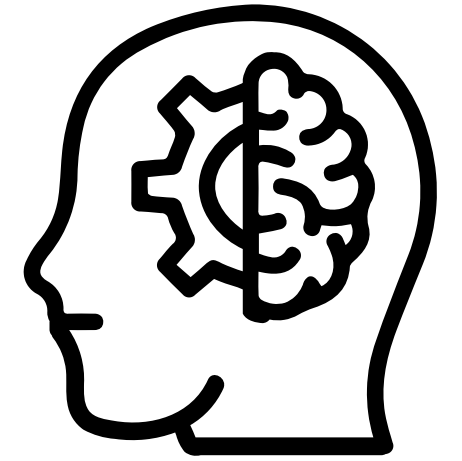


Création d'un Assistant intelligent pour la recommandation d'événements culturels



Mission : Mise en place d'un système RAG pour la recommandation d'événements culturels à Paris

Plan

- I. Contexte et objectifs**
- II. Organisation du dépôt**
- III. Problématique**
- IV. Données utilisées**
- V. Préparation & Vectorisation**
- VI. Architecture technique**
- VII. API et endpoints exposés**
- VIII. Évaluation du système**
- IX. Résultats et valeur ajoutée**
- X. Limites du POC**
- XI. Perspectives d'amélioration**

Conclusion



I. Contexte et objectifs

Objectifs :

- Créer un chatbot intelligent capable de répondre en langage naturel aux questions sur les événements, basé sur les données OpenAgenda.
- Livrer un POC : Démontrer la faisabilité technique et la pertinence métier


















Contexte fictif :

- Les utilisateurs veulent accéder facilement aux événements culturels à venir (concerts, expositions, ateliers...).
- En tant que Data scientist freelance, j'interviens pour l'entreprise "Puls-Events", je suis chargé de piloter la mise en production d'un un nouveau chatbot intelligent capable de répondre à des questions utilisateurs sur les événements culturels à venir, en s'appuyant sur un système RAG (Retrieval-Augmented Generation)

Phases du projet :

- Structuration du dépôt local et distant
- Récupération des données Open Agenda et nettoyage
- Mise en place de la base de données Vectorielle et création du chatbot
- Création de L'API pour exposer le chatbot
- Conteneurisation et déploiement local

