REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Université 8 Mai 1945 – Guelma

Faculté des Mathématiques, d'Informatique et des Sciences de la matière

Département d'Informatique

Réf ...../2025

MEMOIRE

Présenté pour l'obtention du **diplôme** de **MASTER Académique**

Domaine : Mathématiques, d'Informatique et des Sciences de la matière
Filière : Informatique
Spécialité/Option : Systèmes informatiques

Par : GASMI Ahmed Anis & Ayad fadi

Thème

AGRICO.DZ : Plateforme de Coopératives Agricoles Connectées

Soutenu publiquement, le 23/06/2025, devant le jury composé de :

Mme Tadjer Houda	SMMI	Univ. Guelma	Président
Mr Benamira Adel	SMMI	Univ. Guelma	Encadreur
Mr Ferrag Mohamed Amine	SMMI	Univ. Guelma	Examinateur
Mr Ghnia Berkat		Univ. Guelma	Président Pôle Pro

Année Universitaire : 2024/2025

REMERCIEMENT

Le grand remerciement revient à Dieu, qui nous a donné la force et le courage de pouvoir terminer ce travail.

Tout d'abord, ce travail ne serait pas aussi riche et n'aurait pas pu avoir le jour sans l'aide et l'encadrement du **Professeur Benamira Adel**, nous le remercions pour la qualité de son encadrement exceptionnel, pour sa patience, sa rigueur et sa disponibilité durant notre préparation de ce mémoire.

Sans oublier tous ceux qui nous ont aidés et encouragés de près ou de loin à la réalisation de ce travail.

Enfin, je ne saurais terminer ces remerciements sans oublier nos familles, sans exception, ainsi que tous nos amis.

DEDICACES

À mes chers parents, mes frères et ma grande sœur,
votre soutien inébranlable et vos encouragements sans fin ont façonné mon parcours.
Ce mémoire témoigne de votre amour infini et de vos sacrifices.
À mon professeur exceptionnel (Professeur Benamira Adel),
votre guidance et votre sagesse ont éveillé ma passion pour l'apprentissage.
Ce travail est le reflet de votre mentorat inestimable et de votre inspiration.

GASMI Ahmed Anis

À mes chers parents,
pour leur soutien inconditionnel, leurs encouragements,
et bien sûr pour m'avoir permis de réaliser mes études dans les meilleures conditions ;
à toute ma grande famille ;
à ceux qui m'ont soutenu pendant toute la durée de mes études…
…je dédie ce modeste travail.

AYAD Fadi

RÉSUMÉ :

Le travail présenté dans ce mémoire s'inscrit dans le cadre d'un projet de fin d'études en ingénierie logicielle, visant à concevoir et développer une application numérique dédiée à la mise en relation d'acteurs du secteur agricole. Ce projet s'appuie sur une architecture moderne orientée microservices et s'inspire des principes du Domain-Driven Design (DDD) pour garantir modularité, évolutivité et clarté dans la répartition des responsabilités métier.

L'objectif principal de l'application, nommée Agrico, est de faciliter la location de matériels agricoles, la recherche de main-d'œuvre spécialisée (opérateurs) ou de transporteurs, ainsi que la gestion des transactions et évaluations liées à ces services. L'approche adoptée permet de modéliser chaque besoin métier comme un sous-système autonome, géré par un microservice dédié. Le tout est orchestré via une architecture orientée événements (EDA) qui permet des interactions fluides, en temps réel, entre les différents modules.

L'étude comprend une analyse approfondie du domaine agricole, l'identification des différents contextes fonctionnels (utilisateurs, matériels, terrains, réservations, paiements, notifications, recommandations, évaluations), ainsi que leur modélisation détaillée à l'aide de diagrammes UML. L'optimisation de l'expérience utilisateur, la sécurité des échanges, la réactivité des notifications et la traçabilité des opérations sont au cœur des préoccupations du système.

Le résultat est une plateforme cohérente, flexible, et adaptée aux besoins du terrain, offrant une solution numérique pertinente pour digitaliser les interactions agricoles et renforcer la collaboration entre les acteurs du secteur.

Mots clés : Domain-Driven Design, Microservices, Agriculture, Réservation, EDA, Utilisateur, Matériel agricole, Transporteur.

