

# Concevez et déployez un système RAG

Développez un assistant pour la recommandation d'évènements culturels

François Hellebuyck

OPENCLASSROOMS



# Contexte et Objectifs du POC RAG



## Le Client : Puls-Events

Une entreprise technologique spécialisée dans la recommandation culturelle. Elle vise à intégrer un chatbot RAG pour répondre aux questions des utilisateurs sur les événements à venir, basé sur Open Agenda.



## Votre Rôle : Data Scientist POC

Concevoir et livrer un POC RAG fonctionnel complet : de la collecte des données (Open Agenda) à l'API (FastAPI), en passant par l'indexation (Faiss) et la génération (Mistral).



## Les Contraintes & Outils

Stack technique imposée (LangChain, Mistral, Faiss), ciblage des données récentes (< 1 an) et livrable conteneurisé (Docker) prêt pour une démo live pour les équipes Produit et Marketing.

# **Fonctionnement du système**

# Le Pipeline d'Embeddings



## 1. Collecte et Nettoyage des Données

Les données des événements culturels sont extraites de l'API OpenAgenda, en ciblant une région spécifique comme l'Occitanie.

Elles sont ensuite stockées dans une base MongoDB, puis nettoyées pour supprimer les doublons et les événements sans description pertinente.



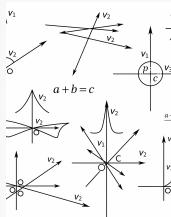
## 2. Préparation et Découpage (Chunking)

Les données des événements nettoyés sont formatées en un texte structuré, puis découpées en petits morceaux qui se chevauchent.

Chaque morceau conserve les informations essentielles de l'événement pour préserver le contexte sémantique. Le framework

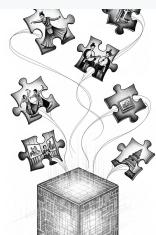


Langchain est utilisé pour réaliser cette étape et les suivantes.



## 3. Génération des Embeddings (Vectorisation)

Chaque morceau de texte est transformé en un vecteur numérique de 1024 dimensions à l'aide du modèle **multilingual-e5-large**, exécuté en local. Ce vecteur représente l'empreinte sémantique unique du texte, le rendant compréhensible pour la machine.



## 4. Crédit de la Base Vectorielle

Tous les vecteurs sont stockés dans un index FAISS, une base de données optimisée pour la recherche par similarité à très grande vitesse.

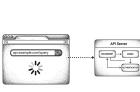
Cet index est ensuite sauvegardé sur le disque, prêt à être interrogé par l'API pour trouver les événements les plus pertinents.

# API d'intérogation



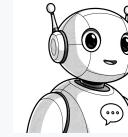
## Architecture et Framework

- Framework : FastAPI**, un framework web Python moderne et performant.
- Serveur : Uvicorn**, un serveur ASGI (Asynchronous Server Gateway Interface) rapide.
- Validation des données** : Les modèles **Pydantic** sont utilisés pour valider automatiquement les données des requêtes et des réponses, garantissant ainsi leur intégrité.
- Documentation automatique** : Une interface Swagger UI est disponible à l'adresse `/docs` pour explorer et tester les endpoints de manière interactive.



## Endpoints principaux

- POST /ask** : Pose une question en langage naturel au chatbot pour obtenir une réponse générée par l'IA.
- POST /search** : Effectue une recherche sémantique pure pour trouver les événements les plus pertinents sans interroger l'IA.
- POST /rebuild** : Lance en arrière-plan la mise à jour de la base de données avec les derniers événements.
- GET /rebuild/status** : Permet de suivre l'état d'avancement de la mise à jour de la base de données.
- GET /health** : Vérifie que tous les composants de l'API sont fonctionnels.
- GET /stats** : Fournit des statistiques sur la base de données vectorielle, comme le nombre d'événements indexés.



## Choix du modèle de chatbot

Le choix de **mistral-small-latest** repose sur plusieurs critères stratégiques, idéaux pour un POC (Proof of Concept) comme celui-ci :

**Qualité sur la langue française** : Le modèle est reconnu pour son excellente performance et sa compréhension nuancée du français, ce qui est crucial pour un chatbot destiné à un public francophone.