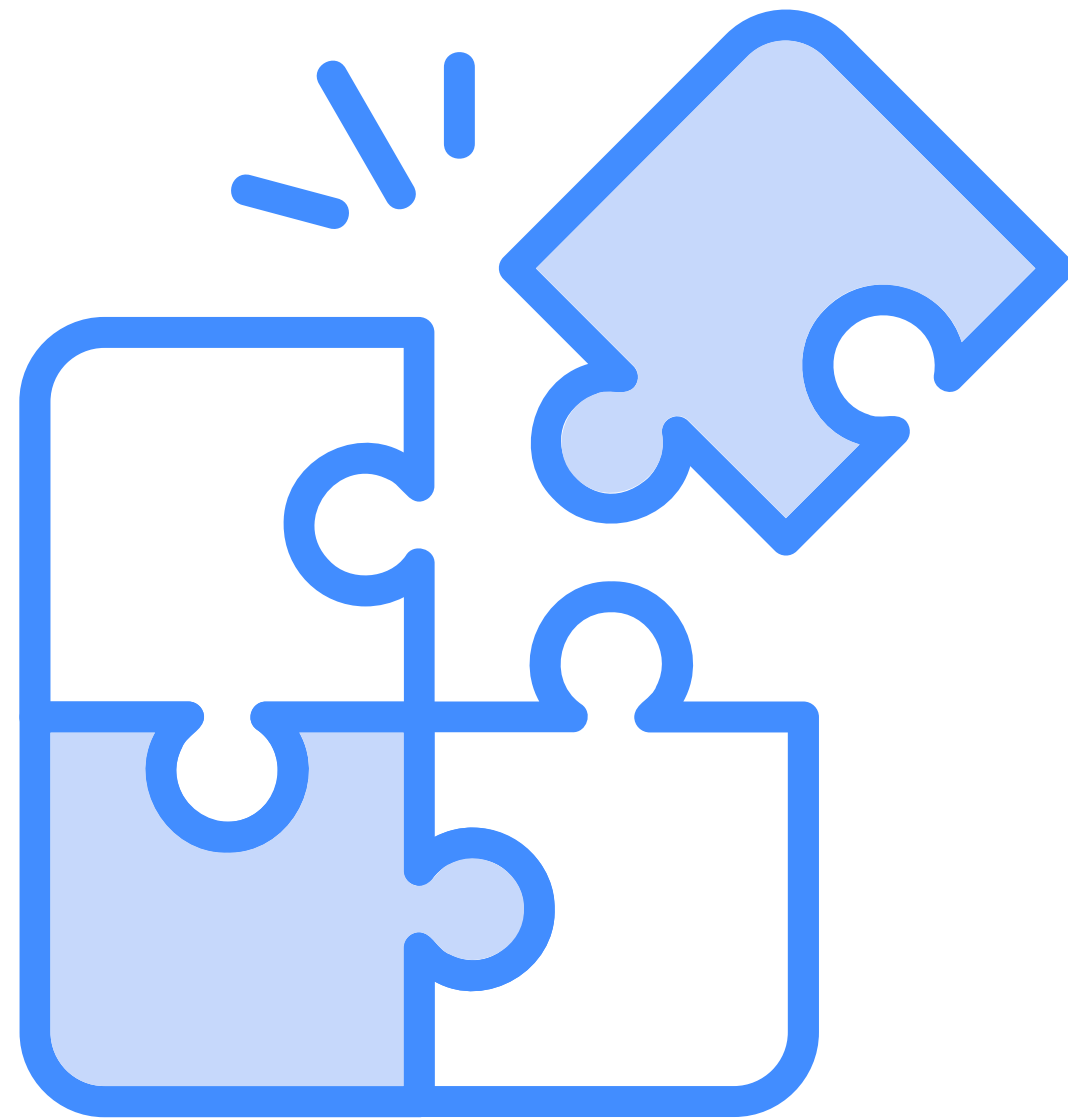
II. Organisation du dépôt

 .git	02/10/2025 12:07
 api	25/09/2025 12:29
 data	24/09/2025 11:53
 docker	22/09/2025 12:54
 env	22/09/2025 12:15
 eval	26/09/2025 12:46
 notebook	23/09/2025 11:51
 rag	25/09/2025 12:34
 scripts	26/09/2025 16:23
 tests	25/09/2025 13:22
 .dockerignore	26/09/2025 16:26
 .env	24/09/2025 11:35
 .gitignore	22/09/2025 12:07
 Dockerfile	02/10/2025 12:03
 Rapport technique.pdf	02/10/2025 12:06
 README.md	30/09/2025 11:51
 requirements.txt	26/09/2025 17:06