Table des matières

Table des Figures

Liste des tableaux

Introduction Générale

Le secteur agricole constitue l'un des piliers fondamentaux de l'économie algérienne. En plus de son rôle central dans la sécurité alimentaire, il contribue activement au développement rural et à la diversification des revenus du pays, notamment en réduisant la dépendance aux hydrocarbures. Face aux enjeux stratégiques que représente ce secteur, l'Algérie a engagé ces dernières années une série de réformes et d'initiatives pour le relancer, en le positionnant comme levier de croissance durable. Cependant, cette ambition se heurte encore à plusieurs contraintes structurelles : l'insuffisance des infrastructures logistiques, l'absence d'un marché structuré pour les services agricoles, le manque de main-d'œuvre qualifiée, ainsi que l'usage persistant de méthodes de gestion traditionnelles. Un besoin particulièrement critique concerne l'accès limité aux services essentiels tels que la location de matériel, le transport des produits agricoles ou encore l'organisation de la location des terres.

Dans ce contexte, le projet AGRICO.DZ est né comme une réponse technologique innovante, visant à structurer l'écosystème agricole informel à travers une plateforme numérique dédiée. Celle-ci permet de mettre en relation les agriculteurs avec différents prestataires (loueurs de matériel, opérateurs de machines, transporteurs, propriétaires fonciers) au sein d'un environnement numérique transparent, efficace et adapté aux réalités du terrain. Le projet repose sur des fondations techniques robustes, notamment une architecture microservices qui permet de construire un système modulaire, évolutif et résilient. Chaque sous-système (gestion des utilisateurs, matériels, réservations, paiements, etc.) est développé comme un microservice indépendant, ce qui facilite l'évolutivité et la maintenabilité de l'ensemble de la plateforme.

Par ailleurs, AGRICO.DZ intègre un système de recommandation intelligent basé sur l'analyse du profil utilisateur, des préférences déclarées, de l'historique d'utilisation et de la localisation. Ce moteur de recommandation améliore significativement l'expérience utilisateur en facilitant la recherche des services pertinents, en valorisant les profils fiables et en renforçant la confiance entre les différentes parties. Le projet dépasse ainsi le cadre d'une simple application de mise en relation pour devenir un véritable écosystème numérique agricole, orienté vers la modernisation des pratiques, l'autonomisation des acteurs ruraux et la promotion de l'économie collaborative.

Ce mémoire vise à analyser en profondeur les fondements conceptuels, fonctionnels et techniques ayant guidé la conception de cette plateforme. Il s'appuie sur une démarche progressive : l'étude des architectures microservices, l'adoption des principes du Domain-Driven Design (DDD), l'intégration d'une architecture orientée événements (EDA), ainsi que la mise en œuvre d'un système de recommandation hybride. Une attention particulière est portée à la modélisation des sous-domaines métiers propres au contexte agricole algérien et à leur traduction en services applicatifs indépendants.

La suite de ce travail est structurée en six chapitres. Le Chapitre I présente les fondements des architectures microservices, en les comparant aux approches monolithiques et SOA, et en détaillant leurs composants et enjeux. Le Chapitre II expose les principes du Domain-Driven Design (DDD) et de l'architecture orientée événements (EDA), qui ont guidé la modélisation des domaines métiers. Le Chapitre III est consacré aux systèmes de recommandation, en explorant leurs types et leur utilité dans la personnalisation des services. Le Chapitre IV traite du déploiement technique, en abordant la conteneurisation, l'orchestration (Kubernetes) et les outils CI/CD. Le Chapitre V approfondit l'analyse et la conception métier de la plateforme AGRICO.DZ à travers les contextes fonctionnels, les flux et l'architecture microservices. Enfin, le Chapitre VI détaille l'implémentation concrète du système, ses technologies principales, ainsi que les interfaces utilisateur développées.

Ce projet, à la croisée de l'innovation technologique et du développement territorial, entend ainsi contribuer concrètement à la modernisation du secteur agricole algérien en apportant des solutions numériques réalistes, évolutives et centrées sur les besoins des utilisateurs.

Chapitre I – Introduction aux Microservices

1 Introduction

L'évolution des architectures logicielles est caractérisée par un passage à des systèmes plus modulaires, plus flexibles et plus adaptatifs. Dans ce cadre, les microservices constituent une approche architecturale qui prend de la vitesse de chacune des entreprises souhaitant automatiser son développement logiciel et accélérer sa transformation numérique.

2 Définition et principes fondateurs

Les microservices constituent une méthode de développement logiciel utilisée pour concevoir une application comme un ensemble de services modulaires.

