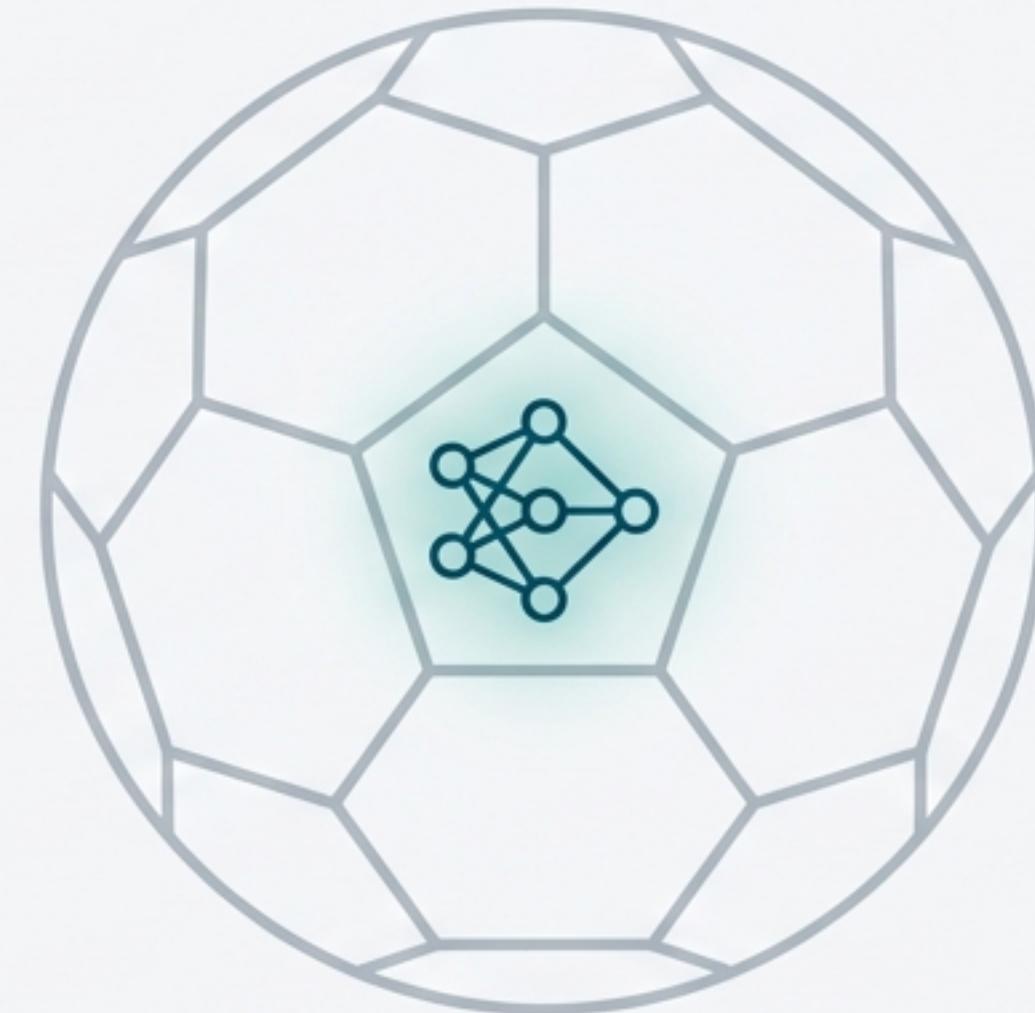


ASSAD AI : Architecture Technique

Conception et implémentation d'un assistant RAG expert pour la CAN 2025

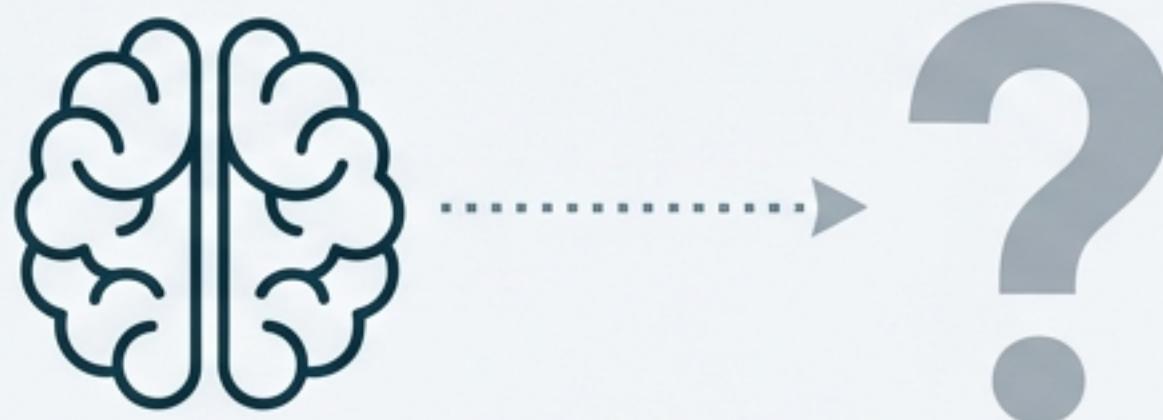


Document de conception v1.0 | Basé sur le modèle Google Gemini 3 Pro

Fournir des réponses fiables et factuelles, la mission fondamentale d'ASSAD AI

Le Défi

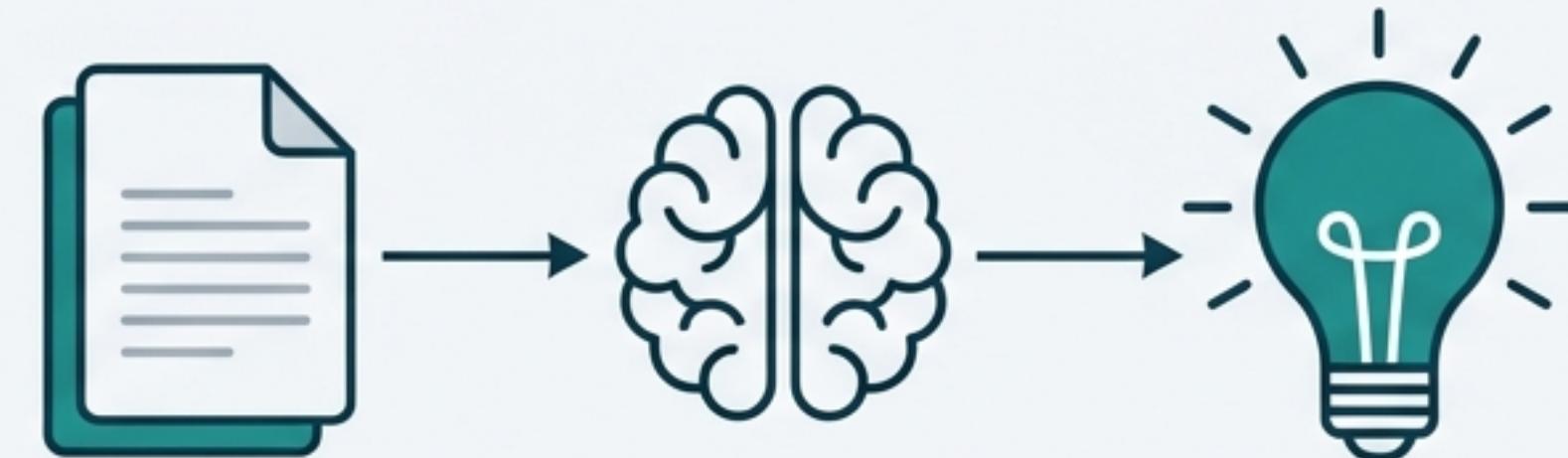