- **Présentation des dossiers :**

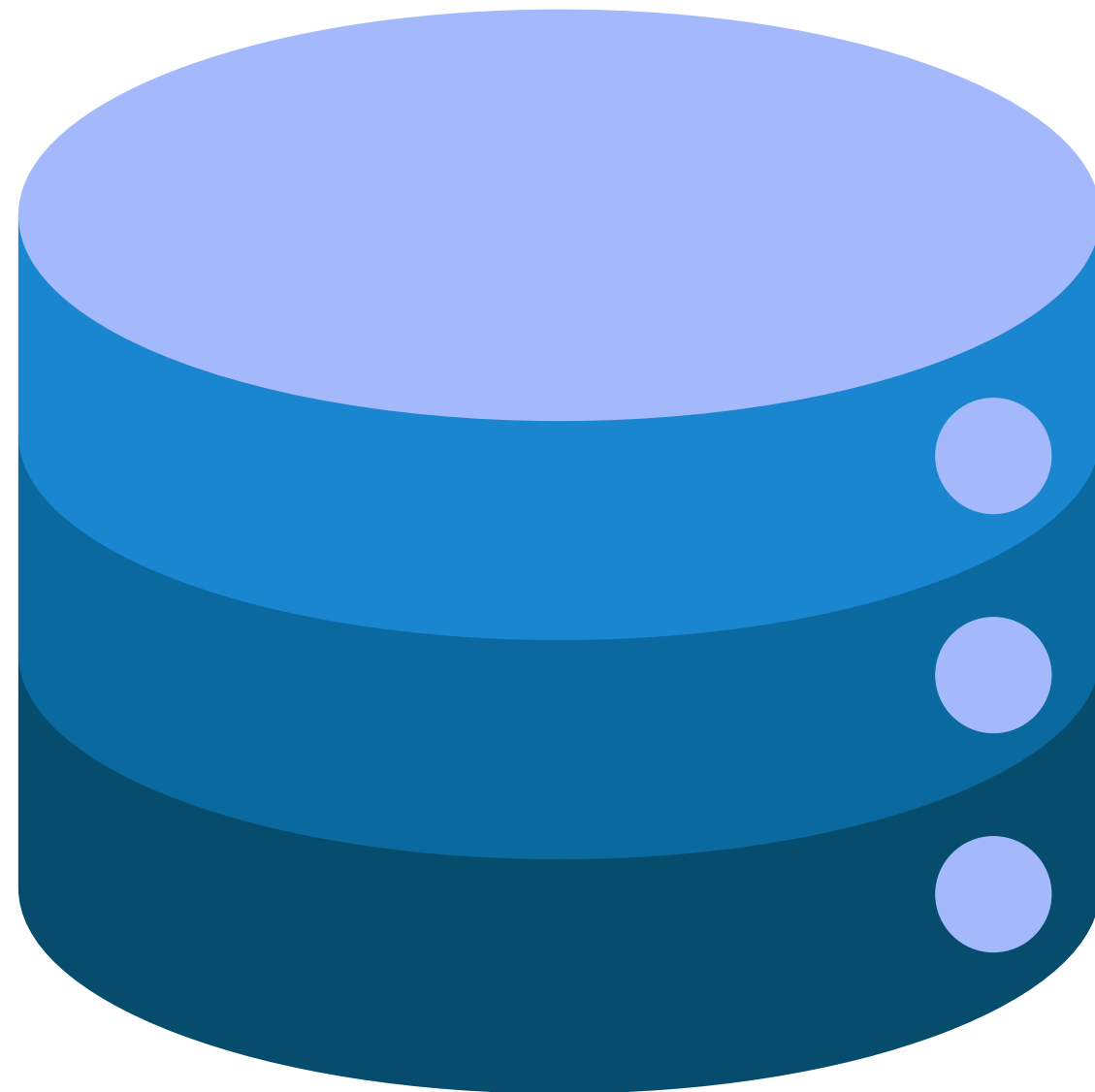
- rag/ (scripts ingestion, vectorisation, chatbot)
-
- api/ (FastAPI main.py)
-
- eval/ (scripts évaluation Ragas + jeu de test)
-
- data/ (datasets et index FAISS)
-
- tests/ (unit tests)

III. Problématique



- Saturation de l'offre culturelle → difficile pour les utilisateurs de trouver des événements pertinents.
- L'utilisateur doit naviguer longuement pour trouver des événements pertinents.
- Besoin d'un outil intelligent qui filtre et recommande selon la demande utilisateur.
- Une recherche complexe et chronophage : Les plateformes actuelles reposent souvent sur des filtres basiques (par date, lieu, catégorie).
- **Objectif métier :**
 - => améliorer l'expérience utilisateur de la plateforme Puls-Events.
 - => réduction des coûts

IV. Données utilisées



Source des données

- API publique OpenAgenda (jeu de données événements publics).
- Données ouvertes et régulièrement mises à jour.
- Contenu : concerts, expositions, festivals, ateliers, spectacles, conférences, etc.