Selon Martin Fowler, dont la définition fait référence dans le domaine : « Le style architectural des microservices est une approche permettant de développer une application unique sous la forme d'une suite de petits services, chacun s'exécutant dans son propre processus et communiquant avec des mécanismes légers, souvent une API de ressources HTTP. » [1]

Le point clé de base du microservice repose sur un principe simple : c'est une application qui fait une chose en une seule mais qu'elle fait au mieux. Chaque microservice est consacré à une attache métier particulière et s'effectue avec les autres composants. Pour Fowler, des microservices doivent avoir besoin du minimum possible en termes de gestion centralisée et peut-être programmé sous différents langages de programmation. [1]

Les caractéristiques clés sont [2] :

- **Modularité :** Chaque service, unique capacité métier, plus facile de maintenance.
- **Autonomie :** Les services peuvent être créés, déployés, d'amplifier, indépendamment.
- **Faible couplage :** Les communications s via API diminuent les dépendances, offrant plus de flexibilité.
- **Résilience :** Une panne d'un service n'empêche pas l'ensemble grâce à des mécanismes comme les disjoncteurs.
- **Scalabilité :** Ceux-ci peuvent être mis à l'échelle de manière individuelle à la demande afin de gérer les ressources.

3 Monolithique vs SOA vs Microservices

Le choix entre une architecture microservices, une architecture monolithique, ou une architecture orientée services (SOA) dépend de plusieurs facteurs spécifiques au contexte de l'entreprise.

3.1 Architecture Monolithique

Les applications monolithiques se composent de composants qui sont tous étroitement liés et doivent être développés, déployés et gérés comme une seule entité, car ils s'exécutent tous comme un processus de système d'exploitation simple. [3]
Les modifications apportées à une partie de l'application nécessitent un redéploiement de l'ensemble de l'application et, au fil du temps, l'absence de limites strictes entre les parties entraîne une augmentation de la complexité et une détérioration conséquente de la qualité de l'ensemble du système en raison de la croissance sans contrainte des interdépendances entre ces parties.

3.2 Architecture Orientée Service (SOA)

SOA est similaire aux microservices en termes de modularité, mais les services SOA sont généralement plus gros et plus complexes. Cette architecture est souvent préférée dans les environnements où l'interopérabilité entre différentes applications est critique, et où les entreprises ont déjà investi dans une infrastructure orientée services. [3]

3.3 Comparaison entre SOA et Microservices

Des voix critiques à l'égard de l'architecture des microservices (MSA) affirment qu'elle ne serait qu'une réédition de l'architecture orientée services (SOA). Vue de loin, il est vrai qu'on peut identifier certaines ressemblances. Ces deux approches conçoivent un système comme un assemblage de services. Toutefois, en y regardant de plus près, comme l'illustre le tableau 1, des différences marquées se font jour. [3]

Critère	SOA	MSA
Communication inter-services	Smart pipes (Entreprise Service Bus), utilisant des protocoles lourds (SOAP).	Dumb pipes (message broker), communication directe de service à service, utilisant des protocoles légers (REST, gRPC).
Données	Modèle de données global et bases de données partagées.	Modèle et base de données par service.
Service typique	Application monolithique plus grande.	Service plus petit.

Table I. 1 -Comparaison entre SOA et les microservices [3]

L'architecture orientée services (SOA) et celle des microservices s'opposent d'abord par les technologies qu'elles empruntent. Les applications SOA s'appuient souvent sur des outils lourds comme SOAP ou les standards WS*, et intègrent fréquemment une ESB, sorte de canal intelligent qui regroupe la logique métier et le traitement des messages pour connecter les services. [3]

À l'inverse, les applications fondées sur les microservices optent pour des technologies légères et souvent open source, avec des services qui échangent via des canaux simples, comme des message brokers ou des protocoles allégés tels que REST ou gRPC. [1]

Un autre contraste est la gestion des données. Dans un cadre SOA, on se retrouve généralement en face d'un modèle de données commun, et bases de données partagées. Les microservices répondent d'une manière différente : chaque service dispose d'une base de données pour lui et il est généralement considéré comme ayant son propre modèle de domaine. [1]

Une autre différence clé est la taille des services. SOA est pensée pour rassembler d'applications monolithiques, volumineuses et complexes, ce qui provoque un petit nombre de services gigantesques. Les microservices sont presque toujours nettement plus petits, conduisant les applications composées de dizaines services d'embryonnaire. [3]

4 Structure et Composants d'une application microservices

La structure d'une application microservices repose sur la décomposition en services indépendants, organisés autour de capacités métiers spécifiques [4]. Les composants essentiels d'une architecture microservices incluent :

4.1.1 Les Services

Unités de base, Chaque microservice au sein de l'architecture remplit une fonction métier spécifique et peut s'exécuter sur plusieurs instances pour plus scalabilité. Les services fonctionnent de manière indépendante mais communiquent entre eux via des appels API ou des courtiers de messages (message brokers). [4]

