UNIVERSITÉ CADI AYYAD

ÉCOLE NATIONALE DES SCIENCES APPLIQUÉES

GÉNIE INFORMATIQUE

Service de Recherche Sémantique de Plantes et Maladies

Service FLORA-BOT du projet SuperEE

Réalisé par :

Hmiddouch Abdessadek
AHMAMO Hamza
ER-RETBY Abdelilah
AOURIR Aissam

Encadré par :

Prof. Bouzid Sara

Rapport de Projet

Année universitaire : 2025-2026

14 décembre 2025

Résumé

Ce projet présente la conception et le développement d'un moteur de recherche sémantique avancé, spécifiquement dédié à l'identification des types de plantes et à la détection des maladies qui leur sont associées. L'objectif principal est de permettre aux utilisateurs, qu'ils soient chercheurs, agronomes ou amateurs, de formuler des requêtes en langage naturel, ou même de fournir des descriptions d'images et de textes, afin d'obtenir des résultats pertinents et précis. Contrairement aux moteurs de recherche traditionnels basés sur la correspondance exacte de mots-clés, ce système repose sur l'analyse sémantique des requêtes et des données, exploitant la vectorisation automatique des contenus et la mesure de la similarité cosinus pour retrouver les éléments les plus proches du sens de la requête. Cette approche permet de capturer des relations implicites entre les différents types de plantes et leurs maladies, offrant ainsi une recherche plus intelligente, flexible et adaptée aux besoins réels des utilisateurs.

Mots-clés : Recherche sémantique, Plantes et maladies, Traitement du langage naturel, FastAPI, React, Elasticsearch, Llama 3.2, HNSW, Indexation vectorielle, Vectorisation automatique, Similarité cosinus, Moteur de recherche intelligent, Requêtes en langage naturel, PDF pertinents, Récupération de documents.

Table des matières

1 Introduction

Dans le cadre du projet global **SuperEYE Search Engine**, le présent travail porte sur la conception et le développement du **service FLORA-BOT (Semantic Search Engine)**, un service dédié à la recherche sémantique et à l'exploitation intelligente de documents techniques liés aux plantes et aux maladies qui leur sont associées. Ce service constitue un composant clé du système, permettant d'interroger automatiquement des sources expertes afin d'enrichir les fonctionnalités globales du projet.

Ce projet présente ainsi la conception et le développement d'un moteur de recherche sémantique avancé, dédié à l'identification des types de plantes et à la détection des maladies associées. L'objectif principal est de permettre aux utilisateurs, qu'ils soient chercheurs, agronomes ou amateurs, de formuler des requêtes en langage naturel ou de fournir des descriptions textuelles, afin d'obtenir des résultats pertinents et précis.

Contrairement aux moteurs de recherche traditionnels, basés sur la correspondance exacte de mots-clés, ce système repose sur l'analyse sémantique des requêtes et des données, exploitant la vectorisation automatique des contenus, la mesure de la similarité cosinus, ainsi que des algorithmes de recherche approximative basés sur HNSW. Le moteur ne se limite pas à l'identification des plantes et maladies : il est également capable de retourner les documents PDF pertinents, chacun accompagné d'un score de pertinence, enrichissant ainsi les réponses et offrant aux utilisateurs des informations détaillées et contextualisées.

Cette approche permet de capturer des relations implicites entre les différentes plantes, maladies et documents associés, offrant ainsi une recherche plus intelligente, flexible et adaptée aux besoins réels des utilisateurs.

Le rapport présente dans un premier temps la méthodologie adoptée, incluant les choix techniques pour l'indexation, la vectorisation et la recherche sémantique. Nous détaillerons ensuite l'architecture du système, les technologies utilisées (FastAPI, React, Elasticsearch, Llama 3.2, HNSW), et enfin nous présenterons une évaluation des performances ainsi que les perspectives d'amélioration du moteur.

2 Présentation du Sujet

2.1 Thématique du Moteur de Recherche

Le projet porte sur le développement d'un moteur de recherche sémantique dédié au domaine des plantes et de leurs maladies associées, avec la possibilité de retourner des documents PDF pertinents en complément des résultats principaux.

Contrairement aux moteurs de recherche traditionnels, qui se basent uniquement sur la correspondance exacte de mots-clés, ce système utilise des techniques avancées de traitement du langage naturel et de recherche d'information pour comprendre le sens des requêtes formulées par l'utilisateur, qu'il s'agisse de textes ou d'images.

Le moteur est capable de retrouver non seulement les plantes et maladies pertinentes, mais aussi les documents PDF associés, offrant ainsi des informations plus complètes et détaillées. Cette approche permet de capturer les relations conceptuelles et implicites entre les différentes

entités et les documents, garantissant une recherche plus précise, adaptée et enrichissante pour les utilisateurs.

2.2 Fonctionnalités Principales

Le système développé offre deux fonctionnalités principales :

Indexation : Cette étape comprend plusieurs sous-processus :

— Chunking du texte à l'aide de NLTK,

— Génération d'embeddings via Sentence Transformer,

— Indexation dans Elasticsearch en utilisant HNSW pour la recherche vectorielle et le calcul de similarité.

