Liste des abréviations

Abréviation	**Désignation**
RAG	Retrieval-Augmented Generation

Table des figures

Liste des tableaux

Table des matières

Chapitre 1

Contexte du Projet

1.1 Projet de Fin d'Études

Dans le cadre de notre formation en Génie Informatique à l'École des Hautes Études d'Ingénierie d'Oujda (EHEIO), nous devons réaliser un Projet de Fin d'Études (PFE), visant à consolider et approfondir les compétences acquises durant notre parcours. Ce stage représente une étape cruciale, permettant de transposer les connaissances théoriques dans un environnement professionnel concret.

L'objectif principal est de se familiariser avec les réalités du marché du travail, particulièrement dans le domaine de l'informatique, en perpétuelle évolution. Ce projet offre ainsi l'opportunité d'appliquer nos acquis académiques à des problématiques réelles, tout en développant une approche pratique et méthodique de la gestion de projets technologiques.

Ce stage s'est déroulé dans une entreprise adoptant une méthodologie Scrum, l'une des approches Agile les plus répandues dans le secteur informatique. Cette immersion professionnelle a été l'occasion découvrir le fonctionnement d'une équipe projet en conditions réelles, d'appliquer les principes Agile (itérations, sprints, réunions quotidiennes) dans un cadre professionnel, et de collaborer avec des experts et assimiler les bonnes pratiques en gestion de projets logiciels.

L'utilisation de Scrum a renforcé ma compréhension des processus modernes de développement, tout en améliorant mes capacités d'adaptation et de travail en équipe. Cette expérience a été déterminante pour affiner ma vision du métier d'ingénieur informatique et préparer mon intégration dans le monde professionnel.

Dans ce chapitre, nous commencerons par présenter l'entreprise d'accueil, avant de procéder à une description détaillée des besoins du projet. Cette description comprendra l'identification du problème posé, l'analyse des besoins fonctionnels et non fonctionnels, ainsi que les solutions envisagées pour y répondre.

1.2 Entreprise d'accueil

Ce stage a été réalisé au sein de l'entreprise Algolus, située à Oujda, spécialisée dans les solutions innovantes en intelligence artificielle. Démarré le 24 février 2024, il m'a permis de m'immerger dans un environnement professionnel exigeant, où j'ai pu collaborer avec des experts en IA et en ingénierie logicielle.

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 1.1 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 Redwan.Berahioui@algolus.ma www.algolus.ma IMMEUBLE OUASSIM, Bd Mohammed VI, Oujda 60000

1.2.3 Organigramme de l'entreprise

La figure 1.2 présente l'organigramme de l'entreprise Algolus :

