MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE DE SOUSSE

INSTITUT SUPERIEUR D'INFORMATIQUE

ET DES TECHNOLOGIES DE COMMUNICATION

المعهد العالي للإعلامية وتكنولوجيات الاتصال

Rapport de stage de fin d'études

Présenté en vue de l'obtention du Diplôme National d'Ingénieur

Spécialité : Téléinformatique

Réalisé Par :

Malek BOUZAYANI

Migration Automatisée des Applications Legacy via un Système Multi-Agent Orchestré par des Modèles Large de Langage (LLMs)

Encadré par :

Encadrante académique : **Mme. Olfa BESBES**

Encadrant professionnel : **M. Marouen BEN MOUSSA**

Réalisé au sein de :

Mobelite

Année universitaire : 2024/2025

Institut Supérieur d'Informatique et des Technologies de Communication **ISITCOM**
Tél/Fax : +216 73 37 15 71 / +216 73 36 44 11

SIGNATURES

Encadrante académique : Mme. Olfa BESBES

Encadrant professionnel : M. Marouen BEN MOUSSA

DEDICATION

À mes parents, pour leur amour inconditionnel, leurs sacrifices et leur soutien indéfectible.

À mes amis et camarades, avec qui j'ai partagé des moments précieux de vie, d'apprentissage et de croissance.

À toutes les personnes qui croient en la puissance de l'apprentissage et de la technologie pour améliorer le monde.

À vous tous,

Je dédie ce travail.

Malek BOUZAYANI

REMERCIEMENTS

Je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont contribué, de près ou de loin, à la réalisation de ce projet.

Mes sincères remerciements vont à l'entreprise **Mobelite** pour m'avoir offert l'opportunité d'évoluer dans un environnement innovant et stimulant. Je remercie particulièrement mes encadrants de stage pour leur disponibilité, leurs conseils précieux et leur accompagnement bienveillant tout au long de ce projet.

Une mention spéciale à l'équipe **BlackBeak**, avec laquelle j'ai eu le privilège de participer au Hackathon organisé par **Optimized AI et Traversaal.ai**. Cette expérience m'a permis de confronter notre solution à un contexte réel, de valider ses performances en conditions intensives et de renforcer mes compétences en travail collaboratif, innovation et prise de décision rapide. Notre distinction à la **2e place du classement final** fut une source de motivation et de reconnaissance que je n'oublierai pas.

J'adresse également mes remerciements les plus chaleureux à ma collègue **Hiba CHAABNIA** pour son engagement, sa rigueur et son esprit d'équipe tout au long du projet et lors de notre participation au hackathon.

Enfin, je remercie mes enseignants pour la qualité de la formation reçue, ainsi que ma famille et mes proches pour leur soutien indéfectible, leur patience et leurs encouragements constants tout au long de cette aventure.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ABRÉVIATIONS

IA Intelligence Artificielle

LLM Large Language Model (Modèle de Langage de Grande Taille)

MAS Multi-Agent System (Système Multi-Agent)

API Application Programming Interface

NLP Natural Language Processing (Traitement du Langage Naturel)

AST Abstract Syntax Tree (Arbre Syntaxique Abstrait)

FIM Fill In the Middle (Remplissage du milieu de code)

INTRODUCTION GÉNÉRALE

L'évolution rapide des technologies numériques pousse les entreprises à adapter continuellement leurs systèmes d'information afin de rester compétitifs, performants et sécurisés. Toutefois, un grand nombre d'organisations dépend encore de logiciels dits « legacy », souvent développés avec des technologies anciennes, rigides, coûteuses à maintenir, et difficilement compatibles avec les standards actuels tels que les microservices, le cloud ou la conteneurisation.

Dans ce contexte, la modernisation de tels systèmes devient un enjeu stratégique. Cependant, la migration manuelle d'applications legacy est une tâche complexe, sujette aux erreurs humaines, mobilisant des compétences rares et des ressources considérables. C'est pourquoi les approches automatisées, basées sur l'intelligence artificielle (IA) et les systèmes multi-agents, offrent une alternative prometteuse.

Le présent travail propose le développement d'un système intelligent, modulaire et évolutif, capable de réaliser de manière autonome les différentes étapes du processus de migration — de l'audit au déploiement — à l'aide de modèles larges de langage (**LLMs**) et d'agents spécialisés orchestrés par le framework **CrewAI**.

Ce travail est présenté d'une en organisant le rapport en quatre chapitres principaux :

Chapitre 1 – Présentation générale du projet : Ce chapitre introduit le contexte professionnel du projet, la problématique rencontrée, les limitations des systèmes legacy, ainsi que la solution proposée basée sur un système multi-agent intelligent.

Chapitre 2 – Étude théorique : Il propose une analyse approfondie des approches classiques de migration, des concepts liés aux LLMs et aux systèmes multi-agents, ainsi qu'une revue critique des outils et des architectures existants dans ce domaine.

Chapitre 3 – Architecture technique du système développé : Ce chapitre détaille l'architecture fonctionnelle et logicielle de la solution développée, présente les différents agents mis en œuvre, les technologies utilisées, et justifie les choix d'intégration comme CrewAI et DeepSeek Coder.

Chapitre 4 – Implémentation et expérimentation : Il décrit concrètement les étapes de développement, l'implémentation des agents, l'interface utilisateur, ainsi que les résultats obtenus lors de l'expérimentation sur des cas réels, notamment dans le cadre d'un hackathon.

Ce projet vise à démontrer qu'il est possible d'automatiser de manière fiable, sécurisée et scalable la migration de systèmes legacy vers des environnements modernes. Il s'inscrit dans une volonté d'innovation technologique continue, au service de l'efficacité logicielle et de la transformation numérique des entreprises.

1 Présentation générale du projet

Sommaire

1.1 Introduction

L'évolution rapide des technologies modernes a mis en évidence la nécessité pour les entreprises de moderniser leurs applications héritées afin de rester compétitives et efficaces. Cependant, la migration des applications anciennes vers des technologies modernes est un processus complexe, nécessitant des ressources importantes et une expertise technique approfondie. Dans ce contexte, notre projet vise à développer un système multi-agent basé sur l'IA pour automatiser la migration des applications vers des technologies modernes. Ce système se base sur des LLMs et des frameworks tels que LangChain, LangGraph, ainsi que CrewAI pour l'orchestration des agents.

1.2 Présentation de l'organisme d'accueil

Mobelite est une agence spécialisée dans le conseil en stratégie mobile, la conception et le développement d'applications mobiles et web, ainsi que dans le marketing mobile. Grâce à son équipe d'experts, Mobelite se distingue par son savoir-faire dans la création de solutions digitales adaptées aux besoins des entreprises. L'entreprise dispose de plusieurs pôles stratégiques : des équipes commerciales et marketing basées à Paris (France), et des équipes de développement situées à Tunis, Sfax et Monastir, garantissant ainsi une approche globale et une exécution efficace des projets.

Fondée avec une vision innovante, Mobelite accompagne ses clients dans la définition et la mise en œuvre de stratégies de mobilité, en développant des produits performants et en évaluant leur impact à travers une analyse approfondie du retour sur investissement. Ses fondateurs, forts d'une expérience significative dans le domaine de l'innovation technologique et du développement mobile, ont su établir un partenariat stratégique avec un centre de développement offshore. Ce centre, existant depuis 2008, rassemble une équipe pluridisciplinaire composée de développeurs, de business analystes et d'ingénieurs QA, contribuant ainsi à la qualité et à la robustesse des solutions proposées par Mobelite.

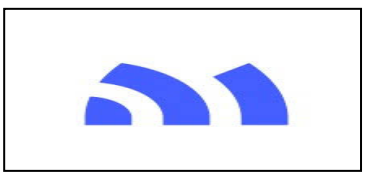


Figure 1 : Logo de Mobelite.

1.3 Contexte du projet

De nombreuses organisations possèdent encore des applications legacy fonctionnant sur des technologies obsolètes. Ces applications peuvent être coûteuses à maintenir, difficiles à intégrer avec de nouvelles solutions ou migrer vers des architectures modernes (microservices, conteneurs, cloud), et vulnérables aux menaces de sécurité et aux erreurs humaines. En réponse à ces défis, notre projet propose une approche automatisée basée sur un système multi-agent intelligent capable d'analyser, de refactorer et de migrer automatiquement ces applications vers des architectures modernes telles que les microservices, la conteneurisation ou les plateformes cloud.

1.4 Problématique

La migration des applications legacy vers des technologies modernes représente un défi majeur pour les entreprises, car elle soulève plusieurs problématiques critiques.

Tout d'abord, le processus est souvent manuel, long et complexe. Il implique l'identification minutieuse des dépendances, la compréhension approfondie du code existant ainsi que la restructuration des composants, ce qui demande un investissement conséquent en temps et en efforts.

De plus, chaque modification introduite nécessite une validation rigoureuse afin d'éviter l'apparition d'erreurs dans la version migrée. Ce travail manuel accroît également le risque d'erreurs humaines, pouvant entraîner des dysfonctionnements, des régressions, et compromettre la cohérence ainsi que la compatibilité avec l'environnement cible.

À cela s'ajoute le manque d'expertise spécialisée : la modernisation d'un système legacy exige des compétences pointues à la fois dans des technologies anciennes et dans les solutions modernes, rendant le recrutement ou la formation des développeurs particulièrement difficile et coûteux. L'impact financier et temporel est non négligeable, puisque la lenteur du processus contribue à l'augmentation des coûts de migration et de maintenance, tout en causant des retards qui freinent la transformation numérique et nuisent à la compétitivité de l'entreprise.

Enfin, l'adoption de nouvelles architectures, telles que les microservices, le cloud ou les conteneurs, reste difficile. Ces environnements nécessitent souvent une refonte complète des applications existantes, et l'absence de mécanismes d'automatisation complique davantage cette transition.

1.5 Étude et critique de l'existant

L'étude de l'existant est une étape fondamentale pour comprendre les caractéristiques des systèmes actuels, leurs limitations et les impacts potentiels d'une migration.

1.5.1 Identification des technologies legacy

Parmi les principales technologies considérées comme legacy, on retrouve plusieurs langages et systèmes largement dépassés mais encore utilisés dans certains domaines critiques tels que :

COBOL, un langage procédural apparu dans les années 1950, reste aujourd'hui incontournable dans certaines applications bancaires où il gère des transactions sensibles.

Visual Basic 6, une technologie événementielle développée par Microsoft et abandonnée depuis 2008, est encore présente dans de nombreuses applications métiers anciennes.

PHP 5, dont le support officiel a pris fin en 2018, est également largement répandu dans des intranets et portails web anciens.

MUMPS, système principalement utilisé dans les environnements hospitaliers pour la gestion des dossiers médicaux, demeure en usage malgré son ancienneté et ses limitations en matière de compatibilité avec les standards modernes.

1.5.2 Analyse critique des systèmes existants

Les systèmes legacy se caractérisent par plusieurs limitations techniques majeures qui freinent leur modernisation. Leur architecture est souvent monolithique, ce qui signifie que les composants sont étroitement couplés. Ainsi, la moindre modification peut impacter l'ensemble du système, rendant les évolutions risquées et complexes. De plus, l'absence de conteneurisation constitue un frein majeur au déploiement de ces applications sur des environnements modernes comme le cloud ou les architectures en microservices, limitant leur portabilité et leur flexibilité. En matière de sécurité et de conformité, ces systèmes souffrent généralement d'un retard important, notamment par l'absence de normes actuelles telles que l'authentification multifacteur (MFA) ou le chiffrement des données sensibles. Enfin, la dette technique accumulée au fil du temps se traduit par un coût de maintenance élevé, aggravé par l'absence de documentation à jour, ce qui complique considérablement toute tentative de reprise, de diagnostic ou d'évolution du système.

1.5.3 Comparaison avec les solutions modernes

Comparées aux systèmes legacy, les solutions technologiques modernes offrent de nombreux avantages significatifs. L'adoption d'une architecture basée sur les microservices permet une meilleure modularisation des composants, ce qui améliore considérablement la scalabilité et facilite les mises à jour indépendantes des différentes parties du système. De plus, la conteneurisation, notamment à travers des outils comme Docker et Kubernetes, optimise le déploiement des applications en assurant une portabilité accrue, une meilleure gestion des ressources et une isolation des environnements. Le cloud computing, quant à lui, permet de réduire les coûts liés à l'infrastructure physique tout en augmentant la résilience et la disponibilité des services. Enfin, l'intégration des pratiques DevOps facilite l'automatisation des déploiements, réduit les erreurs humaines et accélère les cycles de développement, contribuant ainsi à une amélioration continue de la qualité logicielle.

1.5.4 Impact organisationnel

Les limitations techniques des systèmes legacy ont également un impact organisationnel considérable. Les coûts de maintenance sont particulièrement élevés, pouvant engloutir jusqu'à 70 % du budget informatique, ce qui limite les marges de manœuvre pour l'innovation ou le développement de nouveaux services. Cette charge financière est accentuée par la complexité du code ancien, qui accroît le risque d'erreurs et allonge les délais de résolution des incidents, compromettant ainsi la fiabilité opérationnelle.

En parallèle, ces systèmes entravent la capacité des entreprises à intégrer rapidement de nouvelles fonctionnalités, ce qui entraîne un retard technologique face à la concurrence et freine la transformation numérique globale.

1.6 Solution Proposée

Pour répondre à cette problématique liée à la migration des applications legacy, notre projet propose le développement d'un système multi-agents basé sur l'IA, conçu pour automatiser, accélérer et fiabiliser le processus de migration vers des technologies modernes.

Notre approche repose sur l'utilisation de LLMs et d'agents autonomes, orchestrés à l'aide de frameworks tels que LangChain, CrewAI et LangGraph.

1.6.1 Analyse automatique des applications legacy

L'analyse automatique des applications legacy constitue la première étape du processus de migration. Elle consiste à identifier et cartographier les dépendances entre les différents modules et bibliothèques, afin de comprendre la structure globale du système.

Cette phase inclut également l'extraction et l'interprétation de la logique métier sous-jacente, ce qui permet d'assurer une cohérence fonctionnelle tout au long de la migration. Enfin, des rapports détaillés sont générés pour évaluer la complexité du code, détecter les éléments critiques et anticiper les risques liés à la transformation.

1.6.2 Génération intelligente de refactoring

La génération intelligente de refactoring vise à proposer des stratégies de modernisation adaptées à chaque projet, telles que la transformation d'architectures monolithiques en microservices, l'adoption de la conteneurisation ou encore la migration vers des environnements cloud.

Cette étape repose sur la réécriture et l'optimisation automatique du code existant en s'appuyant sur les meilleures pratiques en matière de développement logiciel. De plus, les recommandations de refactoring sont proposées en temps réel, avec la possibilité pour l'utilisateur de les valider ou de les ajuster manuellement, assurant ainsi un équilibre entre automatisation et contrôle humain.

1.6.3 Automatisation du processus de migration

L'automatisation du processus de migration repose sur l'intégration de modèles larges de langage open-source tels que LLaMA, Mistral, GPT-NeoX et DeepSeek-Coder, capables d'analyser le code existant et de générer automatiquement des versions optimisées. Ces modèles sont orchestrés à l'aide de frameworks comme LangGraph, LangChain et CrewAI, ce qui permet de structurer la collaboration entre les agents de manière fluide, modulaire et efficace.

Ce système prend également en charge la conversion vers des architectures modernes, qu'il s'agisse de microservices, de solutions serverless ou encore de conteneurs basés sur Docker et Kubernetes, assurant ainsi une transition technologique complète et adaptée aux standards actuels.

1.6.4 Validation et tests automatisés

La phase de validation et de tests automatisés vise à garantir la qualité et la conformité du code migré. Elle comprend la mise en place automatique de tests unitaires et fonctionnels afin de vérifier le bon fonctionnement de chaque composant et d'assurer que le comportement global du système reste conforme aux attentes.

Une comparaison rigoureuse est effectuée entre l'application legacy et sa version migrée, ce qui permet de détecter d'éventuelles régressions ou anomalies introduites durant la transformation. Enfin, des pipelines d'intégration et de déploiement continus (CI/CD) sont intégrés afin d'assurer un déploiement sécurisé, fiable et reproductible du nouveau système.

1.7 Méthodologie de gestion de projet adoptée

Pour bien comprendre et réussir notre projet, nous avons opté pour la méthodologie Data Driven Scrum (DDS), spécifiquement conçue pour les projets de data science. Cette approche agile et itérative permet de structurer le travail autour d'expérimentations successives et d'adaptations, tout en s'adaptant aux particularités des projets orientés données.

1.7.1 Principes fondamentaux de DDS

La méthodologie Data Driven Scrum (DDS) repose sur trois principes fondamentaux qui en font une approche particulièrement adaptée aux projets de data science. Ces principes sont comme suit :

Le premier principe est basé sur des cycles d'expérimentation itératifs et adaptatifs. Chaque itération est conçue comme un cycle complet incluant la formulation d'une hypothèse, sa mise à l'épreuve par l'expérimentation, puis l'analyse des résultats obtenus. Ce processus permet une amélioration continue tout en validant ou invalidant les hypothèses de départ.

Le second principe concerne la flexibilité dans la gestion des itérations. Contrairement aux méthodes agiles classiques qui imposent souvent des sprints fixes, DDS permet d'adapter la durée des itérations à la capacité de l'équipe et aux exigences du projet ; celles-ci peuvent durer d'un seul jour à plusieurs semaines, tout en garantissant la livraison de blocs de travail cohérents.

Enfin, le troisième principe repose sur une estimation simplifiée des tâches. En raison de l'incertitude inhérente aux projets orientés données, DDS privilégie des estimations globales plutôt que des prévisions précises, ce qui facilite la priorisation des tâches tout en conservant une marge de manœuvre suffisante pour gérer les imprévus.

1.7.2 Processus de DDS : Créer - Observer - Analyser

Chaque itération dans la méthodologie DDS suit un processus structuré en trois étapes clés. La première étape, « Créer », consiste à concevoir un livrable ou une fonctionnalité en partant d'une hypothèse initiale, qui guidera l'expérimentation. L'étape suivante, « Observer », permet de collecter les résultats produits et de s'assurer que les mesures obtenues sont pertinentes, fiables et exploitables. Enfin, l'étape « Analyser » vise à étudier en profondeur les résultats recueillis afin de tirer des enseignements concrets et de définir les prochaines actions à entreprendre. Ce cycle méthodique garantit une adaptation continue aux réalités du projet tout en maximisant l'apprentissage par l'expérimentation.

1.7.3 Illustration des artifacts DDS

La figure ci-dessous illustre le flux des tâches dans le cadre de la méthodologie DDS, incluant les étapes de création, d'observation et d'analyse :

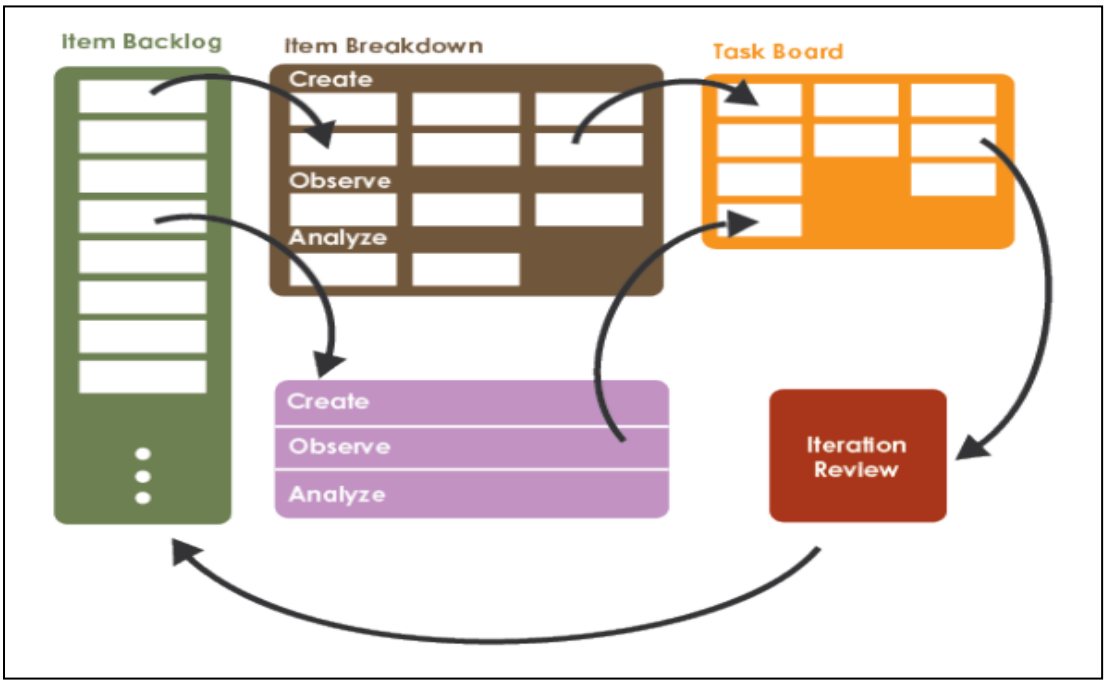


Figure 2 : Illustration de la méthodologie DDS (Créer–Observer–Analyser).

1.7.4 Backlog du produit

Le Backlog du produit est la liste des fonctionnalités attendues d'un produit. Scrum n'impose pas de pratique pour identifier et nommer les éléments du Backlog. L'usage le plus courant est de définir un élément comme étant une story ou un cas d'utilisation. Dans un Backlog du produit, les stories sont classées selon l'ordre envisagé pour leur réalisation. Cette

notion de priorité prend une grande importance dans le développement itératif. La technique utilisée pour prioriser les besoins dans un contexte itératif est celle de MoSCoW.

Tableau 1: Backlog produit - système IA multi-agent.

ID	User Story	Priorité
US1	En tant qu'utilisateur, je peux importer un projet legacy pour lancer la migration.	Haute
US2	En tant qu'utilisateur, je peux suivre l'avancement de la migration en temps réel.	Haute
US3	En tant qu'utilisateur, je peux consulter les rapports d'audit générés par les agents.	Haute
US4	En tant qu'utilisateur, je peux visualiser et corriger manuellement les suggestions de refactoring.	Moyenne
US5	En tant qu'agent, je peux analyser le code source pour détecter les dépendances.	Haute
US6	En tant qu'agent, je peux migrer le code legacy vers Next.js 14.	Haute
US7	En tant qu'agent, je peux générer automatiquement des tests unitaires.	Moyenne
US8	En tant qu'agent, je peux générer la documentation technique du code migré.	Moyenne
US9	En tant qu'agent, je peux vérifier la sécurité du code migré.	Haute
US10	En tant qu'utilisateur, je peux interagir avec le projet migré via un terminal web intégré.	Basse

1.8 Analyse et spécification des besoins

Notre système de migration automatique d'applications legacy doit répondre à un ensemble de besoins fonctionnels et non fonctionnels afin d'assurer une migration efficace, sécurisée et évolutive.