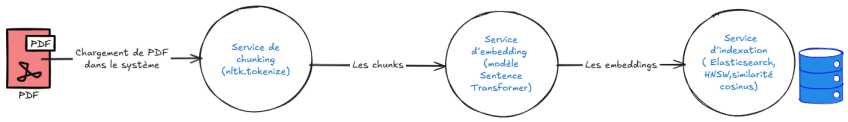

FIGURE 1 – Illustration des étapes de l'indexation

Recherche : Cette fonctionnalité permet de récupérer les documents pertinents en comparant les embeddings de la requête avec ceux des documents indexés.

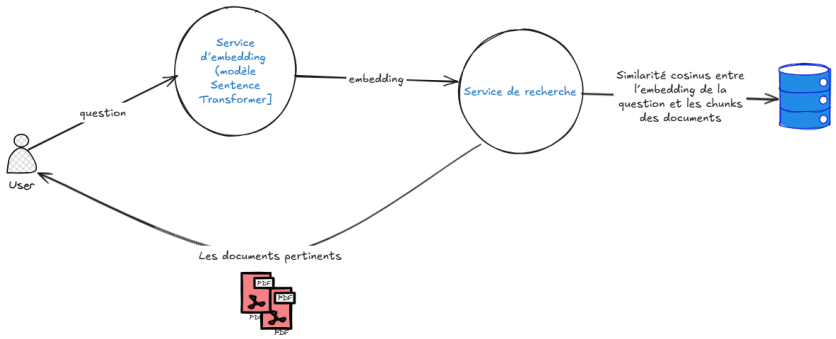

FIGURE 2 – Illustration des étapes de la recherche

2.3 Types de Ressources et Formats

Le système traite plusieurs types de ressources :

1. **Données multimédias** :
   — Documents PDF complets stockés sur le serveur ontenant les informations sur les plantes et maladies(`assets/docs`)
2. **Données dérivées pour la recherche** :
   — Extraction des contenus des PDF en **chunks** stockés sous forme de JSON pour l'indexation
   — Vecteurs d'embeddings générés par le modèle Sentence Transformer (all-mpnet-base-v2) à partir des descriptions textuelles et des chunks PDF
   — Indexation vectorielle via HNSW dans Elasticsearch pour permettre une recherche sémantique rapide et précise

— Métadonnées associées aux chunks (origine, titre du PDF, etc.) pour retrouver les documents complets

3. **Format d'échange** :
   — API REST exposée via FastAPI utilisant JSON pour les requêtes et réponses
   — Structure normalisée pour les messages d'erreur et les résultats de recherche

3 Processus d'Indexation et de Recherche

3.1 Indexation : Automatique sans Prétraitement Linguistique Avancé

Dans ce système de recherche d'information, l'indexation est réalisée de manière automatique et repose principalement sur des représentations vectorielles sémantiques (embeddings) générées à partir du contenu textuel des documents. Contrairement aux approches classiques de recherche lexicale (basées sur la racinisation, la lemmatisation ou la suppression des stop-words), aucune étape de prétraitement linguistique avancé n'est appliquée. Le sens des textes est directement capturé par le modèle d'embeddings.

L'indexation concerne des documents PDF, qui sont découpés en segments textuels (chunks), transformés en vecteurs numériques, puis stockés dans un index Elasticsearch configuré pour la recherche vectorielle.

3.1.1 Choix d'Indexation

— **Type** : Indexation automatique
— **Moteur d'indexation :** Elasticsearch, choisi pour sa robustesse, sa scalabilité et son support natif de la recherche vectorielle.
— **Mesure de similarité :** similarité cosinus (cosine similarity), adaptée à la comparaison de vecteurs normalisés issus de modèles de type Transformer.
— **Algorithme de recherche approximative :** HNSW (Hierarchical Navigable Small World), permettant une recherche rapide et efficace dans des espaces vectoriels de grande dimension.

Les paramètres HNSW retenus sont :
— **m = 32** : nombre maximal de connexions par nœud, assurant un bon compromis entre précision de la recherche et consommation mémoire.
— **ef_construction = 100** : paramètre contrôlant la qualité du graphe lors de la phase de construction de l'index, améliorant la précision des résultats au prix d'un temps de construction légèrement plus élevé.

3.1.2 Méthode d'Indexation

La méthode d'indexation repose sur un processus automatique en trois étapes principales : le découpage des documents, la génération des embeddings et l'indexation vectorielle dans Elasticsearch.

— **Découpage des documents (Chunking)** : Les documents PDF sont d'abord transformés en texte, puis découpés en segments sémantiquement cohérents (*chunks*). Cette étape

est réalisée par la fonction `chunk_by_sentences_with_smart_tools()`, qui s'appuie sur les éléments suivants :

- `clean_text()` pour le nettoyage du texte ;
- `sent_tokenize()` (NLTK) pour la segmentation en phrases ;
- `find_sentence_continuation()` pour reconstituer les phrases coupées entre deux pages ;
- `get_smart_text_start_improved()` et `get_smart_text_end_improved()` pour éviter les coupures non naturelles.

Ce découpage respecte le sens du texte, limite la fragmentation et assure une bonne continuité contextuelle entre les *chunks*.

— **Génération des embeddings** : Chaque *chunk* est ensuite converti en un vecteur numérique dense à l'aide de la fonction `embed()`. Cette fonction utilise le modèle Sentence Transformer (*all-mpnet-base-v2*), qui encode chaque segment textuel en un vecteur de dimension 768 capturant sa signification sémantique.