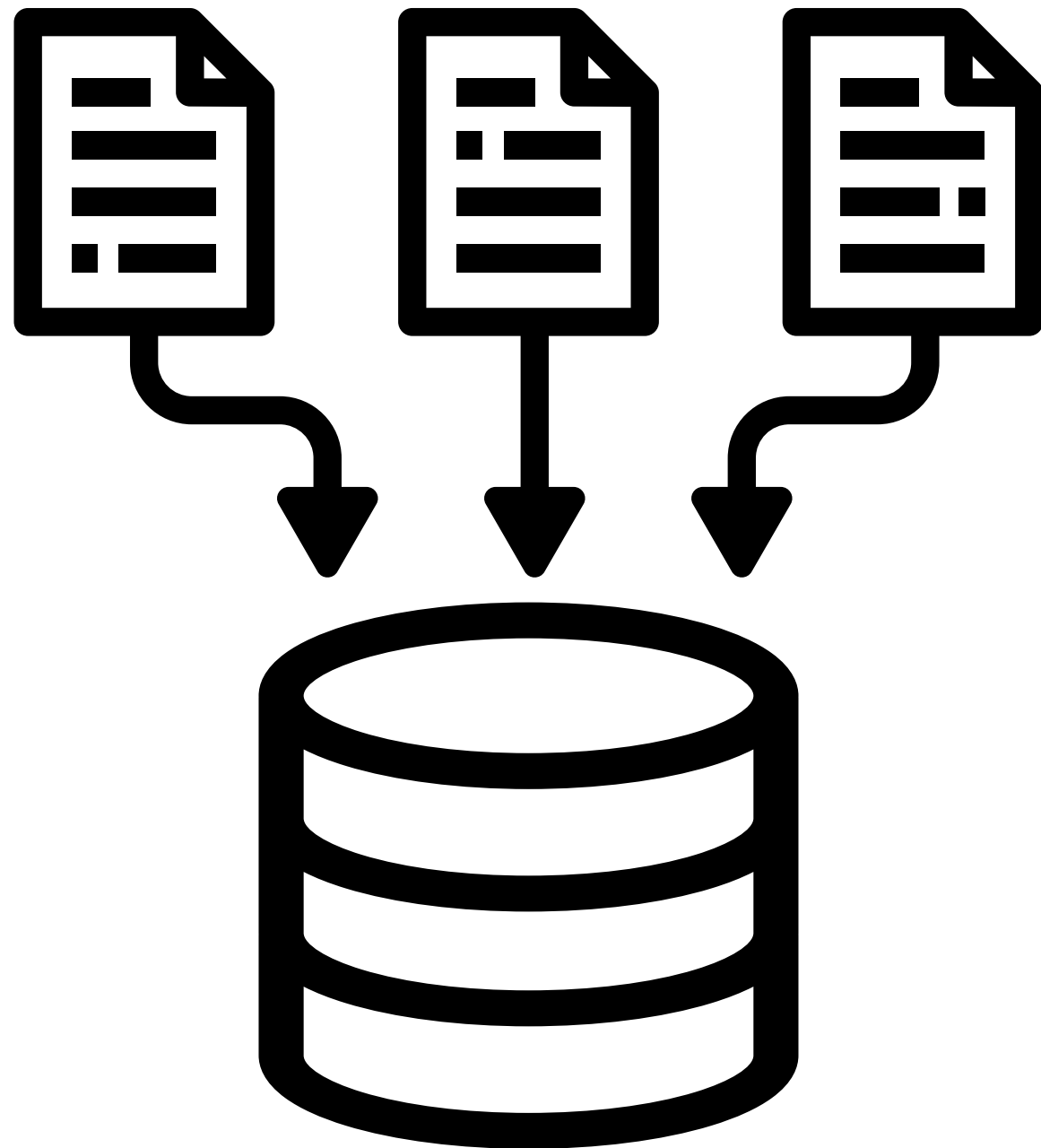
Périmètre retenu pour le POC :

- Localisation : Paris.
- Période : Événements pour l'année 2025.
- Environ 2 200 événements récupérés.

Export et traitement :

- Export initial en JSON (via l'API) puis conversion en CSV pour analyse rapide.
- Nettoyage et structuration réalisés avec Pandas.
- Ajout d'une colonne text_to_embed combinant titre + description pour la vectorisation.
- Ex de champs clés conservés : id, title, description, date_start, city...

V. Préparation & Vectorisation



Nettoyage des données

- Suppression des doublons et gestion des champs manquants
- Concaténation des champs title + description + long_description
- Normalisation du texte (suppression des balises HTML etc)

Création d'une colonne `text_to_embed` :

- Texte unifié destiné à l'indexation vectorielle.
- Assure une cohérence entre les événements (tous représentés par une description exploitable).

Découpage en chunks :

- Découpage du texte en blocs de 500 caractères avec un chevauchement de 50.