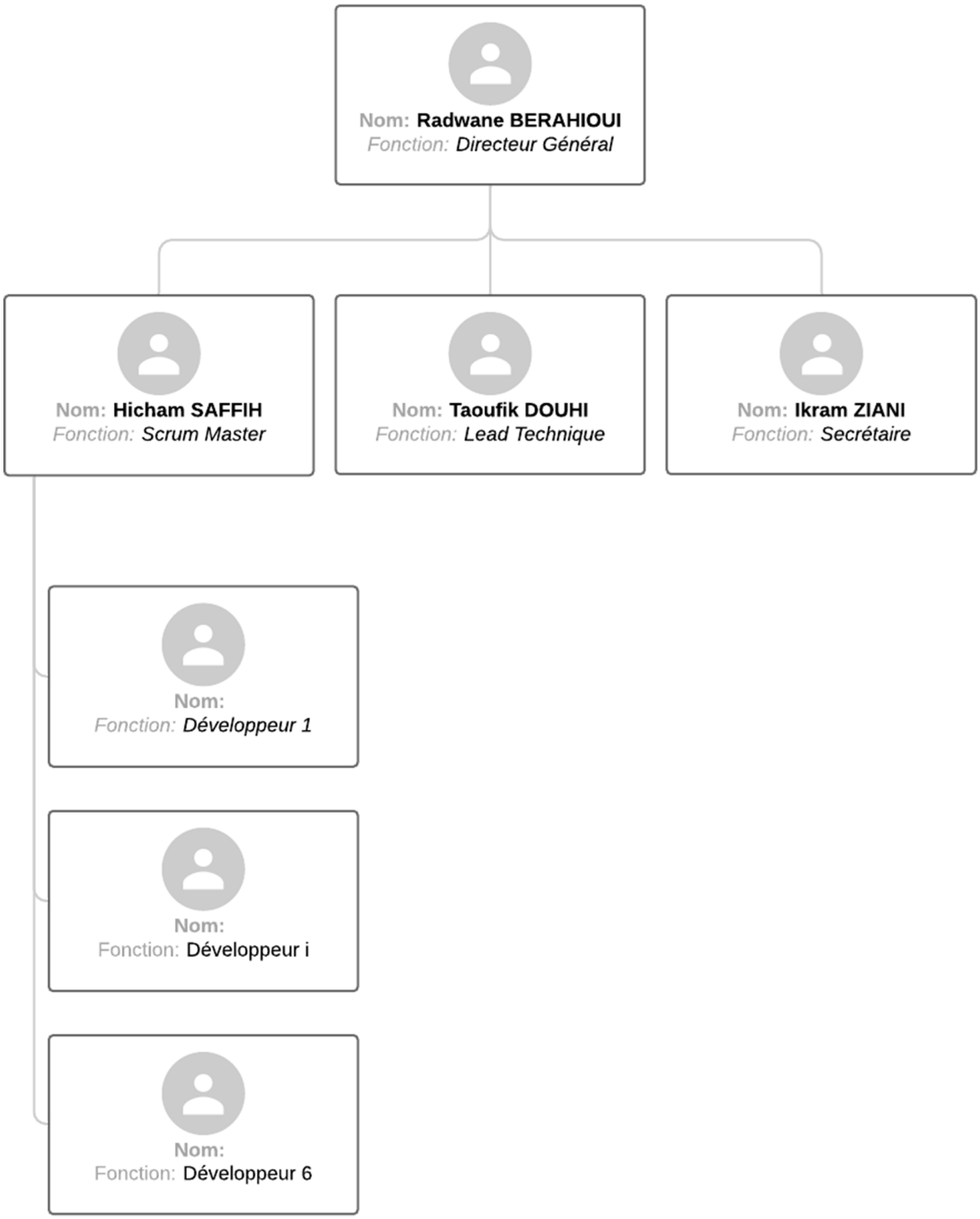


FIGURE 1.2 – *Organigramme de l'entreprise Algolus*

1.3 Description des besoins

1.3.1 Problème

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 générique et intelligente pour interpréter ces anomalies et proposer des correctifs pertinents.

Ce défi prend une dimension particulière dans le cadre des activités de Algolus. La complexité des systèmes gérés par l'entreprise, caractérisés par des architectures microservices distribuées, des intégrations multiples avec des partenaires externes et des exigences strictes de conformité, amplifie les difficultés de diagnostic des anomalies. Les équipes techniques consacrent actuellement un volume considérable de leurs ressources temporelles à l'analyse manuelle des incidents, retardant d'autant les mises en production. Par ailleurs, la variété des clients et des cas d'usage entraîne une hétérogénéité des remontées d'erreurs (rapports techniques détaillés pour les clients corporate vs. simples captures d'écran pour les utilisateurs finaux), ce qui rend inefficaces les outils de monitoring conventionnels utilisés jusqu'à présent. Ce constat a motivé l'entreprise à explorer des solutions d'IA générative capables d'unifier l'interprétation des anomalies.

1.3.2 Les besoins fonctionnels

Les besoins fonctionnels définissent les actions spécifiques que le système doit accomplir pour répondre aux exigences métier. Ils décrivent "quoi" le logiciel doit faire, sous la forme de fonctionnalités concrètes, de processus et d'interactions avec l'utilisateur. Ces exigences sont formulées par les parties prenantes : clients, utilisateurs, et équipe produit, et servent de base à la conception des cas d'usage et des scénarios de test.

Les besoins fonctionnels suivants ont été identifiés pour garantir que le système de diagnostic d'erreurs réponde efficacement aux attentes des utilisateurs et des équipes techniques :

1. **Collecte et Pré-traitement des Données** : Extraction automatique des erreurs et des anomalies des systèmes à partir de : stacktraces, parsing des logs, captures d'écran, retours utilisateurs.
2. **Analyse et Compréhension** : Analyse sémantique de retours d'erreurs, enrichissement contextuel : requêtage d'une base de connaissances (documentation technique, correctifs historiques) via RAG (Retrieval-Augmented Generation).

3. **Génération de Solutions** : Explication en langage naturel des causes racines, génération de correctifs (ex : snippets de code, étapes de résolution).
4. **Interfaces utilisateurs** : Soumission des erreurs via des formulaires web pour uploader des stacktraces et des captures d'écran, visualisation des résultats : Dashboard interactif (erreurs en cours, historiques, statistiques).

1.3.3 Les besoins non fonctionnels

Les besoins non fonctionnels caractérisent "comment" le système doit fonctionner, en précisant ses contraintes de qualité, de performance et d'infrastructure. Contrairement aux besoins fonctionnels, ils ne décrivent pas des fonctionnalités mais des critères tels que la rapidité, la sécurité, la scalabilité ou la facilité de maintenance. Leur respect est essentiel pour assurer la robustesse et l'efficacité du système en conditions réelles.