— **Indexation dans Elasticsearch avec HNSW** : Avant l'indexation, l'index Elasticsearch est créé via la fonction `create_index_if_not_exists()`. Les *chunks* vectorisés sont ensuite stockés à l'aide de la fonction `index_pdf()`, avec un *mapping* incluant un champ `dense_vector` configuré pour :

- la similarité cosinus ;
- l'algorithme HNSW (`index_options : type = hnsw`).

L'algorithme HNSW (Hierarchical Navigable Small World) utilisé par Elasticsearch permet d'optimiser la recherche de similarité vectorielle en organisant les embeddings sous forme de graphes hiérarchiques. Comme illustré dans l'image ci-dessous, la recherche commence par un point d'entrée situé dans les couches supérieures, peu denses, afin de se rapprocher rapidement de la zone pertinente. Ensuite, la navigation descend progressivement vers des couches plus denses, où l'algorithme explore les voisins les plus proches du vecteur de requête. Cette approche hiérarchique réduit considérablement le nombre de comparaisons nécessaires, tout en maintenant une excellente précision, ce qui rend la recherche beaucoup plus rapide et efficace que les méthodes exhaustives.

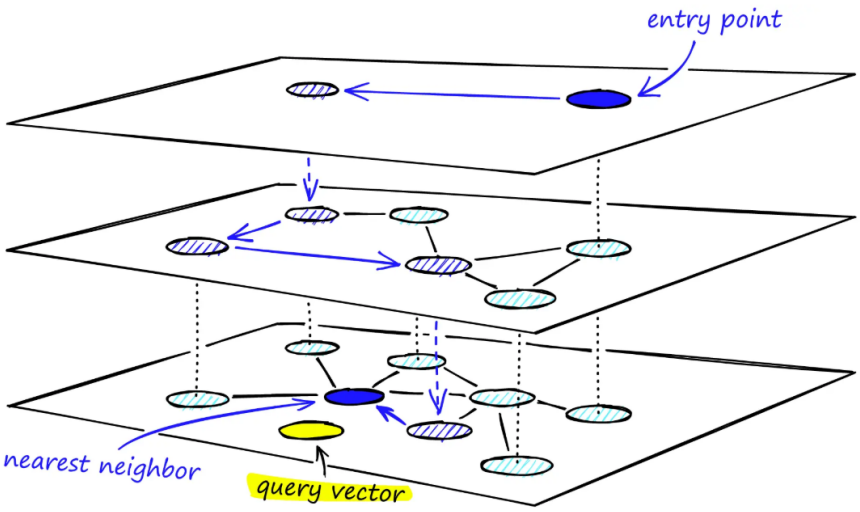

FIGURE 3 – Illustration du fonctionnement de l'algorithme HNSW pour la recherche de similarité vectorielle

3.1.3 Justification des choix

Les choix effectués pour l'indexation sont directement liés aux objectifs du système :

- **Absence de prétraitement linguistique avancé :** les modèles d'embeddings modernes capturent directement le sens du texte, réduisant le besoin de traitements linguistiques complexes et dépendants de la langue.
- **Embeddings sémantiques :** ils permettent de retrouver des passages pertinents même lorsque les termes de la requête ne correspondent pas exactement aux mots du document.
- **Similarité cosinus :** largement utilisée et efficace pour comparer des vecteurs de grande dimension.
- **HNSW :** offre d'excellentes performances en temps de réponse, ce qui est essentiel pour un système de recherche interactif.
- **Indexation par segments :** améliore la précision des résultats et facilite l'exploitation des documents longs.

3.2 Stockage :

Les segments de texte sont stockés dans Elasticsearch avec :

- un identifiant de document (doc_id) et de segment (chunk_id),
- les vecteurs dans un champ dense_vector utilisant la similarité cosinus,
- l'algorithme HNSW pour organiser les embeddings dans un graphe hiérarchique, permettant une recherche rapide et précise,
- la source du document pour retrouver l'origine des chunks.

Cette organisation assure un stockage structuré et performant pour la recherche vectorielle.

FIGURE 4 – Structure de stockage des embeddings dans Elasticsearch avec HNSW

3.3 Recherche : Modèle Vectoriel avec Similarité Cosinus

Le modèle de recherche d'information adopté est basé sur la similarité vectorielle :

3.3.1 Modèle de RI

- **Type** : Recherche vectorielle sémantique
- **Similarité** : Cosinus entre vecteurs normalisés
- **Encodage** : Sentence Transformer `(all-mpnet-base-v2 )`

Chaque requête et chaque segment de document (chunk) sont encodés en vecteurs d'embedding. La recherche se fait en comparant ces vecteurs dans l'espace sémantique.

