Rapport Technique : Mise en place d'un ChatBot soutenu par un RAG, proposant des événements adaptés aux préférences de ses utilisateurs.

Sommaire

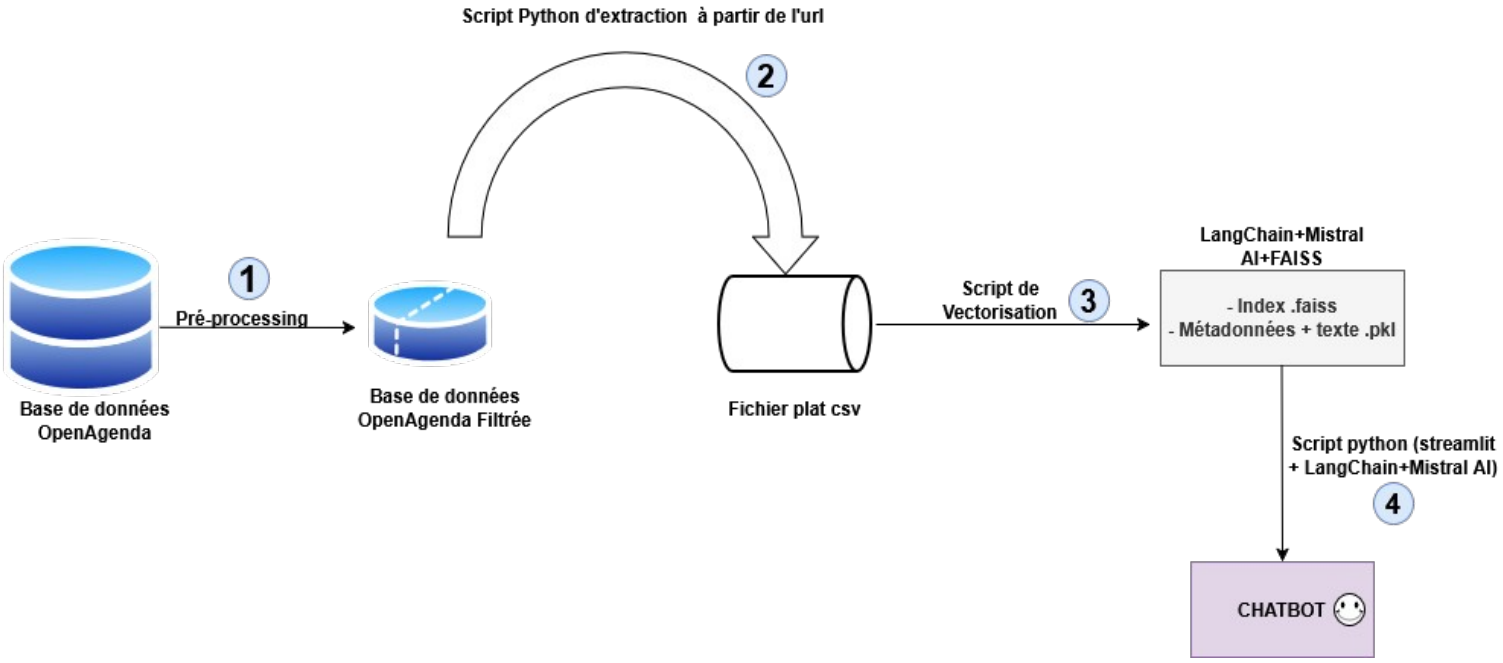
I- INTRODUCTION

Puls-Events est une société innovante dans le domaine de la gestion d'événements culturels. L'entreprise propose une plateforme web permettant aux utilisateurs de découvrir et de suivre des événements culturels en temps réel. Elle collecte des informations à partir de diverses sources, comme Open Agenda, pour proposer des événements adaptés aux préférences de ses utilisateurs, filtrables par lieu et par période.