4.1.2 API Gateway

API Gateway sert de point d'entrée unique pour les applications clientes. Il gère le routage, le filtrage et l'équilibrage de charge des demandes entre différents microservices. Ce composant simplifie les interactions avec les clients en fournissant une interface unifiée pour tous les services. [5]

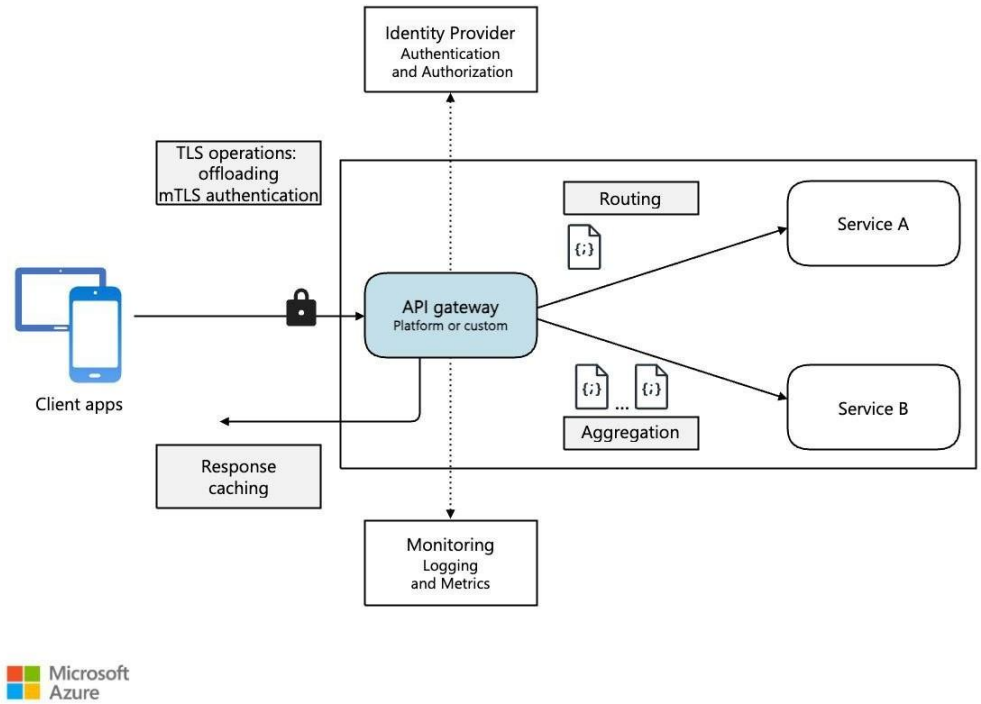

Figure I. 1-API Gateway [5]

4.1.3 Registre de services (Service Registry)

Un registre de services conserve des informations sur tous les services en cours d'exécution, ce qui permet à API Gateway de découvrir et d'acheminer les demandes vers le service approprié. Il permet également de gérer la nature dynamique des microservices, où les instances peuvent démarrer et s'arrêter fréquemment. [4]

4.1.4 Serveur d'autorisation (Authorization Server)

Pour sécuriser les microservices et gérer le contrôle d'accès, un serveur d'autorisation est indispensable. Ce composant gère l'authentification, l'autorisation et la gestion de l'identité de l'utilisateur. Les outils les plus populaires dans ce domaine incluent Keycloak, Clerk et Okta. [4]

4.1.5 Bases de données

Les microservices ont besoin d'une solution de base de données évolutive pour stocker les données d'application. Les bases de données traditionnelles telles que PostgreSQL et MySQL sont largement utilisées en raison de leur fiabilité et de leurs performances. En fonction des exigences de stockage de données du microservice, d'autres bases de données telles

Que MongoDB ou Cassandra peuvent également être utilisées pour des besoins de données non structurées ou NoSQL. [4]

4.1.6 Communication asynchrone entre microservices

Pour la communication asynchrone entre microservices, des messages brokers tels qu'Apache Kafka ou RabbitMQ sont essentiels. Ces outils permettent une communication fiable, scalable et découplée en transmettant des messages dans des files d'attente ou des rubriques, ce qui

Permet aux services de fonctionner de manière indépendante sans attendre de réponses immédiates. [6]