**Rapport performance/coût optimal** : Il offre un équilibre parfait entre une haute qualité de génération, une faible latence (temps de réponse rapide) et un coût par token maîtrisé, ce qui le rend idéal pour un POC (Proof of Concept).

**Intégration facile** : Le modèle est nativement supporté par le framework LangChain, ce qui a permis une intégration simple et rapide dans l'architecture RAG (Retrieval-Augmented Generation) du projet.

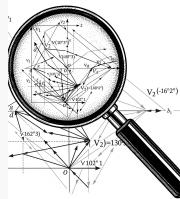
# Le Fonctionnement de l'Endpoint /ask



## 1. La Question de l'Utilisateur

L'utilisateur envoie sa question en langage naturel via une requête POST à l'endpoint /ask.

```
{ "question": "Quels sont les marchés de Noël à Toulouse ?" }
```



## 2. Recherche Sémantique (Retrieval)

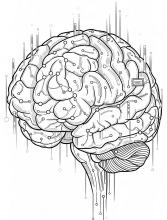
- La question est transformée en un vecteur numérique (embedding) qui capture son sens.
- Ce vecteur est utilisé pour interroger la base FAISS et trouver les 5 extraits d'événements les plus pertinents.



## 3. Enrichissement du Prompt

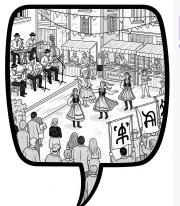
Les extraits trouvés (le contexte) sont assemblés avec :

Le prompt système (ps.md) : qui définit la personnalité ("Puls-Events") et les règles (répondre uniquement avec le contexte, se limiter à l'Occitanie) et la question originale de l'utilisateur.



## 4. Génération par l'IA (Generation)

Le prompt complet et enrichi est envoyé à l'API de Mistral AI (mistral-small-latest). Le modèle génère une réponse en se basant uniquement sur les informations fournies dans le contexte, ce qui empêche les "hallucinations".

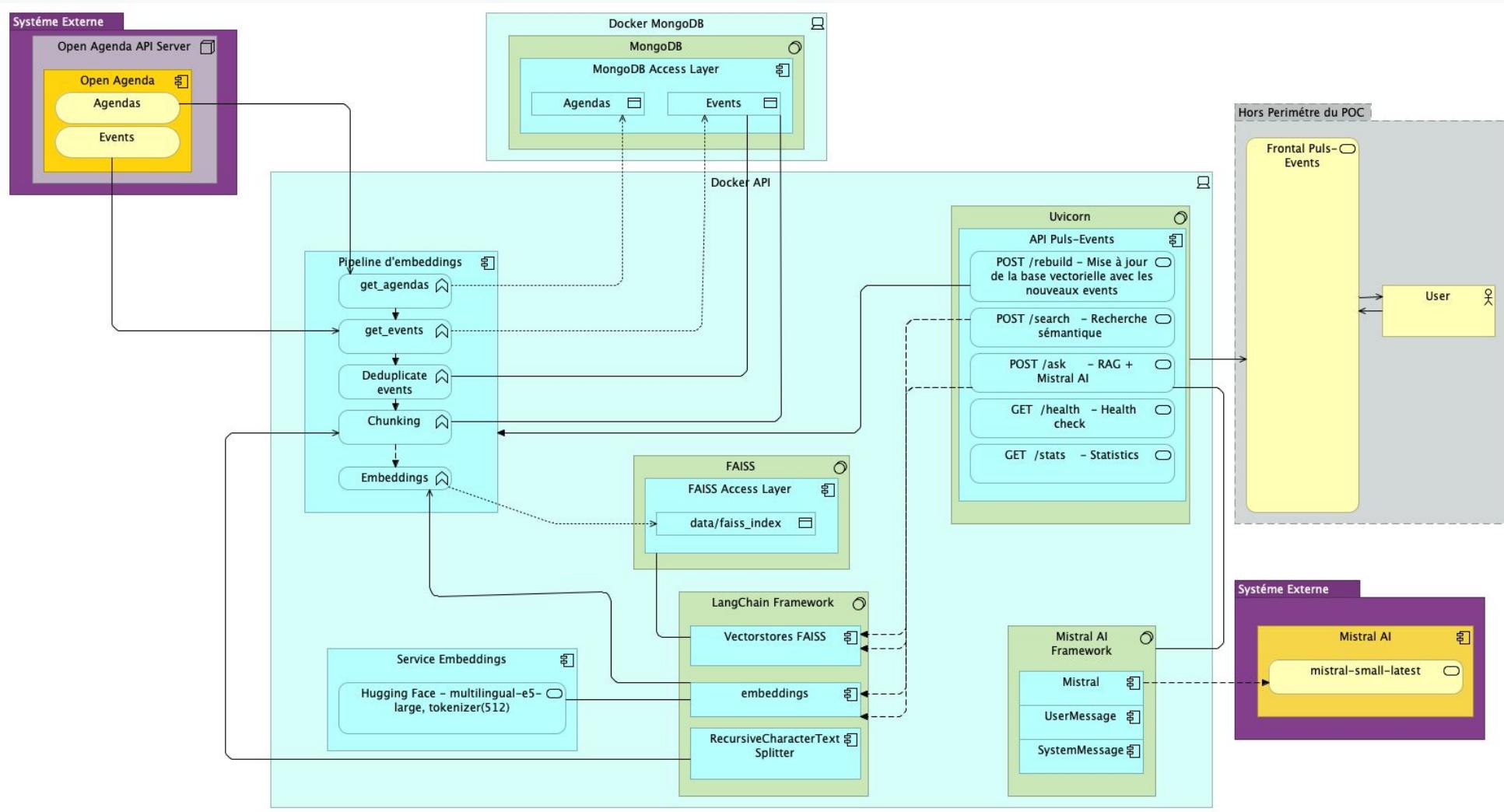
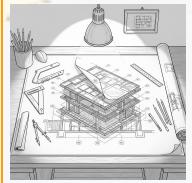