1.8.1 Besoins fonctionnels

Au cours de cette étape, nous allons spécifier les différentes fonctionnalités qu'offre notre portail Web. Le portail permet à l'utilisateur :

Gestion des projets : Dans le cadre du système de migration automatisée, la gestion des projets repose sur deux fonctionnalités principales : Tout d'abord, l'utilisateur a la possibilité d'importer les projets de migration, notamment en chargeant le code source à traiter.

Ensuite, une interface dédiée permet le suivi en temps réel de l'état d'avancement des projets, incluant la progression des différentes étapes, les journaux d'activité générés par les agents d'intelligence artificielle, ainsi que l'identification des éventuelles erreurs rencontrées au cours du processus. Cette gestion centralisée offre une vision claire et continue de l'évolution de chaque migration.

Supervision des agents AI : Elle constitue un élément central du système. Elle permet d'afficher les rapports détaillés générés par les agents tout au long du processus de migration. Ces rapports incluent l'analyse approfondie du code legacy, mettant en évidence les dépendances entre les modules, le niveau de complexité, ainsi que les technologies utilisées. Les agents proposent également des suggestions de refactoring, formulant des recommandations d'amélioration du code, lesquelles peuvent être validées ou corrigées manuellement avant leur exécution. Une fois le code migré généré automatiquement, il est comparé à la version originale afin de mesurer les écarts et s'assurer de la cohérence fonctionnelle. En complément, le système offre une visualisation claire des dépendances entre les modules, aussi bien avant qu'après la migration, facilitant ainsi l'évaluation de l'impact des transformations apportées.

Amélioration et correction des résultats : Le système offre la possibilité d'intervenir manuellement sur les résultats proposés par les agents IA afin d'en affiner la qualité. L'utilisateur peut ainsi modifier ou ajuster les suggestions générées, en fonction de contraintes métier ou de préférences techniques spécifiques. Par ailleurs, il est possible d'ajouter des règles personnalisées qui seront prises en compte lors des prochaines migrations, contribuant ainsi à améliorer le niveau d'automatisation et la pertinence des propositions futures. Toutes les corrections effectuées sont enregistrées dans un système de versioning, permettant de conserver un historique complet des ajustements et de garantir la traçabilité des modifications apportées.

Suivi et Audit de la Migration : Le suivi et l'audit de la migration sont assurés par un système d'enregistrement complet des logs générés au cours du processus. Ces journaux permettent d'effectuer des audits détaillés et facilitent le diagnostic en cas d'anomalie. En complément, le système génère des rapports exhaustifs sur les performances des agents IA, ainsi que sur les résultats obtenus à chaque étape. Pour renforcer la fiabilité du processus, des mécanismes d'alerte et de notification sont mis en place afin d'informer l'utilisateur immédiatement en cas d'erreur critique ou d'incohérence détectée dans la migration.

1.8.2 Besoins non fonctionnels

Pour garantir la qualité, l'efficacité et la sécurité du système de migration automatisée, plusieurs exigences non fonctionnelles doivent être prises en compte.

Performance : Le système doit assurer une exécution rapide et optimisée des processus de migration, même lorsqu'il s'agit d'applications complexes. Il est essentiel de minimiser la consommation des ressources tout en maintenant des temps de réponse courts. Pour cela, une gestion intelligente du cache est mise en place afin d'éviter la répétition de tâches coûteuses en calcul.

Fiabilité : Un taux de succès élevé doit être garanti lors des opérations de migration. Le système doit être capable de détecter automatiquement les erreurs et de les gérer efficacement à l'aide de mécanismes de reprise, comme le rollback en cas d'échec. La validation des résultats à chaque étape du processus est également indispensable pour limiter les risques de régressions fonctionnelles.

Sécurité : La protection des données est primordiale. Toutes les informations sensibles, telles que les codes sources ou les fichiers de configuration, doivent être chiffrées. Un contrôle d'accès basé sur les rôles est mis en place pour éviter toute action non autorisée, tandis qu'un mécanisme de traçabilité et de journalisation garantit que toutes les actions des utilisateurs et des agents IA sont enregistrées et vérifiables.

Scalabilité : Le système doit être capable de gérer une grande variété d'applications, allant de petits projets monolithiques à des architectures complexes en microservices. Il doit également permettre la gestion de plusieurs migrations simultanées sans dégradation des performances. Pour répondre à une montée en charge potentielle, une intégration avec des infrastructures cloud telles qu'AWS, Azure ou GCP est envisagée.

Extensibilité : Enfin, le système doit offrir une extensibilité suffisante pour permettre la personnalisation des pipelines de migration en fonction des besoins spécifiques des entreprises. Cette capacité d'adaptation est essentielle pour assurer la pérennité et l'évolution du système face à des environnements techniques et fonctionnels en constante mutation.

1.9 Conclusion

Ce premier chapitre a permis d'établir le cadre général du projet, en exposant le contexte dans lequel il s'inscrit, les problématiques rencontrées par les entreprises utilisant des systèmes legacy, ainsi que les limites des solutions traditionnelles de migration. Face à ces constats, une solution innovante a été proposée, reposant sur l'orchestration d'un système multi-agent alimenté par des modèles larges de langage. Cette solution ambitionne de rendre le processus de migration plus rapide, plus fiable, plus automatisé et mieux adapté aux environnements technologiques modernes. En adoptant une méthodologie de gestion de projet agile, centrée sur l'expérimentation et l'adaptation continue (DDS), le projet vise à maximiser l'efficacité du pipeline de migration tout en tenant compte des spécificités des applications à transformer. Les besoins fonctionnels et non fonctionnels ont également été détaillés, afin de poser des fondations solides pour la conception technique du système.

Le chapitre suivant approfondira les aspects théoriques liés aux technologies legacy, aux LLMs et aux architectures multi-agents qui constituent le socle de la solution envisagée.

2 Étude théorique

Sommaire

2.1 Introduction

La migration des applications legacy vers des technologies modernes est un défi majeur pour les entreprises cherchant à améliorer leur efficacité opérationnelle, leur scalabilité et leur sécurité. Cependant, ce processus est souvent complexe, coûteux et sujet à des erreurs. Ce chapitre explore les approches actuelles de migration, l'utilisation des systèmes multi-agents pour la gestion des tâches complexes et les modèles de langage pour le refactoring et la génération de code.

2.2 Applications legacy

2.2.1 Définition

Une application legacy désigne un système logiciel ou une application informatique reposant sur des technologies, des langages ou des infrastructures considérés comme obsolètes. Bien que ces applications soient encore opérationnelles et qu'elles continuent à répondre aux besoins des entreprises, elles présentent de nombreuses limitations. Elles ne sont généralement plus évolutives, souffrent de failles de sécurité et sont difficilement compatibles avec les environnements technologiques modernes, notamment en matière d'intégration cloud, de conteneurisation ou de microservices.

Parmi les exemples les plus courants, on retrouve les systèmes bancaires développés en COBOL qui assurent des transactions critiques, les applications d'assurance construites en Visual Basic 6 pour la gestion des sinistres, ainsi que des intranets en PHP 5 toujours en production malgré l'absence de support de sécurité. Dans le domaine de la santé, certains hôpitaux utilisent encore des systèmes comme MUMPS, à l'image de VistA, pour le traitement des dossiers médicaux. Enfin, de nombreux ERP client-serveur ont été développés en PowerBuilder ou Delphi, des technologies aujourd'hui dépassées mais encore présentes dans des structures informatiques vieillissantes.

2.2.2 Quand une application est-elle considérée comme legacy ?

Une application est considérée comme legacy lorsqu'elle présente un ensemble de caractéristiques techniques et structurelles qui la rendent obsolète face aux exigences des environnements modernes. Elle utilise des technologies obsolètes, comme des langages ou frameworks anciens (par exemple COBOL, VB6 ou PHP 5), qui ne sont plus maintenus ni

enseignés. Son architecture est généralement monolithique, ce qui signifie que les composants sont fortement couplés, rendant chaque modification complexe, risquée et coûteuse. Le support technique est souvent limité ou inexistant, le fournisseur d'origine ne fournissant plus ni mises à jour, ni correctifs de sécurité. Des problèmes de sécurité sont également fréquents, ces applications n'intégrant pas les normes actuelles telles que le chiffrement des données ou l'authentification multifacteur (MFA). Elles sont aussi incompatibles avec les infrastructures cloud modernes, car leur conception ne permet pas une conteneurisation ou une migration aisée vers des environnements virtualisés. Enfin, elles ont souvent atteint leur fin de vie (end-of-life), ce qui signifie qu'aucune mise à jour, ni solution de remplacement directe n'est disponible.

Ces différents critères permettent de qualifier clairement une application de legacy, et soulignent l'importance de prévoir sa migration vers une architecture plus moderne, évolutive et sécurisée.

2.2.3 Approches classiques de la migration legacy

Les approches classiques de migration des applications legacy sont nombreuses, mais elles poursuivent toutes un objectif commun : moderniser le système tout en limitant les risques d'interruption. Parmi ces approches, on peut citer :

Encapsulation : Elle consiste à encapsuler la logique métier existante dans des API ou des services web, permettant ainsi d'interagir avec le système sans modifier le code source. Par exemple, une base de données Oracle legacy peut être exposée via une API REST, afin d'être consommée par des services modernes.

Replatforming : Il implique le déplacement de l'application vers une infrastructure plus récente, comme le cloud, sans modification du code source. C'est le cas, par exemple, lorsqu'un mainframe COBOL est migré vers une machine virtuelle AWS EC2 tout en conservant son code intact.

Rehosting : Souvent appelé "Lift & Shift", il désigne le transfert d'une application legacy vers une nouvelle plateforme, qu'il s'agisse d'un environnement cloud ou d'une machine virtuelle, sans aucun changement dans le code ou les dépendances. Un exemple courant est le déplacement d'un ERP fonctionnant sur Windows Server 2003 vers un serveur Windows Server 2019.

Refactoring : Il consiste à réécrire partiellement ou totalement le code source afin de le rendre plus modulaire, maintenable et conforme aux standards d'architectures modernes. Par exemple, un système monolithique peut être restructuré en microservices tout en conservant les fonctionnalités d'origine.

Reengineering : Il va plus loin, puisqu'il s'agit d'un redéveloppement complet de l'application, en réutilisant les mêmes fonctionnalités mais avec des technologies modernes. Une application écrite en VB6 peut, par exemple, être reconstruite à l'aide de frameworks comme Angular et .NET Core.

Replacement : Il correspond au remplacement total de l'application legacy par un progiciel ou une solution SaaS équivalente. Par exemple, un ERP développé en interne peut être abandonné au profit de solutions comme SAP ou Salesforce.

Retirement : Il concerne les cas où l'application est totalement abandonnée, parce qu'elle est devenue redondante ou que ses fonctions ont été intégrées ailleurs. Un CRM interne, par exemple, peut être supprimé au profit d'un outil cloud moderne comme HubSpot ou Zoho.

2.2.4 Pourquoi les approches classiques ont échoué ?

Les approches classiques de migration legacy, bien qu'efficaces dans certains cas, ont montré leurs limites pour les raisons suivantes :

Tableau 2: Les limites des approches classiques.

Approche	Problème Principal	Exemple
Encapsulation	Le code legacy reste intact, maintenant ainsi les problèmes de performance et de sécurité.	Encapsulation d'un mainframe COBOL exposant des vulnérabilités non corrigées.
Replatforming	Les dépendances ne sont pas modernisées, augmentant la complexité de maintenance.	Un ERP Windows NT transféré sur une VM Windows 2019 reste difficile à maintenir.
Rehosting	Aucune optimisation du code, entraînant des coûts d'infrastructure élevés.	Un lift & shift d'un monolithe PHP5 vers AWS sans réduction de la dette technique.

Refactoring	Processus long et coûteux, surtout avec un code source mal documenté.	Refactorer un CRM en VB6 vers une architecture en microservices nécessite des ressources humaines spécialisées.
Reengineering	Risque de régression fonctionnelle et de perte de données.	Recréer une application de gestion des stocks sous Angular + .NET peut entraîner des bugs non prévus.
Replacement	Coût élevé et résistance des utilisateurs aux nouveaux systèmes.	Remplacer un ERP legacy par SAP implique des mois de formation des employés.
Retirement	Risque de perte de données critiques si le système est retiré trop rapidement.	Retirer un CRM sans plan de migration des données entraîne des pertes de contacts clients essentiels.

2.3 Du Deep Learning à l'ère des Transformers

2.3.1 Deep Learning

Le Deep Learning [1] est une sous-discipline du Machine Learning qui repose sur des réseaux de neurones artificiels composés de multiples couches. Ces réseaux, appelés réseaux profonds, permettent d'apprendre des représentations complexes des données à partir de grandes quantités d'exemples. Les architectures les plus courantes incluent :

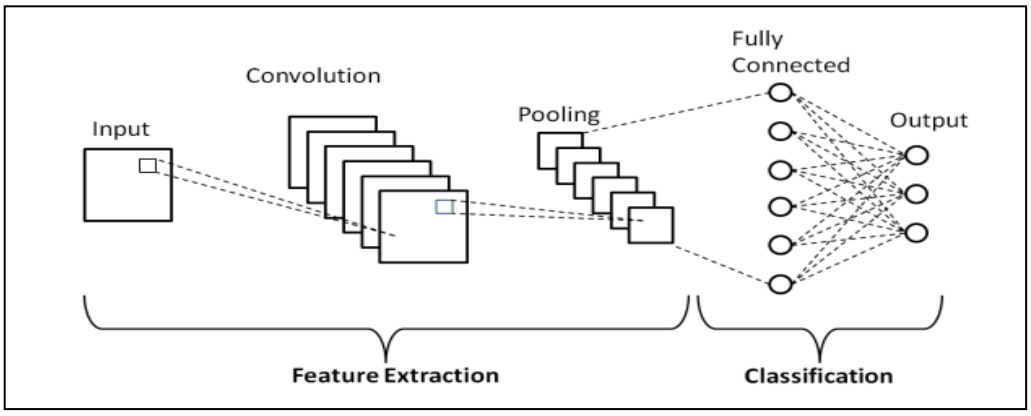


Figure 3 : Architecture CNN.

CNNs (Convolutional Neural Networks) : Utilisées principalement pour le traitement des images, ces architectures extraient les caractéristiques essentielles par le biais de filtres convolutifs.

RNNs (Recurrent Neural Networks) : Exploitées pour les séquences temporelles, ces architectures possèdent une mémoire interne permettant de capturer les dépendances entre les éléments d'une séquence.

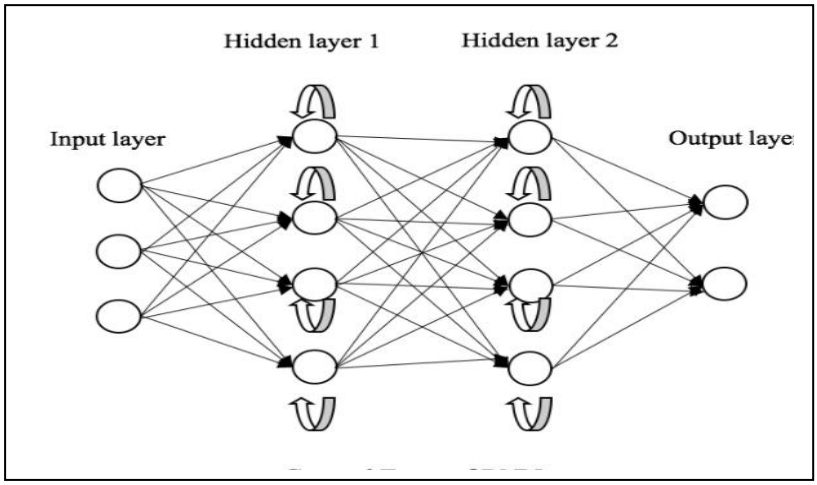


Figure 4 : Architecture RNN.

Bien que ces architectures aient permis des avancées significatives dans divers domaines (vision par ordinateur, NLP), elles présentent les limitations majeures suivantes :

- Difficultés à paralléliser le traitement des séquences longues.
- Perte d'information contextuelle pour les séquences longues (surtout pour les RNNs).
- Temps de calcul élevé pour des réseaux profonds très larges.

2.3.2 Transformers

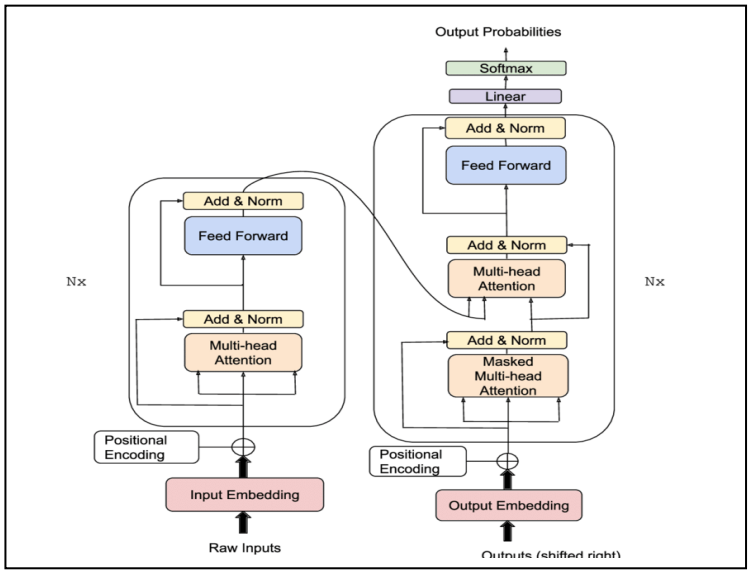


Figure 5 : Architecture du Transformer (Vaswani et al, 2017).

Proposés par Vaswani et al. en 2017 [2], les Transformers ont profondément bouleversé le domaine du Deep Learning. Contrairement aux RNNs qui traitaient les données de manière séquentielle, les Transformers reposent sur un mécanisme d'attention qui permet un traitement parallèle plus efficace et plus puissant, notamment pour les séquences longues.

L'architecture des Transformers repose sur plusieurs composantes fondamentales. Le mécanisme d'attention permet de calculer les relations entre tous les tokens d'une séquence simultanément, ce qui améliore la gestion du contexte global. Le mécanisme *multi-head attention* introduit plusieurs têtes d'attention indépendantes, capables de capturer des types de relations variées entre les éléments d'entrée. Le *positionnal encoding*, quant à lui, ajoute une information sur l'ordre des tokens, indispensable pour compenser l'absence de structure séquentielle inhérente à cette architecture. Enfin, le réseau *feed-forward*, composé de couches denses, est appliqué à chaque token indépendamment afin de mettre à jour ses représentations contextuelles de manière efficace.

Les avantages des Transformers sont nombreux. Le parallélisme accru permet un traitement simultané des tokens, réduisant considérablement le temps d'entraînement par rapport aux RNNs. La scalabilité est également remarquable : cette architecture peut être étendue à des modèles comportant des centaines de milliards de paramètres, sans dégradation des performances. De plus, la gestion des contextes longs est particulièrement performante, car les relations entre les tokens sont calculées globalement, ce qui améliore la compréhension des séquences étendues.

2.3.3 Transition vers les Transformers

La transition des architectures traditionnelles du Deep Learning, telles que les CNNs, RNNs et LSTM, vers les Transformers s'est imposée pour répondre à deux limitations majeures : la difficulté de paralléliser les traitements séquentiels et la faible capacité à gérer efficacement les contextes longs. Les Transformers, en s'appuyant sur un mécanisme d'attention, ont permis de surmonter ces obstacles tout en offrant une structure plus souple et évolutive. Des modèles emblématiques comme BERT et GPT ont ainsi démontré leur supériorité dans des tâches complexes telles que la traduction automatique, le résumé de texte, ou encore le refactoring de code source.

Cette transition vers les Transformers s'est matérialisée par l'abandon des structures séquentielles classiques au profit d'un traitement parallèle basé sur l'attention. L'introduction des *embeddings* a permis de mieux capturer le contexte sémantique de chaque token,

enrichissant les représentations internes du texte. Par ailleurs, des techniques comme le *masking*, utilisées notamment dans BERT, ont été intégrées pour prédire les tokens manquants dans une séquence, favorisant ainsi un apprentissage contextualisé plus fin.

L'impact sur le traitement du langage naturel (NLP) a été considérable. Les Transformers ont permis des avancées spectaculaires dans ce domaine, donnant naissance à des modèles de grande envergure tels que GPT-3, T5 ou encore CodeT5. Ces modèles sont capables non seulement de générer du texte de manière cohérente, mais aussi de traduire des séquences complexes et de restructurer automatiquement du code legacy, rendant possible une automatisation poussée des tâches de programmation.

2.4 IA générative et LLMs

2.4.1 IA générative

L'IA générative [3] se concentre sur la création de nouveaux contenus à partir de modèles préalablement entraînés sur de vastes volumes de données. Ces contenus peuvent prendre des formes très diverses, incluant des images, du texte, du code, des vidéos ou encore des modèles 3D. Les algorithmes génératifs reposent principalement sur des architectures de réseaux neuronaux profonds, telles que les GANs (Generative Adversarial Networks), les VAEs (Variational Autoencoders) et les Transformers.

Les GANs fonctionnent selon une approche adversariale, où un réseau générateur produit des données synthétiques tandis qu'un réseau discriminateur évalue leur authenticité, créant ainsi une boucle d'apprentissage compétitive. Les VAEs, quant à eux, encodent les données dans un espace latent compressé, avant de les décoder pour générer de nouvelles instances similaires aux données d'origine. Les Transformers utilisent le mécanisme d'attention pour générer du contenu séquentiel, comme du texte ou du code, et sont à la base de modèles puissants tels que GPT ou T5.

Les applications de l'IA générative sont extrêmement variées. Elle permet par exemple la création d'images réalistes à partir de descriptions textuelles, comme le fait DALL·E, ou la génération automatisée de séquences textuelles, illustrée par les modèles GPT. Elle est également utilisée pour la synthèse vocale, la création musicale, ainsi que pour des usages techniques tels que le refactoring et la génération de code dans des environnements de développement logiciel.

2.4.2 Modèles larges de langage (LLMs)

Les LLMs sont des modèles de deep learning spécialisés dans le traitement et la génération de texte à grande échelle. Ces modèles sont pré-entraînés sur des corpus massifs de données textuelles, leur permettant d'apprendre les structures linguistiques, les contextes et les relations sémantiques entre les tokens. Les LLMs sont principalement structurés selon trois architectures :

- **Encoder-Only (BERT, RoBERTa) :** Axé sur la compréhension des séquences d'entrée, il excelle dans les tâches de classification et d'extraction d'entités.
- **Decoder-Only (GPT, Codex) :** Conçu pour générer du texte à partir d'un contexte donné, ce type de modèle est performant pour des tâches de génération créative.
- **Encoder-Decoder (T5, BART) :** Combine les avantages des deux précédents, permettant de convertir des séquences d'entrée en séquences de sortie (ex : traduction, résumé).

