction depuis des formats variés comme PDF. Utilisation de librairies comme PyPDF2 ou pdfminer.
- **Nettoyage des données** : Les données extraites doivent être nettoyées pour éviter le bruit dans les résultats, par exemple en supprimant les caractères inutiles ou en corrigeant les erreurs typographiques.
- **Structuration des données** : Les documents doivent être découpés en chunks adaptés (par exemple, 500 mots), optimisés pour le LLM.
- **Base vectorielle** : Nécessité de choisir une solution comme Pinecone ou FAISS pour gérer efficacement les embeddings, tout en tenant compte des délais d'apprentissage.

---

2. Contraintes liées à la technologie :

- **Frameworks Frontend** : Angular est prévu, mais React est aussi évoqué avec des bibliothèques spécialisées comme `react-chatbot-kit`.
- **Backend** : Langage Python avec FastAPI, mais des performances doivent être garanties pour les requêtes utilisateur.
- **Communication API** : L'utilisation de Fetch API ou Axios doit être clarifiée en fonction des besoins du projet.
- **LLM local** : Nécessité de déployer et optimiser le modèle sur une infrastructure universitaire, avec gestion des prompts via LangChain.

---

3. Contraintes fonctionnelles :

- **Authentification** : Implémenter un système robuste de gestion des rôles (utilisateur, administrateur) avec JWT pour sécuriser les endpoints.
- **Gestion des sessions** : Assurer un historique précis des questions/réponses des utilisateurs, tout en optimisant l'accès à MongoDB.
- **Sécurité des données** : Garantir la confidentialité des informations, notamment via des permissions pour l'accès aux documents.

---

4. Contraintes organisationnelles :

- **Temps limité** : Le projet doit être réalisé en plusieurs versions (V0, V1, V2), chaque version augmentant les fonctionnalités.
- **Réunions et communication** : Synchronisation avec les parties prenantes comme Sabrina Drouet et PRO BTP pour des retours réguliers.
- **Livraison intermédiaire** : Présentation de l'architecture prévue lors de la semaine du 20 janvier.

---

5. Contraintes de déploiement :

- **Infrastructure locale** : Hébergement du LLM sur les serveurs de l’université, avec des outils de conteneurisation comme Docker pour simplifier le déploiement.
- **Gestion des ressources** : Limiter la consommation mémoire et assurer une exécution fluide du backend et des recherches vectorielles.

dependances, environments, surcharge LLM (spacy),