## 5. Réponse Structurée

L'API renvoie une réponse JSON complète au client, contenant :

- La question originale.
- La réponse générée par l'IA.
- Le contexte utilisé pour la transparence.
- Les statistiques de tokens pour le suivi des coûts.

# Schéma d'architecture applicative



# **Les résultats**

# Rapport des résultats RAGAS

## 1. Phase de Collecte

- Un jeu de 10 questions de test est posé au chatbot via l'endpoint /ask.
- Pour chaque question, on collecte et sauvegarde 3 éléments clés :
- La réponse générée par l'IA.
- Le contexte (les documents extraits de FAISS).
- La vérité terrain (la réponse idéale, annotée manuellement).

## 2. Phase d'Évaluation

- Le framework RAGAS utilise un modèle de langage (Mistral AI) comme **un juge impartial**.
- Ce juge reçoit les données collectées et les compare pour évaluer la qualité de la réponse sur plusieurs axes.
- Exemple de question au juge : "La réponse générée est-elle entièrement soutenue par le contexte fourni ?"

## 3. Phase de Calcul

- Le juge attribue un score de 0 à 1 pour chaque métrique, qui est ensuite moyenné sur toutes les questions.
- Résultat final : Un rapport HTML visuel est généré, présentant ces scores de manière claire.

### Métriques Moyennes



### Scores Détailrés par Question

#	Question	Faithfulness	Answer Relevancy	Context Precision	Context Recall
1	Y a-t-il des marchés de noel en Occitanie cette année ?	0.963	0.934	0.000	0.824
2	Donne moi plus d'information sur le marché de noel de Nouilles	0.750	0.932	1.000	0.750
3	Quand commence le marché de noel de Nouilles	1.000	0.920	1.000	1.000
4	Quand a lieu le TGS ?	1.000	0.878	0.810	0.667
5	Quel sont les prix du TGS ?	1.000	0.891	1.000	1.000
6	Quel est le programme du TGS ?	0.778	0.873	0.500	0.500
7	Quand a lieu la fête du kiwi ?	1.000	0.924	0.806	1.000
8	Que propose le bal trad de la fête du kiwi ?	1.000	0.905	1.000	0.833
9	Peux-tu me donner une recette de cuisine avec du kiwi ?	0.833	0.000	0.950	0.667

# Évaluation du POC – Les Chiffres Clés

FAITHFULNESS

**0.925**

Excellent

## 1. Fiabilité (Faithfulness)

Le chatbot n'invente pas d'informations et reste fidèle aux données sources.