Pour garantir une intégration harmonieuse dans l'écosystème existant et une expérience utilisateur optimale, les besoins non fonctionnels suivants ont été définies :

1. **Performances** : Temps de réponse optimisé, et 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 par chiffrement des échanges et anonymisation des logs utilisateurs (RGPD), et authentification : JWT pour l'accès aux APIs sensibles.
4. **Expérience Utilisateur** : Ergonomie : interface intuitive, Dark/Light mode et thèmes accessibles.

1.3.4 Solutions envisagées

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 taux de réussite d'au moins 80% les anomalies logicielles sur des sources multimodales.
- Proposer des correctifs pertinents dans plus de 70% des cas.
- Optimiser le temps moyen de résolution d'erreurs de 30% par rapport aux méthodes manuelles.

Pour atteindre ces objectifs, 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.), et 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, et 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), et effectuer un Benchmark : comparaison sur des jeux de données communs.

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.

Dans ce chapitre, nous présentons une introduction à la modélisation UML en rappelant ses principes fondamentaux et son utilité dans le développement logiciel. Par la suite, nous exposerons les différents types de diagrammes utilisés dans notre étude, à savoir : le diagramme de cas d'utilisation, les diagrammes de séquence, ainsi que le diagramme de classes.

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, de favoriser une meilleure communication entre les développeurs, les analystes et les clients, de détecter plus tôt les incohérences ou erreurs potentielles dans la conception, et aussi de 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 Diagramme de cas d'utilisation

La figure 2.1 présente le diagramme de cas d'utilisation de notre application :

