# Table des matières

# Chapitre 1

# Contexte du Projet

## 1.1 Projet de Fin d'Études

Ce projet s'inscrit dans le cadre de la validation de mon cursus de fin d'études en Génie Informatique à l'Ecole des Hautes Etudes d'Ingénierie , et a été réalisé au sein de l'entreprise Algolus, située à Oujda. Le stage a débuté le 3 Mars 2024, avec pour objectif principal la conception et le développement d'une application intelligente de détection et de correction d'erreurs logicielles basée sur une approche RAG multimodale.

Ce stage a constitué une excellente opportunité de mettre en pratique les connaissances acquises durant ma formation, tout en découvrant les réalités professionnelles du développement logiciel orienté intelligence artificielle.

## 1.2 Entreprise d'accueil


FIGURE 1.1 – Logo de Algolus

### 1.2.1 Description de l'entreprise

Algolus est une agence web marocaine, créée en 2020, spécialisée dans la conception et le développement de solutions informatiques adaptées aux besoins des clients. Elle s'engage à offrir à ses clients une communication en ligne efficace et sur mesure.

Ses prestations incluent :

— Création et gestion de sites web (dynamiques, statiques, e-commerce, CMS)

— Développement d'applications web (mode hybride)

— Stratégie digitale complète :

- Infographie
- Publicité en ligne
- Marketing digital
- Community management
- E-réputation

### 1.2.2 Fiche technique de l'entreprise

Le tableau ci-dessous récapitule la fiche technique de l'entreprise Algolus :

TABLE 1.1 – Fiche technique de l'entreprise Algolus

| | |
|---|---|
| **Dénomination sociale** | Algolus |
| **Date de création** | 07/10/2020 |
| **Forme juridique** | SARL |
| **Capital** | 100.000 Dh |
| **Chiffre d'affaires** | Indisponible |
| **Activités** | Développement informatique et marketing digital |
| **Effectif** | 10 |
| **Dirigeant** | Radwane BERAHIOUI |
| **Coordonnées** | +212 6644 35967<br>Redwan.Berahioui@algolus.ma<br>www.algolus.ma<br>IMMEUBLE OUASSIM, Bd Mohammed VI, Oujda 60000 |

### 1.2.3 Organigramme de l'entreprise

La figure ci-dessous présente l'organigramme de l'entreprise Algolus :