Voici le format de la requête utilisée :

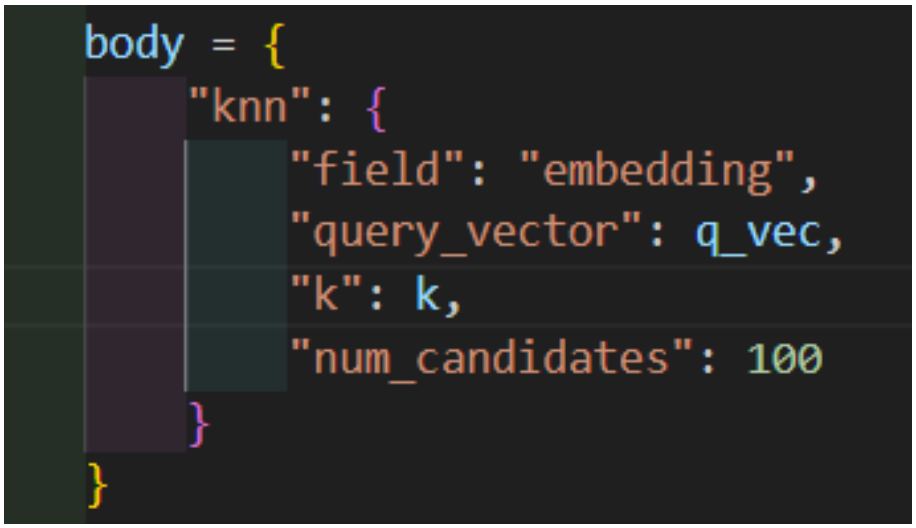

FIGURE 5 – Format de la requête KNN utilisée pour la recherche vectorielle dans Elasticsearch

3.3.2 Formule de Matching

La similarité entre une requête q et une recette r est calculée comme suit :

$$similarit(q, r) = \frac{\mathbf{v}_q \cdot \mathbf{v}_r}{\|\mathbf{v}_q\|\|\mathbf{v}_r\|}$$

Où :
— $\mathbf{v}_q$: Vecteur d'embedding de la requête
— $\mathbf{v}_r$: vecteur d'embedding du document
— $\cdot$: Produit scalaire
— $\|\cdot\|$: Norme euclidienne

3.3.3 Flux de Recherche

Le processus de recherche dans le système sémantique se déroule selon les étapes suivantes :

1. **Récupération de la requête** : La question ou la description saisie par l'utilisateur est extraite afin d'être traitée par le système.
2. **Encodage sémantique** : La requête est convertie en un vecteur d'embedding à l'aide du modèle Sentence Transformer (*all-mpnet-base-v2*).
3. **Calcul de similarité** : Le vecteur de la requête est comparé aux embeddings des *chunks* issus des documents PDF. Dans le graphe HNSW :
   — à chaque niveau, le vecteur de la requête est comparé aux 100 *chunks* les plus proches ;
   — les k *chunks* les plus pertinents sont retenus à chaque niveau ;
   — cette opération est répétée sur l'ensemble des niveaux du HNSW afin d'obtenir une sélection finale des k *chunks* les plus proches.
4. **Retour des documents pertinents** : Les k *chunks* les plus pertinents sont retournés. À partir de ces segments, les documents d'origine correspondants sont identifiés, permettant de présenter à l'utilisateur les documents les plus proches de sa requête.

3.4 Classement des Résultats

3.4.1 Méthode de Classement

— **Critère principal** : Score de similarité cosinus entre la requête utilisateur et les embeddings des descriptions et des chunks PDF
— **Direction** : Tri décroissant (du document le plus pertinent au moins pertinent)
— **Retour des résultats** : Chaque document PDF ou chunk renvoyé est accompagné de son score de pertinence

3.4.2 Caractéristiques du Classement

1. **Uniformité** : Tous les champs textuels et chunks PDF contribuent également à l'embedding
2. **Sémantique pure** : Le classement se base uniquement sur la similarité sémantique, sans pondération par fréquence ou position des termes
3. **Indépendance lexicale** : Les synonymes, paraphrases et variations de formulation sont correctement gérés grâce aux embeddings
4. **Continuité** : Les scores de similarité sont continus, permettant un classement fin et précis des documents et chunks PDF
5. **Pertinence documentée** : Chaque document retourné est accompagné de son score de pertinence pour informer l'utilisateur

4 Choix Technologiques

Le système utilise un ensemble de technologies modernes et éprouvées :

4.1 Backend et API

— **Framework Web** : FastAPI 0.100.x
  — *Justification* : Léger, moderne, performant pour les APIs REST, et bien intégré avec Python
  — *Fonctionnalités utilisées* : Routage, gestion des requêtes, validation des données, sérialisation JSON
— **Gestion CORS** : FastAPI Middleware
  — *Justification* : Permet l'accès depuis des applications web externes, notamment la partie frontend en React

4.2 Conteneurisation et Déploiement

— **Outil de conteneurisation** : Docker
  — *Justification* : Assure l'isolation des dépendances, la portabilité du service FLORA-BOT et la reproductibilité de l'environnement d'exécution.
  — *Usage dans le projet* :
    — Conteneurisation de l'API FastAPI

    - Déploiement du moteur Elasticsearch
    - Exécution du serveur Ollama pour Llama 3.2
  - *Avantages* :
    - Déploiement simplifié du service FLORA-BOT
    - Intégration fluide au projet global **SuperEE**
    - Réduction des problèmes liés aux dépendances et à la configuration