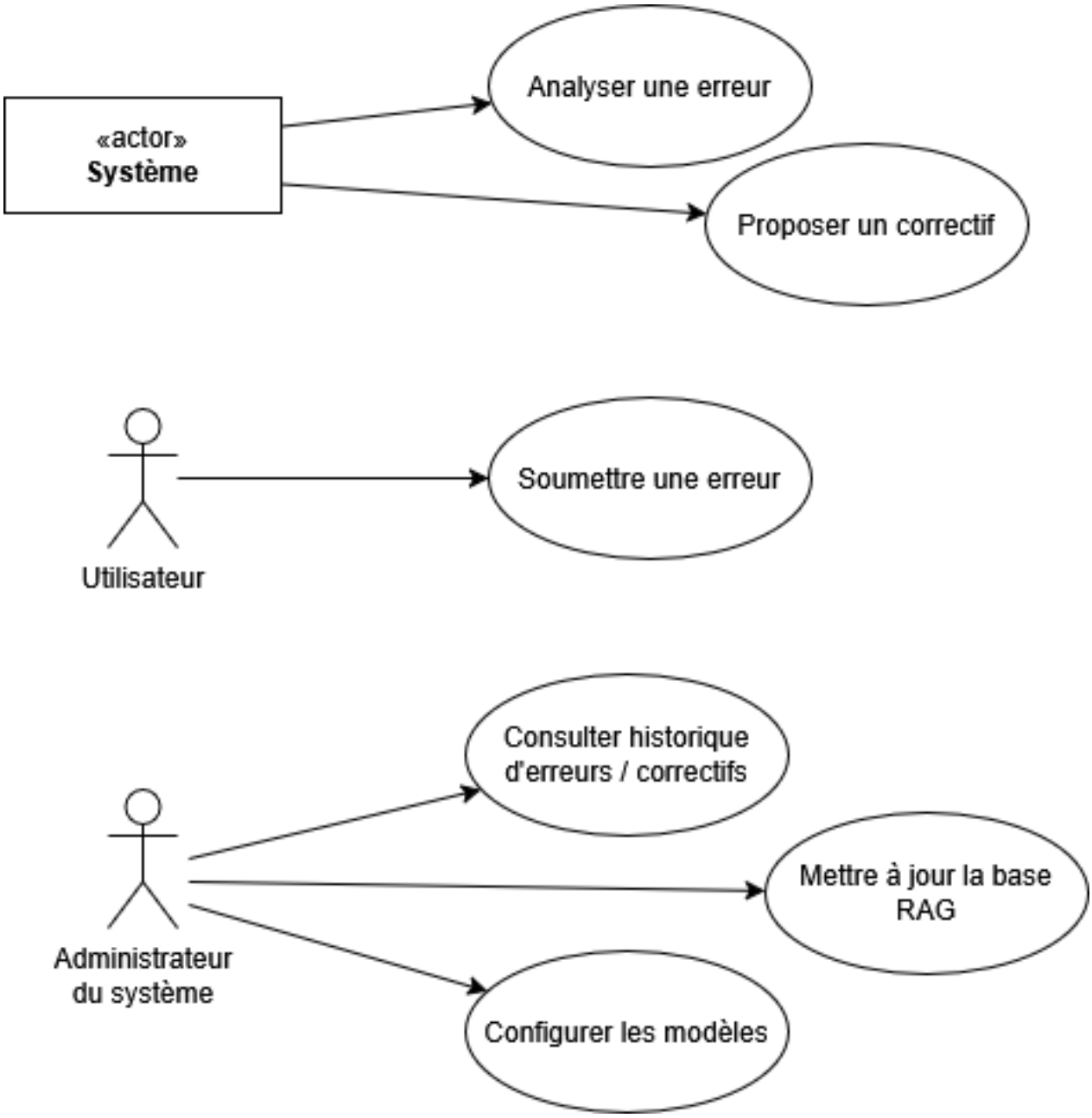


FIGURE 2.1 – *Diagramme de cas d'utilisation*

2.4 Diagrammes de séquences

2.4.1 Définition

Un diagramme de séquences est un type de diagramme UML, utilisé pour modéliser les interactions entre les différents objets ou composants d'un système dans un scénario précis. Il met en évidence l'ordre chronologique des messages échangés entre les acteurs et les objets, les interactions dynamiques entre les éléments du système, et la durée de vie

des objets participant au scénario.

2.4.2 Diagramme de séquences principal

Cette application d'analyse d'erreurs techniques repose sur une architecture modulaire, où chaque composant joue un rôle précis dans le traitement des requêtes. Voici une présentation des éléments clés qui permettent au système de comprendre, contextualiser et répondre aux problèmes soumis par les utilisateurs :

- **Utilisateur** : L'Utilisateur constitue le point de départ du système. Ce composant représente l'acteur humain qui interagit avec l'application via une interface web ou des appels API. Il soumet des requêtes contenant des stacktraces d'erreur et éventuellement des captures d'écran.
- **Agent AI** : sert de chef d'orchestre principal dans l'architecture. Ce service intelligent coordonne l'ensemble du processus d'analyse. Il reçoit les requêtes brutes de l'utilisateur, les enrichit en combinant plusieurs techniques avancées comme le RAG et la mémoire conversationnelle, puis les présente au modèle de langage sous une forme optimale.
- **ChatModel** : incarne le moteur de génération de langage naturel. Ce composant spécifique, configuré pour utiliser des modèles locaux ou externes, transforme les prompts structurés en analyses techniques détaillées.
- **RAG (Retrieval-Augmented Generation)** : apporte une dimension documentaire aux analyses. Composé de trois éléments principaux - un modèle d'embedding, un stockage vectoriel et un moteur de recherche - ce pipeline sophistiqué permet d'enrichir les réponses du LLM avec des connaissances provenant d'une base documentaire technique. Le processus transforme d'abord les requêtes et documents en vecteurs, puis effectue des recherches de similarité dans le stockage vectoriel avant d'injecter les documents pertinents dans le contexte du LLM.
- **LLM (Large Language Model)** : représente le cœur cognitif du système. Ce modèle linguistique à grande échelle possède une compréhension approfondie du langage naturel et des concepts techniques. Capable de traiter des stacktraces complexes et d'identifier des patterns d'erreur, il génère des analyses structurées qui incluent généralement la classification de l'erreur, sa cause probable, et des solutions potentielles. Son fonctionnement est optimisé par le contexte fourni par la mémoire conversationnelle et les documents retrouvés via le RAG.
- **ChatMemory** : joue un rôle crucial dans la maintien du contexte conversationnel. Implémentée comme une mémoire à fenêtre glissante, elle conserve les N dernières interactions de chaque session utilisateur, identifiée par un MemoryId unique. Cette mémoire permet au LLM de maintenir une cohérence dans les échanges et de se souvenir des points clés abordés précédemment dans la conversation, essentiel pour les analyses techniques itératives où l'utilisateur affine progressivement sa requête.

Le diagramme de séquences dans la figure 2.2 décrit le flux de messages entre ces composants.