Les Large Language Models (LLM) standards peuvent "halluciner" ou inventer des informations, ce qui est inacceptable pour des données factuelles comme des résultats de matchs ou des calendriers. Leur connaissance s'arrête à leur date d'entraînement et n'inclut pas les détails spécifiques de la CAN 2025.



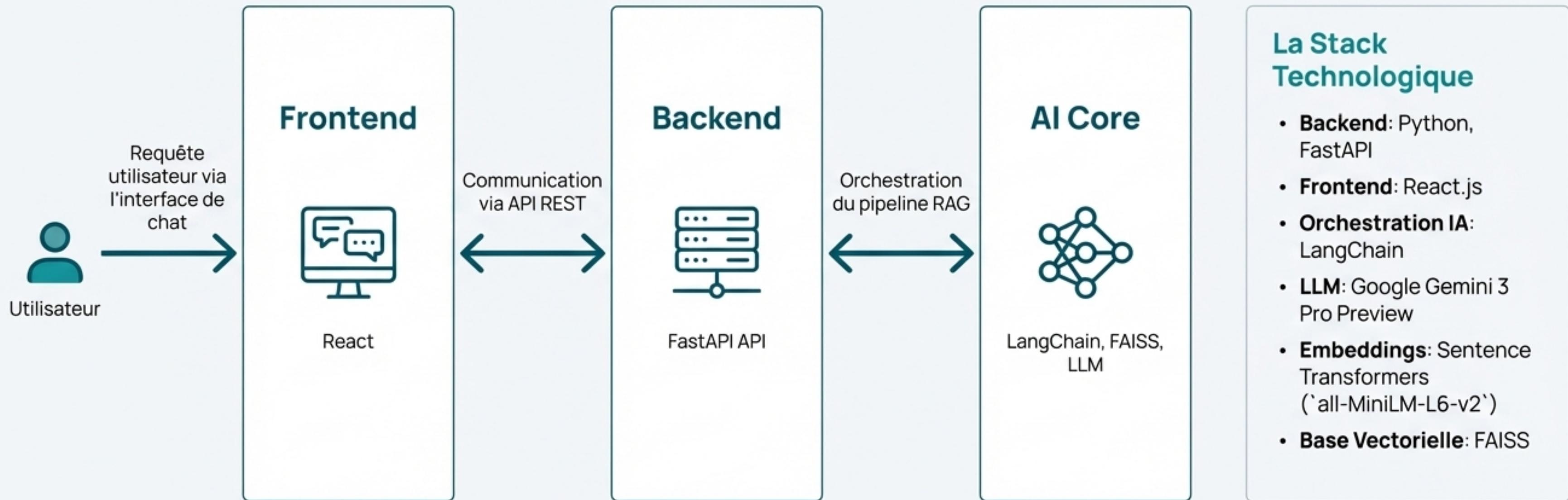
Notre Solution Stratégique : RAG

RAG (Retrieval-Augmented Generation) : Une architecture qui ancre les réponses du LLM sur une base de connaissances privée et contrôlée.