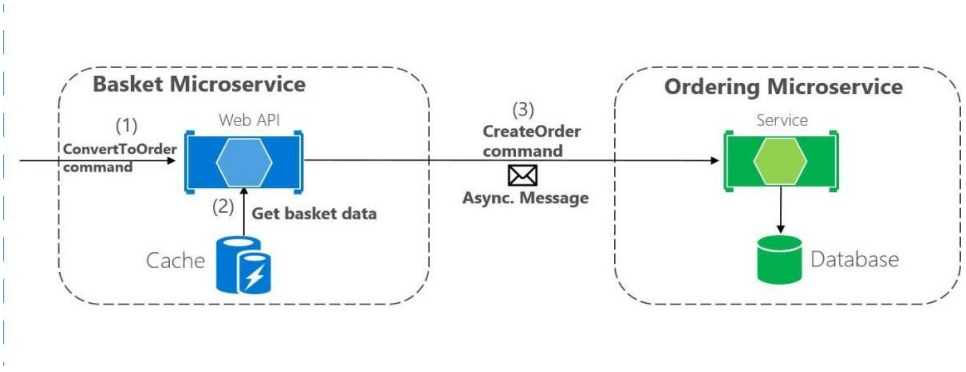

Figure I. 2Communication Asynchrone entre 2 microservices [6]

4.1.7 Visualisation des métriques

Le suivi des performances des microservices est crucial dans les environnements de production. Les métriques peuvent être collectées à l'aide de Prometheus,

Et Grafana peut être utilisé pour visualiser ces métriques, fournissant des informations sur l'intégrité, la latence et le trafic des microservices. [4]

4.1.8 Agrégation et visualisation des journaux

Une journalisation efficace permet de diagnostiquer les problèmes et de suivre les erreurs dans les microservices. Les journaux générés par les services peuvent être agrégés à l'aide d'outils tels que Logstash et stockés

Dans Elasticsearch. Kibana peut ensuite être utilisé pour visualiser et analyser les journaux, offrant ainsi une vue claire du comportement de l'application au fil du temps. [4]

5 Avantages et Inconvénients des Microservices

5.1 Avantages des microservices

L'architecture microservices offre plusieurs avantages significatifs pour les organisations :

Résilience et isolation des pannes : Dans une architecture monolithique, une erreur va affecter toute l'application. Les microservices, en revanche, sont indépendants. Un échec ne va pas affecter les autres parties de l'application [2]. Cette isolation des défaillances permet d'améliorer la stabilité globale du système.

Agilité et rapidité de développement : Le fait de scinder les applications en différents éléments plus réduits permet d'accélérer le développement. Les équipes peuvent ainsi créer, tester et déployer des logiciels plus rapidement [2]. Cette agilité accrue est particulièrement précieuse dans un environnement commercial qui évolue rapidement.

Diversité technologique : Les microservices permettent aux développeurs de choisir les outils et les technologies les mieux adaptés à leurs tâches. L'efficacité et la productivité en sont ainsi augmentées [2]. Cette liberté technologique favorise l'innovation et l'adoption des meilleures solutions pour chaque composant.

Maintenance améliorée : La possibilité de tester des composants individuels permet de détecter et de corriger les bugs plus facilement, sans devoir mettre l'ensemble de l'application hors connexion [2]. Cette approche modulaire facilite également les mises à jour et les évolutions du système.

Évolutivité : Les services individuels peuvent évoluer parallèlement à la demande, afin de réduire les coûts et de garantir la disponibilité [2]. Cette capacité à adapter les ressources en fonction des besoins spécifiques optimise l'utilisation des infrastructures.

5.2 Inconvénients et défis des microservices

Malgré leurs nombreux avantages, les microservices introduisent également des défis importants :

- **Complexité accrue** : Les microservices introduisent une complexité technique importante, notamment en termes de gestion des déploiements, des dépendances, de la sécurité, et de la maintenance [7]. Cette complexité peut être difficile à gérer sans une équipe technique expérimentée.
- **Coûts d'implémentation élevés** : Les microservices nécessitent des investissements significatifs en formation, en infrastructure, et en outils de gestion [7]. Le coût initial de mise en place peut être prohibitif pour certaines organisations, en particulier les petites entreprises.
- **Défis de coordination** : La gestion de multiples services indépendants requiert des mécanismes de coordination sophistiqués pour assurer la cohérence et l'intégrité du système dans son ensemble [7]. Cette orchestration peut devenir un défi majeur à mesure que le nombre de services augmente.

5.3 Critères de choix pour une architecture microservices

Plusieurs facteurs doivent être pris en compte lors de l'évaluation de l'adoption des microservices :

- **Nature du projet** : Pour les projets de petite envergure ou avec des fonctionnalités bien définies et peu évolutives, une architecture monolithique peut être plus appropriée. En revanche, pour les projets complexes, évolutifs, et nécessitant une forte modularité, les microservices peuvent offrir une meilleure flexibilité et évolutivité [7].
- **Objectifs à long terme** : Si l'entreprise prévoit une croissance rapide, des besoins en scalabilité, ou une intégration continue de nouvelles fonctionnalités, les microservices peuvent être le meilleur choix. Cependant, si la priorité est de maintenir des coûts bas et de limiter la complexité, une architecture monolithique ou SOA pourrait être plus adaptée [7].