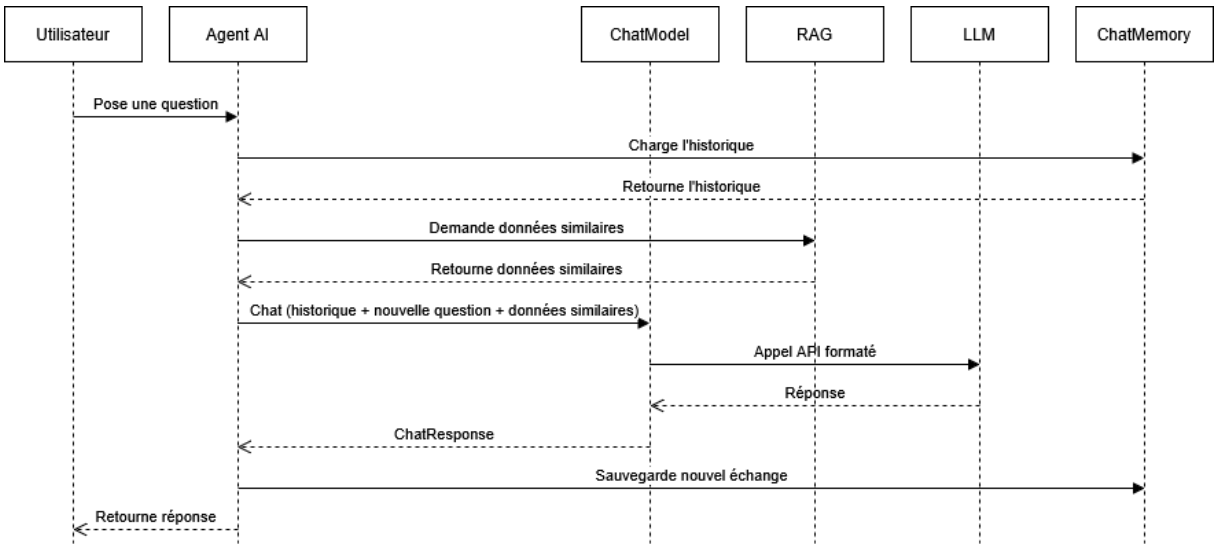


FIGURE 2.2 – *Diagramme de séquences décrivant le fonctionnement d'un agent AI*

2.4.3 Diagramme de séquences spécifique : RAG

Nous allons creuser un peu dans les composants du système RAG, voici une énumération de ces composants avec chacun son rôle :

- **Resource** : Constitue la matière première du système RAG. Il s'agit de la source originelle des données qui alimenteront la base de connaissances. Ces ressources peuvent prendre diverses formes : fichiers PDF contenant de la documentation technique, pages web de référence, extraits de bases de données, ou tout autre support contenant des informations pertinentes. Le système doit être capable de traiter ces ressources hétérogènes, qu'elles proviennent de systèmes internes ou de sources externes. La qualité et l'actualité des resources déterminent en grande partie l'efficacité finale du système RAG.
- **Tokenizer** : Joue un rôle fondamental dans le prétraitement du texte. Ce composant décompose le contenu textuel en unités significatives appelées tokens, qui peuvent correspondre à des mots entiers, des sous-mots ou même des caractères individuels selon la méthode employée. Des algorithmes avancés comme ceux proposés par HuggingFace permettent une tokenisation optimale qui préserve le sens tout en gérant les particularités linguistiques. La tokenisation est cruciale car elle influence directement la qualité des embeddings générés ultérieurement.
- **DocumentParser** : Assure la transformation des ressources brutes en documents structurés. Ce composant doit comprendre divers formats de fichiers (PDF, HTML, Markdown, etc.) et en extraire le contenu textuel significatif tout en conservant

les métadonnées importantes. Des bibliothèques spécialisées comme sont souvent employées pour cette tâche complexe. Le DocumentParser nettoie également le texte en supprimant les éléments non pertinents (en-têtes, pieds de page, balises HTML) pour ne conserver que l'information essentielle.

- **Document** : Représente la forme normalisée et standardisée des informations après traitement. Chaque document contient non seulement le texte brut nettoyé, mais aussi des métadonnées descriptives (titre, auteur, date de création, source) qui faciliteront son identification et son utilisation ultérieure. Un identifiant unique est attribué à chaque document pour permettre son suivi tout au long du pipeline. Cette structuration rigoureuse est essentielle pour maintenir la cohérence des données dans les étapes suivantes du processus RAG.
- **EmbeddingModel** : Est au cœur de la transformation sémantique du système. Ce modèle sophistiqué convertit le texte en représentations vectorielles denses (embeddings) qui capturent le sens profond des contenus. Des modèles sont spécialisés dans cette tâche produisent des vecteurs où la similarité spatiale correspond à la similarité sémantique. La qualité de l'EmbeddingModel détermine directement la capacité du système à retrouver des documents pertinents pour une requête donnée.
- **EmbeddingStoreIngestor** : Orchestre le processus complet d'indexation des documents. Ce composant supervise plusieurs opérations critiques : il applique la tokenisation et la segmentation des textes, déclenche la génération des embeddings via l'EmbeddingModel, et gère le stockage final dans l'EmbeddingStore. L'EmbeddingStoreIngesto implémente souvent des stratégies de traitement par lots pour optimiser les performances et peut gérer des pipelines complexes de prétraitement avant la vectorisation.
- **EmbeddingStore** : Sert de mémoire à long terme au système RAG. Cette base de données vectorielle spécialisée stocke les embeddings générés et permet des recherches rapides de similarité. L'EmbeddingStore doit supporter des opérations massives d'insertion tout en maintenant des temps de réponse faibles pour les requêtes.
- **Retriever** : Est le composant qui établit le pont entre les questions des utilisateurs et la base de connaissances. Lors d'une requête, le Retriever transforme d'abord la question en embedding, puis recherche dans l'EmbeddingStore les documents dont les vecteurs sont les plus proches. Ce composant implémente des algorithmes de similarité vectorielle (cosine similarity par exemple) et peut être finement paramétré (nombre de résultats retournés, seuil de similarité minimal). Le Retriever joue ainsi un rôle déterminant dans la pertinence des résultats fournis au LLM.