2.4.3 LLMs pour le refactoring de code

Les LLMs jouent un rôle crucial dans le processus de migration des applications legacy. Ils permettent d'analyser, de reformater et de générer du code automatiquement. Les LLMs open-sources entrainés sur les codes et qui sont les plus connus sont :

- **Codestral-2501** [4]**:** C'est un modèle spécialisé dans la génération de code, la correction syntaxique et la création de tests unitaires.
- **Codellama 70B** [5] **:** C'est un modèle de 70 milliards de paramètres, performant pour les tâches de refactoring, mais limité en termes de contexte (4K tokens).
- **DeepSeek Coder V2** [6] **:** Ce modèle est conçu pour le refactoring complexe et le multi-langage, avec une capacité contextuelle étendue à 128K tokens.

2.4.4 Comparaison de LLMs pour le refactoring de code

Les LLMs utilisés pour le refactoring de code varient en termes de capacités, de longueur de contexte, de fonctionnalités et de cas d'utilisation. Pour plus de clarté, la comparaison est divisée en deux tableaux : le premier se concentre sur les fonctionnalités et les cas d'utilisation, tandis que le second se concentre sur les performances et le support linguistique.

Tableau 3 : Fonctionnalités et cas d'utilisation des LLMs pour le refactoring.

Modèle	Cas d'utilisation	Fonctionnalités
Codestral-2501	- Complétion du code - FIM (Fill in the middle) tasks - Génération de tests - Correction des codes	- Spécialisé dans les tâches à faible latence et à haute fréquence comme FIM - Prise en charge de plus de 80 langages de programmation - Architecture et tokenizer améliorés pour une génération et une complétion de code plus rapides (environ deux fois plus rapides que les versions précédentes)
Codestral-2405 22B	- Génération de code - Débogage - Tests unitaires - Synthèse et documentation du code	- Similaire à Codestral-2501, mais avec une taille de paramètre différente (22 milliards). - Prend en charge une longueur de contexte de 32K jetons. - Excellent pour les tâches FIM dans divers langages de programmation.
Codellama 70B Instruct	- Synthèse et compréhension du code - Instructions / chat	- Modèle optimisé pour les instructions avec 70 milliards de paramètres. - Conçu pour suivre les instructions de codage et faciliter les tâches liées au code. - Prend en charge une longueur de contexte de 4K jetons.
DeepSeek Coder V2 Lite	- Saisie de code simplifiée - Suggestions de refactorisation de code - Fonctionnalités de correction et d'explication de base du code - Latence de réponse minimale	- Architecture Mixte d'Experts (MoE) avec 16 milliards de paramètres au total (dont 2,4 milliards actifs), compatible avec plus de 338 langages de programmation. - Prise en charge d'un contexte étendu de 128K jetons. - Capacités de codage et de raisonnement améliorées. - Adapté à l'assistance au code IA et au développement logiciel.

DeepSeek Coder 33B Instruct	- Génération de code - Complétion de code au niveau du projet - Tâches de remplissage - Réparation de code - Tâches de code multilingue - Tâches de langage général	- Modèle optimisé pour les instructions, compatible avec plus de 80 langages de programmation. - Pré-entraîné sur 2 000 milliards de jetons dans plus de 80 langages de programmation (87 % de code et 13 % de langage naturel en anglais et en chinois). - Prend en charge un contexte de 16K jetons. - Performances de pointe parmi les modèles de code open source.

Tableau 4 : Comparaison entre des LLMs pour le refactoring.

Model	Context length	RepoBench (Python)	LiveCode-Bench (Python)	Spider (SQL)	CanItEdit	HumanEval JavaScript	HumanEval Bash	HumanEval Typescript	HumanEval FIM JS		Local Run
									Single Line Exact Match	FIM pass@1	
Codestral-2501	256k	38.0%	37.9%	66.5%	50.5%	82.6%	43.0%	82.4%	87.96%	96.1%	No
Codestral-2405 22B	32k	34.0%	31.5%	63.5%	50.5%	71.4%	40.5%	74.8%	86.08%	95.0%	Yes
Codellama 70B instruct	4k	11.4%	20.0%	37.0%	29.5%	62.7%	32.3%	61.0%	-	-	Yes
DeepSeek Coder 33B instruct	16k	28.4%	27.0%	60.0%	47.6%	73.3%	39.2%	77.4%	86.80%	-	Yes
DeepSeek Coder V2 lite	128k	20.0%	28.1%	72.0%	41.0%	80.8%	34.2%	82.4%	85.90%	-	Yes

Les LLMs orientés instruction sont optimisés pour suivre efficacement les instructions humaines. Dans le contexte des LLMs axés sur le codage, ces modèles sont entraînés à comprendre et à exécuter des tâches de programmation basées sur des instructions en langage naturel. Ils peuvent compléter des extraits de code partiels en comprenant le contexte avant et après le segment manquant (*Fill-in-the-Middle (FIM)*). Cette capacité est particulièrement utile pour refactoriser et améliorer les bases de code existantes.

En tenant compte de la complexité des applications à migrer, de la nécessité d'un support multilingue, et de la capacité à traiter des contextes très larges, notre choix s'est porté sur DeepSeek Coder V2 Lite comme modèle principal intégré à notre pipeline multi-agent. Les indicateurs utilisés pour cette comparaison sont décrits en détail dans l'Annexe A. Ils permettent une évaluation fiable et reproductible des performances des LLMs dans des contextes concrets de développement logiciel.

2.5 Agents IA

2.5.1 Qu'est-ce qu'un agent IA ?

Un agent IA est une entité logicielle autonome capable de percevoir son environnement, de prendre des décisions basées sur ces perceptions et d'exécuter des actions pour atteindre un objectif spécifique. Un agent est généralement programmé pour effectuer des tâches spécifiques et peut interagir avec d'autres agents pour coordonner des actions complexes. Les agents IA peuvent être :

- **Réactifs :** Ils exécutent des actions simples en réponse à des événements.
- **Proactifs :** Ils planifient des actions pour atteindre des objectifs complexes.
- **Collaboratifs :** Ils communiquent avec d'autres agents pour coordonner les tâches.

Exemples d'agents dans le contexte de la migration legacy :

- Agent d'analyse de code : Analyse le code source, détecte les dépendances et génère un rapport sur la complexité du code.
- Agent de refactoring : Réécrit le code legacy en utilisant des modèles IA pour le rendre compatible avec les technologies modernes.
- Agent de test : Génère des tests unitaires pour le code migré et vérifie son intégrité.

2.5.2 Architecture cognitive d'un agent à base de LLM

Un agent n'est pas simplement un LLM isolé : il s'agit d'un système orchestral capable de percevoir, raisonner, planifier et agir de façon autonome. Un agent intelligent se compose alors de plusieurs éléments fondamentaux qui collaborent pour lui permettre de raisonner, d'interagir avec son environnement et de s'adapter aux tâches qui lui sont confiées.

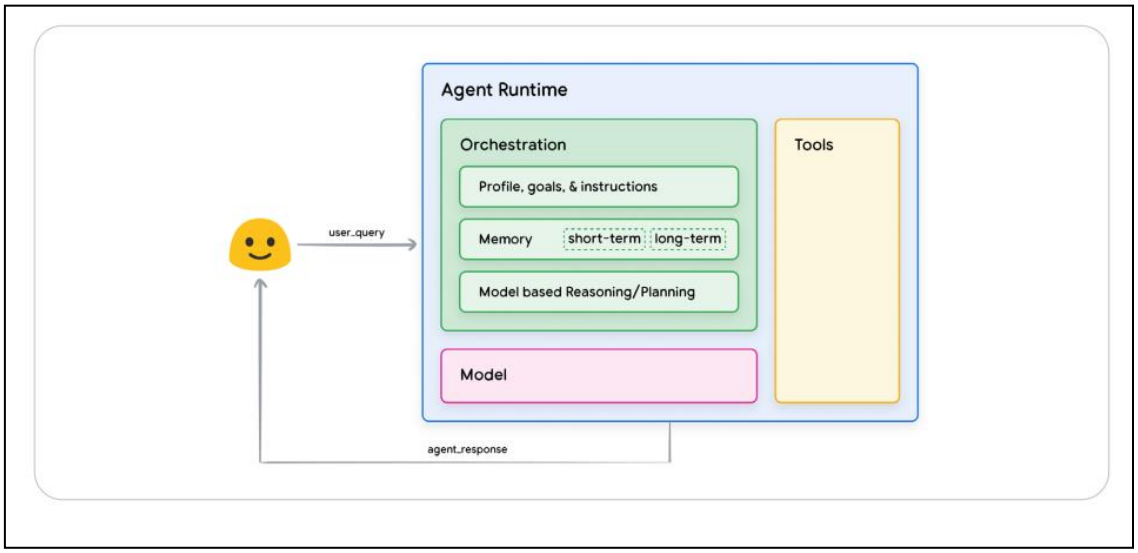


Figure 6 : Architecture d'un agent à base de LLM.

Modèle : Il constitue le cœur décisionnel de l'agent. C'est lui qui applique les logiques de raisonnement comme ReAct (*Reasoning and Acting*), CoT (*Chain-of-Thought*) ou ToT (*Tree-of-Thoughts*), afin d'élaborer des plans d'action et de prendre des décisions cohérentes. Le modèle peut être généraliste, spécialisé ou multimodal, en fonction du domaine d'application ou du niveau d'autonomie requis.

Outils : Ils permettent à l'agent d'interagir avec le monde extérieur, notamment avec des interfaces comme des APIs, des bases de données, des systèmes de fichiers ou des applications métier. Ces outils prennent différentes formes : fonctions côté client, extensions intégrées côté agent ou encore connecteurs vers des data stores. Par exemple, un agent peut interroger une API météo afin d'adapter ses recommandations en fonction des conditions climatiques.

Mémoire : Elle joue un rôle essentiel dans la continuité contextuelle. Elle se divise en deux types : la mémoire à court terme, qui conserve le contexte immédiat d'une session ou d'un échange, et la mémoire à long terme, qui enregistre des préférences, des apprentissages ou des épisodes passés importants. Différents mécanismes peuvent enrichir cette mémoire,

comme la réflexion, le partage d'informations entre agents ou l'utilisation de mémoires vectorielles.

Orchestration : Elle est responsable de la gestion complète du cycle de traitement : perception, raisonnement et action. Elle repose sur des frameworks de prompts pour guider le raisonnement logique de l'agent, en appliquant par exemple des méthodes comme ReAct, CoT ou ToT. L'orchestration prend également en charge la définition et le suivi de l'état de l'agent, de ses objectifs, ainsi que la sélection des outils appropriés à chaque situation.

Registre d'agents et d'outils (Mesh) : Il constitue une base de données centralisée dans laquelle sont enregistrés les agents et les outils disponibles. Ce registre contient également les ontologies, les capacités fonctionnelles, les descriptions techniques et les métadonnées de performance associées à chaque composant, facilitant leur découverte et leur gestion.

Boucles de feedback et d'apprentissage : Elles permettent aux agents d'évaluer leur propre performance et d'ajuster leur comportement. Ces boucles peuvent s'appuyer sur des métriques précises et s'exécuter de manière manuelle, avec un humain dans la boucle (*human-in-the-loop*), ou de façon automatisée, par exemple via des systèmes d'évaluation tels qu'"*autorater*" ou "*LLM-as-a-Judge*", qui permettent une régulation continue du raisonnement et des décisions de l'agent.

2.5.3 Rôle des LLMs dans les agents IA

Les LLMs constituent le cœur cognitif des agents IA. Contrairement à un simple outil de génération de texte, un LLM intégré dans une architecture agentique devient un composant décisionnel, doté de capacités de raisonnement, de planification et de communication.

De moteur de texte à moteur d'action : Seul, un LLM est limité à son entraînement initial et à la prédiction statistique de texte. Mais lorsqu'il est orchestré dans un cadre agentique, il devient capable de :

- Comprendre un objectif global
- Segmenter cet objectif en sous-tâches
- Choisir les outils les plus adaptés pour chaque tâche
- Générer des actions pilotées (API calls, requêtes, scripts…)
- S'auto-réévaluer et ajuster son comportement

Cette transformation repose sur l'ajout de composants externes (orchestration, mémoire, outils), mais aussi sur des stratégies internes de raisonnement, comme :

- **ReAct** (*Reasoning* + *Acting*) : boucle où le modèle réfléchit, choisit une action, observe le résultat, et recommence jusqu'à l'objectif.

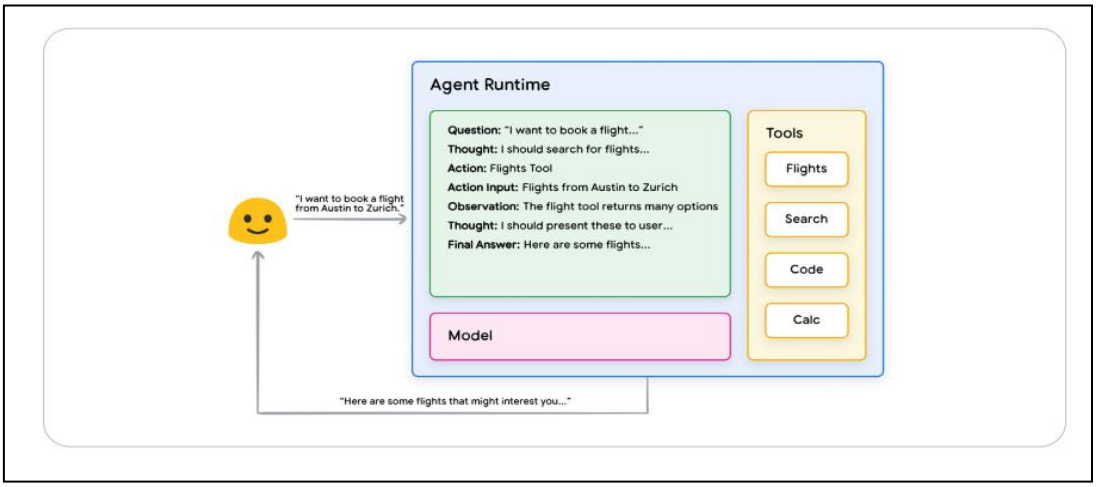


Figure 7 : Exemple d'agent avec raisonnement ReAct dans la couche d'orchestration.

- **CoT** (*Chain-of-Thought*) : structure la réflexion du LLM en étapes intermédiaires explicites.
- **ToT** (*Tree-of-Thoughts*) : permet l'exploration parallèle de plusieurs chaînes de raisonnement avant de choisir la meilleure

Tableau 5 : Rôle du LLM dans l'agent IA.

Capacité	Description
Multimodalité	Le LLM peut intégrer texte, image, audio selon le besoin de l'agent.
Contextualisation	Grâce à la mémoire, le modèle ajuste ses réponses selon l'historique des interactions.
Suivi d'instruction	Le LLM comprend des rôles complexes, ex. : « Tu es un agent de refactoring de code… »
Interprétation sémantique	Capable d'abstraction, reformulation, catégorisation, etc.

2.5.4 Agent vs. LLM

Bien que les agents et les LLMs puissent sembler similaires, ils ont des objectifs et des fonctionnements différents comme souligné dans le tableau suivant :

Tableau 6 : Comparaison entre un Agent IA et un LLM.

Caractéristiques	LLM (Modèle seul)	Agent IA (LLM orchestré)
Accès aux données	Limité aux données vues lors de l'entraînement	Étendu par l'utilisation d'outils externes (API, extensions, RAG, etc.)
Raisonnement	Peut suivre une prompt mais n'a pas de structure de logique native	Implémente un cadre cognitif (ReAct, CoT, ToT) pour planifier, décider, raisonner
Mémoire	Pas de mémoire native (stateless)	Dispose d'une mémoire de session, voire d'une mémoire long terme (vecteurs, résumés)
Actions	Répond par génération textuelle uniquement	Exécute des actions concrètes : appels API, navigation, requêtes, création de fichiers
Interactions continues	Fonctionne en mode question-réponse simple	Capable de gérer des sessions multi-turns, avec suivi d'état
Autonomie	Dépend de chaque requête utilisateur	Peut opérer de manière proactive ou autonome selon son objectif et ses règles
Outils intégrés	Aucun outil natif	Peut intégrer des outils, fonctions, extensions, data stores
Ciblage d'objectif	Répond à une prompt spécifique sans stratégie globale	Suit un objectif global défini via prompt + planificateur
Architecture logicielle	LLM seul (ex. : API OpenAI, Gemini, Claude)	Composé de plusieurs modules : LLM + orchestration + mémoire + outils

2.5.5 Architecture cognitive basée sur un seul agent IA

Un agent unique utilise un seul LLM pour toutes les étapes : compréhension, recherche d'information, raisonnement et génération. Cette approche est plus simple mais montre ses limites pour les cas d'usage complexes.

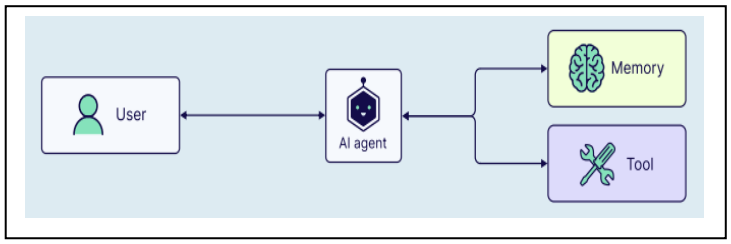


Figure 8 : Architecture basé sur un seul agent IA.

Tableau 7 : Avantages et limites d'un seul agent IA.

Avantages	Limites
- Faible complexité de développement - Pas de coordination requise - Moins de ressources nécessaires	- Difficulté à gérer les tâches longues ou complexes - Risque d'erreurs si trop d'outils sont appelés - Peut nécessiter un LLM plus puissant (et coûteux)

2.6 Systèmes multi-agents et outils pour la modernisation logicielle

2.6.1 Architecture multi-agent

Une architecture multi-agent répartit les tâches entre plusieurs agents spécialisés, chacun ayant son propre rôle (ex. : planification, récupération d'information, génération, évaluation).

Tableau 8 : Avantages et limites d'une architecture multi-agent.

Avantages	Limites
- Division du travail, spécialisation - Scalabilité facile (ajout ou retrait d'agents) - Plus robuste et traçable	- Complexité d'orchestration - Coordination nécessaire - Débogage plus délicat

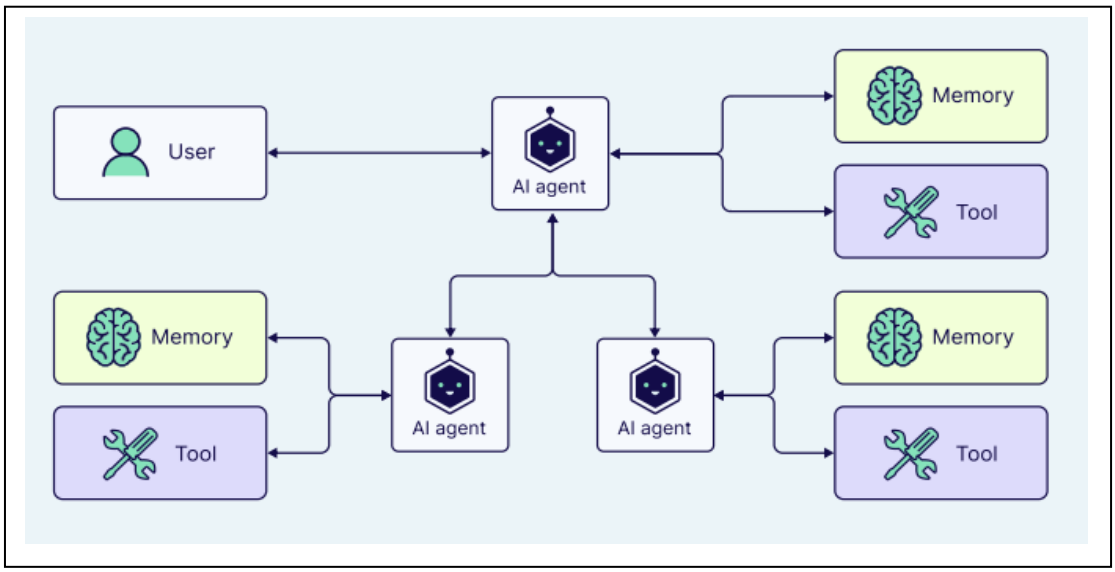


Figure 9 : Architecture multi-agent.

Afin de mieux illustrer les différences fondamentales entre une architecture basée sur un seul agent et celle reposant sur plusieurs agents, le tableau suivant présente une comparaison synthétique selon plusieurs critères techniques et opérationnels.

Tableau 9 : Un seul agent vs système multi-agent.