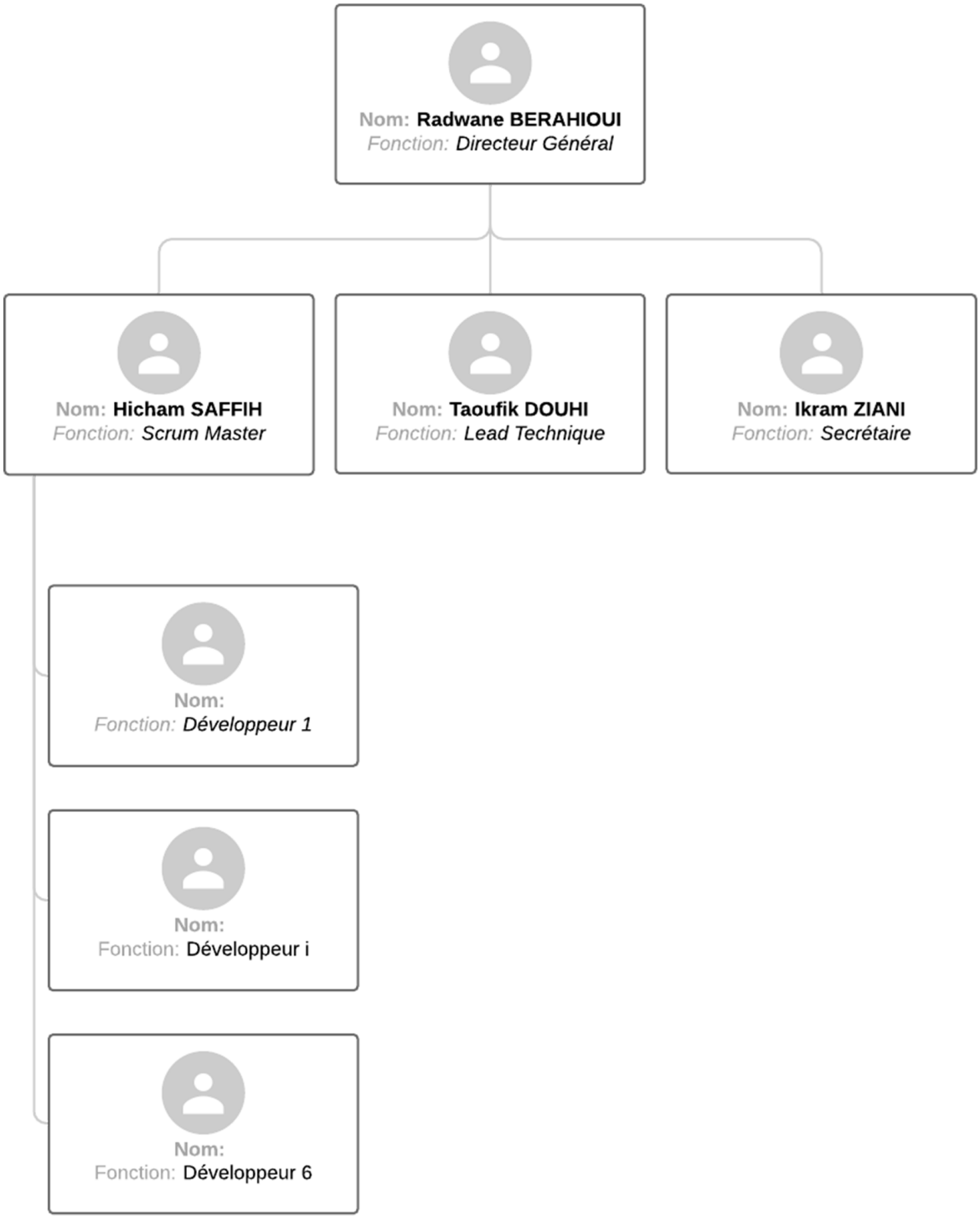



FIGURE 1.2 – Organigramme de l'entreprise Algolus

## 1.3 Description des besoins

### 1.3.1 Problématique

Dans le développement logiciel, la détection et la correction des erreurs représentent un défi majeur, notamment en raison de la diversité des sources d'anomalies (logs, stack traces, captures d'écran, retours utilisateurs, etc.) et de la complexité croissante des applications. Les méthodes traditionnelles de débogage reposent souvent sur une analyse manuelle, ce qui est chronophage et sujet à des erreurs humaines. De plus, les solutions existantes peinent à offrir une approche unifiée et intelligente pour interpréter ces anomalies et proposer des correctifs pertinents.

L'absence d'un système capable de :

— Comprendre multimodalement les erreurs (texte, images, logs structurés).

— Proposer des corrections automatisées en s'appuyant sur des modèles de langage (LLMs) et des techniques de RAG (Retrieval-Augmented Generation).

— S'intégrer facilement dans des environnements DevOps existants.

rend ce projet particulièrement pertinent. Comment concevoir une application IA capable d'analyser efficacement ces différentes sources d'erreurs et de générer des solutions adaptées et contextualisées ?

### 1.3.2 Solutions envisagées

Pour répondre à cette problématique, le projet s'appuie sur une architecture innovante :

1. Analyse Multimodale des Erreurs :
   — Implémenter un système capable d'interpréter des données hétérogènes (stack traces, logs texte, captures d'écran, etc.).
   — Utiliser des techniques de RAG pour enrichir les requêtes avec une base de connaissances (documentation technique, résolutions d'erreurs courantes).
2. Génération Automatique de Correctifs :
   — Exploiter des LLMs (via Ollama) pour suggérer des corrections précises et contextualisées.
3. Intégration et Scalabilité :
   — Développer un backend Spring Boot flexible, couplé à LangChain4J pour orchestrer les appels IA, et un système de gestion de base de données qui prend en charge les bases de données vectorielles, comme PostgeSQL.
   — Permettre une extension future via des connecteurs pour différents outils de monitoring.
4. Optimisation et Évaluation :

— Mesurer l'efficacité du système via des métriques de précision (taux de détection, pertinence des correctifs).
— Benchmark : Comparaison sur des jeux de données communs.