4.3 Traitement du Langage Naturel

- **Modèle d'Embeddings** : Sentence Transformer `( all-mpnet-base-v2 )`
  - *Caractéristiques* : Capable de générer des embeddings vectoriels pour le texte et le contenu extrait des PDF
  - *Avantages* : Haute qualité sémantique, adapté à la recherche par similarité
  - *Justification* : Permet de comparer efficacement les requêtes utilisateurs avec les descriptions des plantes et maladies, ainsi qu'avec les chunks PDF
- **Modèle LLM** : Llama 3.2
  - *Caractéristiques* :Modèle de langage de grande taille capable de générer des réponses en langage naturel à partir d'un contexte fourni
  - *Avantages* : Génération de réponses cohérentes, contextualisées et pertinentes
  - *Justification* : Exploite le contexte composé des k *chunks* les plus pertinents retournés par le moteur de recherche sémantique afin de produire des réponses précises et informatives pour l'utilisateur

4.4 Base de Données Vectorielle

- **Moteur de Recherche** : Elasticsearch avec HNSW
  - **Index utilisé** : Index vectoriel pour les embeddings des textes et des chunks PDF
  - **Avantages** :
    - Recherche sémantique rapide et scalable
    - Supporte les mises à jour dynamiques et la recherche approximative
    - Intégration facile avec FastAPI

4.5 Traitement des Données

- **Extraction des chunks PDF** :
  - Découpage des PDF en morceaux cohérents pour indexation
  - Stockage des chunks sous forme JSON pour l'indexation dans Elasticsearch
- **Manipulation JSON** : Module json standard de Python
  - Chargement des fichiers source (plantes, maladies, chunks PDF)
- **Calcul Numérique et Normalisation** : NumPy 1.24.x
  - Normalisation des vecteurs pour calcul de similarité cosinus

4.6 Justification des Choix Architecturaux

1. **Séparation des préoccupations** :

Architecture du Backend

Le backend est organisé de manière modulaire afin que chaque composant ait une responsabilité bien définie :

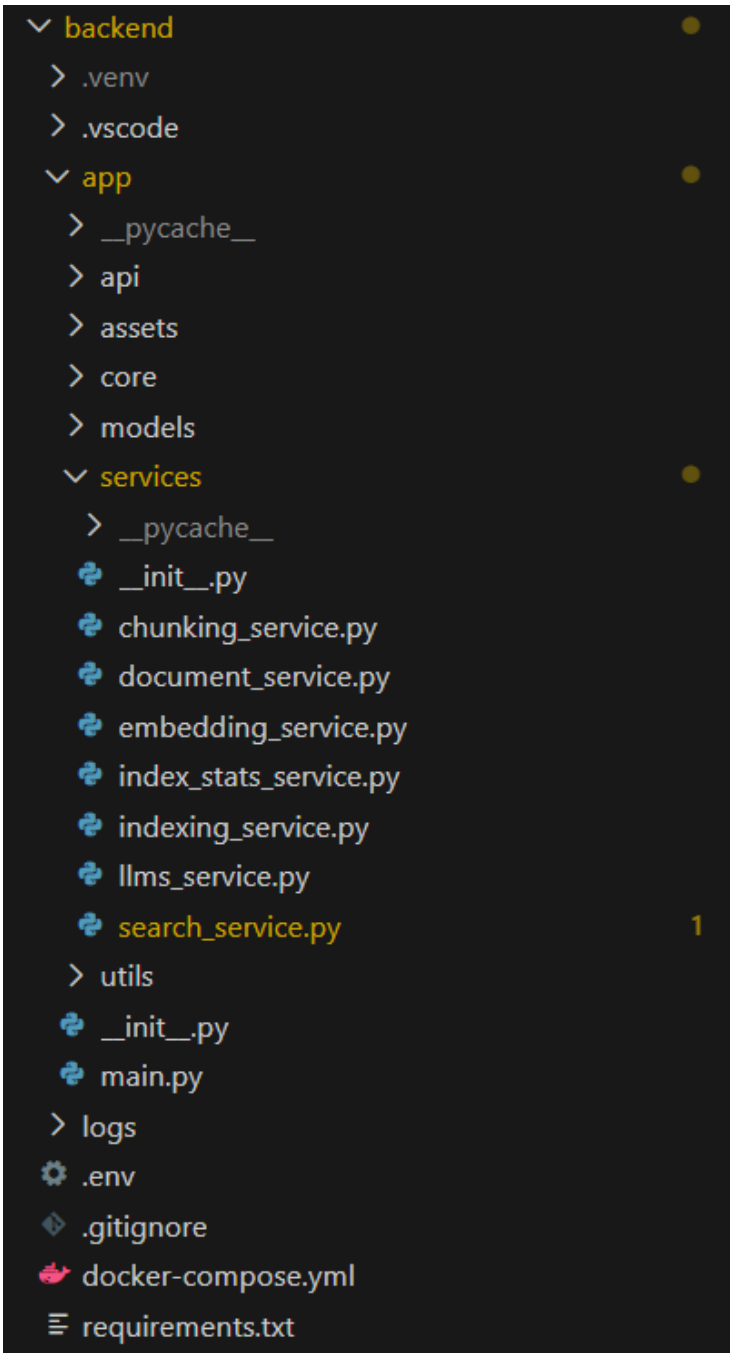

FIGURE 6 – Schéma de l'architecture Backend

— **api**/ : Contient les endpoints FastAPI.
— **core**/ : Gère la configuration générale et la gestion des exceptions.
— **models**/ : Définit les schémas de données et assure la validation.
— **services**/ : Regroupe les différents services, tels que le chunking, l'embedding et l'indexation.
— **utils**/ : Fournit des fonctions utilitaires réutilisables dans tout le projet.