Critère	Agent Seul	Multi-Agent
Complexité	Faible	Élevée
Modulaire	Non	Oui
Résilience	Faible (point unique de défaillance)	Haute (agents isolés et remplaçables)
Adaptabilité	Limitée	Excellente
Performance sur tâches	Moyenne	Excellente si agents spécialisés

2.6.2 Systèmes multi-agents

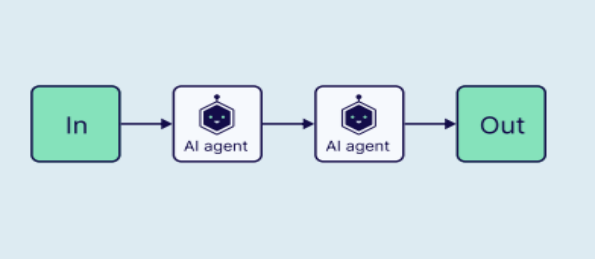
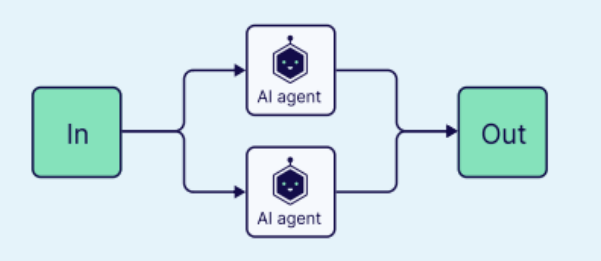
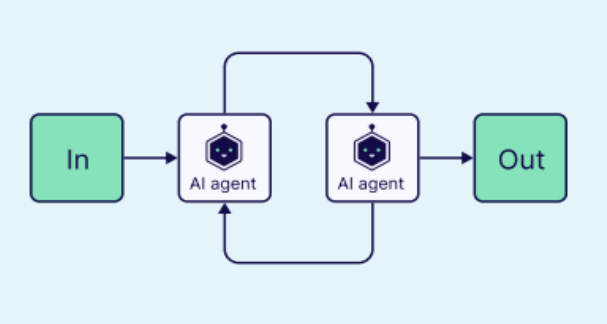
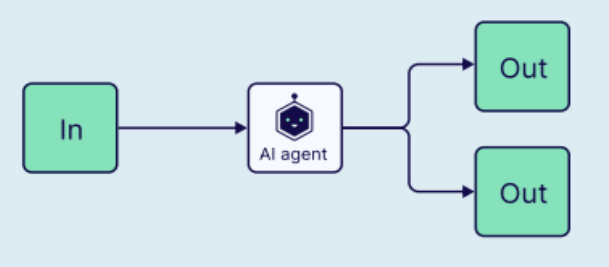
Les systèmes multi-agents (*Multi-Agent Systems* (MAS)) sont des architectures distribuées où plusieurs agents intelligents collaborent pour atteindre un objectif global. Chaque agent peut agir de manière autonome, tout en étant capable de coopérer avec les autres. Il est particulièrement adapté pour :

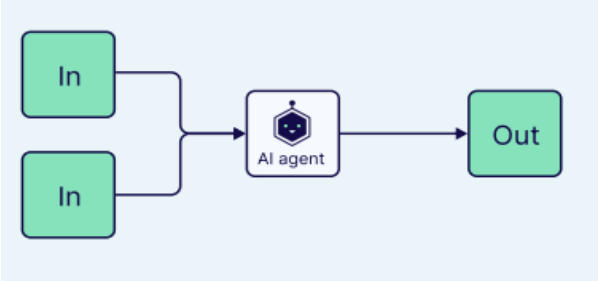
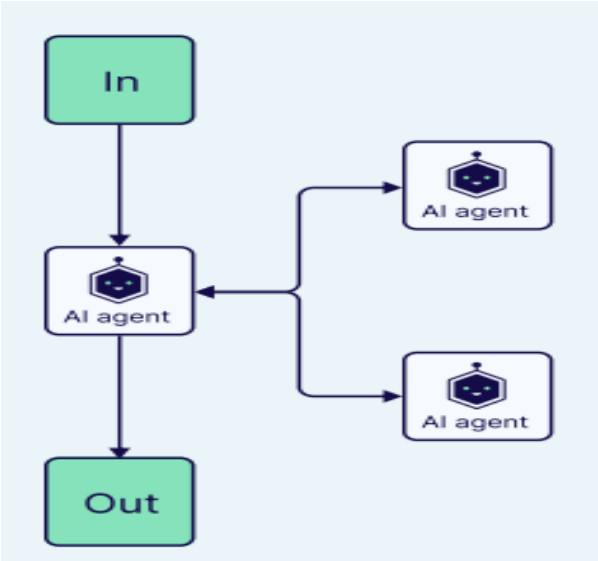
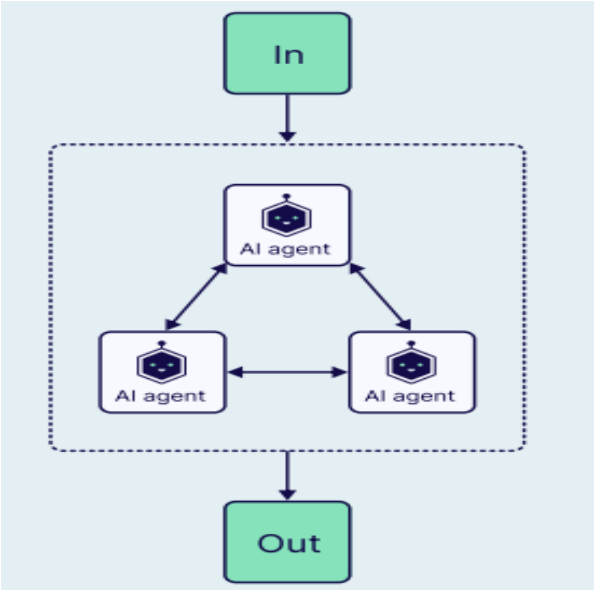
- La gestion des workflows complexes.
- La coordination des tâches parallèles (analyse, refactoring, tests).

❖ La gestion des états contextuels partagés entre plusieurs agents.

Plusieurs patrons peuvent être utilisés pour concevoir une architecture MAS, comme présenté dans le tableau suivant :

Tableau 10 : Patrons d'architectures MAS.

Patron	Description	Illustration
Séquentiel	Agents agissent en chaîne (l'un après l'autre)	
Parallèle	Plusieurs agents exécutent des tâches simultanément	
Boucle	Agents s'évaluent et s'améliorent mutuellement	
Routeur	Un routeur choisit quel agent exécute une tâche spécifique	

Agrégateur	Un agent regroupe les sorties des autres pour synthèse finale	
Hiérarchique	Organisation avec superviseur et agents subordonnés	
Réseau	Collaboration en pair-à-pair, sans hiérarchie	

2.6.3 Frameworks de développement

Afin de développer un système multi-agent, plusieurs frameworks de développement sont disponibles dans l'écosystème IA, comme présenté dans le tableau suivant :

Tableau 11 : Frameworks de développement de systèmes multi-agents.

Framework	**AutoGen**	**LangGraph**	**CrewAI**	**MetaGPT**	**Google ADK**	**LlamaIndex**
Objectif principal	Flux de travail de chat multi-agents	Orchestration LLM basée sur des graphes	Flux modulaires pour les agents IA	Agents de codage de type SOP	Kit d'orchestration d'outils/agents de Google	Flux de travail des agents centrés sur les documents
Type de flux de travail	Chats scriptés, utilisation d'outils	Exécution DAG/graphe	Flux piloté par les événements (@start, @listen)	Exécution séquentielle des SOP	Plans d'agent + mémoire + boucle d'exécution	Pipelines de requêtes, récupération et actions
Agents	Agents Python avec outils	Nœuds avec entrées/sorties	Agents/tâches définis par YAML	Rôles d'agent de style développeur	Configuration JSON ou basée sur du code	QueryEngine et agents utilisant des outils
Enchaînement de tâches/outils	Manuel via des scripts Python	Chaînage de graphes natifs	Enchaînement d'équipage modulaire	Flux d'exécution SOP défini	Planification d'agent intégrée	Déclencheurs basés sur des documents et des outils
Visualisation du flux	Non natif	Interface utilisateur Graphviz ou Streamlit	Diagramme de flux HTML	Non natif	Prévu dans le SDK	Via les carnets/l'interface utilisateur
Humain dans la boucle	Pause/reprise, nœuds d'entrée	Déclencheurs d'événements, entrée d'interface utilisateur	Injection par étapes de débit possible	Non pris en charge	Pris en charge via la mémoire et les outils	Basé sur les invites + outil de secours

Mémoire / État	Chat + mémoire globale	État transmis aux nœuds	Suivi de l'état pydantique	État implicite du SOP	Mémoire d'exécution riche	Mémoire vecteur + agent
Modularité	Réutilisation de script uniquement	Nœuds réutilisables	Équipages réutilisables et composables	Rôles statiques	Chaînes d'agents modulaires	Requête + réutilisation d'outils
Expérience du développeur	Centré sur Python, clair	Courbe d'apprentissage plus élevée	Adapté aux développeurs, configuration + code	Édition facile des SOP	Natif de l'écosystème Google	LLM-first, Pythonic
Idéal pour	Assistants, agents de recherche	Pipelines LLM conditionnels	RAG, automatisation, flux asynchrones	Simulation d'équipes de codage	Piles d'agents prêtes pour la production	Récupération, assurance qualité agentique, outils
Exemples de cas d'utilisation	Copilotes propulsés par GPT	Flux de travail graphiques multimodaux	Écrivain de livres, robots de réunion	Créer une base de code via des agents	Agents d'espace de travail, agents IDE	Agents RAG, chat avec outils
Maturité / Soutien	Stable, soutenu par Microsoft	Bibliothèque principale LangChain	OSS actif, en évolution rapide	Utilisation stable et de niche	Nouveau	Cadre RAG mature

2.6.4 MAS pour la migration legacy

Autonomous Legacy Web Application Upgrades Using a Multi-Agent System : Publiée le 31 janvier 2025 [7], cette solution propose un système multi-agent structuré autour de plusieurs agents spécialisés qui coopèrent pour automatiser efficacement le processus de refactoring d'applications legacy. Au centre de ce dispositif, on trouve le « *Manager Agent* », qui reçoit les exigences de l'utilisateur, les analyse, puis les décompose en sous-tâches séquentielles. Ce module est chargé d'assurer l'organisation et le suivi de l'ordre d'exécution de ces tâches.

Ces sous-tâches sont ensuite transmises à un « *Task Pipeline* », composé de deux agents complémentaires. Le premier, le « *Prompt Maker Agent* », est responsable de la génération des prompts nécessaires au refactoring du code legacy, en tenant compte des contraintes techniques et fonctionnelles. Le second, « *Execution Agent* », se charge de mettre en œuvre les actions décrites dans ces prompts, en réalisant concrètement les transformations du code. Une fois les tâches exécutées, le « *Verification Agent* » entre en jeu pour analyser le code généré et détecter d'éventuelles erreurs ou incohérences. Si des problèmes sont identifiés, ils sont transmis au « *Finalizer Agent* », dont le rôle est d'appliquer les corrections nécessaires. Ce dernier garantit ainsi que le code final est propre, fonctionnel et conforme aux attentes définies initialement par l'utilisateur.

Ce système multi-agent a été mis en œuvre dans le cadre d'une expérimentation concrète visant à migrer une application legacy développée en CakePHP 1.2 (2008) vers la version plus moderne CakePHP 4.5 (2023). Cette application, vieillissante et non compatible avec les standards actuels, représentait un cas typique de migration nécessitant des ajustements tant au niveau du backend que du frontend.

Les résultats de cette mise en œuvre ont été évalués à partir de plusieurs fichiers confrontés à des problèmes spécifiques, notamment en matière de compatibilité JavaScript, de format de données, ainsi que de refactoring AJAX. Trois indicateurs principaux ont été utilisés pour mesurer la performance du système. Le premier était le nombre d'erreurs détectées, englobant les erreurs fatales, les erreurs de syntaxe, ainsi que les fonctionnalités manquantes. Le second critère portait sur le temps d'exécution, évalué en mesurant la durée moyenne requise pour l'accomplissement de chaque tâche de migration. Enfin, une analyse comparative des lignes de code (LOC) a permis d'observer les différences structurelles entre le code legacy et le code généré après migration.

Les conclusions de l'expérimentation montrent que les agents basés sur des LLMs ont obtenu des résultats satisfaisants sur des tâches simples, telles que les mises à jour de syntaxe ou les corrections de compatibilité mineures. En revanche, ils ont rencontré des difficultés notables dans le traitement de tâches complexes, en particulier lors du refactoring de logique métier ou de la gestion de dépendances avancées. Le système a généré un taux d'erreurs plus élevé que les interventions manuelles pour les cas complexes, mais a montré un gain de temps significatif sur les tâches simples, soulignant ainsi un bon potentiel d'automatisation partielle dans un pipeline de migration hybride.

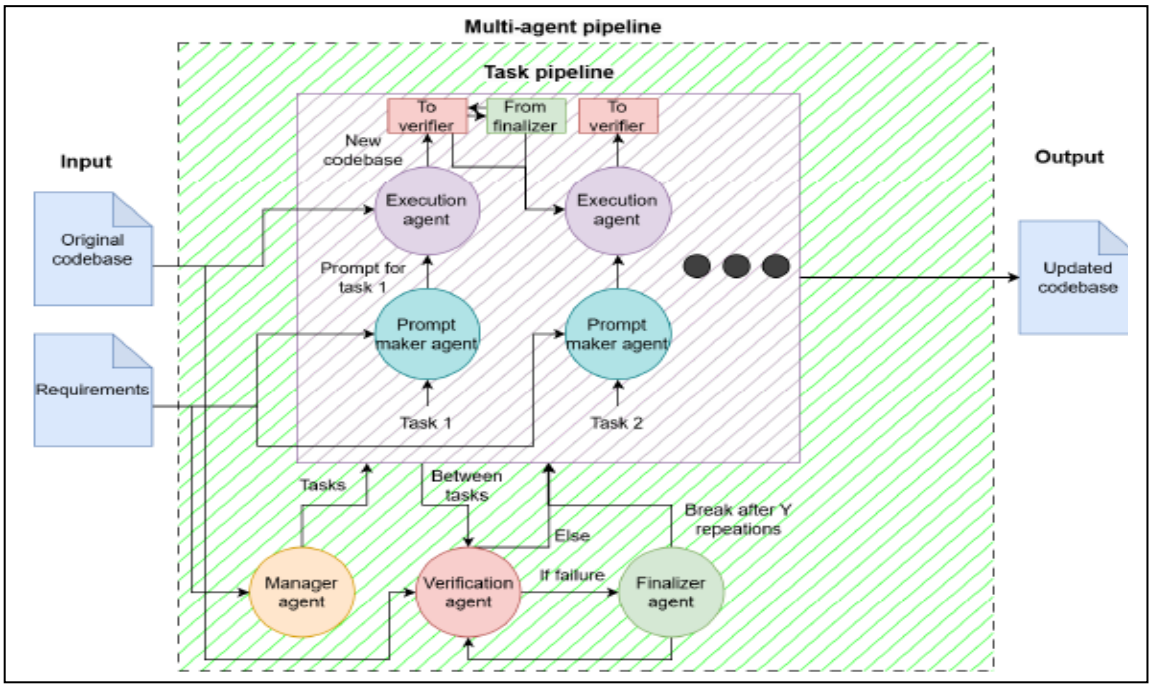


Figure 10 : Système multi-agent de mise à jour de code legacy.

AgentCoder: Multi-Agent Code Generation with Effective Testing and Self-Optimisation : Publiée le 24 mai 2024 [8], la solution propose une approche multi-agent axée sur la génération de code, la conception de tests et l'optimisation par autoréflexion. Cet AgentCoder intègre plusieurs agents autonomes qui coopèrent pour assurer un cycle complet de production logicielle automatisée :

En premier, le « *Programmer Agent* » est chargé de la génération du code. Il reçoit des prompts spécifiques fondées sur le contexte de la tâche à accomplir et s'appuie sur des représentations graphiques structurées pour mieux appréhender les complexités inhérentes au code à produire. Cette approche visuelle favorise une meilleure structuration logique des blocs fonctionnels générés.

Ensuite intervient le « *Test Designer Agent* », dont la mission est de produire les cas de test nécessaires à la validation du code généré par le programmeur. Cet agent identifie en priorité les scénarios critiques, c'est-à-dire ceux qui exigent une couverture de test rigoureuse afin de garantir la stabilité et la fiabilité de l'application cible.

Enfin, le « *Test Executor Agent* » entre en jeu pour tester le code dans un environnement réel. Il interagit directement avec le terminal, exécute les tests générés et vérifie que le comportement du code est conforme aux attentes. Si des anomalies sont détectées lors de cette phase, l'agent est également capable d'appliquer des corrections finales, assurant ainsi la livraison d'un code fonctionnel et robuste.

Toutefois cette solution présente plusieurs limitations. Tout d'abord, ces agents fonctionnent de manière indépendante et sans synchronisation, ce qui peut entraîner des redondances. De plus, l'autoréflexion est limitée aux tests, sans retour d'information structuré pour le refactoring global.

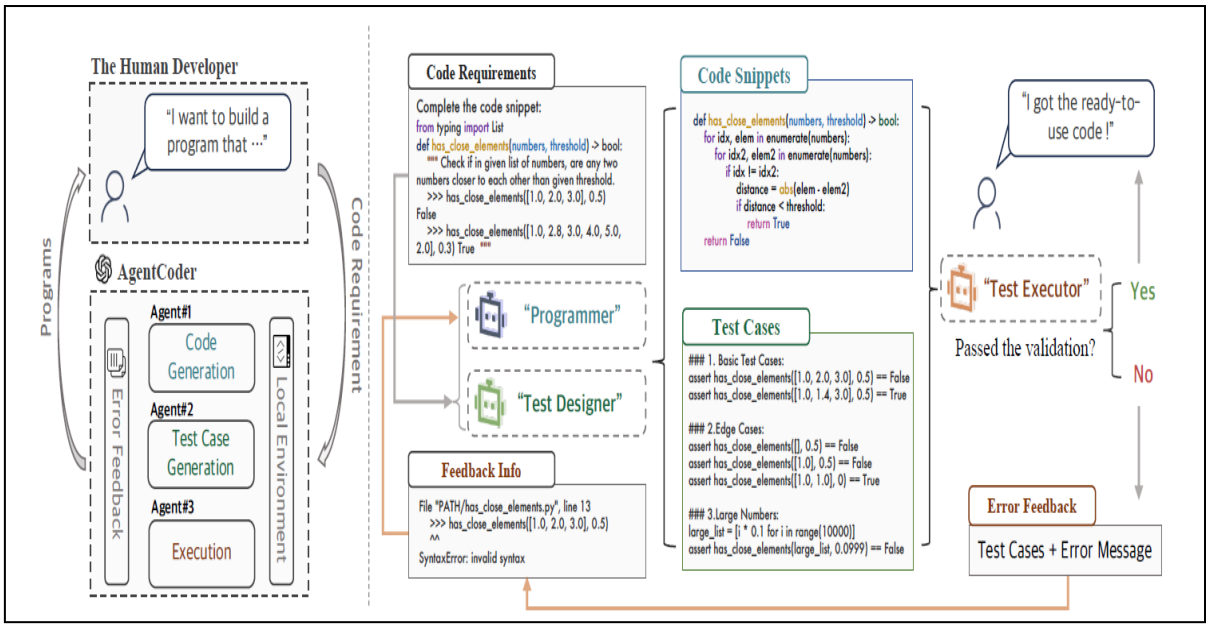


Figure 11 : Système multi-agent de AgentCoder.

2.7 Conclusion

L'étude théoriques des LLMs et des systèmes multi-agents nous a permis de structurer efficacement le pipeline de migration. Dans le chapitre suivant, nous aborderons la méthodologie utilisée pour concevoir et implémenter ce pipeline, en mettant en avant

l'orchestration des agents, les stratégies de refactoring et les outils d'évaluation des performances.

3 Architecture du système multi-agent proposé

Sommaire

3.1 Introduction

Dans le cadre de ce projet de migration d'applications legacy, nous avons conçu un système multi-agent basé sur l'IA, orchestré par le framework CrewAI. Ce système repose sur une architecture modulaire, distribuée et orientée tâche, permettant de traiter chaque phase du processus de migration de façon autonome, parallélisée et sécurisée. Ce chapitre détaille l'architecture fonctionnelle et logicielle mise en œuvre.

3.2 Architecture globale du système

3.2.1 Architecture fonctionnelle du système multi-agent

Le diagramme ci-dessous illustre le flux de travail du système multi-agent conçu pour la migration des applications legacy. Chaque module est représenté par un agent spécialisé, assurant une gestion autonome et synchronisée des différentes étapes du pipeline.

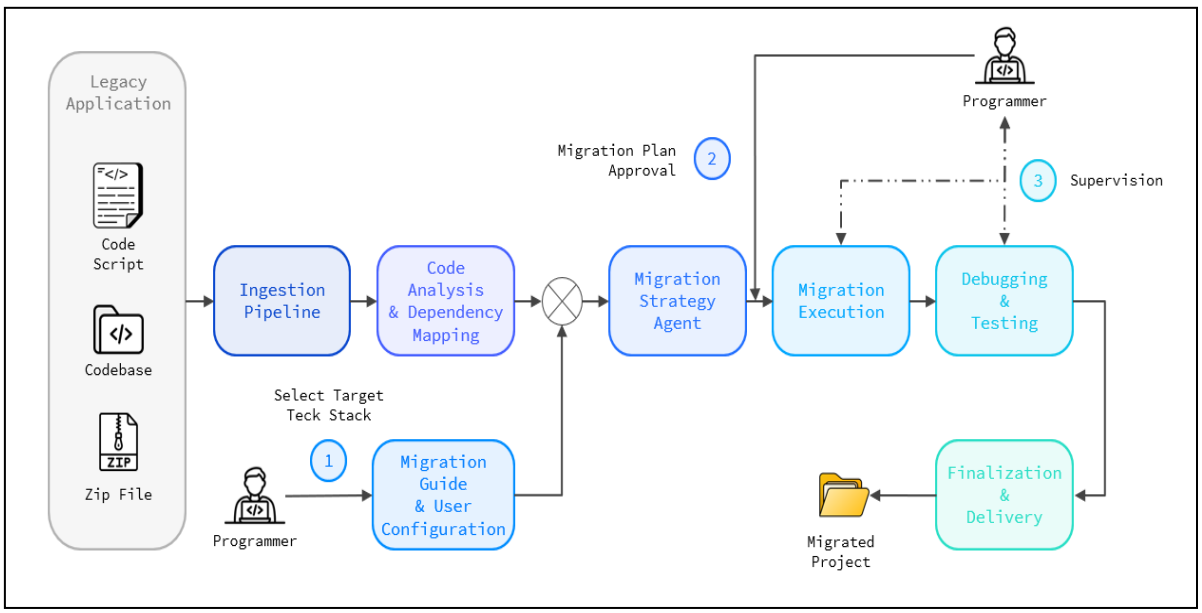


Figure 12 : Architecture du système IA multi-agent proposé.

Notre pipeline multi-agent est organisé selon une structure modulaire, articulée autour de plusieurs étapes clés qui se succèdent de manière fluide et synchronisée pour assurer une migration automatisée efficace et fiable des applications legacy.

Le processus débute avec le **module d'ingestion**, qui prend en charge l'extraction des fichiers sources, la segmentation du code en blocs fonctionnels exploitables, ainsi que la génération des graphes de dépendances. Vient ensuite le **module d'analyse**, chargé de l'analyse syntaxique à l'aide d'arbres de syntaxe abstraits (AST), de la génération des schémas de données et de l'identification des dépendances critiques entre les différents composants du système.

Une fois cette base analysée, le **module de refactoring** entre en action. Il applique le modèle **DeepSeek Coder 33B** pour réécrire les modules identifiés, en tenant compte des meilleures pratiques de développement moderne. Le résultat est ensuite transmis au **module de test**, qui se charge de générer des tests unitaires et de valider le bon fonctionnement des modules refactorés.

Enfin, le **module de validation** effectue un contrôle qualité final. Il vérifie les performances des modules migrés, compare les comportements entre la version legacy et la version modernisée, et génère des logs d'erreurs détaillés pour documenter l'ensemble du processus.

Cette architecture modulaire permet non seulement de gérer efficacement la complexité des tâches, mais garantit également une synchronisation optimale entre les différents agents, assurant ainsi la robustesse et la cohérence de l'ensemble du pipeline de migration.