Le principe: Au lieu de demander au LLM de répondre de mémoire, nous lui fournissons les extraits de documents pertinents pour qu'il synthétise une réponse basée *uniquement* sur ces faits.



L'architecture d'ASSAD AI : un système complet, de l'interface à la base vectorielle



Notre conception repose sur trois piliers fondamentaux



Récupération de Données Intelligentes

Transformer et indexer efficacement les données de la CAN 2025 pour une recherche sémantique ultra-rapide et pertinente.

*Mots-clés : *Embeddings, FAISS, Métadonnées**



Génération de Texte Maîtrisée

Contraindre le LLM pour garantir des réponses factuelles, déterministes et traçables, basées exclusivement sur le contexte fourni.

*Mots-clés : *RAG, Prompt Engineering, Temperature=0**



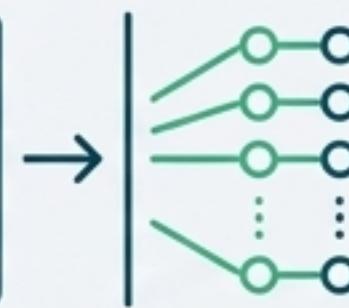
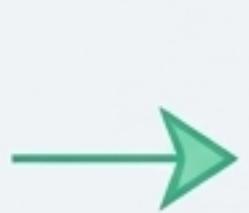
Une Architecture Applicative Robuste

Intégrer la logique IA dans une application client-serveur performante, modulaire et prête pour le déploiement.

*Mots-clés : *FastAPI, React, API REST**

Pilier 1 : Au cœur de la récupération de données, l'indexation sémantique

La phase d'indexation (offline) : préparer les connaissances du système



- 1.** Étape 1 :
Chargement & Transformation
(`load_docs.py`)

Les données JSON brutes sont chargées et transformées en documents structurés via LangChain. Des métadonnées critiques (type, équipe, date) sont ajoutées à chaque document.

- 2.** Étape 2 :
Vectorisation
(`embeddings.py`)

Chaque document est converti en un vecteur numérique de 384 dimensions par le modèle `all-MiniLM-L6-v2`, capturant son sens sémantique.

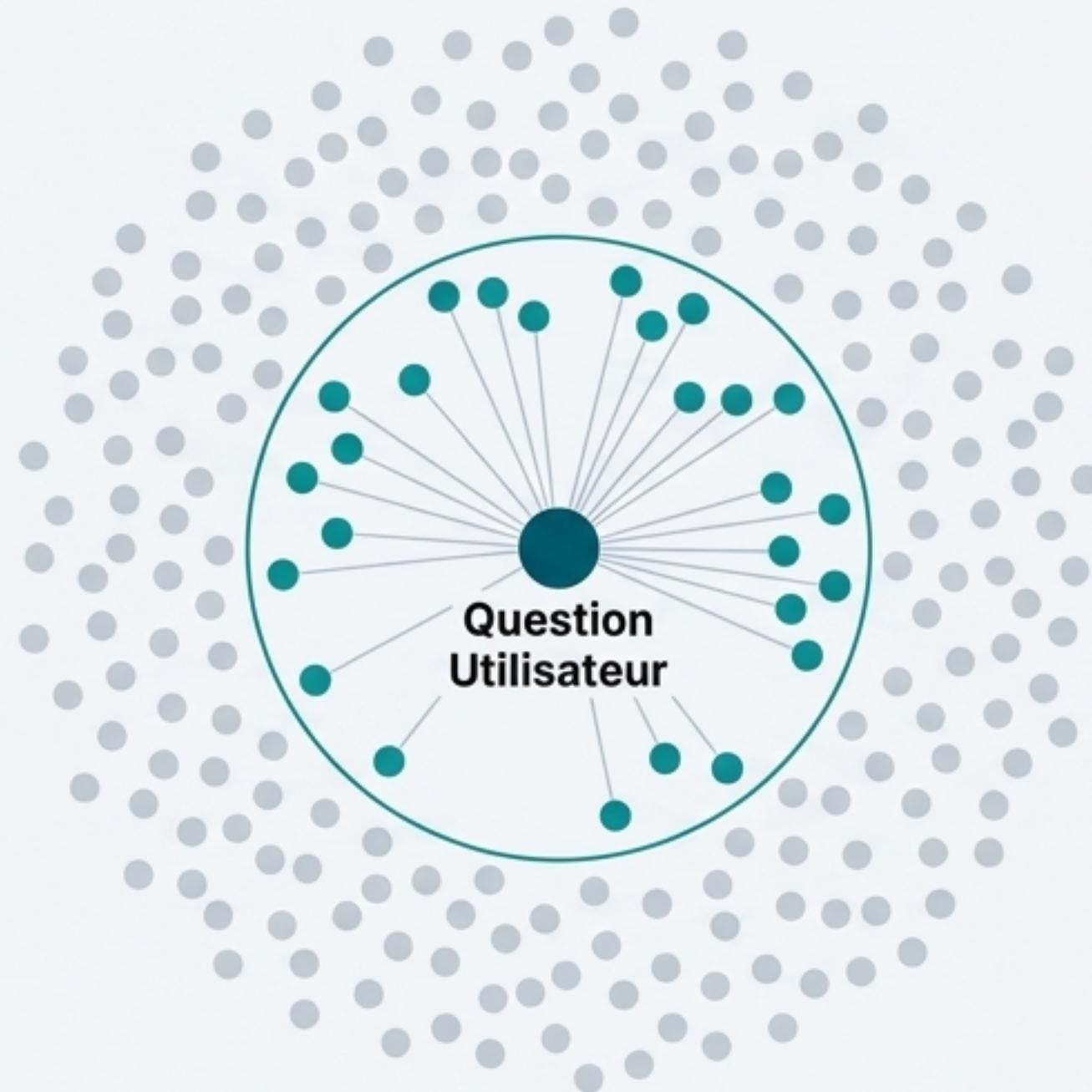
- 3.** Étape 3 :
Indexation ('FAISS')