— **assets/** : Contient les ressources statiques nécessaires à l'application.

Architecture du Frontend

Pour structurer notre frontend de manière modulaire, nous avons adopté deux pages essentielles :

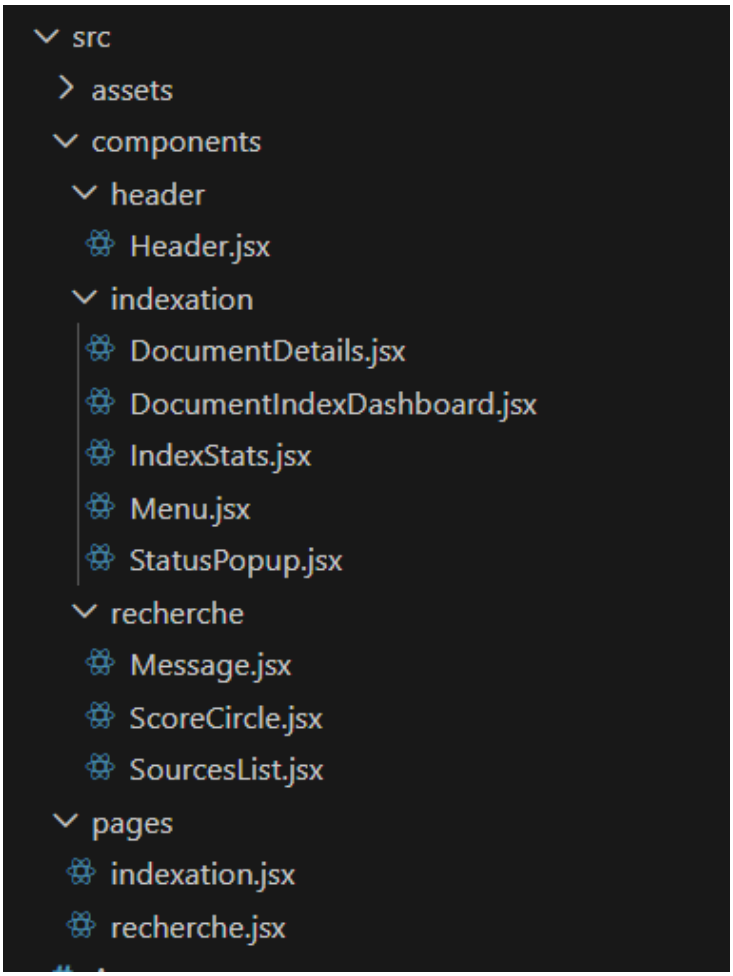

FIGURE 7 – Schéma de l'architecture Frontend

— **recherche** :
  — Permet d'effectuer des recherches et d'afficher les résultats avec leurs sources et scores.
— **indexation** :
  — Permet de télécharger et d'indexer de nouveaux documents, ainsi que de visualiser les documents déjà indexés.

2. **Optimisation des performances** :
   — Pré-calcul des embeddings pour le texte et les chunks PDF
   — Indexation dans Elasticsearch pour une recherche rapide
   — Recherche vectorielle via HNSW pour l'approximate nearest neighbors
   — Déploiement conteneurisé permettant une exécution stable et reproductible
3. **Maintenabilité** :

— Code modulaire et séparé par responsabilités
— Configuration externe des paramètres (chemins, modèles, seuils de similarité)
— Logs détaillés pour le suivi et le debug

4. **Extensibilité** :
   — API REST standard pour intégration avec d'autres services
   — Architecture prête pour ajouter d'autres types de ressources (images, vidéos, données externes)
   — Modularité pour remplacer facilement le modèle d'embeddings ou le moteur de recherche

5 Réalisation et Évaluation

5.1 Méthodologie d'Évaluation

Pour évaluer les performances du système de recherche, deux requêtes représentatives ont été sélectionnées et exécutées. Chaque résultat a été évalué en comparant la question de l'utilisateur avec la réponse générée par le LLM à partir des chunks les plus pertinents. Si la réponse est correcte, cela signifie que le système a identifié les documents correspondant correctement à la question.

5.2 Résultats des Requêtes Tests

5.2.1 Requête 1 : "What diseases can cause sudden wilting in plants, and how can they be identified ?"

reponse :

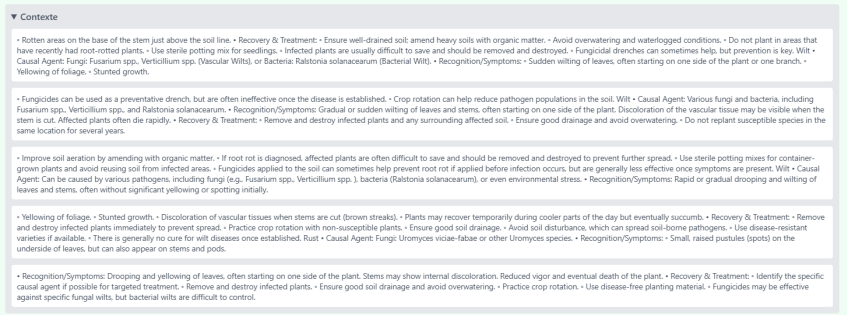

FIGURE 8 – Les 5 chunks retournés par le système de recherche les plus proches de la question