### 1.3.3 Objectifs

Ce projet vise à développer une application intelligente et modulaire permettant de détecter et de corriger automatiquement les anomalies logicielles. Les objectifs spécifiques incluent :

— Détecter avec un bon taux de réussite les anomalies logicielles sur des sources multimodales.
— Proposer des correctifs pertinents dans la majorités des cas.
— Optimiser le temps moyen de résolution d'erreurs par rapport aux méthodes manuelles.

### 1.3.4 Les besoins fonctionnels

1. Collecte et Pré-traitement des Données

   Extraction automatique des erreurs à partir de :
   — Stacktraces : Parsing des logs pour identifier les exceptions (ex : NullPointerException).
   — Captures d'écran : Utilisation de modèles de vision par IA pour détecter les messages d'erreur visuels.
   — Retours utilisateurs : Analyse NLP des descriptions textuelles (ex : "Le bouton ne répond pas").
2. Analyse et Compréhension
   — Analyse sémantique des messages d'erreur.
   — Enrichissement contextuel : requêtage d'une base de connaissances (documentation technique, correctifs historiques) via RAG.
3. Génération de Solutions
   — Explication en langage naturel des causes racines.
   — Génération de correctifs par Ollama (ex : snippets de code, étapes de résolution).
4. Interfaces utilisateurs
   — Soumission des erreurs : Formulaires web pour uploader des stacktraces et des captures d'écran
   — Visualisation des résultats : Dashboard interactif (erreurs en cours, historiques, statistiques).

### 1.3.5 Les besoins non fonctionnels

Les besoins non fonctionnels spécifient comment le système doit fonctionner, sans décrire des fonctionnalités précises. Ils couvrent des aspects comme les performances, la

sécurité, ou l'ergonomie.

Les besoins non fonctionnels définies dans notre système sont :

1. Performances
   - Temps de réponse optimisé.
   - Scalabilité : Support de plusieurs requêtes simultanées.
2. Intégration et Interopérabilité
   - API REST : Endpoints standardisés et format de réponse avec schéma cohérent.
   - Support offline : Fonctionnement local avec Ollama.
3. Sécurité et Confidentialité
   - Protection des données : chiffrement des échanges et anonymisation des logs utilisateurs (RGPD).
   - Authentification : JWT pour l'accès aux APIs sensibles.
4. Expérience Utilisateur

   Ergonomie : interface intuitive, Dark/Light mode et thèmes accessibles.
   - Soumission des erreurs : Formulaires web pour uploader des stacktraces et des captures d'écran
   - Visualisation des résultats : Dashboard interactif (erreurs en cours, historiques, statistiques).

# Chapitre 2

# Analyse et modélisation

## 2.1 Importance de l'analyse

L'analyse constitue une étape clé dans tout projet de développement informatique, car elle permet de bien comprendre les besoins du client et les contraintes du système à réaliser. Elle sert à identifier les fonctionnalités attendues, à détecter les éventuelles incohérences et à poser les bases d'une conception solide. Une analyse bien menée réduit considérablement les risques d'erreurs en phase de développement, facilite la planification du travail et améliore la qualité globale du produit final. Elle est donc essentielle pour assurer la réussite du projet.

## 2.2 Unified Modeling Language

Dans le cadre d'un projet de développement informatique, la modélisation UML (Unified Modeling Language) joue un rôle essentiel en facilitant la compréhension, la conception et la communication autour du système à développer. UML propose un ensemble normalisé de diagrammes qui permettent de représenter visuellement les différentes dimensions d'un logiciel, telles que la structure, le comportement et les interactions entre les composants.

L'utilisation de diagrammes UML, comme les diagrammes de cas d'utilisation, de classes ou de séquence, permet de :

- Clarifier les besoins fonctionnels et non fonctionnels dès les premières phases du projet,
- Favoriser une meilleure communication entre les développeurs, les analystes et les clients,
- Détecter plus tôt les incohérences ou erreurs potentielles dans la conception,
- Servir de documentation technique structurée pour le développement, les tests et la maintenance future du logiciel.