Les vecteurs sont stockés dans un index FAISS (`index.faiss`). Cet index permet une recherche de similarité quasi-instantanée (voisin le plus proche, ou *nearest neighbor*).

Bénéfice Clé : Le résultat est une base de connaissances où la recherche de documents pertinents se fait en quelques millisecondes, non pas par mots-clés, mais par sens.

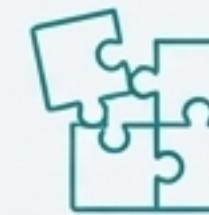
Un choix clé pour la pertinence : pourquoi récupérer les 20 documents les plus proches ($k=20$) ?

Le paramètre ' k ' définit le nombre de documents extraits de FAISS pour construire le contexte du LLM.
Le choix de ' $k=20$ ' est un équilibre stratégique.



1. Complétude

Assure que le contexte est suffisamment riche pour formuler une réponse complète, même pour des questions complexes.



2. Couverture

Gère efficacement les informations fragmentées (ex : un match dont les détails sont répartis sur plusieurs documents).



3. Performance

Représente un excellent compromis entre la quantité de tokens envoyés à l'API du LLM (coût, latence) et la richesse du contexte.



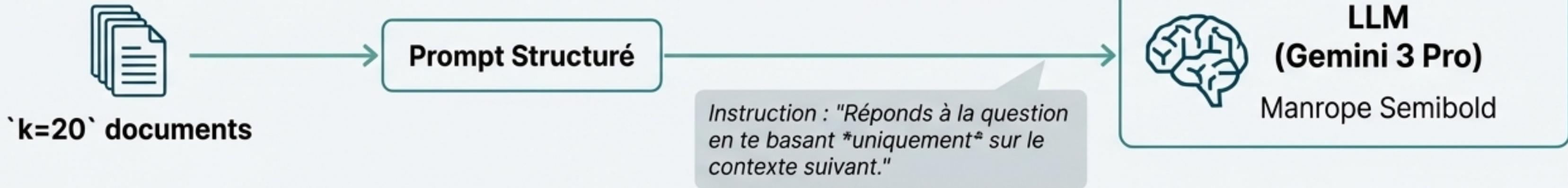
4. Diversité

Permet de récupérer différentes facettes ou perspectives sur un même sujet (ex : vue d'une équipe, vue du calendrier).

Pilier 2 : Maîtriser la génération pour une factualité absolue

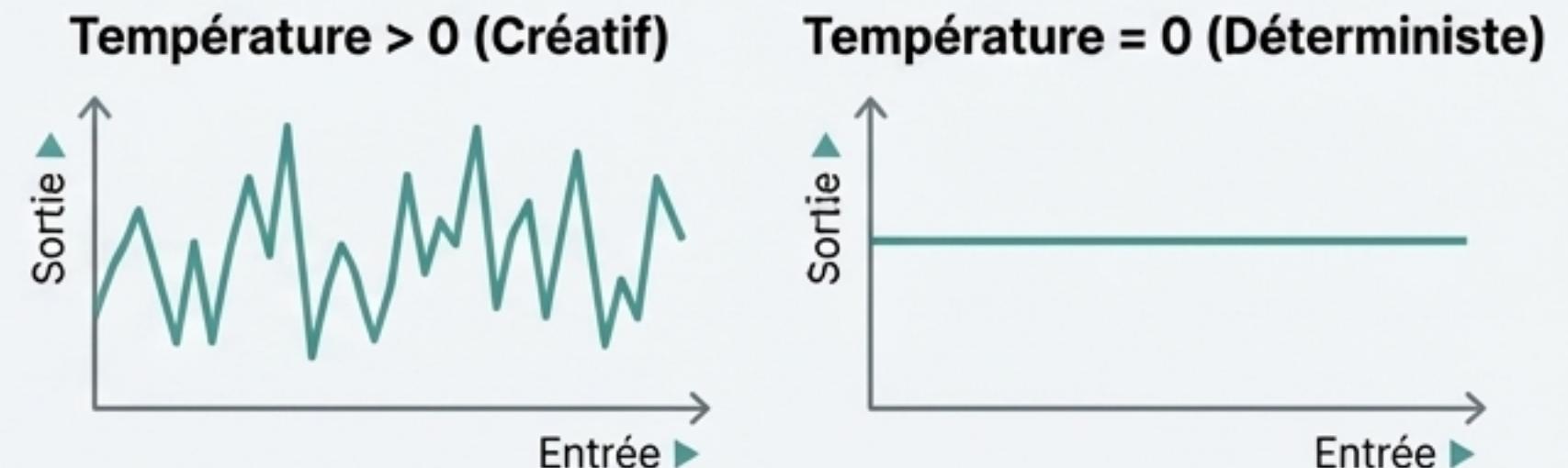
Le déterminisme comme principe directeur