3.3 Description fonctionnelle des agents IA

La méthodologie adoptée repose sur un ensemble d'agents multi-tâches orchestrés par CrewAI pour garantir une exécution fluide des processus de migration des applications legacy. Chaque agent est spécialisé dans une tâche précise, permettant une gestion modulaire et scalable du pipeline de migration.

Tableau 12 : Description fonctionnelle des agents IA.

Agent	Rôle	Fonctionnalités
Code Ingestion Agent	Extraction des fichiers source et segmentation du code	Liste les fichiers, filtre le contenu, nettoie le code
Structure & Dependency Agent	Analyse des dépendances et génération des graphes	Détecte les classes, les fonctions, les imports et les relations

Database Extraction Agent	Analyse des requêtes SQL et des schémas de base de données	Génère des cartes de schéma, identifie les relations clés
Migration Guide & User Config Agent	Interaction utilisateur et génération du contexte	Crée un fichier JSON structuré, configure les préférences utilisateur
Migration Strategy Agent	Génération du plan de migration	Produit un plan structuré, assigne des tâches aux agents
Language Migration Agent	Conversion du code source vers le langage cible	Réécrit les modules, ajuste les interfaces
Database Migration Agent	Conversion des couches d'accès aux données	Génère des scripts ORM, optimise les requêtes SQL
Architecture Refactoring Agent	Restructuration des modules en microservices	Séparation des modules, redéfinition des interfaces API
Debugging & Testing Agent	Exécution des tests unitaires et validation du code	Exécute des tests, génère des rapports d'erreurs
Finalization & Delivery Agent	Préparation du projet migré pour le déploiement	Crée des Dockerfiles, génère des README et des artefacts

Ces agents interagissent de manière synchronisée tout au long du pipeline, garantissant ainsi une migration fluide, une détection proactive des erreurs et une optimisation des modules refactorés.

3.4 Outils et technologies utilisés

Les outils et technologies sélectionnés pour notre projet sont centrés autour des frameworks d'IA, des modèles larges de langage spécialisés et des plateformes de gestion des workflows multi-agents. Chaque composant a été choisi pour sa capacité à s'intégrer dans le pipeline multi-agent tout en répondant aux exigences de modularité, de précision et de scalabilité.

3.4.1 Orchestrateur multi-agent CrewAI

Le diagramme ci-dessous illustre l'architecture de CrewAI [9], où chaque composant est structuré pour assurer une orchestration fluide des workflows multi-agents. L'architecture est conçue autour des composants suivants :

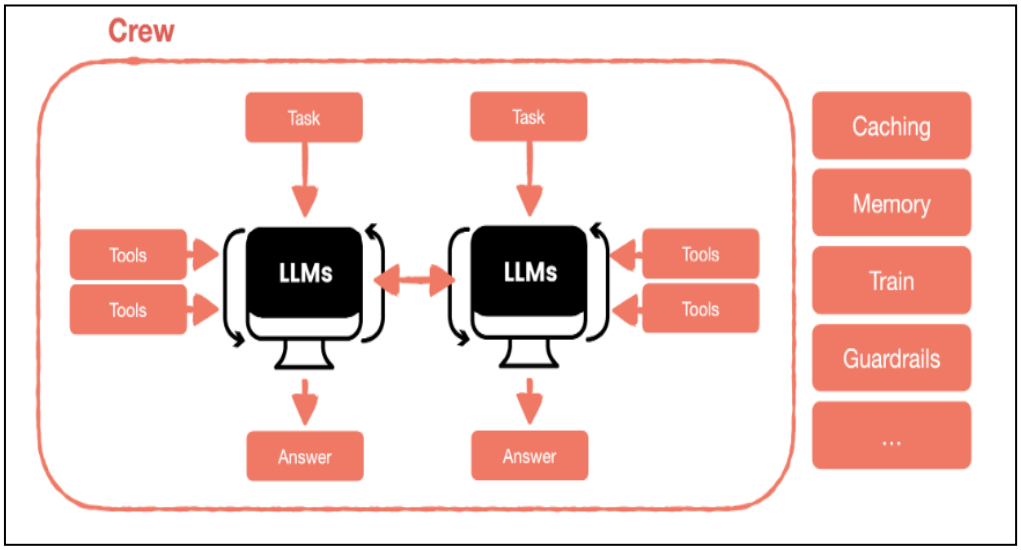


Figure 13 : Architecture basée sur CrewAI.

- *Crew* : Orchestrateur central des agents et des tâches.
- *LLMs* : Moteurs de traitement des tâches complexes (analyse, refactoring).
- *Tasks* : Modules de tâches spécifiques gérés par les agents.
- *Tools* : Outils associés aux agents (par exemple : analyseur de code, générateur de tests).
- *Caching* : Gestion des données temporaires pour minimiser le temps d'exécution.
- *Memory* : Suivi des états contextuels et historiques des tâches.
- *Guardrails* : Vérification des outputs pour garantir la cohérence des résultats.

Le rôle de l'orchestrateur CrewAI consiste à la coordination des agents, la gestion des états et l'orchestration des tâches complexes.

Pourquoi choisir une architecture agentique ?

Ce choix repose sur plusieurs avantages déterminants qui renforcent la robustesse et l'efficacité du pipeline de migration. Tout d'abord, l'approche adoptée permet une exécution asynchrone et modulaire des agents, ce qui facilite le découplage des tâches et améliore la scalabilité du système. En parallèle, le pipeline intègre une gestion de mémoire contextuelle, essentielle pour assurer la continuité logique entre les étapes et permettre un suivi précis de

l'évolution des tâches tout au long du processus. Ce mécanisme contribue également à réduire les risques d'incohérence, en synchronisant intelligemment les actions des agents à travers chaque module du pipeline.

Par ailleurs, le choix du framework CrewAI se justifie pleinement par sa capacité à gérer des flux complexes de manière fluide et granulaire. Comparé à des alternatives comme AgentCoder, CrewAI se distingue par une modularité plus avancée, offrant une meilleure orchestration des agents, ainsi qu'une gestion fine des interactions entre les composants. Cette flexibilité en fait une solution particulièrement adaptée aux environnements nécessitant une automatisation intelligente et distribuée.

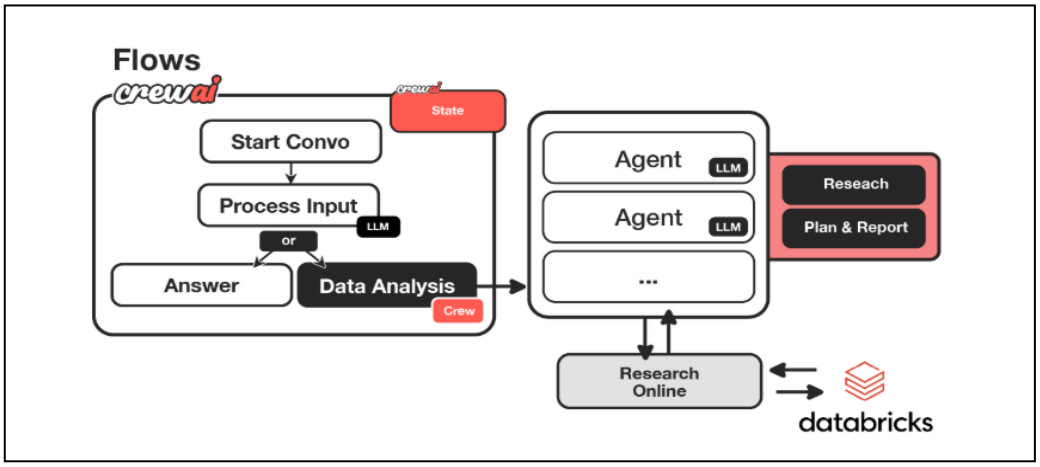


Figure 14 : CrewAI Flows : gestion des workflows multi-agents.

Le diagramme CrewAI Flows met en évidence la structure d'orchestration des workflows complexes à l'aide d'un ensemble d'agents autonomes. Chaque composant de CrewAI Flows est structuré pour traiter des tâches spécifiques de manière asynchrone et modulaire, permettant ainsi une gestion efficace des processus.

Pourquoi utiliser CrewAI Flows ?

Le choix de CrewAI Flows repose sur plusieurs atouts majeurs qui répondent aux exigences d'un système multi-agent évolutif et intelligent. L'un de ses principaux avantages est sa scalabilité, permettant d'ajouter ou de retirer dynamiquement des agents en fonction des besoins, sans perturber l'ensemble du flux de travail. Cette capacité d'adaptation est renforcée par une flexibilité opérationnelle, les agents pouvant être configurés pour exécuter des tâches spécialisées telles que l'analyse de code, la génération de tests ou la documentation automatisée des résultats. Un autre élément clé réside dans la gestion centralisée des états, qui assure un

suivi en temps réel de l'évolution des tâches tout en garantissant la continuité des processus, même en cas de défaillance ponctuelle d'un composant.

En parallèle, CrewAI Flows offre une orchestration avancée, autorisant une communication fluide entre agents pour le partage d'informations critiques, la synchronisation des opérations et la minimisation des redondances. Cette infrastructure est naturellement intégrée avec des LLMs, facilitant le traitement des données, la génération de code et le refactoring automatisé à partir de LLMs comme DeepSeek Coder.

Lors de la phase de conception, une comparaison avec *l'Agent Development Kit* (ADK) de Google a été réalisée. Bien que ADK se soit distingué par une orchestration efficace et un flux séquentiel bien maîtrisé, il a montré des limites importantes dans la gestion des structures parallèles et du partage d'état entre agents. En revanche, CrewAI a démontré une plus grande aisance à orchestrer des processus parallèles, à gérer des états complexes et à s'intégrer nativement avec des LLMs de pointe.

En définitive, le choix de CrewAI Flows est justifié par sa robustesse, sa modularité, sa capacité à gérer des tâches complexes de manière distribuée et son potentiel d'extension élevé pour les projets futurs nécessitant une coordination intelligente entre agents autonomes.

3.4.2 Refactoring et génération de code avec DeepSeek Coder

DeepSeek Coder est un modèle de langage open-source développé par DeepSeek AI, conçu spécifiquement pour la génération, le refactoring et la compréhension de code source multilingue. Il constitue une alternative crédible à des modèles comme Codellama ou Starcoder, avec des performances compétitives sur des tâches complexes liées à l'IA générative pour le code.

Architecture technique : DeepSeek Coder repose sur une architecture Transformer de type *decoder-only*, spécifiquement conçue pour les tâches de complétion et de génération de code. Ce modèle unidirectionnel, à l'image de GPT, permet une génération séquentielle fluide et cohérente, particulièrement adaptée aux langages de programmation. Dans sa version la plus récente, V2 Lite, DeepSeek Coder adopte une architecture *Mixture-of-Experts* (MoE), où seuls certains experts sont activés dynamiquement lors de chaque inférence, optimisant ainsi la performance sans alourdir les ressources nécessaires.

Le modèle est décliné en plusieurs tailles, incluant des versions 1B, 6.7B, 33B Instruct, ainsi que la version V2 Lite qui active 2.4 milliards de paramètres parmi un ensemble de 16 experts, offrant ainsi un compromis optimal entre capacité de génération et efficacité. Un autre

point fort du modèle réside dans sa fenêtre de contexte pouvant atteindre 128 000 tokens dans la version V2, ce qui en fait un outil particulièrement performant pour le traitement de gros projets ou de bases de code legacy complètes, où la compréhension à long terme du contexte est essentielle.

Enfin, DeepSeek Coder intègre un *tokenizer* spécifiquement optimisé pour le code, avec un vocabulaire multilingue adapté aux langages de programmation les plus couramment utilisés. Cela lui permet de gérer avec précision des structures syntaxiques complexes, tout en assurant une compatibilité fluide entre plusieurs environnements de développement.

Données d'entraînement : Le modèle DeepSeek Coder a été entraîné sur un corpus massif de 2000 milliards de tokens, dont environ 87% sont constitués de code source couvrant plus de 80 langages de programmation, parmi lesquels on retrouve Python, Java, JavaScript, PHP, C++, et bien d'autres. Les 13% restants du corpus sont composés de langage naturel, principalement en anglais et en chinois, afin de permettre au modèle de comprendre et suivre des instructions complexes exprimées en langage humain.

Les données proviennent exclusivement de sources ouvertes rigoureusement sélectionnées, incluant des dépôts GitHub, des discussions techniques sur Stack Overflow, des documentations officielles, ainsi que des notebooks Jupyter. Cette diversité permet à DeepSeek Coder de maîtriser les conventions modernes de développement logiciel, de générer automatiquement des commentaires de code, des cas de test et de la documentation, tout en intégrant des patterns de refactoring à jour. Grâce à cette base d'apprentissage étendue, le modèle est capable de produire du code précis, lisible et conforme aux meilleures pratiques actuelles du développement.

Capacités de refactoring et de migration : Le modèle DeepSeek Coder se distingue par ses performances remarquables dans les domaines du refactoring et de la migration de code legacy. Il excelle notamment dans le refactoring syntaxique et structurel, en étant capable de supprimer les duplications de code, de réécrire des fonctions de manière plus modulaire, et d'appliquer des améliorations stylistiques et architecturales conformes aux standards modernes. Il offre également des capacités puissantes de traduction inter-langage, permettant par exemple de convertir du code écrit en PHP vers JavaScript, ou encore de migrer des applications en Visual Basic 6 vers Python, facilitant ainsi la modernisation d'environnements obsolètes.

Une autre fonctionnalité phare réside dans la complétion intelligente en mode *Fill-in-the-Middle*, avec un score FIM pass@1 de 85,9% pour la version V2 Lite, ce qui illustre sa capacité à compléter efficacement des fragments de code en tenant compte du contexte global. En complément, le modèle est en mesure de générer automatiquement des tests unitaires à partir du code existant, ce qui contribue à sécuriser les phases de migration ou de refactoring.

Enfin, DeepSeek Coder propose des fonctions avancées d'explication de code, utiles pour la documentation, la formation ou l'audit technique, en rendant explicites les logiques sous-jacentes d'un programme.

Tableau 13 : Benchmarks de référence du DeepSeek Coder V2 Lite.

Benchmark	**Résultats**	**Tâche**
HumanEval JS	80.8%	Génération fonctionnelle de code JS
RepoBench	20.0%	Complétion dans des repositories
CanItEdit	72.0%	Capacité de modification de code existant
HumanEval FIM JS	85.9%	Complétion contextuelle Fill-in-the-Middle

En conséquence, DeepSeek Coder est retenu comme le model principal pour notre projet, permettant une migration efficace des applications legacy tout en assurant une gestion multi-langage et une génération de code robuste.

3.7 Étude conceptuelle

Dans le cadre du développement du système de migration automatisée basé sur une architecture multi-agent, une étude conceptuelle a été réalisée. Elle vise à modéliser les interactions fonctionnelles entre les acteurs (utilisateurs et agents) et le système.

3.7.1 Diagramme de cas d'utilisation général

Le diagramme de cas d'utilisation générale est un diagramme utilisé pour donner une vision globale du comportement fonctionnel de notre système. Il est utile pour des présentations auprès de la direction ou des acteurs du projet.

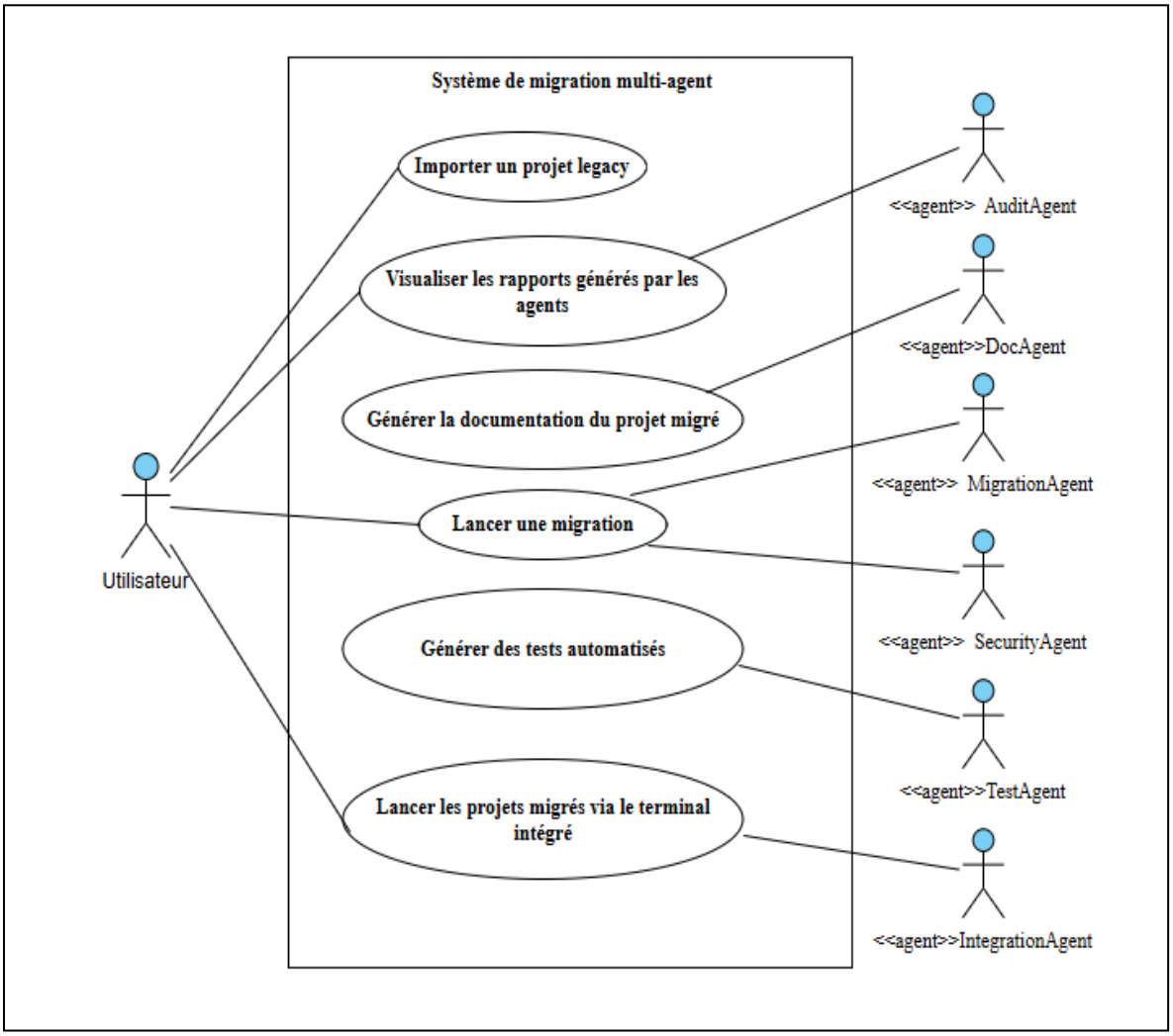


Figure 15 : Diagramme de cas d'utilisation générale.

3.7.2 Description textuelle du cas d'utilisation « Importer un projet legacy »

Tableau 14 :Description textuelle du cas d'utilisation « Importer un projet legacy ».

Cas d'utilisation	**Importer un projet legacy**
Acteurs	Utilisateur
Pré-condition	– Le système est lancé. – Le projet n'a pas encore été importé.

Post-condition	Le projet legacy est importé et disponible dans l'espace de travail.
Description du scénario principal	1. L'utilisateur clique sur "Importer un projet". 2. Le système ouvre une boîte de dialogue pour sélectionner un dossier ou un fichier. 3. L'utilisateur sélectionne un dossier ou fichier. 4. Le système valide le format. 5. Le projet est chargé et analysé.
Exception	– Si un fichier est corrompu ou au mauvais format, un message d'erreur est affiché. – L'importation est annulée.

Avant de lancer toute opération de migration, l'utilisateur doit charger un projet legacy dans le système. Cette fonctionnalité est essentielle pour préparer les fichiers à analyser et à transformer. Le processus permet de sélectionner un répertoire ou un fichier source, puis de le valider avant son intégration dans l'espace de travail.3.7.3 Diagramme de séquence du cas d'utilisation « Importer un projet legacy »

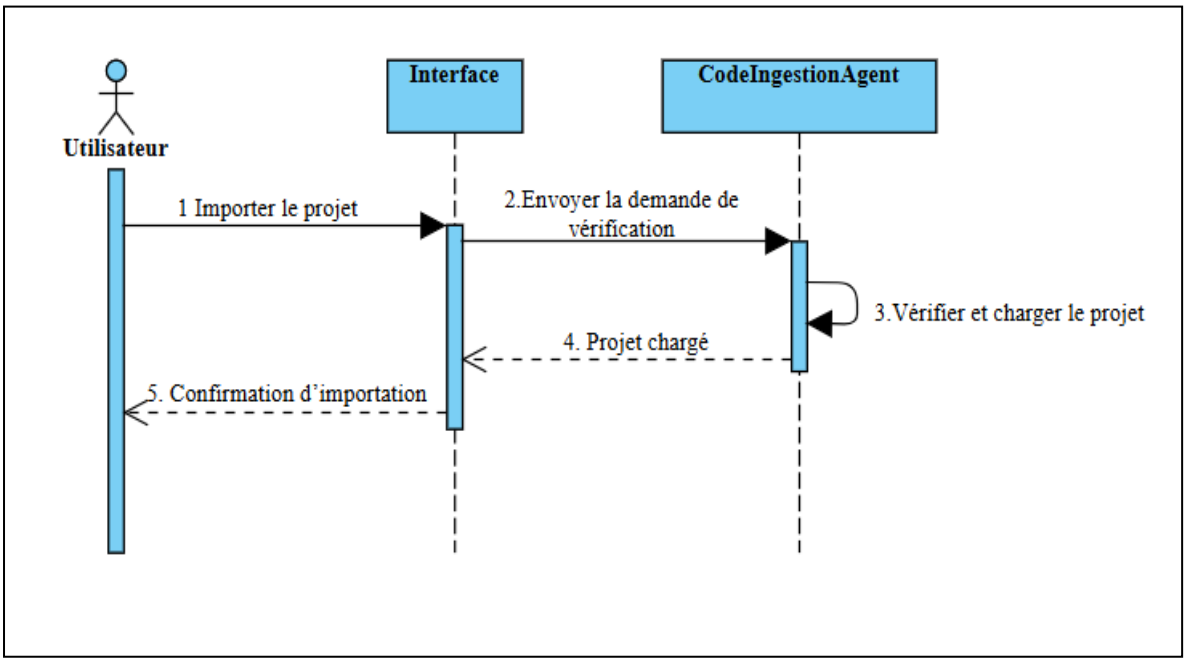


Figure 16 : Diagramme de séquence du cas d'utilisation « Importer un projet legacy ».