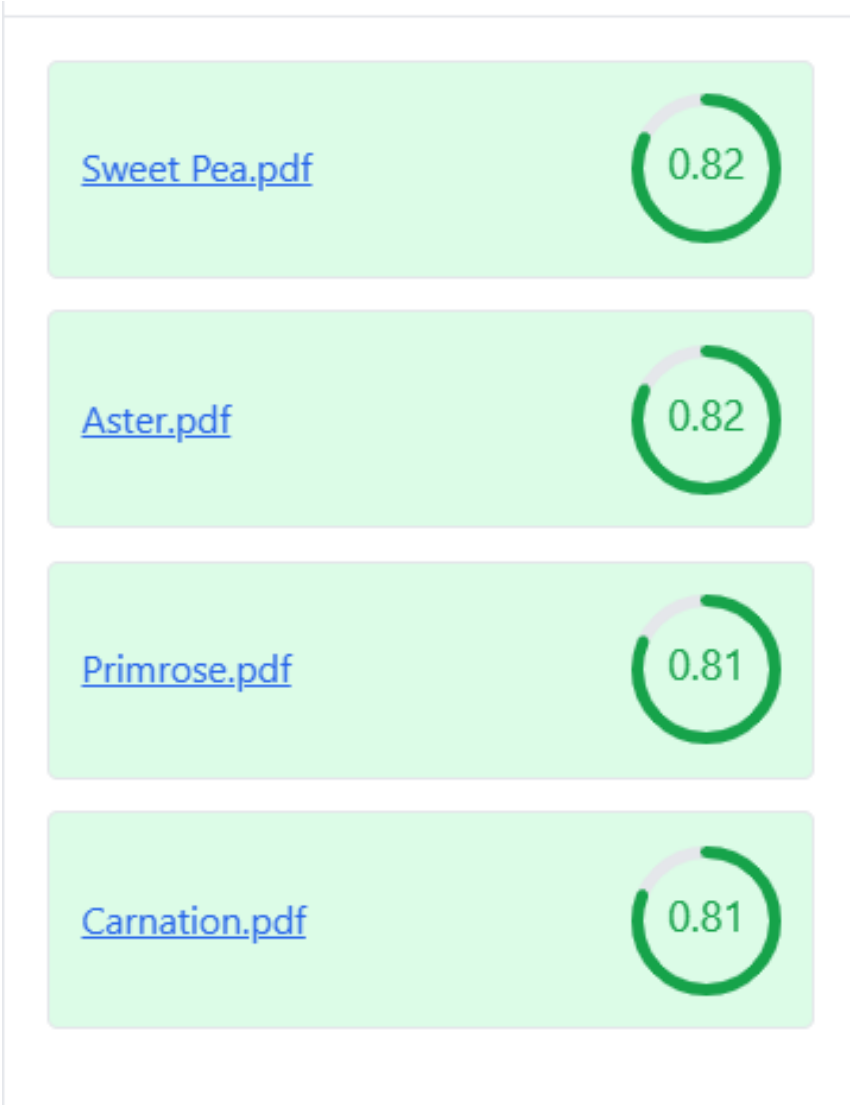

FIGURE 9 – Les documents correspondant aux chunks pertinents retournés avec leur score

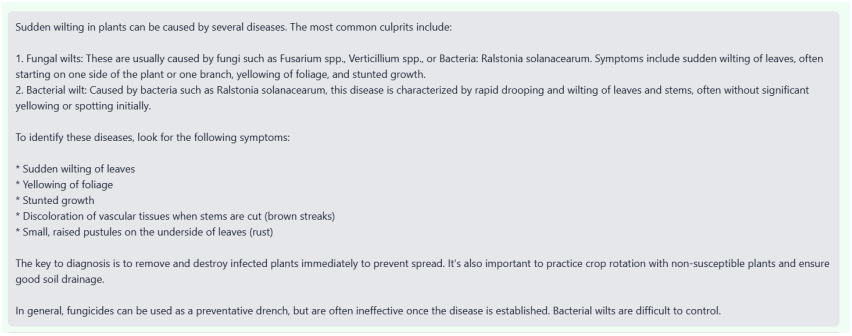

FIGURE 10 – Réponse générée par le LLM en se basant sur les chunks extraits comme contexte

Analyse : Le système retourne des documents avec un bon score, ce qui est justifié par le fait que la réponse générée par le LLM est cohérente et répond correctement à la question de l'utilisateur.