Processus d'Augmentation



Le paramètre critique : `temperature=0`

La température contrôle le degré de "créativité" ou d'aléatoire dans la réponse du LLM. Une température de `0` rend la sortie du modèle entièrement déterministe.



Pourquoi le déterminisme est essentiel ici

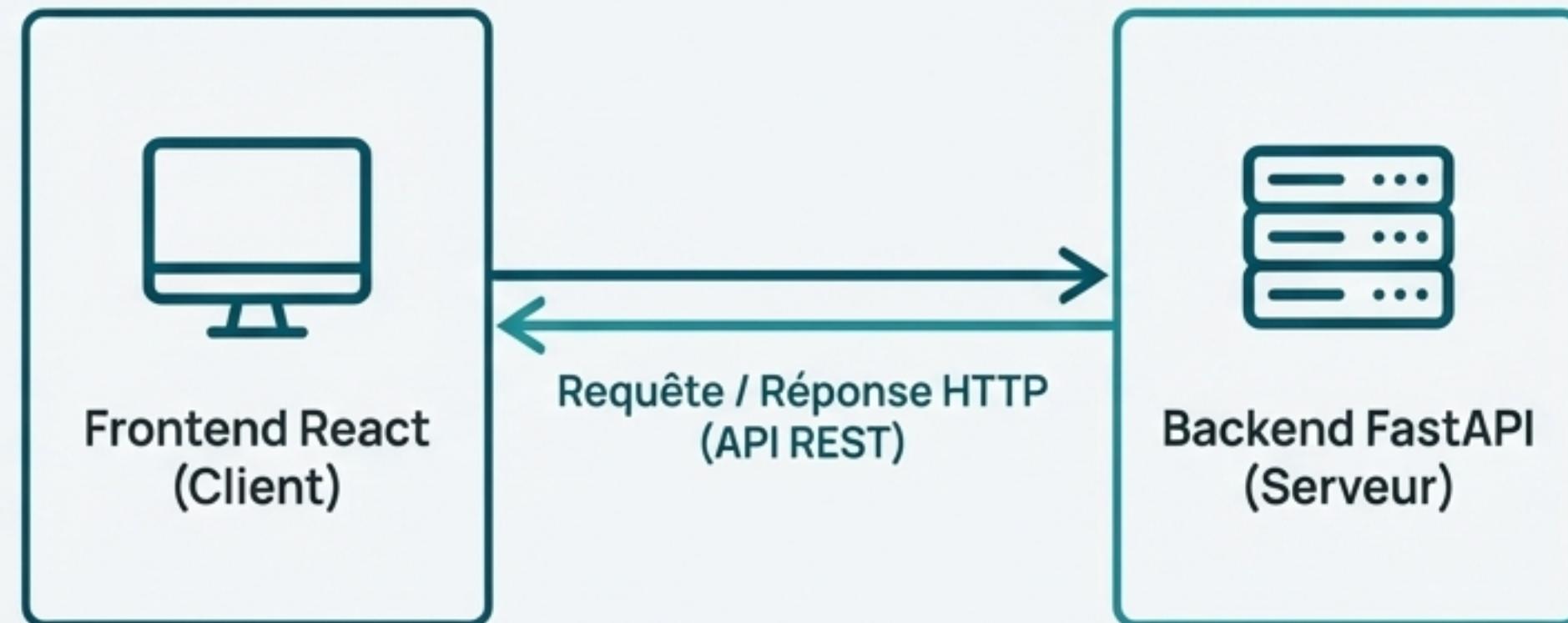
- * **Reproductibilité**: La même question produira toujours la même réponse.
- * **Exactitude**: Élimine toute variabilité créative qui pourrait nuire à la précision factuelle, ce qui est crucial pour des données sportives (scores, dates, noms).