En tant que data engineer au service de Puls-Events, j'ai pour mission de mettre sur pied un chatbot qui en connexion avec une base de données contenant les informations (provenant de OpenAgenda) sur des évènements de Montpellier qui ont lieu en 2025, fera des recommandations aux utilisateurs. Notons que le chatbot pourra aussi répondre à des questions portant sur d'autres problématiques, ceci en fonction du modèle LLM ou SLM qui sera utiliser. A fin de mener à bien la mission, nous avons :

- Mis sur pied un pipeline d'extraction, dont les données sont les évènements de 2025 de la ville de Montpellier.
- Préparer les données textuelles à indexer
- vectoriser des données textuelles et sauvegarder les résultats dans la base de données vectorielle FAISS.
- Mis sur pied le chatbot qui fera des recommandations aux utilisateurs.

II- ARCHITECTURE DU SYSTEME



A. **Pre-processing & Extraction.**

L'interface du site OpenAgenda offre la possibilité de faire un filtrage par date, région, département et ville sur données. Ainsi après filtrage géographique (ville=Montpellier) et temporel (date de début= 01 janvier 2025) sur les données des évènements sauvegardées sur OpenAgenda, à partir de la console d'API, nous avons accès à l'url pour extraction des données filtrées. Comme technique d'extraction, nous écrivons un script python qui utilise cet url.

Résultat : un fichier csv constituer de 268 évènements distincts, où chaque évènement est caractérisé par : son identifiant (**iud**), son titre (**title_fr**), sa description (**description_fr**), sa ville (**location_city=Montpellier**) son lieu (**location_address**) sa date (**daterange_fr**), son lien pour information approfondie (**canonicalurl**), etc……

Remarque : Des tests unitaires (pytest) ont validé la fiabilité du pipeline d'extraction.

B. Vectorisation

Après extraction des données, nous utilisons LangChain+Mistral AI + FAISS dans un script python pour la vectorisation. Premièrement, on prépare les données textuelles qui seront indexées. Dans notre cas, elles correspondent à la concaténation des champs : title_fr, description_fr, location_address, daterange_fr et canonicalurl (**Ici LangChain**

intervient via Document (module importé depuis langchain_core.documents) : chaque texte est transformé en objet structuré avec du contenu (page_content) et des métadonnées= fichier extrait). Par la suite l'on utilise LangChain et la clé d'API du modèle d'embedding **mistral-embed** de Mistral AI pour transformer les données textuelles en vecteurs mathématiques utilisables par FAISS. Notons que nous n'avons pas procédé au chunk des données textuelles car celles-ci ne sont pas de grande taille. Enfin les vecteurs sont sauvegardés dans la bibliothèque FAISS et les documents et métadonnées sont sauvegardés en fichier pickle pour une utilisation future.

Remarque : La clé d'API de Mistral AI est sauvegarder dans un fichier .evn pour éviter de la rendre public lorsque le système **RAG** est versionné sur Github.

C. ChatBot

Ici, nous utilisons la bibliothèque streamlit pour la création de l'interface d'échange entre les utilisateurs et le système RAG+LLM. Le modèle de LLM utiliser est le modèle Large le plus réccent du framework Mistral AI ("mistral-large-latest").

Remarque : Nous avons également mis sur pied un query-classifier qui permet à notre chatbot de répondre aussi aux questions qui font références aux données sur lesquelles le modèle LLM utilisé a été entraîné. En effet lorsqu'une requête est émise par un utilisateur le système vérifie d'abord si on aura besoin du système RAG pour générer la réponse ou pas, et lorsque ce n'est pas le cas la mémoire statistique du LLM est utilisée pour répondre à la requête.

IV- RECOMMANDATIONS POUR LA VERSION FINALE

Pour la version finale, il est recommandé :

- Evaluez et améliorez votre système RAG en utilisant Ragas.

- Intégrez un mécanisme de feedback dans Streamlit.

- D'optimiser les coûts en utilisant un SLM.

- Automatiser le système Récupération de données + RAG + LLM + Chatbot.

V- Conclusion

Le POC démontre la faisabilité technique du projet, avec une architecture claire, scalable et performante. Les améliorations proposées permettront une version robuste et industrialisée.