3.7.4 Description textuelle du cas d'utilisation « Lancer une migration »

Ce cas d'utilisation représente l'une des fonctions clés du système. Il permet à l'utilisateur de déclencher la migration automatisée d'un projet legacy. Cette opération active plusieurs agents en arrière-plan, notamment l'*AuditAgent* pour l'analyse préliminaire et le

MigrationAgent pour la conversion du code. Le tout s'exécute sans intervention manuelle, garantissant fluidité et rapidité.

Tableau 15: Description textuelle du cas d'utilisation « Lancer une migration ».

Cas d'utilisation	**Lancer une migration**
Acteurs	Utilisateur, <<agent>> *MigrationAgent*, <<agent>> *AuditAgent*
Pré-condition	– Un projet legacy est déjà importé. – L'utilisateur a accès à l'espace de migration.
Post-condition	Le code legacy est transformé automatiquement en un nouveau format.
Description du scénario principal	1. L'utilisateur sélectionne le projet importé. 2. Il clique sur "Lancer migration". 3. *AuditAgent* analyse le code. 4. *MigrationAgent* applique les règles de transformation. 5. Le système génère les fichiers migrés.
Exception	– Si le projet est invalide, un message d'erreur s'affiche. – Si un agent échoue, l'action est annulée avec retour d'état.

3.7.5 Diagramme de séquence du cas d'utilisation « Lancer une migration »

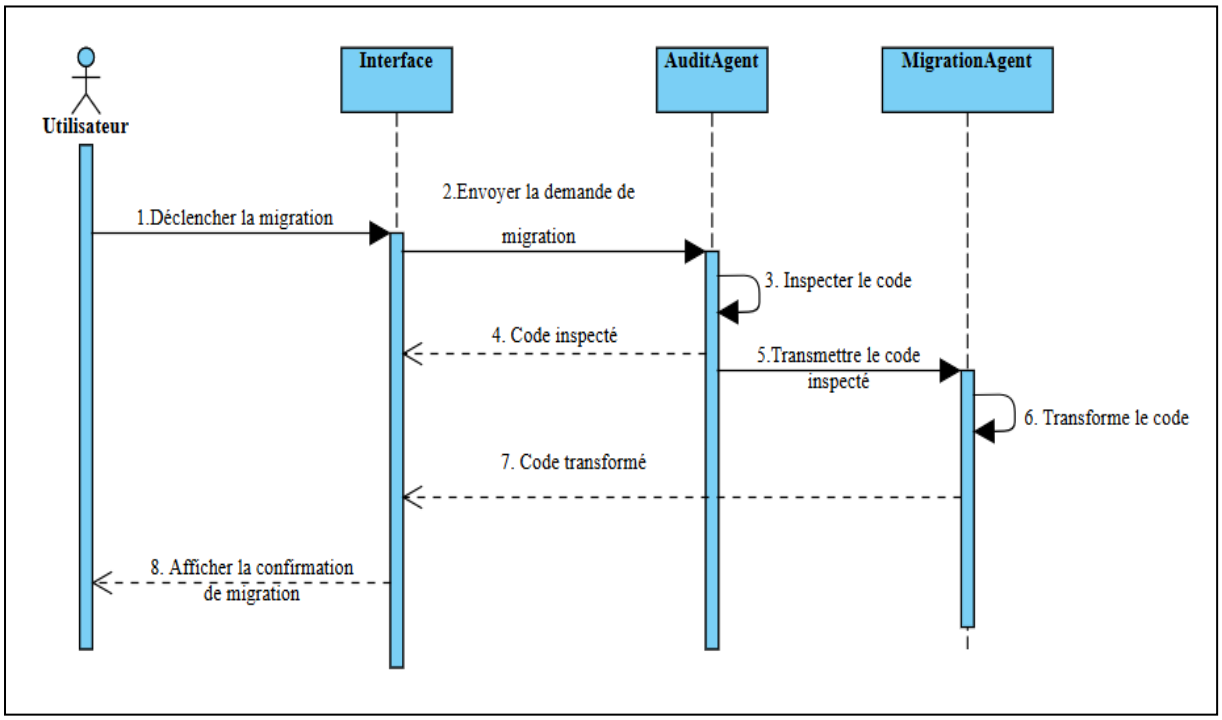


Figure 17 : Diagramme de séquence du cas d'utilisation « Lancer une migration ».

3.7.6 Description textuelle du cas d'utilisation « Visualiser les rapports générés par les agents »

Tableau 16 : Description textuelle du cas d'utilisation « Visualiser les rapports générés par les agents ».

Cas d'utilisation	**Lancer une migration**
Acteurs	Utilisateur, <<agent>> *AuditAgent*
Pré-condition	– Une migration a déjà été effectuée. – Les rapports ont été générés.
Post-condition	L'utilisateur peut consulter les rapports d'analyse et d'exécution.
Description du scénario principal	1. L'utilisateur accède à l'onglet "Rapports". 2. Le système charge les données générées par les agents. 3. Les rapports sont affichés avec options de téléchargement ou d'impression.
Exception	– Si aucun rapport n'est disponible, un message l'indique.

Après l'exécution de certaines tâches critiques, les agents génèrent des rapports d'analyse, d'audit ou de performance. Ce cas d'utilisation permet à l'utilisateur d'accéder à ces rapports via l'interface Streamlit.

3.7.7 Description textuelle du cas d'utilisation « Générer la documentation du projet migré »

Tableau 17 : Description textuelle du cas d'utilisation « Générer la documentation du projet migré »

Cas d'utilisation	**Générer la documentation du projet migré**
Acteurs	<<agent>> *DocAgent*
Pré-condition	– Le projet a été migré avec succès.
Post-condition	Une documentation technique est générée et consultable.
Description du scénario principal	1. *DocAgent* génère la documentation. 2. Une documentation est générée en Markdown ou PDF et affichée.
Exception	– Si la génération échoue, un message d'erreur est retourné. – Si le projet ne contient pas assez de contenu, la documentation est partielle.

L'automatisation de la documentation est un atout majeur dans ce système. L'agent *DocAgent* peut déclencher la création de fichiers décrivant le fonctionnement du code migré. Cela permet une meilleure compréhension, une maintenance facilitée, et une documentation toujours à jour.

3.7.8 Description textuelle du cas d'utilisation « Générer des tests automatisés »

Tableau 18 : Description textuelle du cas d'utilisation « Générer des tests automatisés ».

Cas d'utilisation	**Générer des tests automatisés**
Acteurs	<<agent>> *TestAgent*
Pré-condition	– Le projet migré est disponible et conforme.
Post-condition	Des scripts de test unitaires sont générés.
Description du scénario principal	1. *TestAgent* analyse les fonctions critiques. 2. Le système génère et affiche les fichiers de test.
Exception	– Si le code contient des erreurs, un message bloque la génération.

Les tests unitaires permettent de vérifier que le code migré reste fonctionnel. Dans ce cas, le *TestAgent* analyse le code et produit automatiquement des scripts de test. Cela réduit les erreurs humaines et accélère le processus de validation du projet transformé.

3.7.9 Description textuelle du cas d'utilisation « Lancer les projets migrés via le terminal intégré »

Tableau 19 : Description textuelle du cas d'utilisation « Lancer les projets migrés via le terminal intégré ».

Cas d'utilisation	**Lancer les projets migrés via le terminal intégré**
Acteurs	Utilisateur
Pré-condition	– Le projet migré est prêt à l'exécution. – Le terminal est disponible dans l'interface.
Post-condition	Le projet s'exécute avec tests et contrôles déclenchés automatiquement.
Description du scénario principal	1. L'utilisateur ouvre le terminal intégré… 2. Il lance un script (run, test ou vérification).
Exception	– Si une erreur est détectée, le processus est stoppé.

Une fois le code migré, l'utilisateur peut exécuter le projet dans un environnement simulé via un terminal intégré.

3.5 Validation expérimentale via le hackathon d'Optimized AI

3.5.1 Contexte et objectif du hackathon

Lors du hackathon organisé par Optimized AI Conference [10] et Traversaal.ai, notre équipe BlackBeak a mis en œuvre un système multi-agent innovant basé sur le framework AgentPro. L'objectif principal était de migrer des applications PHP legacy vers une architecture Next.js 14 de manière modulaire et autonome.

3.5.2 Solution développée

La solution mise en œuvre repose sur le framework AgentPro, choisi pour sa capacité à orchestrer des agents intelligents dans un environnement de développement modulaire et automatisé. Elle intègre plusieurs technologies clés, dont Mistral Codestral pour le traitement

du code, Next.js 14 pour la modernisation de l'interface web, ainsi que des modules spécifiques tels que *Tools* et *Test Generator*, qui facilitent l'analyse et la validation du code.

Le cœur de cette solution est centré sur l'automatisation du refactoring du code PHP vers le framework Next.js 14, permettant ainsi une migration fluide vers une architecture web moderne basée sur React. En complément, le système prend en charge la génération automatique de tests unitaires, aussi bien pour les composants frontend que backend, garantissant la fiabilité du code migré. Chaque composant généré est accompagné d'une documentation technique automatisée, facilitant la maintenabilité et la compréhension du système.

Enfin, une attention particulière a été portée à la sécurité, avec l'intégration d'un module d'analyse des vulnérabilités ciblant les codes PHP et JavaScript, afin de détecter les failles potentielles avant déploiement.

3.5.3 Architecture et fonctionnalités des agents

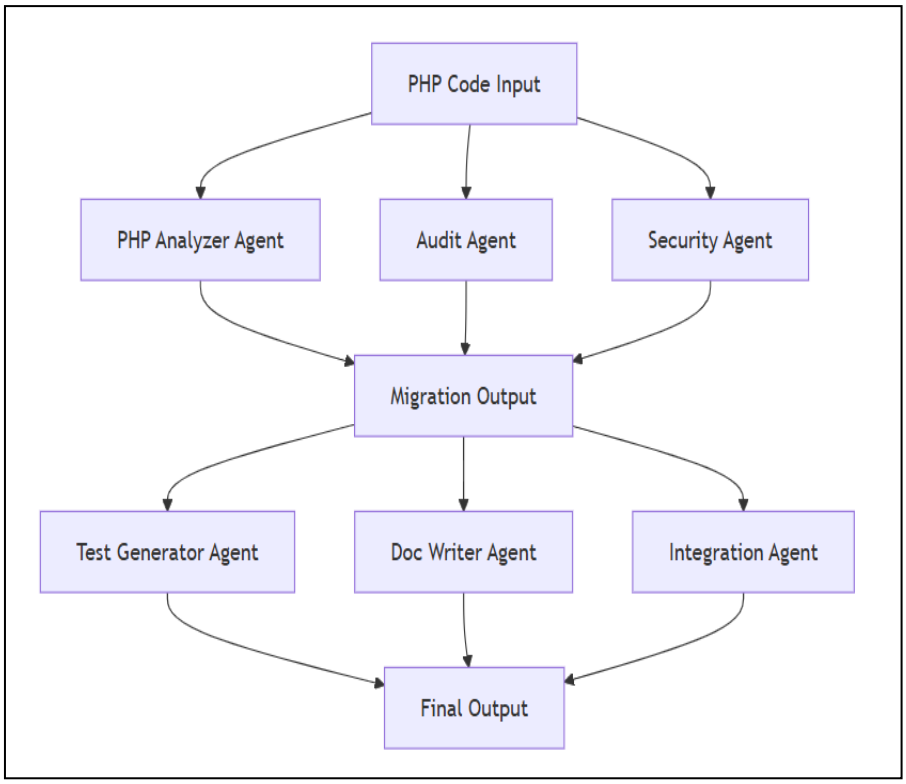


Figure 18 : Architecture AgentPro utilisée au hackathon.

Tableau 20 : Rôle et fonctionnalités des agents IA basés sur AgentPro.

Agent	Rôle	Fonctionnalités
PHP Analyzer	Analyse du code PHP	Conversion du code PHP vers Next.js 14, identification des modules critiques.
Audit Agent	Extraction des dépendances	Génère un graphe de dépendance, extrait la structure des modules.
Security Agent	Analyse des vulnérabilités	Scans automatiques des vulnérabilités dans le code PHP/JS.
Test Generator	Génération des tests unitaires	Créer des tests basés sur le code migré, validation des modules critiques.
Doc Writer	Génération de la documentation	Créer une documentation pour un nouveau projet migré.
Integration Agent	Assemblage final des modules	Génère la structure finale du projet, incluant les tests et la documentation.

3.5.4 Points forts et distinctions

Nous avons remporté la 2e place au Hackathon grâce à une approche modulaire fondée sur des agents autonomes, orchestrés de manière fluide via la plateforme AgentPro. Le processus de migration a été structuré en plusieurs étapes clés : ingestion, analyse, refactoring, tests et génération de documentation.

3.6 Conclusion

La méthodologie proposée garantit une migration efficace des applications legacy vers des environnements modernes, tout en préservant l'intégrité du système et en réduisant les risques de régression. Dans le chapitre suivant, présentera en détails les étapes d'implémentation et les résultats des expérimentations.

4 Implémentation et expérimentations

Sommaire

4.1 Introduction

Ce chapitre détaille la mise en œuvre technique du système multi-agent pour la migration des applications legacy. Chaque étape du pipeline est exécutée par un agent spécialisé, conçu pour fonctionner de manière autonome, orchestrée via CrewAI. Les sections de ce chapitre décrivent les mécanismes internes de chaque agent, leurs objectifs, les données traitées, les méthodes utilisées et les résultats produits.

4.2 Choix technologiques

Le projet repose sur une pile technologique robuste, intégrant des frameworks d'IA, des plateformes de conteneurisation et des outils de gestion des workflows multi-agents.

Tableau 21 : Technologies choisies.

Technologie	Rôle	Version
CrewAI	Orchestration multi-agent	V2.3
DeepSeek Coder	Refactoring automatisé	33B
Streamlit	Interface utilisateur (import de projet, visualisation des rapports)	Dernière version
YAML	Configuration modulaire des agents et tâches pour CrewAI	-
Bash / Terminal intégré	Exécution et test du code migré depuis l'interface Streamlit	-

4.3 Architecture implémentée

L'architecture repose sur une chaîne de traitement orchestrée par CrewAI. Chaque agent est responsable d'une tâche bien spécifique dans le pipeline de migration, comme suit :

- *PHP Analyzer Agent* : Analyse statique du code PHP et détection des composants clés.
- *Audit Agent* : Inspection structurelle du code, production d'un rapport d'audit détaillé.
- *Security Agent* : Détection de failles de sécurité dans le code PHP.

- *Migration Agent* : Migration automatique vers Next.js 14 (composants React, API routes, Prisma, Tailwind).
- *Test Generator Agent* : Génération automatique de tests unitaires adaptés au code migré.
- *Documentation Agent* : Génération automatique de documentation technique (README.md).
- *Extract & Integration Agent* : Extraction des fichiers générés et insertion physique dans un projet Next.js structurellement valide.

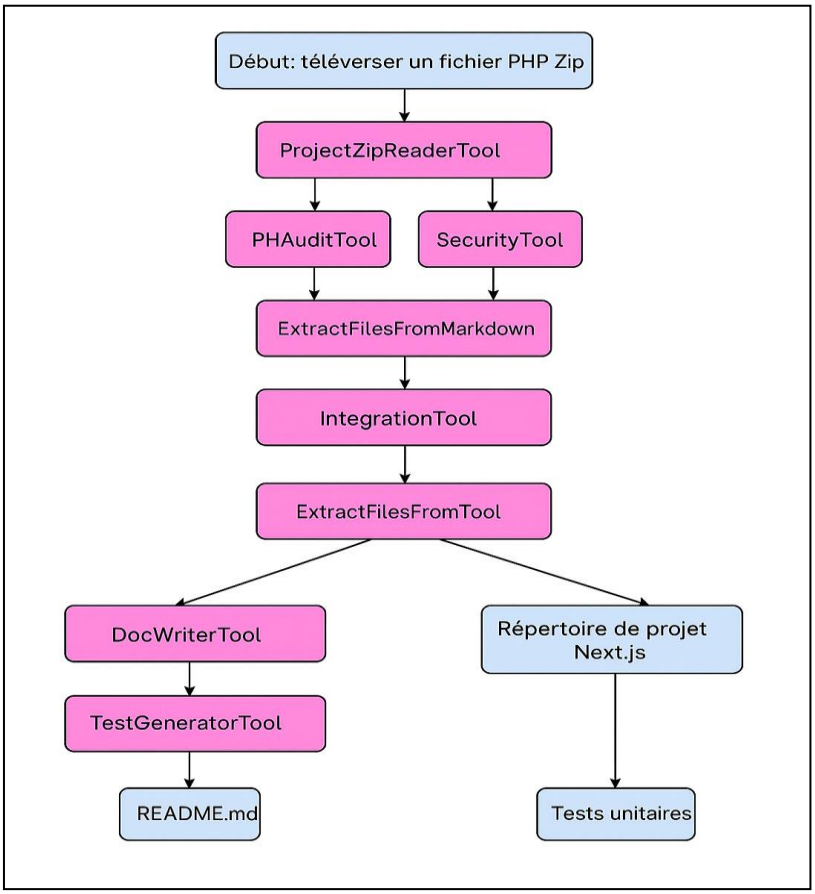


Figure 19 : Flux du processus de migration.

Chaque agent est encapsulé dans une Crew, instancié via des fichiers agents.yaml et tasks.yaml, ce qui rend le système totalement modulaire.

4.4 Architecture logicielle

La structure du répertoire est organisée pour assurer une séparation claire des modules, des modèles IA, des workflows et des scripts de tests.

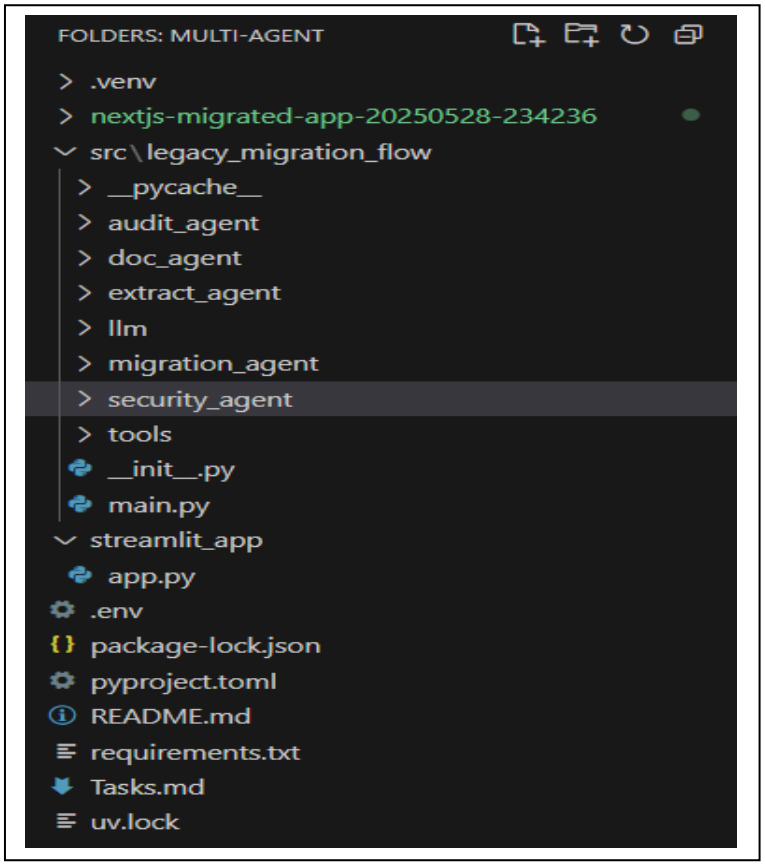


Figure 20 : Structure du répertoire.

4.5 Implémentation des agents

Le système multi-agent développé pour la migration des applications legacy vers Next.js repose sur une série d'agents spécialisés, chacun chargé d'une tâche bien définie au sein du pipeline. Cette approche modulaire repose sur des définitions CrewAI via des fichiers de configuration agents.yaml et tasks.yaml, permettant une instanciation indépendante et dynamique de chaque agent.

4.5.1 Agent 1 : Audit Agent

L'*Audit Agent* a pour objectif de réaliser une inspection structurelle complète du code source, en traitant des fichiers PHP ou JavaScript, ainsi que des archives compressées (au format .zip). Il produit en sortie un rapport d'audit détaillé, contenant les métriques d'analyse statique, les dépendances et la structure du projet initial.

4.5.2 Agent 2 : Security Agent

Le *Security Agent* utilise ce rapport d'audit ainsi que le code source brut pour effectuer une analyse approfondie des failles de sécurité, notamment sur les composants PHP. Il en ressort un rapport de vulnérabilités, classant les menaces détectées par niveau de criticité.

4.5.3 Agent 3 : Migration Agent

Le *Migration Agent* prend le relais pour effectuer la transformation automatique du code legacy vers le framework Next.js. Il utilise comme entrées les segments de code identifiés, leur contexte d'exécution et les fichiers de configuration, et génère un code Next.js modulaire et réutilisable.

4.5.4 Agent 4 : Test Generator Agent

Le *Test Generator Agent* s'appuie sur le code migré pour produire automatiquement des fichiers de tests unitaires ciblant aussi bien le frontend que le backend. Ce processus garantit la vérifiabilité du code refondu tout en réduisant le temps de validation.

4.5.5 Agent 5 : Documentation Agent

Le *Documentation Agent* intervient ensuite pour créer une documentation technique complète du projet migré, principalement sous forme de fichier README.md, en s'appuyant sur l'analyse des composants et leur logique métier.

4.5.6 Agent 6 : Extract & Integration Agent

L'*Extract & Integration Agent* regroupe l'ensemble des fichiers générés (code, tests, documentation) et les intègre dans une structure complète et fonctionnelle de projet Next.js. Ce dernier assure une cohérence finale du livrable en respectant les standards de structuration du framework.

Grâce à cette architecture, chaque agent est autonome, réutilisable et orchestrable indépendamment ou en parallèle, offrant une flexibilité totale et un fort potentiel de scalabilité pour l'extension future du système.

4.6 Fonctionnement de l'interface utilisateur

L'interface utilisateur a été développée avec Streamlit pour garantir une expérience simple, fluide et interactive. Elle propose deux modes d'interaction à savoir une migration automatique et une console interactive.

4.6.1 Migration automatique

L'utilisateur sélectionne un projet PHP à migrer. L'interface affiche ensuite en temps réel les agents en cours d'exécution, les étapes complétées et l'état final de chaque module.