Le diagramme de séquences dans la figure 2.3 résume ce processus.

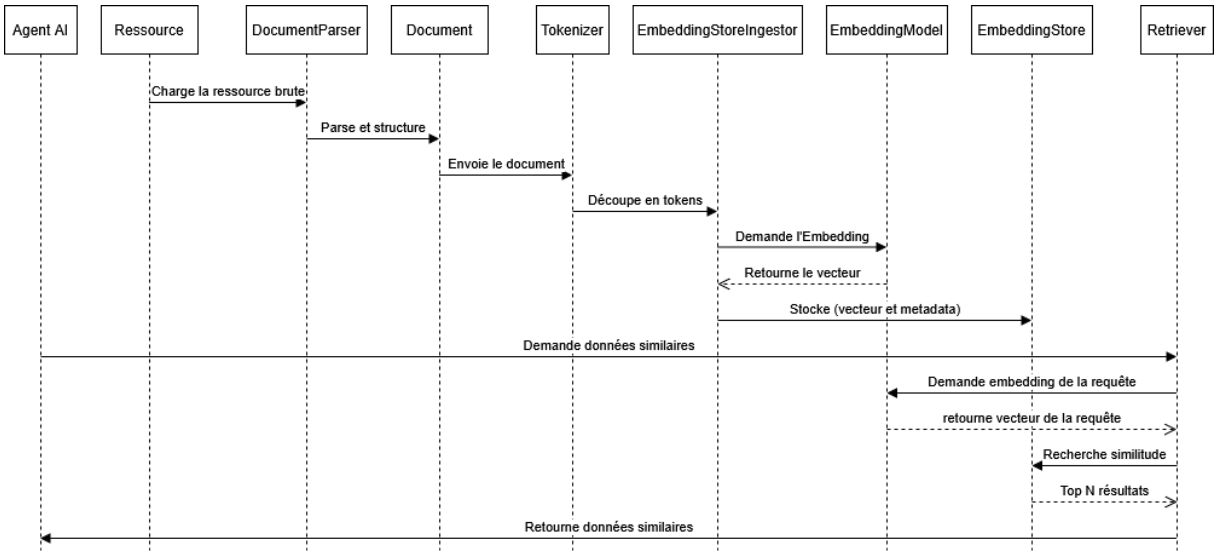


FIGURE 2.3 – *Diagramme de séquences décrivant le fonctionnement d'un RAG*

Chapitre 3

Réalisation

Ce chapitre présente la phase de concrétisation du projet, où les choix techniques et architecturaux se traduisent en une implémentation fonctionnelle. Il décrit l'environnement de développement (outils, frameworks, librairies, etc.), l'architecture logicielle retenue et son adéquation avec les besoins, les défis techniques rencontrés et les solutions apportées, et les composants clés implémentés avec des extraits de code significatifs, et des captures d'écran illustrant les résultats obtenus.

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

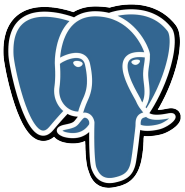
Apache Commons

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

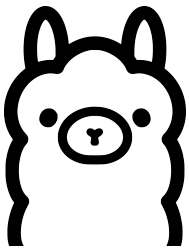
PostgreSQL est un système de gestion de base de données relationnelle (SGBDR) open-source, robuste et extensible. Dans le cadre de ce projet, il joue un rôle central pour stocker et gérer les données structurées nécessaires au bon fonctionnement de l'application, et fournit des plugins pour l'IA, notamment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

Le projet repose sur une architecture modulaire et évolutive, construite autour des meilleures pratiques du développement Java moderne, de l'IA générative et de l'ingénierie logicielle. Il s'appuie sur les technologies suivantes :

- **Spring Boot** : Est le framework retenu pour le développement de la couche back-end. Il s'agit d'un choix stratégique largement justifié par les besoins du projet en termes de performance, de maintenabilité et d'intégration avec des composants d'intelligence artificielle.

  Spring Boot permet de structurer l'application de manière modulaire, en séparant clairement les responsabilités (contrôleurs, services, configuration, etc.). Cette organisation favorise une bonne lisibilité du code et facilite son évolution.

  L'application est également portable, dans la mesure où elle peut être conditionnée sous forme de JAR exécutable et déployée facilement sur tout environnement compatible Java, sans dépendance à un serveur externe.