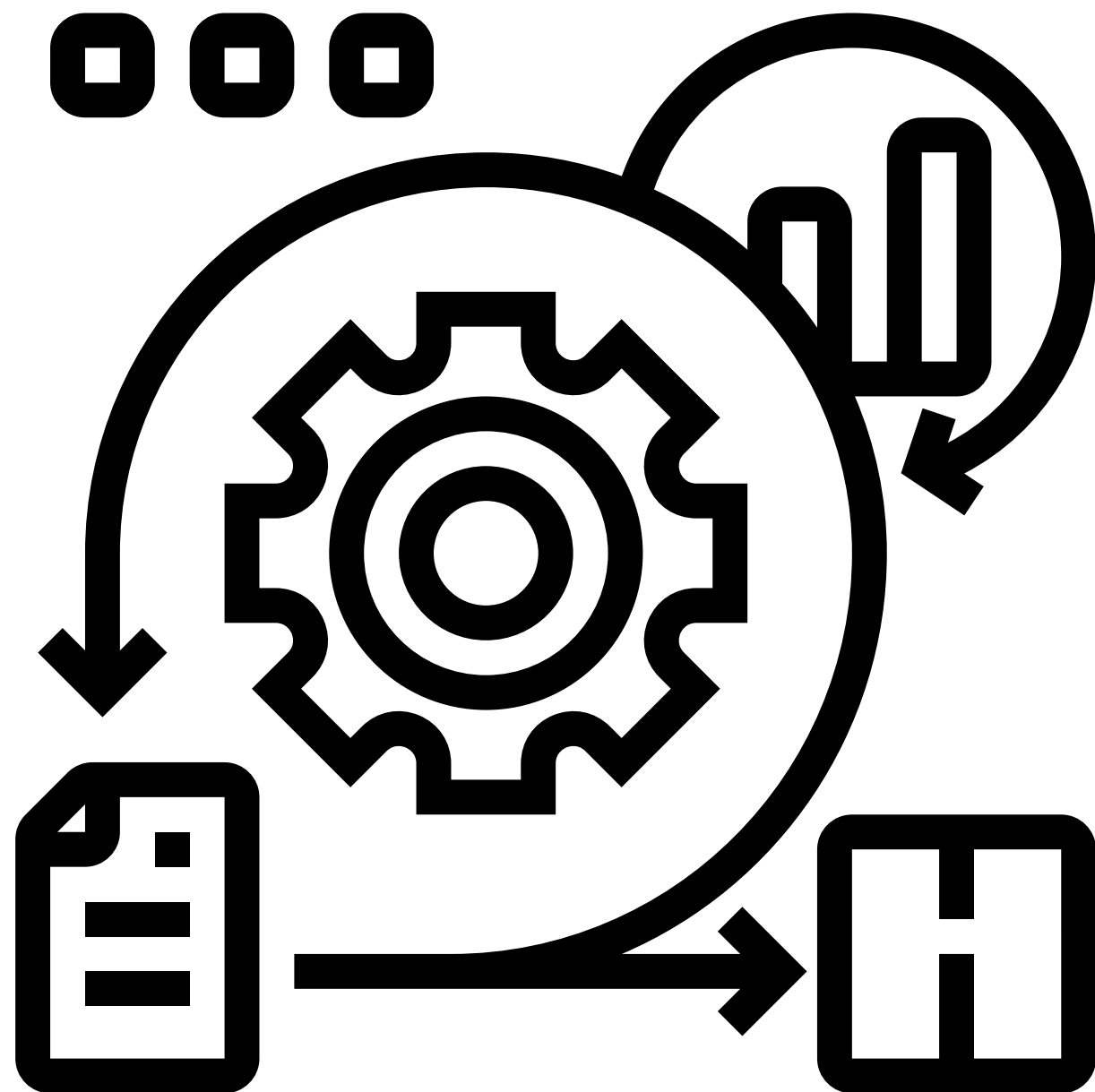
Vectorisation :

- Utilisation du modèle mistral-embed via l'API Mistral.
- Conversion de chaque chunk en vecteur numérique (embedding).

Objectif final :

- Transformer les événements en vecteurs exploitables par FAISS et stockés dans une base vectorielle.

VI. Architecture technique



Ingestion & Nettoyage

Préparation & Vectorisation

Indexation FAISS :

- Construction d'une base vectorielle pour stocker et rechercher les vecteurs + Association des metadonnées

RAG (Retrieval-Augmented Generation) :

- Recherche de documents pertinents via FAISS.
- Génération de réponses contextualisées avec LLM **mistral-small-2503**.

API REST (FastAPI) :

- Exposition de deux endpoints principaux :

/ask → poser une question et obtenir une réponse.

/rebuild → reconstruire la base vectorielle si nécessaire.