- **Ressources disponibles** : Les microservices nécessitent des compétences techniques avancées et une infrastructure adaptée. Les DSI doivent s'assurer que leur organisation dispose des ressources nécessaires pour soutenir cette architecture à long terme [7].
- **Retour sur investissement (ROI)**: Les DSI doivent comparer les coûts d'implémentation et de maintenance de chaque architecture avec les avantages attendus. Pour les microservices, le ROI peut être élevé si l'entreprise bénéficie de la scalabilité, de la rapidité de mise sur le marché, et de la résilience accrue [7].

6 Conclusion

En conclusion, l'architecture microservices représente un changement significatif dans la conception logicielle, offrant scalabilité, résilience et efficacité de développement. Malgré des défis comme la gestion de la complexité et les changements organisationnels, leurs bénéfices en font un choix stratégique pour les applications modernes.

La décision d'adopter les microservices doit donc résulter d'une analyse approfondie qui tient compte de la complexité du projet, des objectifs à long terme, des ressources disponibles et du retour sur investissement attendu. Cette approche équilibrée permettra aux organisations de tirer le meilleur parti de cette architecture prometteuse tout en minimisant les risques associés.

Chapitre II – Domain-Driven Design et Architecture Événementielle

1 Introduction

La complexité croissante des systèmes logiciels modernes, notamment dans les architectures distribuées comme les microservices, nécessite des approches de conception orientées métier afin d'assurer cohérence, évolutivité et maintenabilité. Le Domain-Driven Design (DDD) s'impose comme une méthodologie stratégique pour aligner la conception logicielle avec les besoins métiers réels, en plaçant le domaine fonctionnel au cœur du développement. Cette approche favorise une collaboration étroite entre les experts métiers et les développeurs, à travers un langage commun et une modélisation explicite des règles métiers.

Par ailleurs, dans les systèmes à forte modularité comme ceux basés sur les microservices, la communication entre services constitue un enjeu central. C'est dans ce contexte que l'architecture orientée événements (EDA) prend tout son sens, en assurant une interaction asynchrone, réactive et faiblement couplée entre les composants. Ce chapitre explore ainsi l'interaction entre DDD et EDA, en présentant les principes, les concepts fondamentaux (entités, agrégats, contextes délimités), ainsi que les mécanismes de communication événementielle et leurs implications sur l'organisation du système.

2 Principes fondamentaux du Domain-Driven Design (DDD)

Le Domain-Driven Design repose sur des principes fondamentaux visant à aligner la conception logicielle sur le métier. Cette section présente les notions de base nécessaires pour en comprendre les fondements.

2.1 Définition et objectifs du DDD

Le Domain-Driven Design (DDD) est une approche de conception logicielle qui met l'accent sur la création d'un modèle de domaine qui représente de manière fidèle la complexité du domaine métier. Le but du DDD est de rapprocher la conception du logiciel de la réalité du domaine en travaillant étroitement avec les experts métiers et en utilisant un langage commun. Cette approche aide les équipes de développement à mieux comprendre les besoins des utilisateurs et à éviter les erreurs dues à une mauvaise interprétation des exigences métiers [8].

2.2 Importance de la modélisation du domaine métier

Le modèle de domaine est au cœur de l'approche DDD. Il sert de représentation abstraite des processus métiers, des entités et des règles d'affaires. Ce modèle est constamment raffiné et amélioré au fur et à mesure que l'équipe acquiert de nouvelles connaissances sur le domaine. Un modèle bien conçu permet non seulement de mieux comprendre le domaine, mais aussi de communiquer efficacement avec les parties prenantes tout en évitant les incohérences dans le code [9]. L'importance du modèle de domaine réside dans sa capacité à guider le développement tout en facilitant la gestion des changements, surtout dans des environnements complexes et dynamiques.

2.3 Le rôle des contextes délimités (Bounded Context)

Les contextes délimités (Bounded Contexts) sont un concept clé du DDD. Un contexte délimité définit les frontières dans lesquelles un modèle de domaine particulier est appliqué. À l'intérieur de ce contexte, les termes et concepts ont une signification spécifique, mais ils peuvent avoir une signification différente dans d'autres contextes. Par exemple, le terme "commande" pourrait désigner un concept différent dans le contexte d'un système de gestion des commandes et dans le contexte d'un système de facturation. La délimitation claire des contextes permet de prévenir les conflits de modèles et d'assurer la cohérence des données au sein de chaque sous-système [10].