Ainsi, UML constitue un outil précieux pour assurer la qualité, la cohérence et la

pérennité d'un projet informatique, en apportant une vision globale et partagée du système.

## 2.3 Diagramme de cas d'utilisation

### 2.3.1 Définition

Un diagramme de cas d'utilisation est une représentation visuelle des interactions entre les acteurs (utilisateurs, systèmes) et les fonctionnalités d'une application. Il identifie les besoins métiers sous forme d'actions (cas d'utilisation) et montre qui fait quoi, sans entrer dans les détails techniques.

### 2.3.2 Acteurs

Dans un diagramme de cas d'utilisation, les acteurs sont les entités qui interagissent avec le système pour accomplir un objectif précis. Un acteur peut être primaire (s'il est déclencheur d'un cas d'utilisation) ou secondaire (intervient dans un cas d'utilisation mais ne le déclenche pas). D'une autre part, un acteur peut être humain ou bien un acteur système.

Trois types d'acteurs sont impliqués dans notre cas :

— **Utilisateur** : peut être un développeur ou un testeur qui rapporte une erreur, et peut interagir via une API REST ou bien une interface web.

— **Administrateur du système** : responsable de la mise à jour des connaissances du système et de la configuration des modèles.

— **Système** : le moteur de traitement intelligent, responsable d'analyser les anomalies, et de proposer des correctifs appropriés.

### 2.3.3 Notre diagramme de cas d'utilisation

La figure ci-dessous présente le diagramme de cas d'utilisation de notre application :