Conteneurisation :

- L'ensemble du système est packagé dans un conteneur Docker.
- Avantage : exécution reproductible, facile à déployer

VII. API et endpoints exposés



Framework utilisé :

- FastAPI, choisi pour : sa rapidité d'exécution, sa documentation interactive intégrée (Swagger UI), sa simplicité de mise en place et de test.

Endpoints principaux :

- **GET /health** : Vérifie que l'API est en ligne et opérationnelle.
- **POST /ask** : Input : une question utilisateur en langage naturel.

=> Recherche des événements pertinents dans FAISS.

=> Génération d'une réponse augmentée via Mistral.

=> Output : answer (réponse textuelle), sources

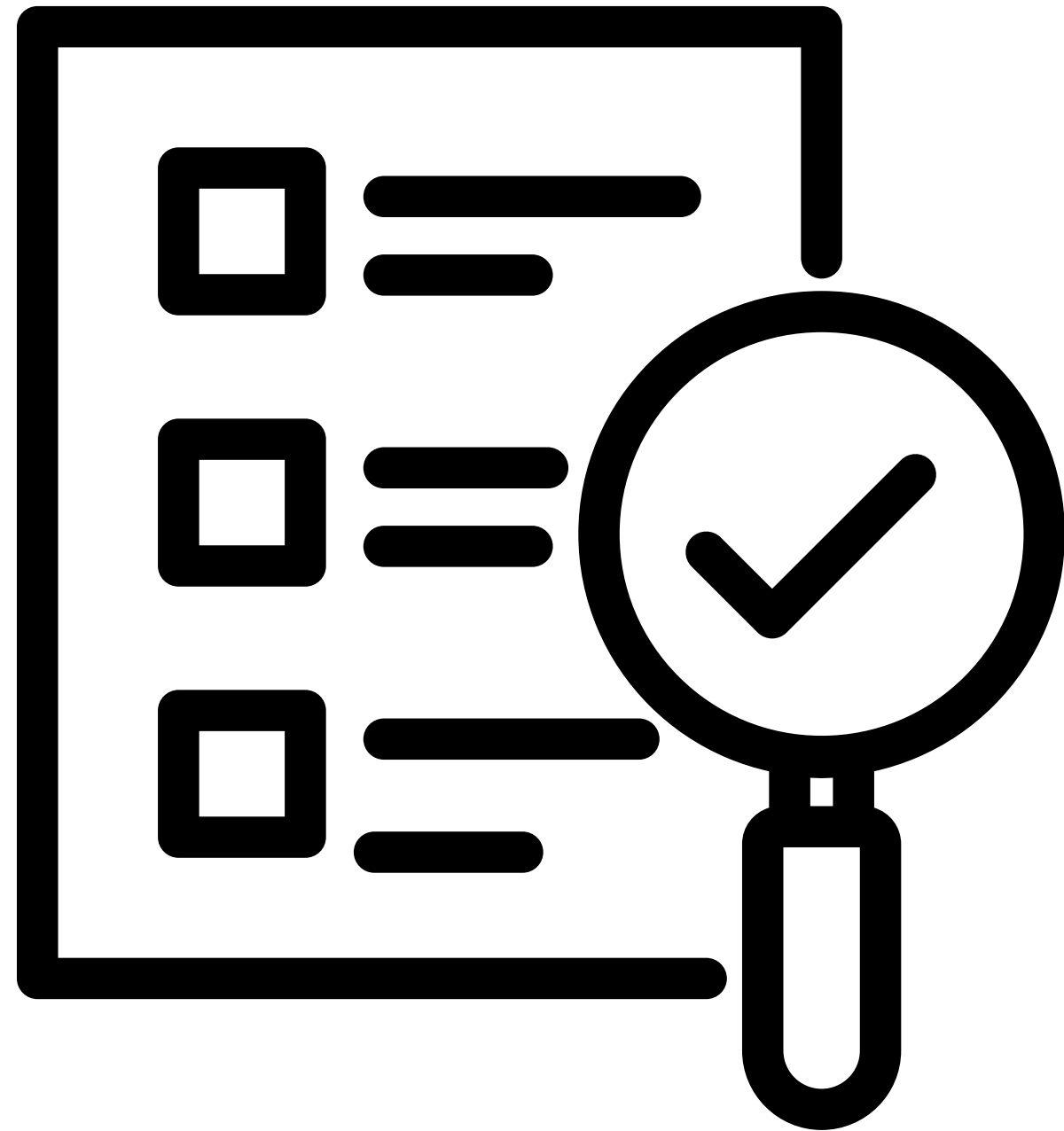
- **POST /rebuild** : Reconstitue complètement la base vectorielle FAISS à partir du fichier events_clean.json.