Pilier 3 : Une architecture applicative robuste et découplée

La séparation des responsabilités entre le backend et le frontend

Composant Frontend : Interface React (`App.js`)

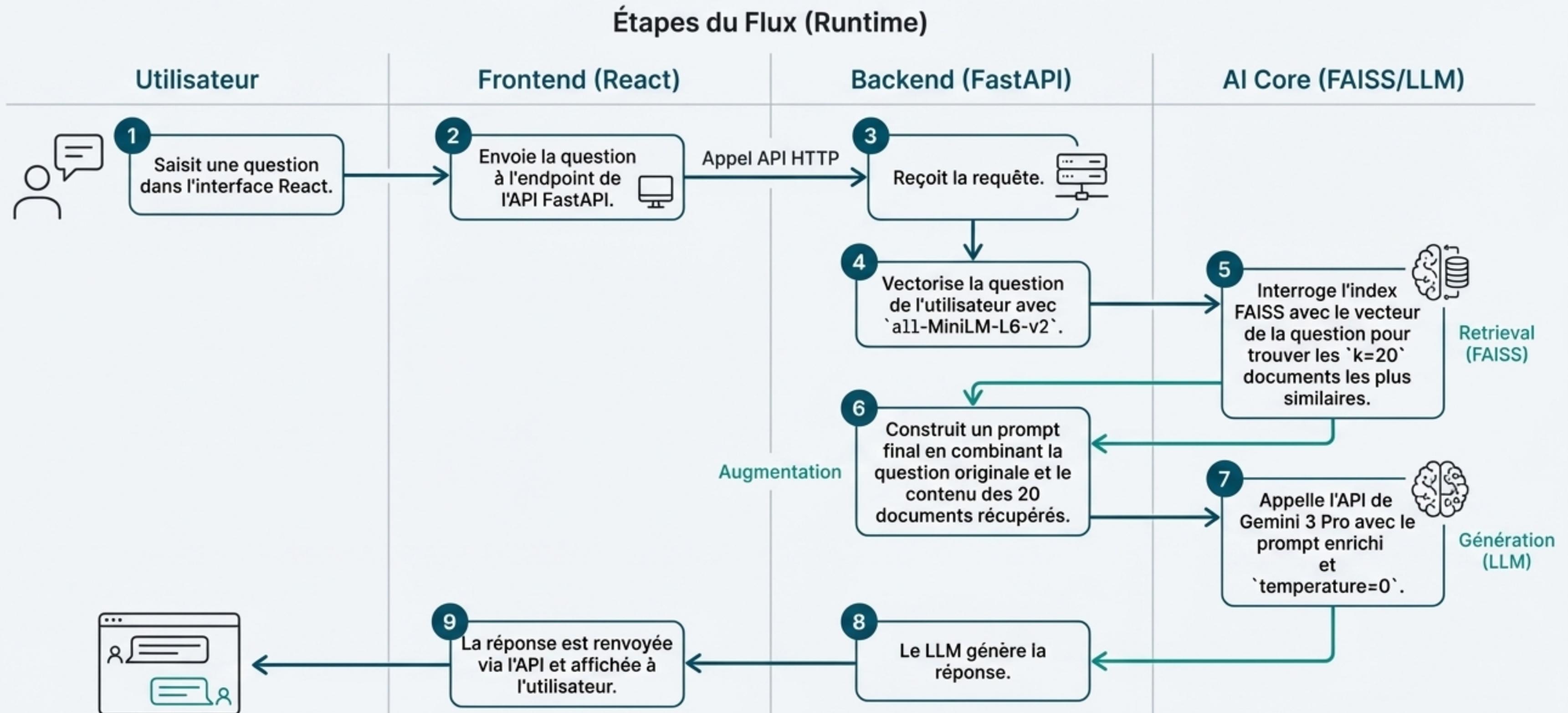
- **Rôle**: Offrir une interface de chat moderne, responsive (Tailwind CSS) et intuitive.
- **Flux utilisateur**: L'utilisateur saisit une question, l'envoie à l'API backend, reçoit la réponse et l'affiche dans l'interface, avec gestion des états (chargement, erreur).



Composant Backend : API FastAPI (`rag_chain.py`)

- **Rôle**: Exposer la chaîne RAG via une API REST sécurisée. Gérer la logique de réception des requêtes, d'orchestration de la recherche vectorielle et de l'appel au LLM.
- **Configuration CORS**: Paramétrée pour restreindre les appels à l'origine du frontend en production, assurant la sécurité.

Le cycle de vie complet d'une requête utilisateur



Exemple d'exécution : "Quand joue le Maroc en phase finale ?"

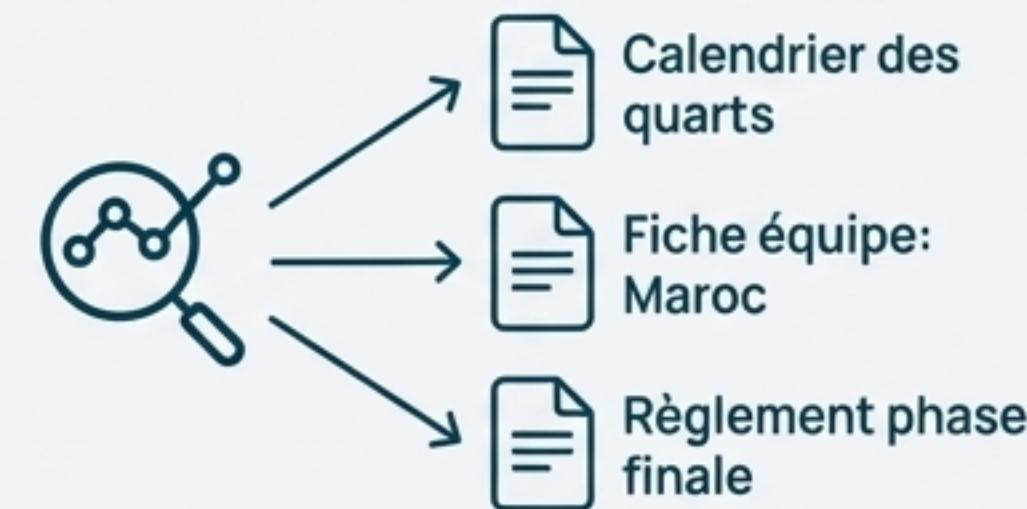
Interprétation de la question

Quand joue le Maroc en phase finale ?