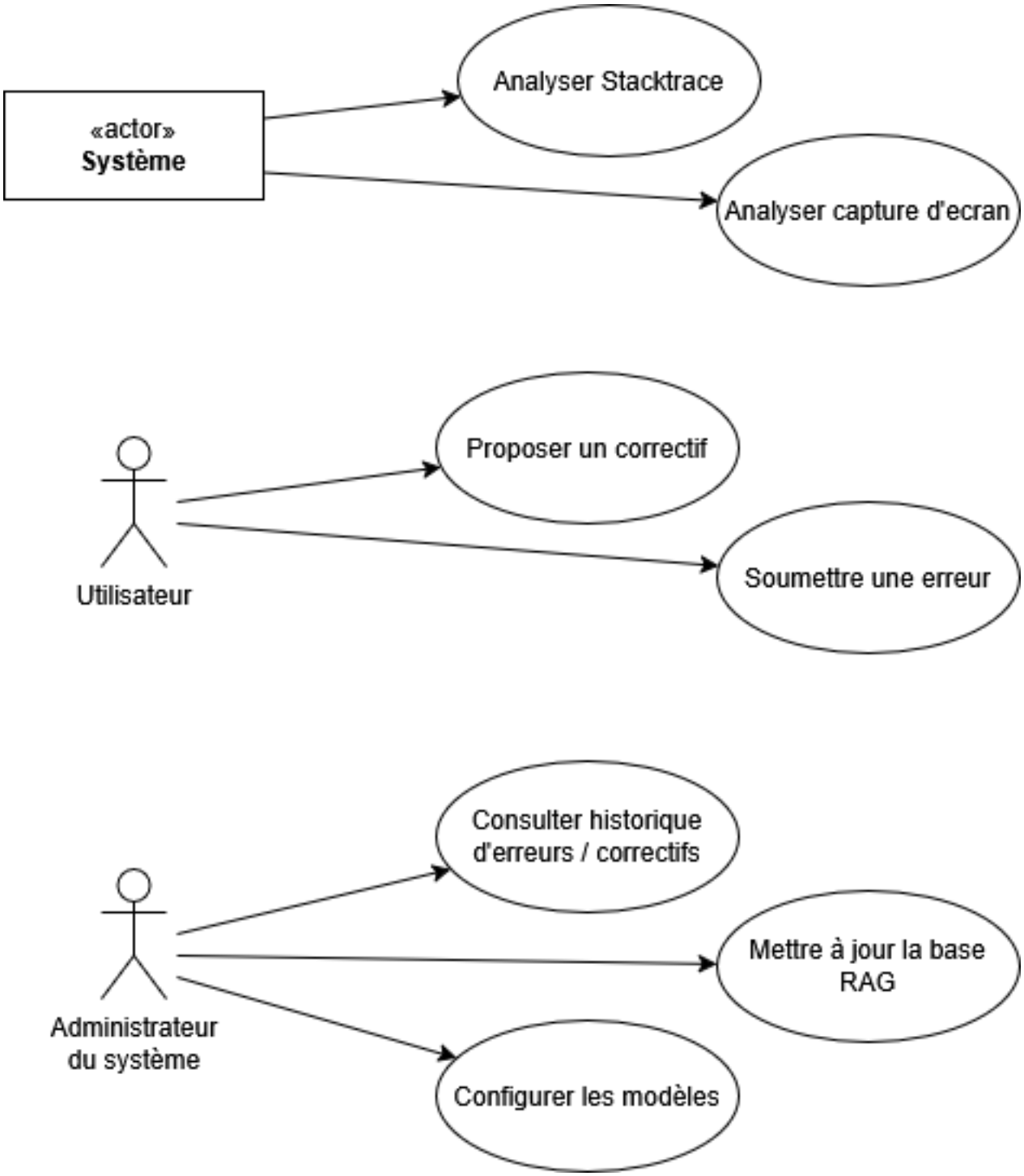



FIGURE 2.1 – Diagramme de cas d'utilisation

# Chapitre 3

# Réalisation

## 3.1 Stack technique

### 3.1.1 Langages

#### Java

Java est un langage de programmation orienté objet, robuste et multiplateforme, largement utilisé dans le développement d'applications d'entreprise. Sa forte typographie, sa gestion automatique de la mémoire (via le garbage collector) et son écosystème riche (bibliothèques, frameworks) en font un choix idéal pour les systèmes backend complexes.

### 3.1.2 Frameworks

#### Spring

Spring est un framework modulaire pour Java, simplifiant le développement d'applications grâce à l'inversion de contrôle (IoC) et la programmation orientée aspect (AOP).

#### Spring Boot

Spring Boot étend Spring en fournissant des configurations automatiques, un serveur embarqué (Tomcat, Netty) et des outils clés en main (Spring Data, Spring Security), permettant de créer des applications standalone rapidement.

#### LangChain4j

LangChain4J est une bibliothèque Java inspirée de LangChain (Python), conçue pour intégrer facilement des LLMs (Modèles de Langage) dans des applications. Elle offre des abstractions pour la gestion des prompts, le RAG, les appels aux modèles (OpenAI, Ollama, etc.), et la connexion à des bases de données vectorielles.

### 3.1.3 Bibliothèques

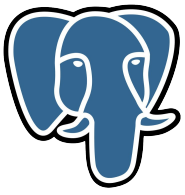
#### Apache Commons

Apache Commons est une bibliothèque Java open-source fournissant des composants réutilisables pour simplifier le développement. Dans ce projet, elle sert à combler des besoins techniques récurrents avec des solutions optimisées et robustes.

### 3.1.4 Systèmes de gestion de bases de données

#### PostgreSQL

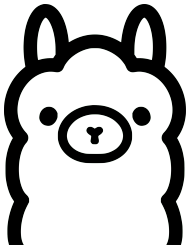
PostgreSQL est un système de gestion de base de données relationnelle (SGBDR) open-source, robuste et extensible. Dans le cadre de ce projet, il joue un rôle central pour stocker et gérer les données structurées nécessaires au bon fonctionnement de l'application, et fournit des plugins pour l'IA, notamment PgVector, qui gère les bases de données vectorielles.



### 3.1.5 Outils et environnement

#### Ollama

Ollama est un outil open-source permettant d'exécuter localement des LLMs (comme Llama 3, Mistral, Gemma) sans dépendre d'une API externe. Il est idéal pour prototyper des solutions IA offline, contrôler les coûts et la confidentialité des données, et personnaliser finement les modèles via des modelfiles.



#### Maven

Outil de build automatisé pour projets Java, qui gère Les dépendances (téléchargement auto), le packaging (JAR/WAR), et les cycles de compilation/test.