=> Utile si les données changent ou doivent être mises à jour.

Documentation interactive :

- Swagger UI disponible sur /docs.
- Permet de tester chaque endpoint directement depuis un navigateur.

VIII. Évaluation du système



Jeu de test annoté :

- Créé manuellement avec 10 questions représentatives
- Chaque question est associée à une réponse de référence (ground truth), validée à partir des données OpenAgenda

Métriques utilisées (via Ragas) :

- Answer Relevancy : correspondance entre la question et la réponse.
- Faithfulness : fidélité au contexte (éviter les hallucinations).
- Context Precision : proportion de contexte pertinent utilisé.

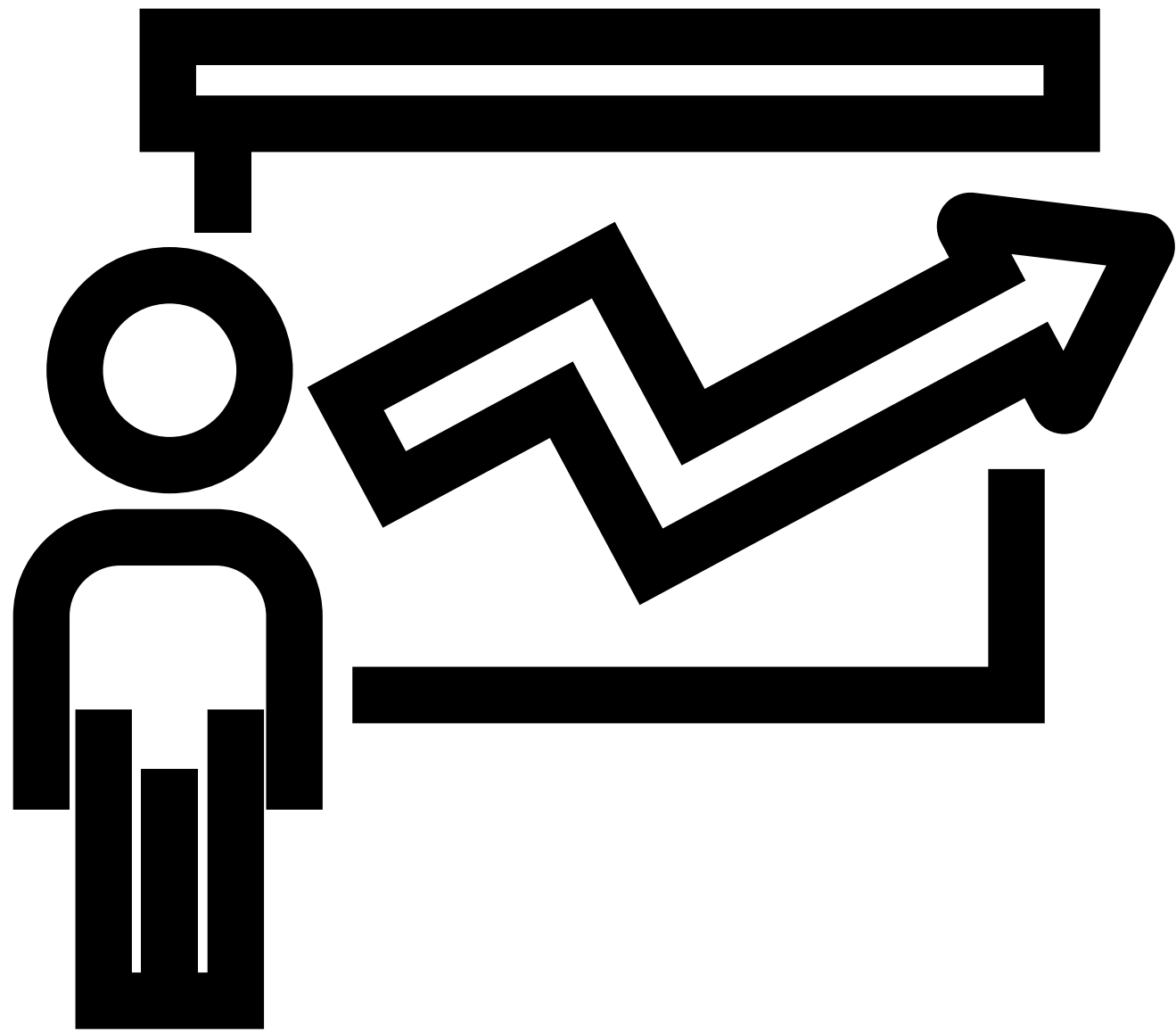
Résultats globaux obtenus :