ANSWER RELEVANCY

**0.806**

Excellent

## 2. Pertinence de la Réponse (Answer Relevancy)

La réponse générée est bien en adéquation avec la question posée par l'utilisateur.

CONTEXT RECALL

**0.804**

Excellent

## 3. Rappel du Contexte (Context Recall)

Le système réussit à trouver toutes les informations nécessaires pour répondre complètement.

CONTEXT PRECISION

**0.785**

Bon

## 4. Précision du Contexte (Context Precision)

La recherche est précise, mais il y a une marge pour améliorer le classement des informations les plus pertinentes en premier.

# Évaluation du POC – Analyse Qualitative



## Ce qui fonctionne très bien

### Fiabilité:

Le système a démontré une tolérance zéro aux "hallucinations". Il reconnaît honnêtement quand il ne sait pas, ce qui renforce la confiance de l'utilisateur.

### Compréhension Sémantique Avancée :

Le chatbot comprend les questions en langage naturel, même sans mots-clés exacts, et gère intelligemment les questions hors-sujet en redirigeant l'utilisateur.

### Pipeline Robuste et Automatisé:

L'architecture est solide, de la collecte des données à la réponse, avec une mise à jour incrémentale fonctionnelle (/rebuild) et un monitoring intégré (/health).



## Axes d'Amélioration Identifiés

### Dépendance à la Qualité des Données Sources:

La principale limite vient de la source de données (Open Agenda). Si un événement manque de détails (ex: le programme détaillé d'un festival), la réponse sera factuelle mais générique.

### Précision de la Recherche pour les Requêtes Fines:

Pour des questions très spécifiques, le système pourrait bénéficier d'une étape de "reranking" pour mieux classer les informations les plus pertinentes et éviter le bruit.

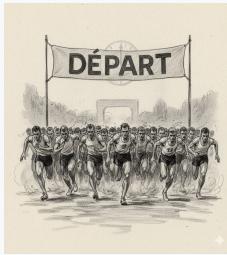
## Conclusion :

Le système RAG est validé comme une solution fiable et performante pour répondre au besoin de Puls-Events.

Les axes d'amélioration sont bien identifiés et concernent principalement l'optimisation plutôt que la conception de base.

# **Les perspectives**

# Perspectives



## Optimisations à Court Terme

### Améliorer la Recherche (Retrieval) :

- Ajouter une étape de "reranking" après la recherche initiale pour mieux classer les résultats et augmenter la Context Precision (passer de 0.65 à > 0.80).
- Mettre en place une recherche hybride (sémantique + mots-clés) pour les requêtes très précises.

### Réduire les Coûts Opérationnels :

- Mettre en place un cache (Redis) pour les questions fréquentes.
- Explorer l'auto-hébergement d'un modèle open-source (ex: Mistral 7B) pour passer d'un coût par requête à un coût d'infrastructure fixe.



## Vision à Long Terme

### Personnalisation et Mémoire :

- Gérer des sessions utilisateur pour mémoriser le contexte de la conversation et les préférences (lieux, types d'événements favoris).

### Extension des Données et Multimodalité :

- Intégrer de nouvelles sources de données (Eventbrite, sites de mairies) pour une couverture plus large.
- Enrichir les réponses avec des images d'événements et des cartes interactives.

# Roadmap Technique



## Passage en Production

### Infrastructure Scalable :

- Migrer l'architecture sur Kubernetes pour la haute disponibilité et l'auto-scaling.
- Mettre en place un Load Balancer et un monitoring avancé (Grafana, Prometheus).

### CI/CD et Sécurité :

- Déployer un pipeline CI/CD (ex: GitHub Actions) pour automatiser les tests et les déploiements.
- Implémenter des mesures de sécurité essentielles : authentification, rate limiting et HTTPS.



# Questions ?