#### IntelliJ IDEA

IntelliJ IDEA est un IDE puissant pour Java/Kotlin, développé par JetBrains. Ses avantages incluent une analyse intelligente du code (suggestions, détection d'erreurs), une intégration native avec Spring Boot et Maven/Gradle, des outils pour le débogage, le profiling et les tests, et des extensions pour l'IA (ex : GitHub Copilot).


#### Git

Git est un système de contrôle de version distribué, essentiel pour le développement collaboratif. Il permet de suivre les modifications du code source, de gérer les branches, et de fusionner les travaux de plusieurs contributeurs. Grâce à des plateformes comme GitHub, il facilite le partage et la revue de code. Son utilisation améliore la traçabilité, la qualité et la productivité dans les projets logiciels.

## 3.2 Architecture technique du projet

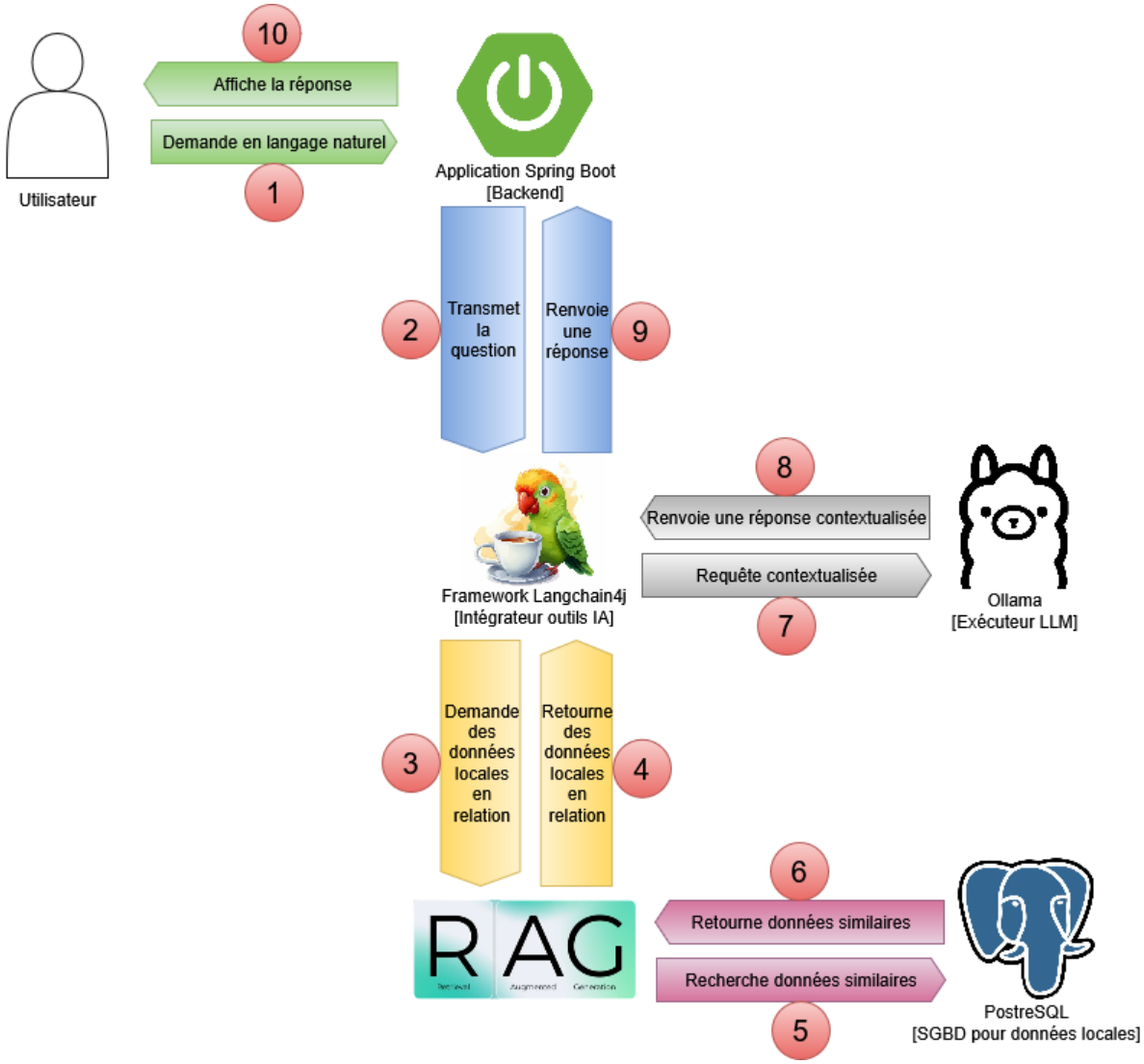



FIGURE 3.1 – Diagramme d'architecture

## 3.3 Premier prototype

### 3.3.1 Introduction

Pour atteindre cet objectif ambitieux, nous avons suivi une approche incrémentale. Dans un premier temps, un prototype fonctionnel a été réalisé afin de valider les fondements techniques du projet. Ce prototype est une application basée sur un agent intelligent exploitant une architecture RAG (Retrieval-Augmented Generation), capable de répondre aux questions de l'utilisateur à partir d'un document texte ou PDF fourni. Cette première version a permis de :