- Answer Relevancy ≈ 0.56
- Faithfulness ≈ 0.59
- Context Precision ≈ 0.10
- Context Recall ≈ 0.18

Conclusion de l'évaluation :

- Le système est pertinent sur son cœur d'usage
- Le POC démontre la faisabilité mais nécessite des améliorations pour élargir la couverture et renforcer la robustesse.

IX. Résultats et valeur ajoutée



Un chatbot culturel fonctionnel :

- Capable de répondre à des questions formulées en langage naturel.
- Fournit des réponses contextualisées

API REST prête à l'emploi :

- Endpoints /ask, /rebuild et /health disponibles.
- Documentation interactive via Swagger UI (/docs).
- Permet aux équipes de tester directement la solution.

Reproductibilité et portabilité :

- Projet conteneurisé avec Docker.
- Facile à déployer et exécuter sur n'importe quelle machine locale ou serveur.

Évaluation automatisée (Ragas)

Valeur ajoutée métier :

- Amélioration de l'expérience utilisateur : accès rapide aux événements pertinents.
- Gain de temps pour la recherche culturelle.
- Base solide pour un futur produit intégré dans Puls-Events.

X. Limites du POC



Contraintes techniques liées à l'API Mistral

- Limitation à 1 requête par seconde → ralentit la génération d'embeddings en masse et Risque d'erreurs 429 (dépassement de quota)

Contexte restreint :

- Données limitées à Paris et à l'année 2025 pas encore généralisable

Couverture thématique inégale :

- Très efficace pour la musique et les expositions.
- Moins pertinent pour des thématiques avec peu d'événements (sport, ateliers spécifiques).

Performance utilisateur :

- Potentiel problème de scalabilité si usage intensif en production.

POC non encore industrialisé :

- Pas de gestion avancée de l'historique de conversation.
- Pas encore optimisé pour un usage en production (monitoring, CI/CD cloud).

XI. Perspectives d'amélioration



Enrichissement des données

- Étendre la couverture géographique à d'autres villes française, Intégrer plusieurs années d'événements pour permettre des recherches prospectives, Ajouter des thématiques plus variées (sport, conférences, ateliers jeunesse, etc.).

Optimisation technique de l'index vectoriel :

- Explorer des structures de recherche avancées comme HNSW pour accélérer la recherche

Réduction de la dépendance à l'API Mistral :

- Tester l'utilisation d'embeddings locaux (ex. Sentence Transformers) pour l'étape de vectorisation.
- Avantages : suppression des quotas, meilleure rapidité, réduction des coûts.

Déploiement cloud et industrialisation :

- Héberger l'API sur un cloud provider (AWS, GCP, Azure).
- Mettre en place une pipeline CI/CD pour les mises à jour automatiques.
- Ajouter du monitoring (temps de réponse, erreurs API, logs des requêtes)

III. Conclusion

Preuve de concept validée

- Mise en place réussie d'un système RAG (Retrieval-Augmented Generation) appliqué aux événements culturels.
- Démonstration d'un chatbot culturel intelligent interrogeable via API.

Résultats tangibles :

- Index vectoriel construit et interrogeable.
- API REST exposée et conteneurisée avec Docker.
- Évaluation automatique avec Ragas démontrant la pertinence des réponses

Apports métier :

- Amélioration de l'expérience utilisateur dans la recherche d'événements.
- Solution réutilisable et extensible pour une plateforme comme Puls-Events.

Limites actuelles

- Quotas API Mistral limitant la fréquence d'interrogation (1 req/sec).
- Couverture limitée des données (uniquement Paris et 2025).
- Pertinence variable selon les thématiques : certaines requêtes n'ont pas de résultats fiables.

Perspectives :

- Passage à l'échelle en intégrant plus de données, optimisations techniques et déploiement cloud.
- Amélioration des performances du chatbot
- Base solide pour une industrialisation future et une intégration dans un produit réel.