Ayant pour but de construire un chatbot (agent conversationnel), nous avons développé une application qui propose un service fiable et qualitatif, dans laquelle il est possible de dialoguer avec un LLM (Grand modèle de langage) génératif, de sauvegarder et visualiser l'historique de discussions ainsi que de supprimer l'historique si souhaité.

Cette application est développée sur la base du framework de *Uvicorn* et *FastAPI* avec également la sollicitation au modèle Llama3.2-3b via la plateforme Ollama et à la base de données Apache Solr. Ce travail a été commencé par la mise en œuvre d'un simple fichier de HTML (index.html) en tant que page d'accueil (la racine). Dans cette interface, il se retrouve une fenêtre de dialogue dans laquelle l'interaction humain-IA aura lieu et un bouton qui dirige l'utilisateur vers la page de l'historique. En parallèle, un fichier de python est créé. Jouant le rôle du point de départ de l'application, il contient le code pour toutes les fonctionnalités nécessaires, la logique métier et il assure la responsabilité d'envoyer dynamiquement les données ou le résultat d'une requête lancée de la part d'utilisateur à un template (page HTML) destinataire.

Concrètement, ce fichier python spécifie d'abord l'adresse de la base de données Solr pour le sauvegarde de l'historique et démarrer les deux applications nécessaires (Solr et Ollama). Ensuite, il explicite le chemin des templates (pages HTML), le chemin des fichiers statiques (fichiers CSS et JavaScript). Puis, il y a des fonctions de python qui servent à normaliser le texte, ajouter les conversations dans la base de données, récupérer l'historique et la réinitialiser. Enfin, il définit les fonctions de FastAPI telles que l'appel aux pages de conversation, la requête au modèle Llama3.2 et l'envoi de sa réponse à l'interface, le switching entre la page de conversation et celle de l'historique ainsi que la suppression de l'historique.

Grâce à FastAPI, il est possible de simplifier d'autres templates (pages HTML) en faisant l'extension de la page d'accueil ou en faisant des boucles pour par exemple l'affichage d'une séquence de données. Une fois que l'utilisateur envoie son premier message à l'IA, l'application passe discrètement de la page d'accueil à la page de conversations (chat.html), qui a exactement la même mise en page que le précédent mais avec le dialogue courant

affiché dans la fenêtre de conversations. Cette page de conversations est en réalité l'extension de la page d'accueil avec simplement un ajout du dialogue. Au moment où l'utilisateur souhaite visualiser l'historique de conversations, il clique sur le bouton destiné et la page sera dirigée de "chat.html" à "chatlogs.html". Ici, les données sont récupérées dynamiquement depuis la base de données via le connecteur du fichier python et sont représentées par une boucle.

Les trois templates partagent le même fichier de style CSS. Un script JavaScript dédiée à envoyer les messages d'utilisateurs à l'IA et à maintenir les messages les plus récents dans la fenêtre de conversations actuelle est utilisé sur les templates "index.html" et "chat.html".

Difficultés rencontrées :

Lors de la phase de déploiement sur Docker, l'intégration d'applications tierces telles que Ollama et Apache Solr a complexifié le processus, car ces outils devaient être pleinement accessibles au sein du conteneur pour garantir le bon fonctionnement du chatbot. Pour surmonter cet obstacle, nous avons adapté le Dockerfile en y ajoutant des instructions spécifiques via la commande RUN, permettant l'installation directe de ces deux composants. De plus, deux volumes ont été créés pour assurer leur disponibilité au sein de l'environnement Docker. Cependant, une fois cette étape franchie, un nouveau problème est apparu : le lancement automatique de l'application, initialement géré par le script app.py via open_app.py, ne s'exécutait plus correctement. Pour résoudre cette situation, nous avons développé un script bash (entrypoint.sh) qui agit comme point d'entrée. Ce script garantit le démarrage automatique des deux composants essentiels, rendant ainsi le chatbot opérationnel dès le lancement du conteneur.

Fonctionnalités supplémentaires :

Dans le but d'améliorer l'accessibilité, une fonctionnalité audio pourrait être implémentée pour les personnes malvoyantes, permettant de transcrire leurs requêtes vocales en texte et de lire à voix haute les réponses générées par le chatbot. Par ailleurs, une autre amélioration consisterait à enrichir les interactions en autorisant les utilisateurs à intégrer des fichiers variés (PDF, images, etc.) à leurs requêtes, offrant ainsi une plus grande flexibilité dans les

échanges avec le modèle. De plus, une recherche contextuelle dans l'historique pourrait être ajoutée à la page chatlogs.html, autorisant les utilisateurs à retrouver rapidement des conversations ou informations spécifiques via une barre de recherche basée sur des mots-clés. Enfin, une fonctionnalité de personnalisation permettrait aux utilisateurs d'adapter l'interface (choix de thèmes clair/sombre, ajustement de la taille de police) et de modifier le style des réponses du chatbot (ton formel, amical ou concis), répondant ainsi à leurs préférences individuelles.