— comprendre le fonctionnement du framework LangChain4j ;
— tester l'intégration avec le LLM Ollama ;
— valider le concept de récupération de contexte à partir de documents externes.

### 3.3.2 Fonctionnement d'un agent AI dans LangChain4j

Dans LangChain4j, un Agent AI orchestre les interactions entre un ChatLanguageModel (abstraction des LLMs comme OpenAI/Gemini/Ollama) et un ChatMemoryProvider (gestion de l'historique conversationnel). L'Agent formate les requêtes, intègre la mémoire contextuelle, et peut utiliser des outils externes, tandis que le LLM génère les réponses. Cette architecture modulaire permet une intégration flexible avec différents modèles et systèmes de stockage, tout en maintenant un état conversationnel cohérent.

Le diagramme de séquences suivant décrit ce processus :

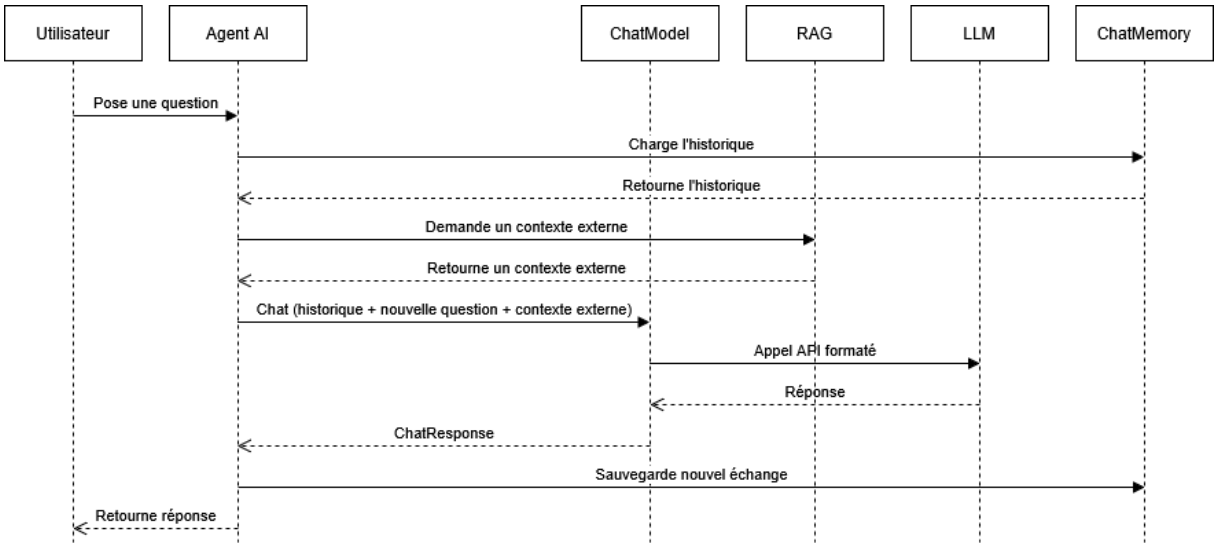



FIGURE 3.2 – Diagramme de séquences décrivant le fonctionnement d'un agent AI

Le RAG combine deux phases clés pour améliorer la génération de réponses par un LLM : la rétroinformation (retrieval) et la génération contextuelle. Dans ce prototype, le système RAG :