  Un autre atout majeur est le caractère évolutif de Spring Boot : il s'intègre naturellement avec des bibliothèques telles que LangChain4j ou des bases de données comme PostgreSQL, ce qui permet d'ajouter de nouvelles fonctionnalités (IA, recherche vectorielle, mémoire contextuelle) sans remise en cause de l'existant.

  Le framework est aussi testable : il propose des outils natifs pour la réalisation de tests unitaires et d'intégration, garantissant la qualité et la stabilité du code produit.

  Enfin, Spring Boot est hautement extensible. Si le projet venait à croître en complexité, il serait tout à fait envisageable de faire évoluer l'architecture vers un modèle microservices avec des outils comme Spring Cloud.
- **LangChain4j** : Pour permettre l'analyse intelligente des erreurs à l'aide d'un modèle de langage (LLM), le projet s'appuie sur la bibliothèque LangChain4j, une adaptation Java du framework LangChain initialement développé pour Python. Ce composant joue un rôle central dans l'intégration des fonctionnalités d'intelligence artificielle générative.

LangChain4j facilite la mise en œuvre d'un mécanisme de RAG (Retrieval-Augmented Generation), en combinant génération de texte via un LLM et récupération de documents pertinents à partir d'une base vectorielle. Il s'intègre naturellement avec un modèles de langage, et avec des composants tels que la mémoire de conversation, le modèle d'embedding et le store d'embeddings.

L'utilisation de LangChain4j rend l'application extensible et modulaire, car ses composants (LLM, mémoire, embeddings, etc.) sont interchangeables via des interfaces. Elle offre également un haut niveau de configurabilité, permettant d'adapter dynamiquement les modèles utilisés, la taille de la mémoire contextuelle ou encore les critères de pertinence documentaire.

Enfin, son intégration avec Spring Boot via des beans injectables simplifie grandement sa mise en œuvre dans l'architecture globale du projet. Cela permet d'enrichir les traitements métier avec une couche d'IA tout en conservant la lisibilité et la testabilité du code.

- **Ollama** : Pour exécuter les modèles de langage en local sans dépendre de services cloud externes, le projet intègre Ollama, une plateforme légère permettant de servir des LLM open-source tels que Mistral, Llama3 ou Qwen qui offre des versions multimodales. Ollama agit comme un point d'accès HTTP local à un modèle de génération de texte, que LangChain4j peut interroger de manière transparente.

  Ce choix présente plusieurs avantages : tout d'abord, il rend l'application autonome et portable, car aucun appel à une API cloud (comme OpenAI ou Hugging Face) n'est requis. Cela permet un déploiement sur des machines locales ou en environnement isolé (on-premise), tout en respectant les contraintes de confidentialité des données.

  Grâce à une configuration centralisée (adresse du serveur, modèle utilisé, température, etc.), Ollama est également hautement configurable. Son intégration dans le projet se fait via des beans Spring instanciés dynamiquement dans la classe de configuration (AiConfig), ce qui permet d'adapter ou changer le modèle utilisé sans modifier la logique métier.

  Enfin, en travaillant de concert avec LangChain4j, Ollama permet la génération de réponses contextualisées et pertinentes, en tenant compte des documents récupérés et des interactions passées. Cela renforce la capacité de l'application à fournir des analyses d'erreurs enrichies, précises et directement exploitables.
- **PostgreSQL** : Utilisé comme système de gestion de base de données relationnelle, avec une orientation spécifique vers le stockage vectoriel, dans le cadre de l'indexation et de la recherche de documents sémantiques. Grâce à l'extension pgvector, PostgreSQL devient capable de stocker des vecteurs d'embedding et d'effectuer des recherches de similarité, essentielles dans une approche RAG (Retrieval-Augmented Generation).

  Le choix de PostgreSQL repose sur plusieurs critères clés : sa fiabilité, sa scalabilité et sa maturité en production. En plus de gérer des données relationnelles

classiques (logs, utilisateurs, paramètres...), il peut aussi indexer efficacement des vecteurs issus des modèles d'embedding (ex. OllamaEmbeddingModel), et permettre des requêtes de type nearest neighbor search.

Son intégration avec Spring Boot est fluide grâce à JPA ou JDBC, et son usage dans ce projet est évolutif : dans un premier temps, les embeddings sont stockés en mémoire (InMemoryEmbeddingStore), mais la bascule vers PostgreSQL permettra une persistance et une scalabilité bien supérieures, tout en conservant une interface compatible avec les composants LangChain4j.

Ainsi, PostgreSQL joue un rôle fondamental dans la stratégie d'enrichissement contextuel des requêtes utilisateur, en permettant à l'IA de s'appuyer sur des documents pertinents stockés localement de façon fiable et interrogeable.

La figure 3.1 illustre de manière synthétique les interactions entre les principaux composants techniques de l'architecture mise en place dans le cadre de ce projet.