Le système vectorise la question. Il comprend sémantiquement "Maroc" (même si l'utilisateur avait écrit "MAR" grâce à la normalisation), "joue" et "phase finale".



Récupération des documents pertinents



FAISS identifie les 20 documents les plus pertinents. Il ne s'agit pas d'une recherche par mot-clé. Il peut récupérer des documents contenant des informations sur :
Le calendrier des quarts de finale mentionnant le Maroc, la fiche de l'équipe du Maroc avec son parcours, le règlement de la phase finale.



Génération de la réponse

Le Maroc jouera son prochain match de la phase finale le [Date] à [Heure] contre [Adversaire] au [Stade].

Traçabilité

Les métadonnées des documents sources sont conservées, permettant de savoir d'où provient l'information.

Une logique IA cohérente, guidée par des principes de conception clairs

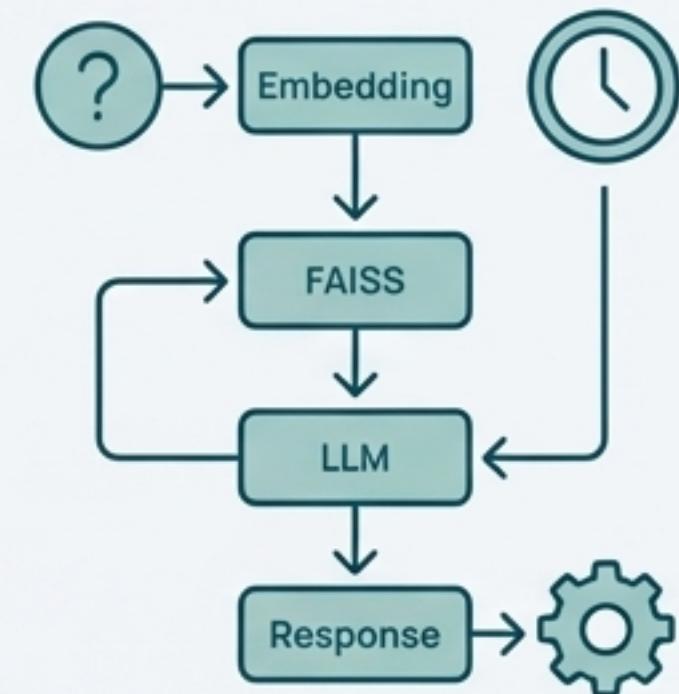
Principes de Design IA

| Principe | Implémentation | Avantage |
|---------------------------|--|---|
| Factualité | RAG avec données JSON vérifiées | Aucune hallucination possible |
| Déterminisme | `temperature=0` | Réponses stables et reproductibles |
| Contexte enrichi | `k=20` documents pertinents | Compréhension profonde de la question |
| Français natif | Prompt Engineering + LLM multilingue | Réponses fluides et naturelles |
| Normalisation | Système d'alias d'équipes (MAR → Maroc) | Flexibilité des requêtes utilisateur |
| Traçabilité ("Grounding") | Documents source inclus dans les métadonnées | Auditabilité et vérification des réponses |

Des performances mesurées pour une expérience utilisateur en temps réel

Performance et Ressources

| Métrique | Valeur | Notes |
|-----------------------------------|----------------------|---|
| Temps d'indexation | ~30 secondes | Exécuté une seule fois (offline) |
| Temps total par requête | ~2-3 secondes | De la question à la réponse affichée |
| <i>Détail du temps de requête</i> | | |
| ↳ Embedding question | ~50 ms | Sentence Transformers (local) |
| ↳ Récupération FAISS | ~5 ms | Recherche <i>nearest neighbor</i> |
| ↳ Génération LLM | ~1-2 sec | Appel API Gemini |
| Mémoire (runtime) | ~500 MB | Chargement des embeddings et de l'index |
| Espace disque | ~50 MB | Fichier `index.faiss` |



Un déploiement simple et une configuration centralisée

Configuration & Démarrage

Manrope Semibold, #005F73

Configuration

- Toutes les constantes sont gérées dans `config.py` pour éviter le 'hardcoding'.
- La clé d'API est sécurisée via un fichier `.env`, séparant le code de la configuration.