2.4 L'utilisation du langage ubiquitaire (Ubiquitous Language)

Le langage ubiquitaire fait référence à un vocabulaire commun que les développeurs, les experts métiers et toutes les parties prenantes utilisent pour discuter du modèle de domaine. L'idée est que tout le monde parle le même langage pour éviter toute ambiguïté. Ce langage est directement intégré dans le code source, ce qui garantit que la compréhension du domaine est reflétée dans la structure du logiciel. Le langage ubiquitaire aide à maintenir une cohérence tout au long du projet, notamment lorsque les équipes de développement et les experts métiers échangent régulièrement des informations [8].

2.5 Structuration d'un modèle orienté domaine

La structuration d'un modèle orienté domaine permet de traduire les besoins métier en composants logiciels clairs et organisés. Elle facilite la compréhension du système, améliore sa maintenabilité et prépare le terrain pour une architecture cohérente et évolutive. Cette section présente l'architecture en couches ainsi que la manière de découper logiquement le domaine en sous-parties compréhensibles et maintenables.

2.6 L'architecture en couches appliquée au DDD

En architecture logicielle, le partitionnement d'un programme complexe en différentes couches est essentiel. Cela permet de gérer la complexité du développement et de rendre le code plus maintenable. L'architecture en couches permet de séparer les préoccupations et d'assurer que chaque couche a une responsabilité clairement définie, réduisant ainsi les risques d'interférences entre les différentes parties du système. Voici les quatre couches typiques dans ce type d'architecture :

- **Interface utilisateur (couche de présentation) :** Responsable de la présentation de l'information et de l'interprétation des commandes de l'utilisateur.
- **Couche application :** Coordonne l'activité de l'application, mais ne contient pas de logique métier.
- **Couche domaine :** Contient l'état des objets métier et le cœur de la logique métier.
- **Couche infrastructure :** S'occupe des aspects techniques comme la persistance des données, la gestion des connexions réseau et les interfaces externes.

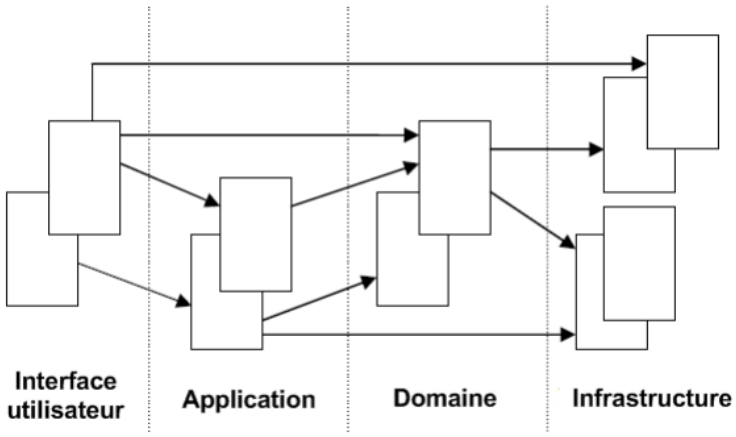

Figure II. 1 - illustration de l'architecture en couches [11]

2.7 Décomposition logique du domaine

Dans une approche Domain-Driven Design (DDD), la décomposition logique du domaine consiste à diviser un système complexe en sous-domaines cohérents, chacun correspondant à une partie distincte de la logique métier. Cette décomposition repose sur l'identification des différentes zones fonctionnelles du domaine, avec leurs propres règles, langages et contraintes. L'objectif est de clarifier les responsabilités et de permettre une meilleure compréhension et modularisation du système global.

Chaque sous-domaine peut ensuite être classé selon sa nature en domaine principal (Core Domain), domaine générique (Generic Subdomain) ou domaine de support (Supporting Subdomain). Le domaine principal contient la logique métier stratégique, celle qui différencie l'entreprise ou l'application. Le domaine de support contient des éléments nécessaires mais non stratégiques (comme la gestion des utilisateurs ou la facturation), tandis que le domaine générique est réutilisable dans d'autres contextes (par exemple, une bibliothèque d'envoi d'e-mails) [8] .

Cette décomposition est étroitement liée à la notion de Bounded Context, qui délimite les frontières au sein desquelles un modèle est appliqué de manière cohérente. Cela permet d'éviter les ambiguïtés conceptuelles et de faciliter l'alignement entre l'équipe technique et les experts métier. La clarté des responsabilités au sein de chaque contexte favorise également une architecture modulaire et extensible, en particulier dans les architectures de type microservices [9].

