Objectif Global : Conception et implémentation d'un système RAG (Retrieval-Augmented Generation) utilisant des composants locaux (Ollama, PostgreSQL/pgvector) pour interroger un corpus de documents PDF.

État Actuel : Les pipelines d'indexation des documents et de récupération d'informations sont fonctionnels et testés. L'intégration de la génération de réponse finale est en cours de finalisation.

A. Fonctionnalité Administrateur : Indexation de PDF

- **Objectif :** Permettre l'upload de fichiers PDF pour traitement (découpage en chunks, génération d'embeddings via Ollama) et stockage/mise à jour dans la base vectorielle pgvector.
- **Endpoint API :** POST /api/admin/ingest
- **Requête :**
    - Type : multipart/form-data
    - Paramètres :
        - file : Le fichier PDF (obligatoire).
        - chunkSize : Taille des chunks (entier, optionnel, défaut actuel 500).
- **Réponse (Format JSON) :**
    - Structure : EmbeddingSaveResponse
    - Contenu Exemple : { "message": "Indexation complète réussie pour 'nom_fichier.pdf'.", "sourceId": "nom_fichier.pdf", "chunksProcessed": N, "embeddingsSaved": N, "savedIds": [id1, id2, ...] }
- **Composants Backend Impliqués :**

1. **AdminIngestionController :** Point d'entrée API REST, valide la requête, délègue à AdminIndexingService.
2. **AdminIndexingService :** Service principal orchestrant l'indexation (annoté @Transactional).
    - Supprime les enregistrements précédents pour le même sourceId via ChunkEmbeddingRepository.deleteBySourceId().
    - Appelle PreprocessingService pour le découpage du PDF.
    - Appelle EmbeddingService2 pour générer les embeddings via Ollama (nomic-embed-text).
    - Appelle EmbeddingPersistenceService pour la sauvegarde finale.
3. **PreprocessingService :** Utilise **Apache PDFBox** pour extraire le texte et le découper en chunks selon chunkSize.

4. **EmbeddingService2 :** Appelle l'API /api/embeddings d'**Ollama** (local) via RestTemplate pour obtenir les vecteurs float[] du modèle **nomic-embed-text**.

5. **EmbeddingPersistenceService :** Convertit les données en entités ChunkEmbedding et appelle ChunkEmbeddingRepository.saveAll() (annoté @Transactional).

6. **ChunkEmbeddingRepository :** Interface **Spring Data JPA** gérant l'accès à la table chunk_embeddings. Fournit saveAll(), deleteBySourceId(), et les méthodes de recherche.

7. **ChunkEmbedding (Entity) :** Classe JPA représentant la table chunk_embeddings avec une colonne de type vector gérée par **pgvector**.

- **Flux Technique Admin :**
  PDF (Upload) → AdminIngestionController → AdminIndexingService → [deleteBySourceId → PreprocessingService → EmbeddingService2 (Ollama Embed) → EmbeddingPersistenceService → saveAll (pgvector)] → Réponse JSON

B. Fonctionnalité Client : Chat RAG (Basé sur Ollama Local)

- **Objectif :** Fournir des réponses générées par un **LLM Ollama local (Llama 3)**, réponses basées sur les informations pertinentes extraites des documents PDF indexés dans pgvector.
- **Endpoint API :** POST /api/rag/query
- **Requête (Format JSON) :**
  - Structure : RagQueryRequest
  - Contenu Exemple : { "question": "Quelle est la question de l'utilisateur ?" }
- **Réponse (Format JSON) :**
  - Structure : RagQueryResponse
  - Contenu Exemple : { "answer": "La réponse générée par Llama 3...", "retrievedSources": [{chunk1...}, {chunk2...}] }
- **Composants Backend Impliqués :**

1. **RagController :** Point d'entrée API REST, valide la requête, délègue à RagService.

2. **RagService :** Service principal orchestrant le flux RAG.
   - Appelle RetrievalService pour obtenir les chunks pertinents (SearchResult).
   - Construit le prompt final (instructions, contexte récupéré, question).
   - Appelle l'API /api/chat d'**Ollama** (local) via RestTemplate en spécifiant le modèle **llama3** (configurable via application.properties).
   - Extrait la réponse textuelle du JSON retourné par Ollama.

- Construit et retourne le RagQueryResponse.

3. **RetrievalService :** Service responsable de la récupération.
    - Appelle EmbeddingService2 pour obtenir l'embedding de la question (Ollama nomic-embed-text).
    - Appelle ChunkEmbeddingRepository.findSimilarChunksByCosineDistance() p our interroger **pgvector** via une requête SQL native (<=>).
    - Retourne le SearchResult avec les RetrievedChunk et leurs scores.

4. **EmbeddingService2 :** (Utilisé par Retrieval) Appelle Ollama /api/embeddings (nomic-embed-text).

5. **ChunkEmbeddingRepository :** (Utilisé par Retrieval)
Fournit findSimilarChunksByCosineDistance().

6. **RestTemplate, ObjectMapper, VectorUtils :** Beans/Utilitaires utilisés pour les appels API et la manipulation des données.

7. **DTOs**
: RagQueryRequest, RagQueryResponse, SearchRequest, SearchResult, RetrievedChunk.

- **Flux Technique RAG :**
  Question (JSON) → RagController → RagService → [RetrievalService → [EmbeddingService2 (Ollama Embed) → ChunkEmbeddingRepository (pgvector Search)] → Contexte] → Construction Prompt → Appel LLM (RestTemplate → Ollama Chat llama3) → Réponse JSON

Technologies Clés Utilisées :

- **Backend :** Java (JDK 21), Spring Boot 3.x, Spring Data JPA, Spring Web
- **Traitement PDF :** Apache PDFBox
- **Embedding & Génération LLM :** Ollama (local) avec nomic-embed-text et llama3
- **Base de Données Vectorielle :** PostgreSQL + pgvector (via Docker)
- **Communication API :** REST (Spring Web), RestTemplate
- **Conteneurisation :** Docker / Docker Desktop
- **Build :** Maven

État Actuel : Les deux flux principaux (Indexation Admin et Requête RAG Client avec LLM local) sont implémentés et fonctionnels. Les prochaines étapes concerneront l'intégration avec les interfaces utilisateur et potentiellement des optimisations/améliorations