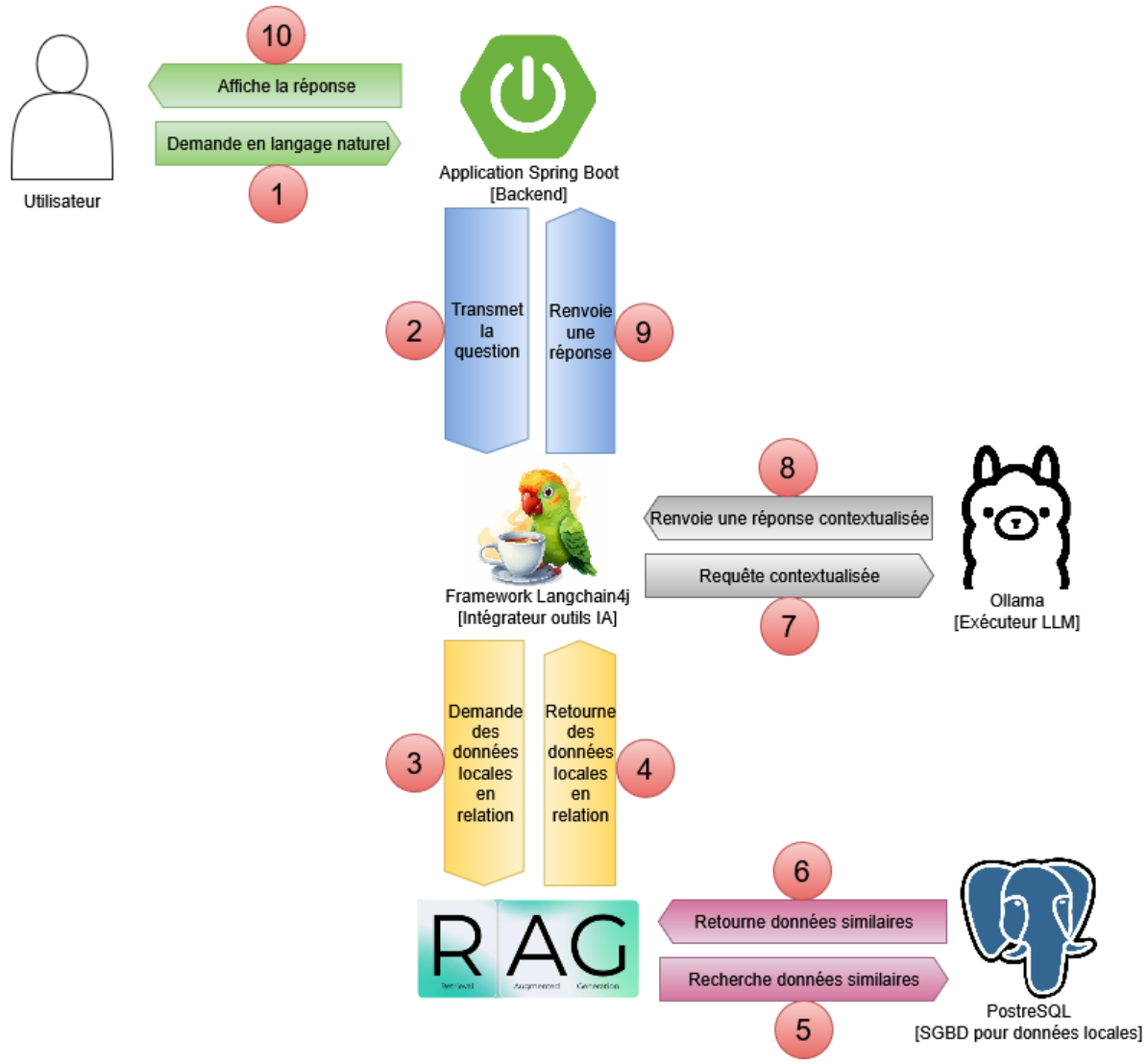


FIGURE 3.1 – *Diagramme d'architecture technique*

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.

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 :

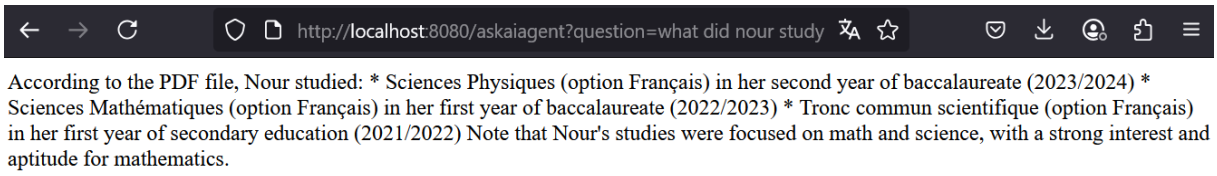


FIGURE 3.2 – *Test du RAG*