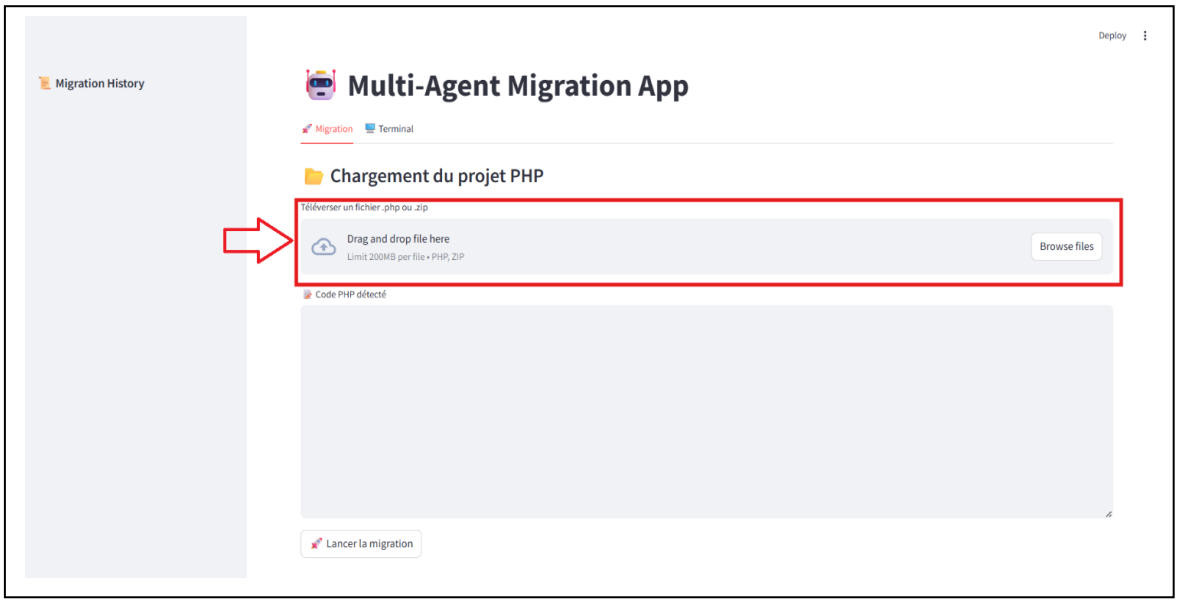


Figure 21 : Interface de sélection du projet PHP à migrer.

Ensuite, l'utilisateur peut d'importer un fichier ZIP contenant un projet PHP legacy à migrer. Ceci marque le point de départ du pipeline de migration.

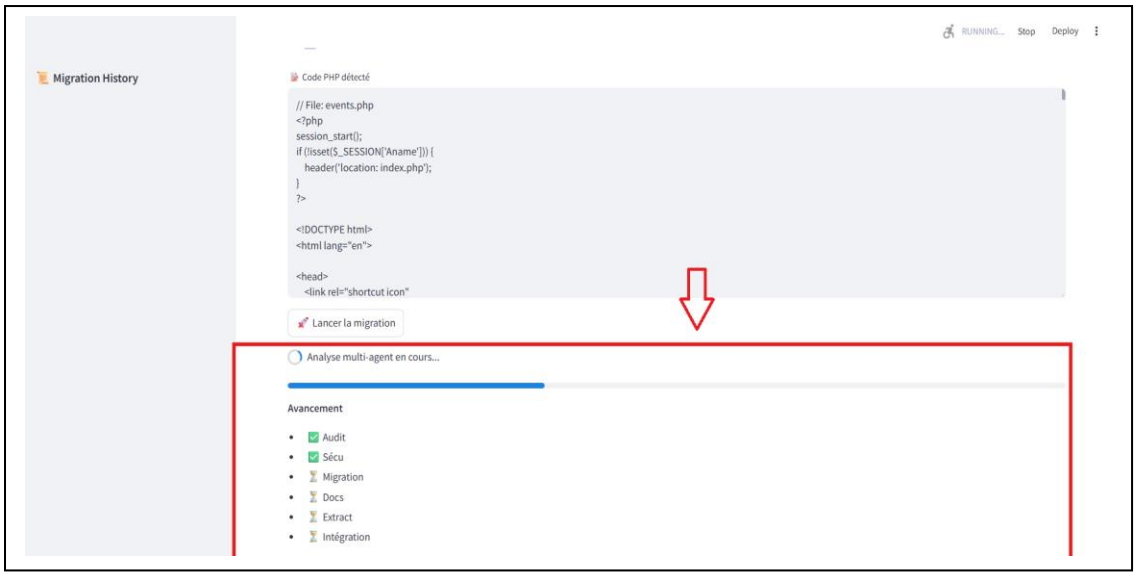


Figure 22 : Suivi en temps réel des étapes de migration.

Une fois le projet uploadé, les différentes étapes prises en charge par les agents (audit, sécurité, migration, tests, etc.) s'affichent dynamiquement pour donner une visibilité complète sur l'avancement.

4.6.2 Console interactive

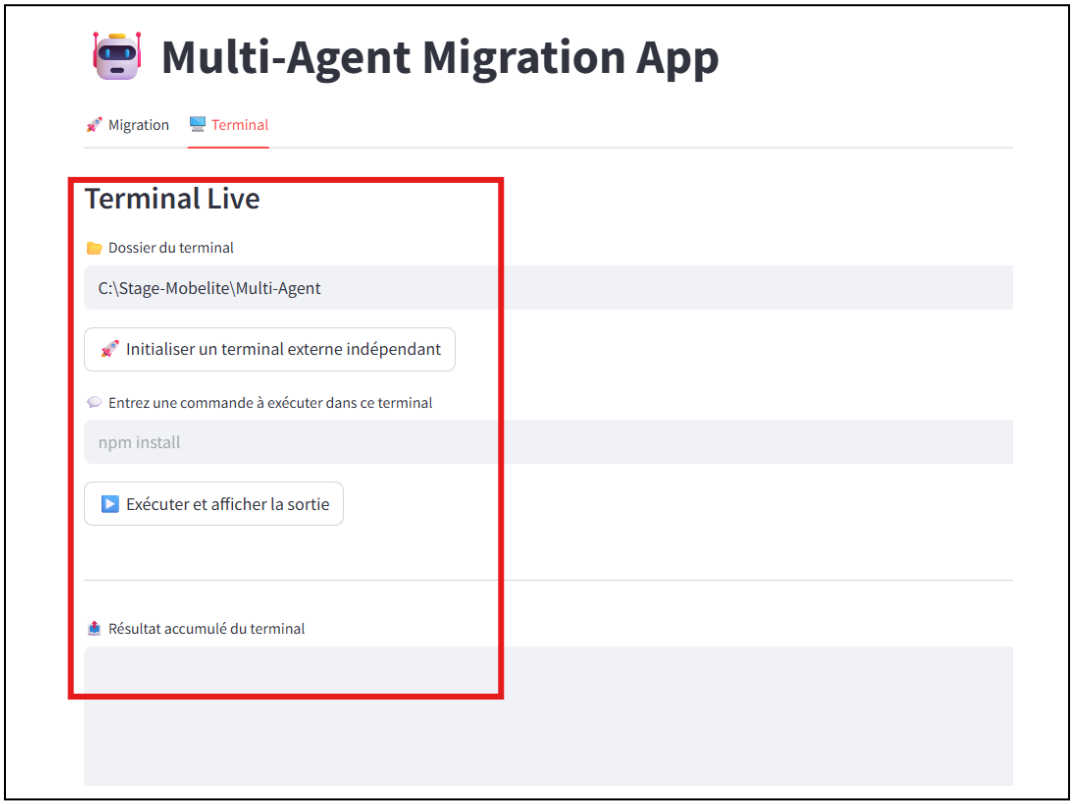


Figure 23 : Console interactive

Permet d'envoyer des commandes et d'accéder à un terminal bash intégré (simulateur) pour interagir avec le projet migré (exécution locale, débogage, npm run...). L'utilisateur peut ainsi exécuter des commandes telles que npm run dev ou consulter les logs. Cela permet de tester et vérifier manuellement le bon fonctionnement du projet après migration.

4.7 Illustration complète du processus de migration

4.7.1 Application PHP à migrer

L'application auditée est un portail d'administration d'événements en PHP [11] : gestion des événements, utilisateurs, participants et statistiques. Les principales étapes de la migration vers Next.js 14 avec Prisma sont illustrées ci-dessous.

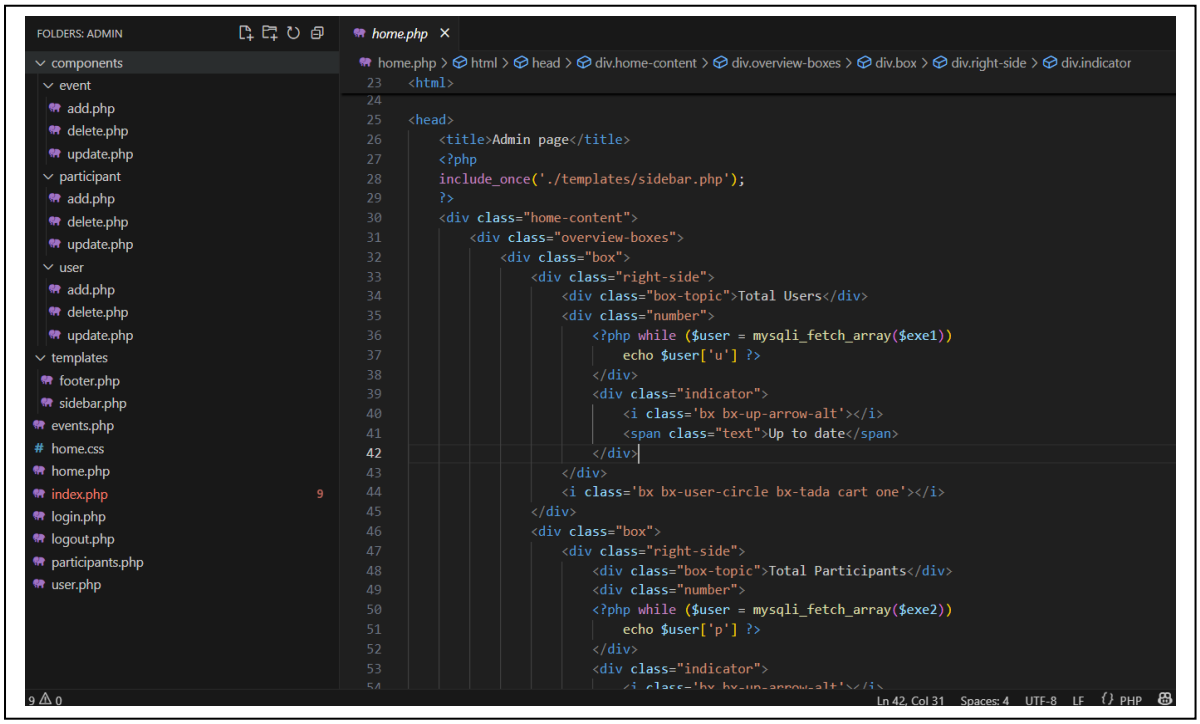


Figure 24 : Extrait du code source original en PHP.

La figure 24 affichage le code legacy avant transformation.

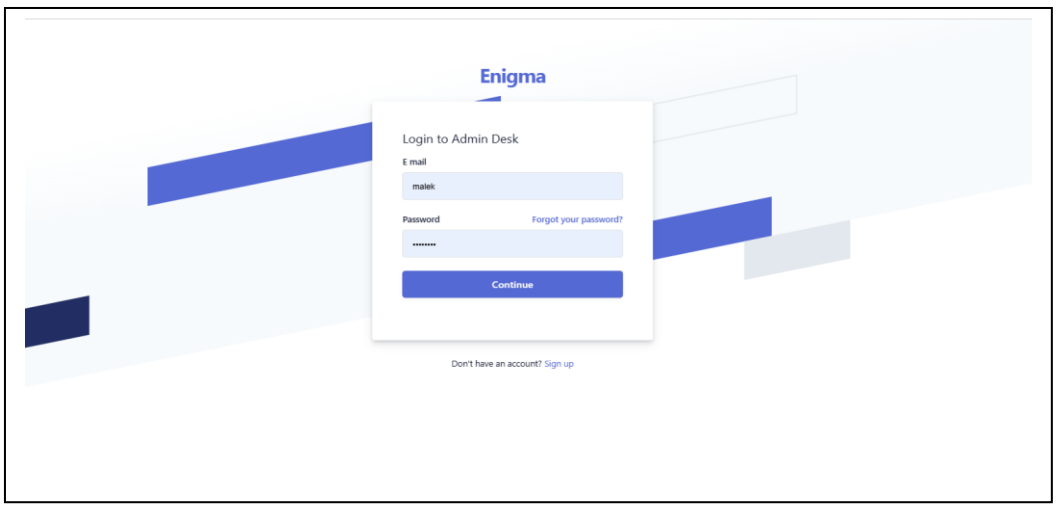


Figure 25 : Interface Login originale de l'application.

La figure 25 montre une page de connexion simple avec des champs pour le mail et le mot de passe. Cette interface est destinée à l'administration des événements.

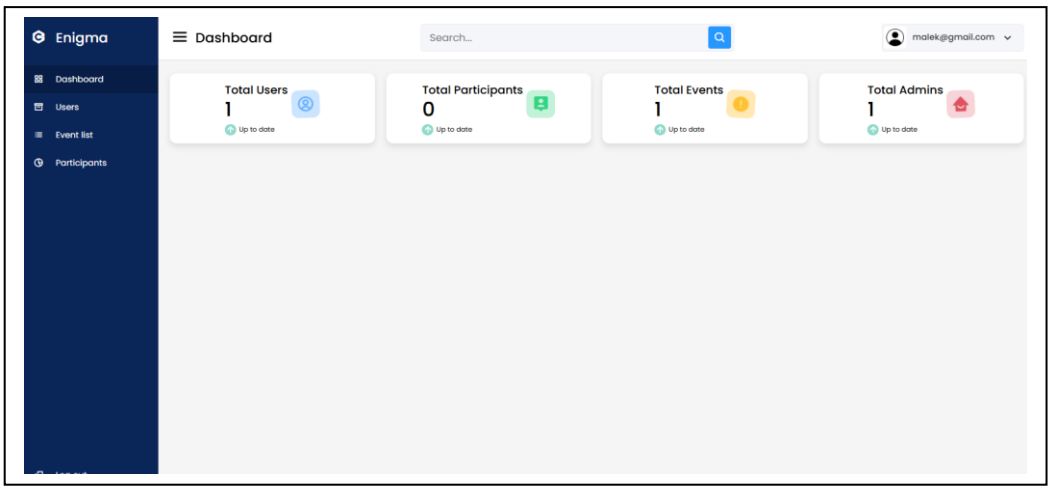


Figure 26 : Interface tableau de bord de l'application legacy.

La figure 26 illustre la page d'accueil admin qui affiche des statistiques comme le nombre total d'utilisateurs, d'événements, et de participants, avec un design classique en PHP.

4.7.2 Étapes de migration

La figure 27 affiche des logs détaillés des appels agents.

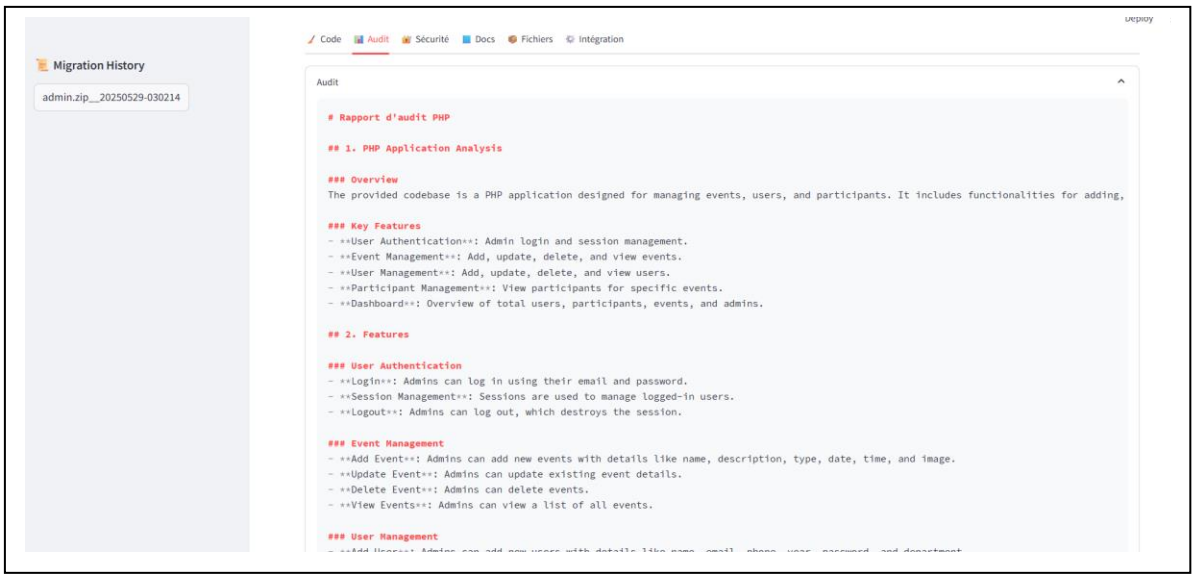


Figure 27 : Audit du code PHP

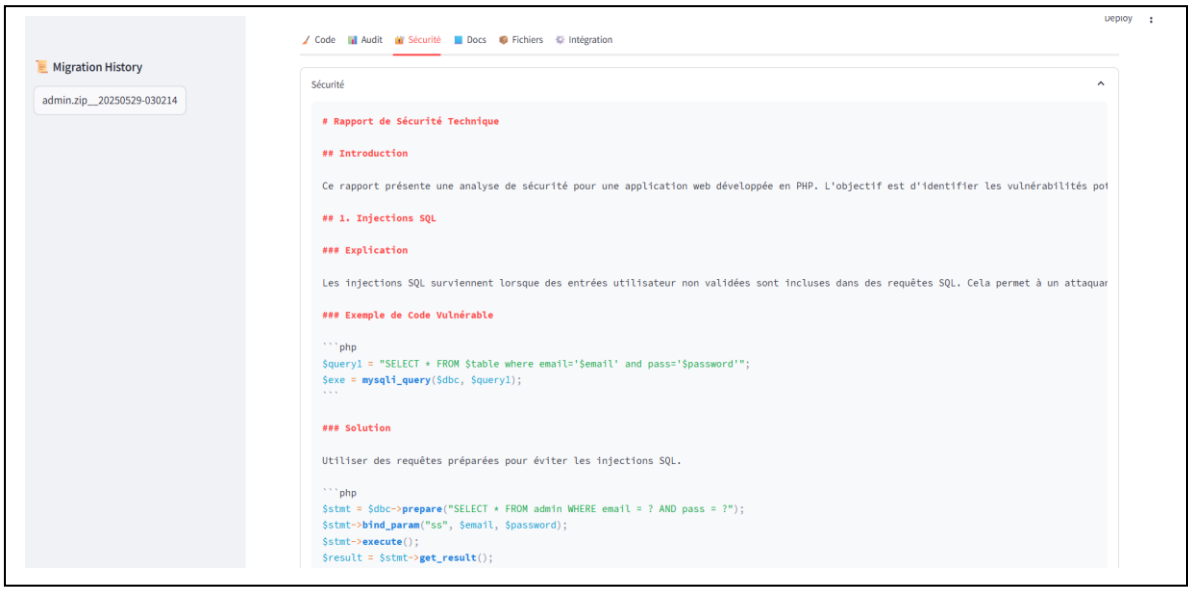


Figure 28 : Analyse des erreurs du code PHP.

La figure 28 montre l'analyse des erreurs du code PHP.

Figure 30 : Documentation du code migré.

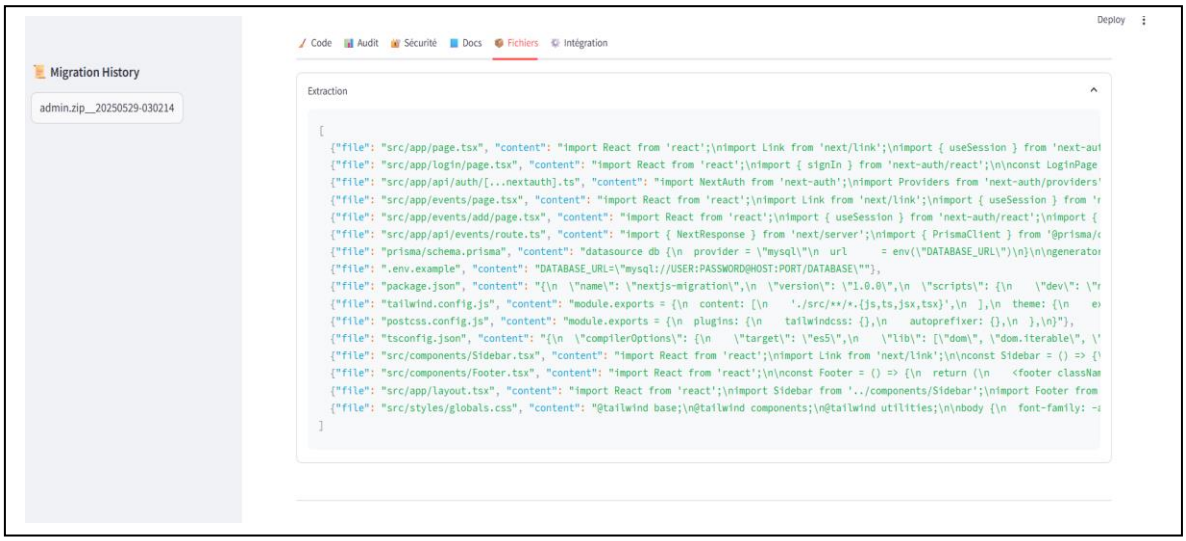


Figure 29 : Fichiers du code migré.

Les figures 29 et 30 illustrent respectivement la documentation générée à partir du code migré et les différents fichiers associés.

4.7.3 Projet migré

La figure 31 affiche la liste des projets précédemment migrés et permet à l'utilisateur de démarrer un projet migré via un terminal web intégré.