Étapes de Démarrage Rapide

Manrope Semibold, #0A9396

1. `python embeddings.py`
 - *Créer l'index FAISS (une seule fois).* Public Sans Italic, #212529
2. `uvicorn rag_chain:app --reload`
 - *Lancer le backend FastAPI.* Public Sans Italic
3. `npm start`
 - *Lancer le frontend React.* Public Sans Italic

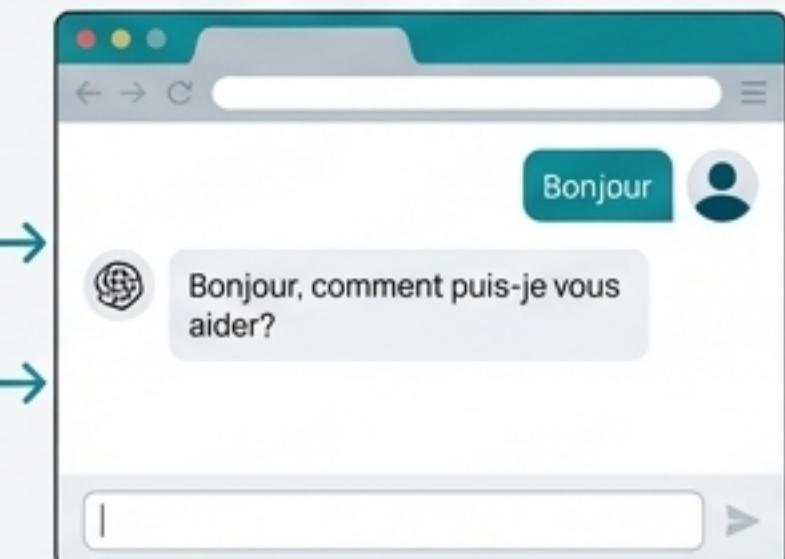
Diagramme de Déploiement

Manrope Semibold, #005F73

Terminal 1: Backend

```
$ uvicorn rag_chain:app --reload
```

Browser: localhost



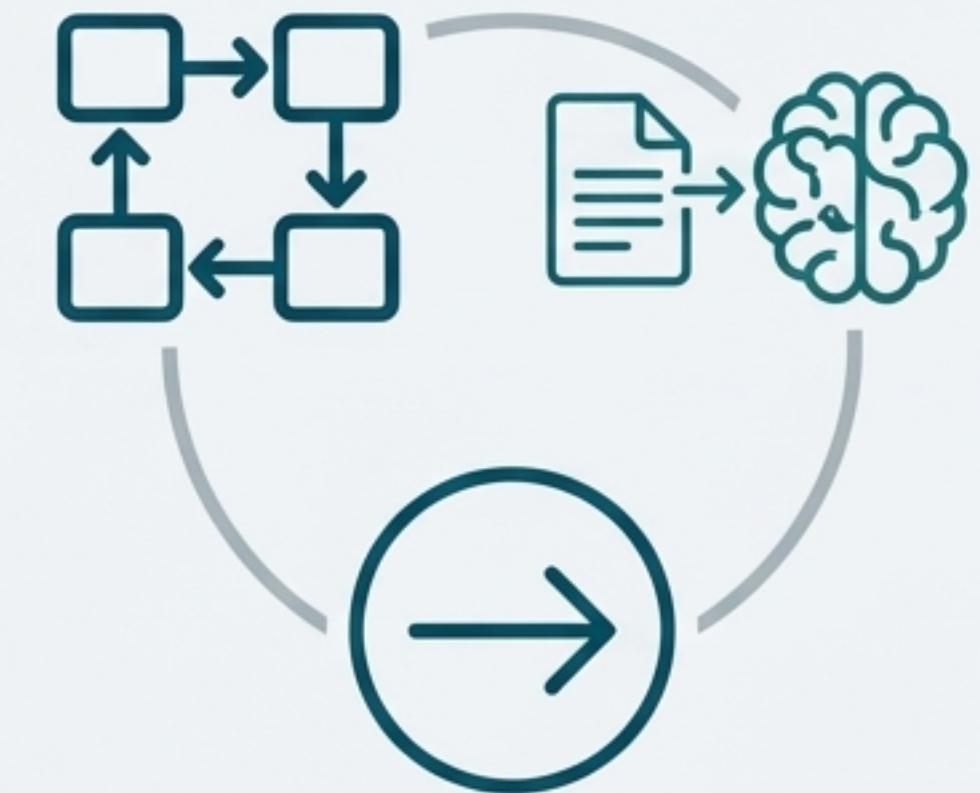
Terminal 2: Frontend

```
$ npm start
```

Résumé de l'architecture ASSAD AI

Les Points Clés de l'Architecture

- **Architecture client-serveur** découplée (React ↔ FastAPI).
- **Pipeline RAG complet** assurant la factualité des réponses.
- **Embeddings sémantiques** (`all-MiniLM-L6-v2`) pour une compréhension fine du langage.
- **Index vectoriel FAISS** pour une recherche de similarité quasi-instantanée.
- **LLM déterministe** (`temperature=0`) pour des réponses fiables et reproductibles.
- **Contexte enrichi** (`k=20`) pour une compréhension complète des requêtes.
- **Métadonnées riches** pour un filtrage et une traçabilité avancés.



Vue d'Ensemble de l'Architecture

Cette architecture garantit une logique IA cohérente, factuelle et performante, spécifiquement adaptée aux exigences de la CAN 2025.