5.2.2 Requête 2 : "What is a Azalea ?"

reponse :

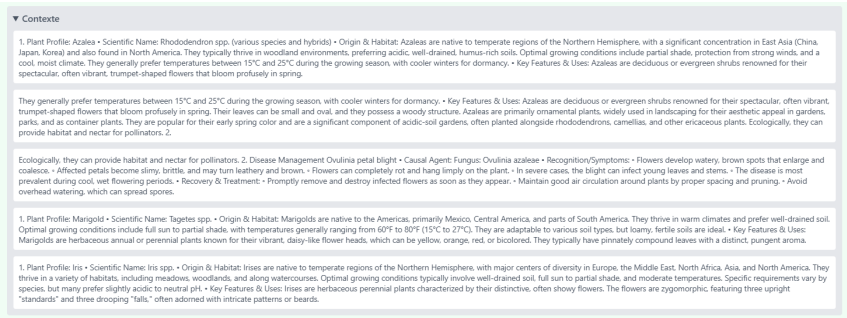

FIGURE 11 – Les 5 chunks retournés par le système de recherche les plus proches de la question

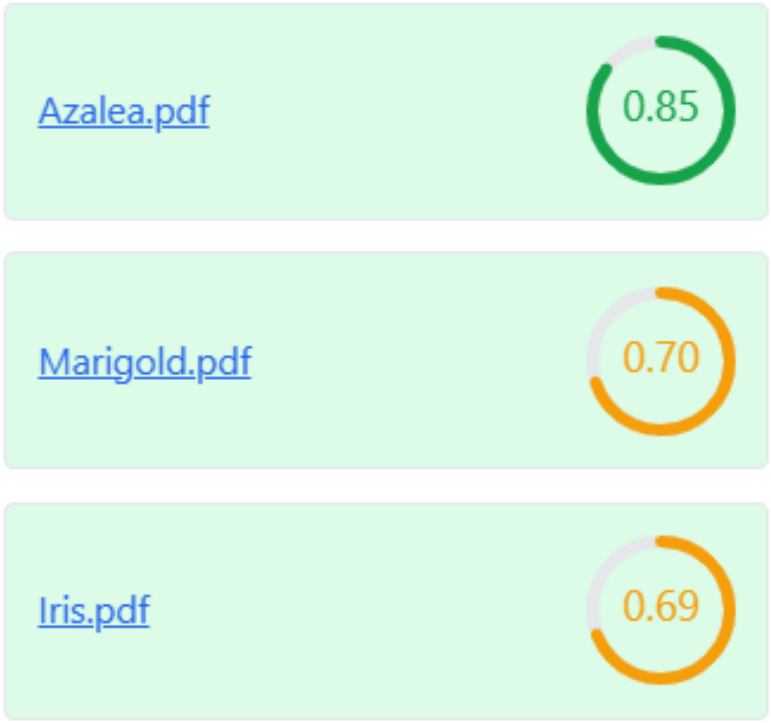

FIGURE 12 – Les documents correspondant aux chunks pertinents retournés avec leur score

FIGURE 13 – Réponse générée par le LLM en se basant sur les chunks extraits comme contexte

Analyse : Le système retourne un document pertinent, intitulé «Azalea.pdf», correspondant

à la question posée : «What is an Azalea ?». Il retourne également deux autres documents qui ne correspondent pas à la question et qui ont un score moyen. Dans l'ensemble, le système a réussi à identifier le document le plus pertinent.

5.3 Perspectives d'Amélioration

1. **Optimisation des embeddings** : Utiliser des modèles plus performants ou adaptés au domaine, éventuellement fine-tunés sur le corpus.
2. **Amélioration de l'index HNSW** : Ajuster les paramètres pour un meilleur équilibre précision/vitesse et gérer le re-indexing dynamique.
3. **Recherche hybride** : Combiner similarité cosinus et scores textuels (BM25) pour des résultats plus pertinents.
4. **Reranking et post-traitement** : Utiliser un LLM pour réordonner les résultats selon leur cohérence avec la question.
5. **Filtrage contextuel et pondération** : Ajouter des filtres (type de document, date) et pondérer les documents selon leur pertinence ou fiabilité.

6 Conclusion

Ce projet a permis de concevoir et d'implémenter un système de recherche sémantique dédié à l'identification des types de plantes et à la détection des maladies qui leur sont associées, avec la capacité de retourner des documents PDF pertinents. Le système exploite des techniques avancées de traitement du langage naturel via Llama 3.2 et une indexation vectorielle dans Elasticsearch utilisant HNSW pour la recherche rapide par similarité.

L'architecture choisie, combinant extraction de chunks depuis les PDFs, vectorisation automatique et recherche sémantique, démontre une approche efficace pour résoudre les limitations des moteurs de recherche traditionnels basés sur la correspondance exacte de mots-clés. Les principaux choix techniques, notamment l'utilisation des embeddings pour les textes et chunks PDF, ainsi que la similarité cosinus pour mesurer la pertinence, se sont avérés adaptés au domaine botanique et phytopathologique.

L'approche modulaire, incluant des pages distinctes pour le chatbot et pour l'indexation des données, offre une expérience utilisateur flexible tout en permettant la maintenance et l'extension du système.

Les perspectives d'amélioration incluent l'intégration de techniques de réordonnancement (reranking), l'apprentissage de seuils adaptatifs pour la pertinence, l'hybridation avec des méthodes de recherche lexicale pour couvrir un spectre plus large de requêtes, et l'ajout de nouveaux types de documents et médias.

Ce travail illustre comment les techniques avancées de recherche d'information et de traitement sémantique peuvent être appliquées à des domaines spécialisés pour offrir des résultats plus précis, des informations enrichies et une expérience utilisateur plus intuitive.