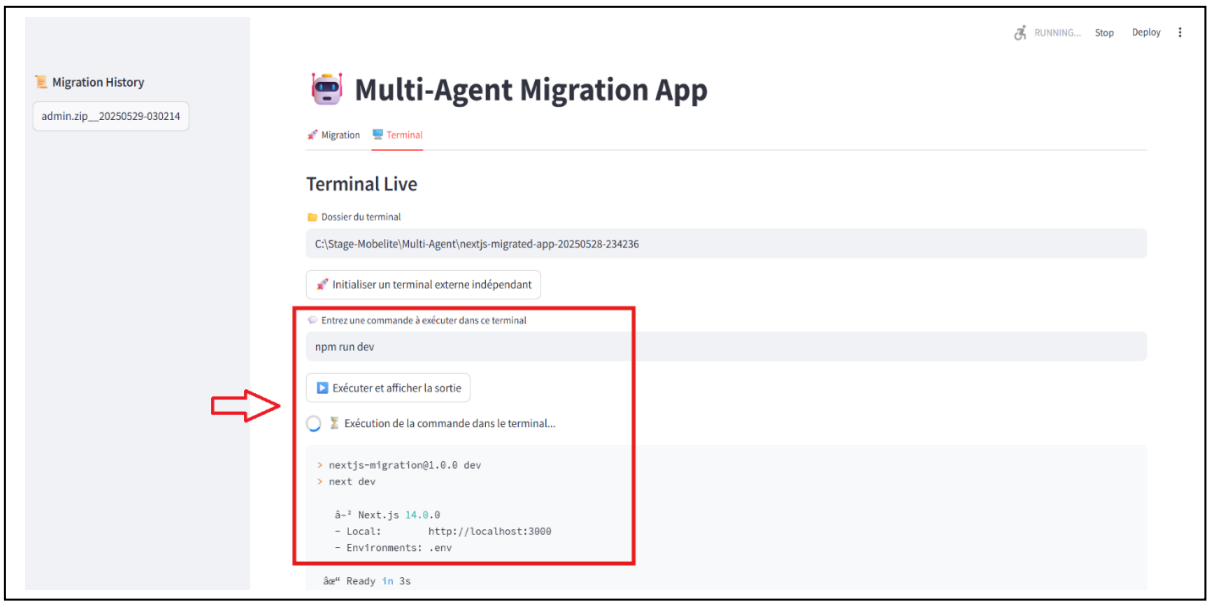


Figure 31 : Exécution du projet migré à travers le terminal interactif

Une version modernisée et minimaliste de l'interface login, générée automatiquement lors de la migration est illustrée par la figure 32.

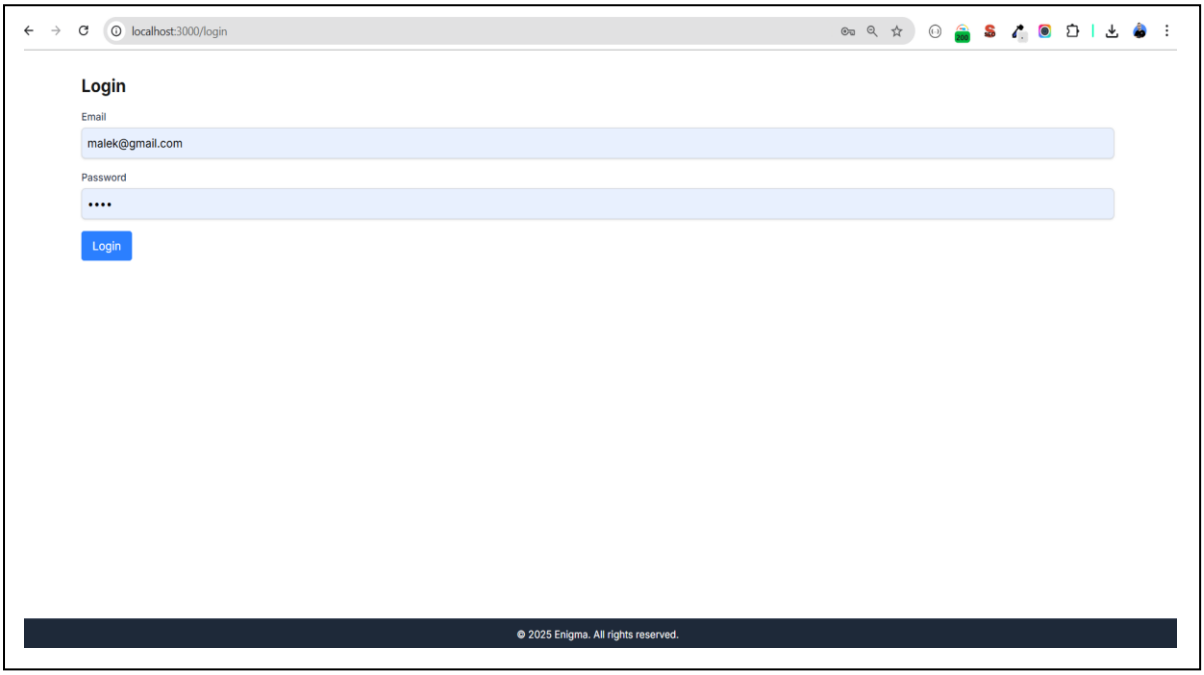


Figure 32 : Nouvelle interface de connexion développée en Next.js.

La figure 33 illustre une nouvelle version du tableau de bord avec des composants React et une meilleure structure UI, propulsée par Tailwind CSS et Prisma.

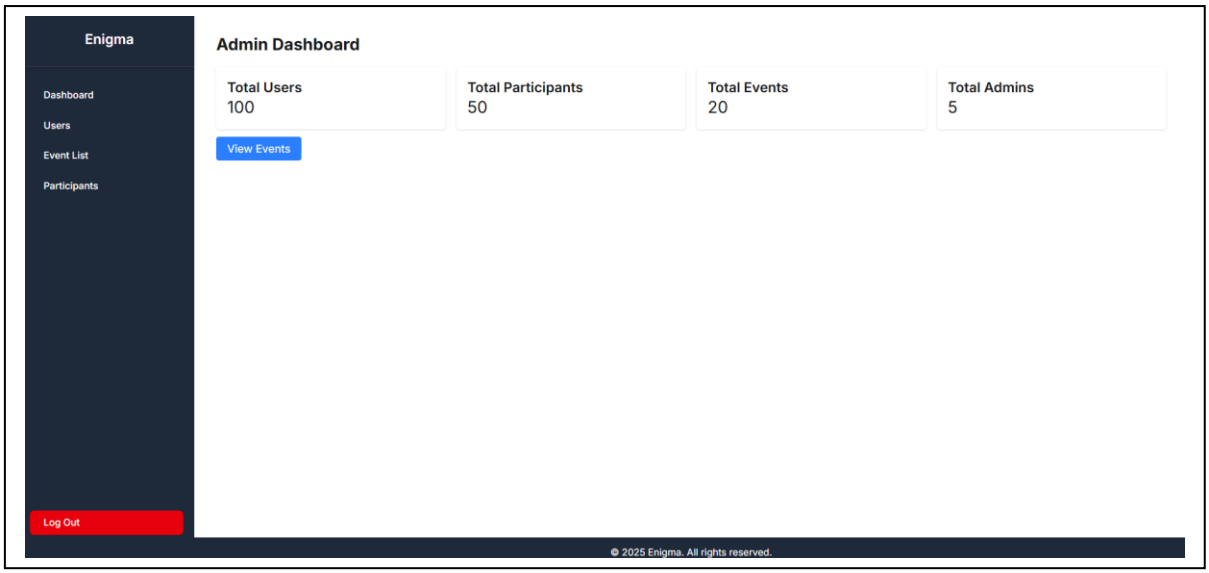


Figure 33 : Dashboard admin migré vers Next.js.

La figure 34 montre le composant Next.js listant les utilisateurs avec leurs rôles, e-mails, et actions possibles (édition, suppression).

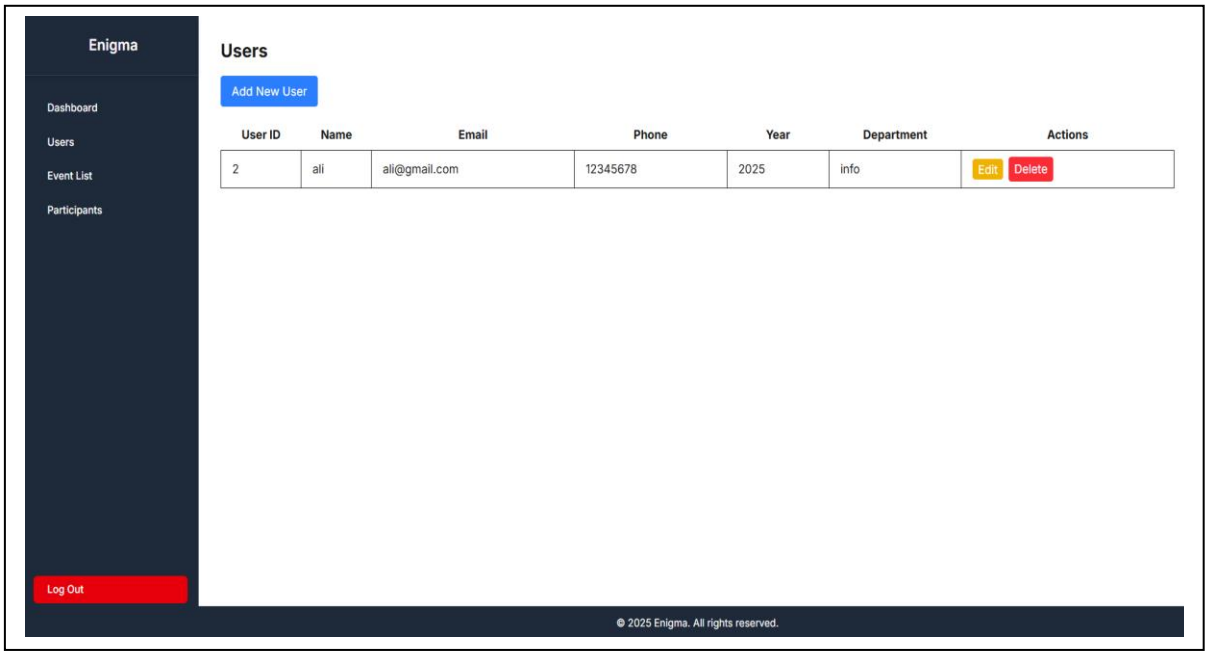


Figure 34 : Tableau des utilisateurs dans l'interface migrée.

La figure 35 montre le formulaire permettant d'ajouter un utilisateur avec nom, mail, mot de passe et rôle. Ce formulaire fait partie des interfaces recréées automatiquement.

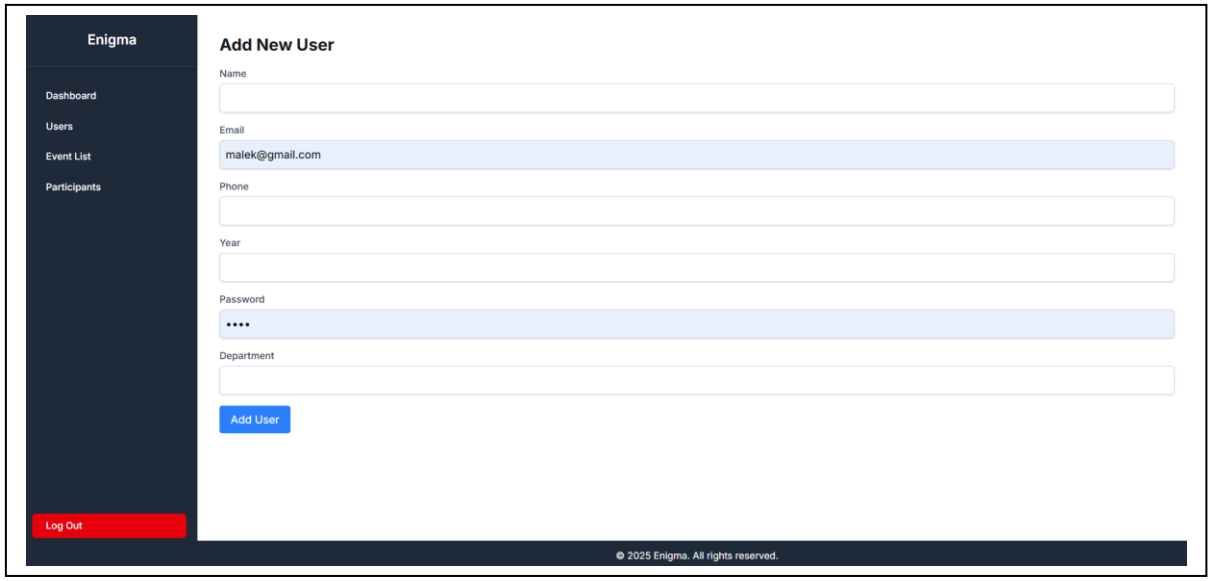


Figure 35 : Formulaire d'ajout d'utilisateur.

4.8 Conclusion

L'ensemble de ces agents permet de construire un pipeline intelligent, automatisé et supervisable, adapté à des environnements complexes et des systèmes legacy hétérogènes. L'architecture implémentée est modulaire, évolutive, et extensible vers d'autres langages, cibles ou contraintes métiers. Elle offre un compromis optimal entre contrôle humain et autonomie IA.

CONCLUSION GÉNÉRALE

Ce travail a permis de concevoir, développer et expérimenter un système intelligent de migration d'applications legacy, combinant des modèles larges de langage (LLMs) à une architecture multi-agent orchestrée via CrewAI. Cette approche répond efficacement à une problématique d'actualité : comment moderniser des systèmes obsolètes de manière fiable, scalable et automatisée, tout en limitant les risques, les coûts et les délais.

Le système proposé se distingue par sa modularité, sa capacité à distribuer les tâches entre agents spécialisés, et son intégration fluide avec des modèles IA adaptés à la génération de code, la refactorisation, la documentation et la sécurisation. Les résultats expérimentaux obtenus démontrent la pertinence de cette solution, notamment en termes de performance, de réduction de la dette technique, et de facilité de déploiement dans des contextes réels.

En plus de sa robustesse technique, l'interface utilisateur développée permet une prise en main fluide, rendant la solution accessible aux développeurs sans expertise spécifique en IA. Ce projet contribue ainsi à démocratiser l'usage des agents intelligents dans des pipelines de migration logicielle, et ouvre la voie à une automatisation accrue dans les processus de transformation digitale

PERSPECTIVES

Ce projet pose les bases d'une solution robuste pour la migration automatisée d'applications legacy, mais il ouvre également plusieurs pistes prometteuses pour les travaux futurs. L'un des axes majeurs d'évolution concerne l'extension multi-langage : l'intégration de langages supplémentaires comme Java, .NET ou Python permettrait d'élargir significativement le champ d'application du pipeline de migration. En parallèle, le raffinement du raisonnement multi-agent est envisagé grâce à l'ajout d'agents décisionnels capables d'ajuster dynamiquement les stratégies de migration en fonction de la complexité du code ou des contraintes métiers détectés.

Une autre amélioration clé réside dans l'introduction de mécanismes d'apprentissage itératif : en intégrant du feedback automatique après chaque migration, les agents pourraient affiner leurs décisions et gagner en efficacité au fil du temps. Le déploiement cloud sécurisé constitue également une perspective centrale, avec l'intégration directe à des plateformes comme AWS, Azure ou GCP, afin d'automatiser le déploiement des projets modernisés dans des environnements cloud natifs.

Enfin, une interface collaborative représente un levier important pour l'usage en contexte professionnel. L'enrichissement de l'interface Streamlit avec des fonctionnalités multi-utilisateurs permettrait de piloter les migrations en équipe, avec une gestion des rôles, des validations croisées et une supervision partagée des résultats.

Annexe A – Indicateurs de performance pour l'évaluation des LLMs

Les résultats comparatifs présentés en section 2.4.5 sont issus de benchmarks standards largement utilisés dans la communauté IA. Voici un aperçu de chacun :

Tableau 22 : Indicateurs d'évaluation.

Référence	Description	Domaine d'intervention	Langages	Méthode d'évaluation
HumanEval	Une référence pour évaluer les modèles de génération de code basés sur l'exactitude fonctionnelle.	Génération de code	Multiple	Tests unitaires
LiveCodeBench	Évalue les modèles d'IA sur des tâches de programmation en direct qui nécessitent un débogage et un raffinement itératifs (pour les scénarios de codage avec intervention humaine)	Codage en direct, raffinement du code, débogage	Python	Basé sur l'exécution
RepoBench	Teste les LLMs sur des tâches réelles au niveau du référentiel, telles que la correction de bugs et la génération de documentation.	Compréhension, complétion et récupération du code au niveau du référentiel	Python, Java	Précision de la récupération et de l'achèvement
Spider	Un ensemble de données à grande échelle pour évaluer les modèles de génération de requêtes SQL.	Conversion de texte en SQL, génération de requêtes, interaction avec la base de données	SQL	Exécution et équivalence logique
CanItEdit	Mesure la capacité des modèles à modifier et à améliorer le code existant.	Édition de code, refactorisation, correction de bugs	Python	Cas de test cachés
MultiPL-E	Étend HumanEval et MBPP à plusieurs langages de programmation pour l'évaluation de la génération de code multilingue.	Génération de code multilingue	Python, JavaScript, C++, Java, Go, Ruby, Rust, etc.	Tests unitaires
DS-1000	Un ensemble de données axé sur l'évaluation des modèles d'IA sur les tâches de codage liées à la science des données.	Tâches de science des données et de ML	Python	Basé sur l'exécution
APPS	Un benchmark contenant des problèmes de codage de différents niveaux de difficulté pour évaluer les modèles de génération de code.	Génération de code, résolution de problèmes	Python	Exactitude des entrées-sorties

BIBLIOGRAPHIE

[1] I. G. a. Y. B. a. A. Courville, Deep Learning. MIT Press, MIT Press, 2016.

[2] A. S. N. P. N. e. a. Vaswani, "Attention is All You Need. Advances in Neural Information Processing Systems," 2017.

[3] D. Foster, Generative Deep Learning, O'Reilly Media, 2019.

[4] M. AI, "Codestral-2501: Enhanced Code Generation and FIM Capabilities.," 2025. [Online]. Available: https://mistral.ai/news/codestral-2501. [Accessed 3 3 2025].

[5] M. AI, "Code Llama: Open Foundation Models for Code.," 2025. [Online]. Available: https://ai.meta.com/research/publications/code-llama-open-foundation-models-for-code. [Accessed 3 3 2025].

[6] H. Face, "DeepSeek Coder V2 Lite Instruct Model.," 2025. [Online]. Available: https://huggingface.co/deepseek-ai/DeepSeek-Coder-V2-Lite-Instruct. [Accessed 1 3 2025].

[7] Z. R. A. M. S. Z. Z. K.-K. K. J. R. S. S. M. S. P. A. Valtteri Ala-Salmi, "Autonomous Legacy Web Application Upgrades Using a Multi-Agent System," 31 1 2025. [Online]. Available: https://arxiv.org/abs/2501.19204. [Accessed 15 2 2025].

[8] J. M. M. L. Q. B. Y. Q. H. C. Dong Huang, "AgentCoder: Multi-Agent Code Generation with," 24 5 2024. [Online]. Available: https://arxiv.org/pdf/2312.13010. [Accessed 3 3 2025].

[9] CrewAi, "CrewAi," 2025. [Online]. Available: https://docs.crewai.com/introduction. [Accessed 20 02 2025].

[10] Hackathon, "Traversaal-x-Optimized-AI-Hackathon," 15 4 2025. [Online]. Available: https://github.com/Hiba-Chaabnia/Traversaal-x-Optimized-AI-Hackathon.

[11] vijaisuria, "admin-panel-event-management," 2023. [Online]. Available: https://github.com/vijaisuria/admin-panel-event-management. [Accessed 20 4 2025].

[12] Arxiv, "Advanced Techniques in LLM-based Code Refactoring.," 2025. [Online]. Available: https://arxiv.org/pdf/2406.11931.

[13] F. Martin, Refactoring: Improving the Design of Existing Code, Boston: Addison-Wesley, 2018.

[14] H. Geoff, Building Intelligent Systems: A Guide to Machine Learning Engineering, Cham: Springer, 2022.

[15] R. Z. S. A. M. Z. Z. K. K.-K. R. J. S. S. S. M. A. P. Ala-Salmi V., "Autonomous Legacy Web Application Upgrades Using a Multi-Agent System," *arXiv preprint, arXiv:2501.19204v1,* p. 13 pages, 2025.

[16] T. J. J. H. Y. Q. d. O. P. H. P. K. J. E. H. B. Y. e. a. Chen M., "Evaluating Large Language Models Trained on Code," *arXiv preprint, arXiv:2107.03374,* p. 33 pages, 2021.

[17] Z. J. M. L. M. B. Q. Q. Y. C. H. Huang D., "AgentCoder: Multi-Agent Code Generation with Effective Testing and Self-optimisation," *arXiv preprint, arXiv:2312.13010v3,* p. 24 pages , 2024.

[18] OpenAI, "GPT-4 Technical Report," *arXiv preprint, arXiv:2303.08774,* p. 40 pages, 2023.

[19] D. AI, "DeepSeek Coder V2 Repository.," 1 5 2024. [Online]. Available: https://github.com/deepseek-ai/DeepSeek-Coder-V2. [Accessed 3 3 2025].

[20] D. AI, "DeepSeek Coder: Bridging General LLMs and Code Intelligence," 2023. [Online]. Available: https://huggingface.co/deepseek-ai. [Accessed 3 3 2025].

[21] DataCamp, "Introduction to Codestral Models.," 2025. [Online]. Available: https://www.datacamp.com/blog/codestral-mistral-introduction. [Accessed 1 3 2025].

[22] H. Face, "CodeLlama-70B Instruct Model.," 2025. [Online]. Available: https://huggingface.co/codellama/CodeLlama-70b-Instruct-hf. [Accessed 1 3 2025].

[23] H. Face, "DeepSeek Coder 33B Instruct Model.," 2025. [Online]. Available: https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct. [Accessed 1 3 2025].

[24] Google, "Agent Development Kit," 9 4 2025. [Online]. Available: https://google.github.io/adk-docs/. [Accessed 9 4 2025].

Migration Automatisée des Applications Legacy via un Système Multi-Agent Orchestré par des Modèles Larges de Langage (LLMs)

Malek BOUZAYANI

Résumé :

La modernisation des applications legacy est devenue un enjeu stratégique pour les entreprises confrontées à des systèmes obsolètes. Ce projet propose une solution automatisée, basée sur des modèles larges de langage (LLMs) et une architecture multi-agent orchestrée par CrewAI, pour migrer efficacement ces applications vers des environnements modernes.

Le système développé assure l'audit, le refactoring, le test et le déploiement de projets PHP vers Next.js, avec le support de DeepSeek Coder et d'une interface intuitive via Streamlit. Une expérimentation réussie lors d'un hackathon a démontré la faisabilité et l'efficacité de cette approche innovante.

Mots clés : migration legacy, intelligence artificielle, LLM, système multi-agent, refactoring de code, automatisation, CrewAI, DeepSeek Coder.

Abstract:

Modernizing legacy applications is a key challenge for organizations aiming to embrace digital transformation. This project presents an automated migration system based on Large Language Models (LLMs) and a multi-agent architecture managed by CrewAI.

The system handles auditing, refactoring, testing, and deployment of PHP applications into modern frameworks like Next.js, using DeepSeek Coder and a Streamlit interface. A successful hackathon experiment validated the solution's performance and potential for real-world adoption.

Keywords: legacy migration, artificial intelligence, LLM, multi-agent system, code refactoring, automation, CrewAI, DeepSeek Coder.