— Charge d'abord un fichier source (texte, PDF, etc.).
— Utilise un DocumentPaser pour extraire les données brutes, et les découper en 'chunks'.
— Utilise un EmbeddingModel qui convertit ces chunks en vecteurs d'embedding (représentations numériques sémantiques).
— Utilise un EmbeddingStoreIngestor pour stocker ces vecteurs dans un EmbeddingStore.
— Utilise un Retriever qui interroge l'EmbeddingStore pour trouver les documents les plus pertinents par rapport à la question de l'utilisateur.

Le diagramme de séquences suivant détaille ces étapes :

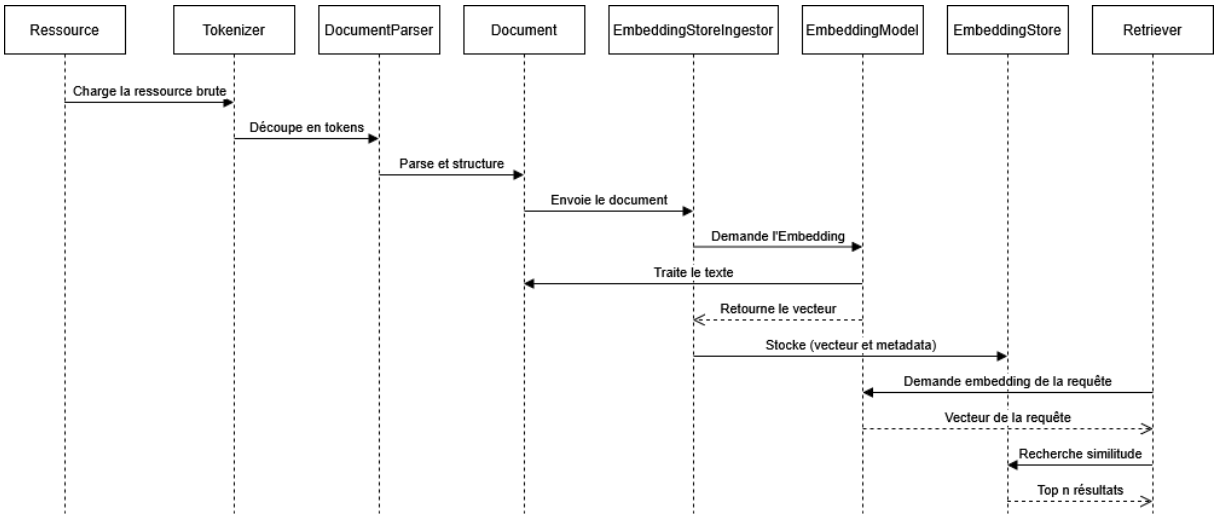



FIGURE 3.3 – Diagramme de séquences décrivant le fonctionnement du RAG

Pour tester ce prototype, nous avons fourni un fichier PDF, son contenu est une lettre de recommandation pour une étudiante appelée Nour, on a ensuite envoyé une question à propos de cette étudiante à l'agent AI, en utilisant un contrôleur web :

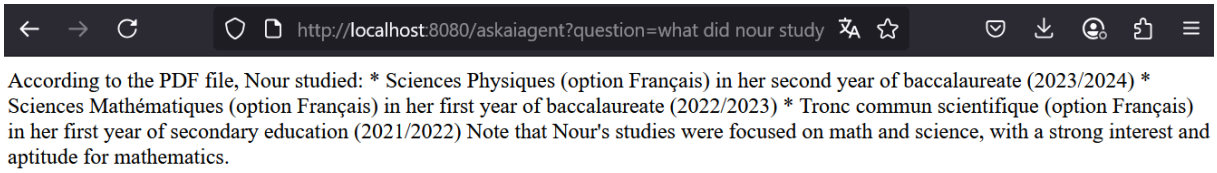



FIGURE 3.4 – Test du RAG