3 Les éléments de base d'un modèle DDD

Un modèle orienté domaine repose sur des éléments fondamentaux qui permettent de structurer la logique métier de manière précise. Cette section présente les composants clés du DDD et explique leur rôle dans la modélisation du système.

3.1 Les Entités et leur identité

Les entités sont des objets définis par une identité unique et persistante, et non par leurs attributs. Leur identité, souvent représentée par un ID (ex. : numéro de compte), reste constante dans le temps. Cela permet de distinguer et suivre les entités malgré les changements d'état. Une bonne conception d'entité garantit une identification fiable, essentielle pour la cohérence du système [11].

3.2 Les Objets-Valeurs : représentation immuable des concepts

Les objets-valeurs sont des objets qui ne nécessitent pas d'identité propre, contrairement aux entités. Ils sont définis principalement par leurs attributs et non par une identité unique. Dans la modélisation, il est important de ne pas confondre entités et objets-valeurs, car cela pourrait entraîner des inefficacités, comme la gestion inutile d'identités uniques pour des objets qui n'en ont pas besoin [11].

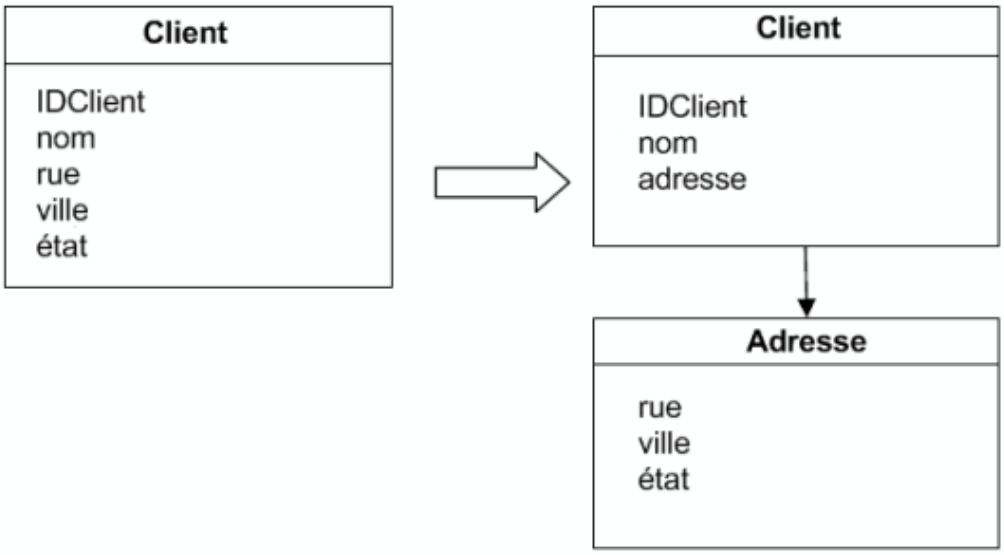

Figure II. 2 - Représentation des objets-valeurs [11]

En résumé, les objets-valeurs permettent de simplifier la conception en réduisant les coûts associés à la gestion d'identités uniques, tout en assurant l'intégrité des données grâce à leur immutabilité.

3.3 Les Services : logique métier sans état propre

Dans une modélisation orientée objet, certains comportements du domaine ne s'intègrent naturellement ni aux entités ni aux objets-valeurs. Ces actions, bien qu'importantes, n'ont pas leur place dans un objet précis sans compromettre sa cohérence. Pour répondre à ce besoin, on introduit le concept de Service.

Un Service est un objet sans état interne, conçu pour encapsuler une logique métier significative qui n'appartient naturellement à aucun objet du domaine. Il expose des opérations centrées sur le domaine et souvent transversales à plusieurs objets.

Les services permettent de préserver la clarté du modèle, en évitant un couplage excessif entre objets et en respectant les responsabilités naturelles de chaque composant.

Il est crucial de distinguer les services du domaine (qui encapsulent de la logique métier) des services applicatifs (qui orchestrent les appels entre les couches) et des services d'infrastructure (qui gèrent l'accès aux ressources techniques comme les bases de données ou les fichiers) [11].

3.4 Les Modules : organisation sémantique du modèle

Les modules permettent de structurer un modèle complexe en regroupant des concepts connexes pour réduire la complexité, renforcer la cohésion et réduire le couplage. Chaque module doit représenter un concept clair du domaine, avoir une interface bien définie, et porter un nom issu du Langage omniprésent. Une bonne modularisation facilite la compréhension, la maintenance et l'évolution du système [11].

3.5 Les Agrégats : unités de cohérence transactionnelle

Les agrégats sont des regroupements d'objets du domaine traités comme une unité cohérente. Chaque agrégat a une racine (entité principale) qui contrôle l'accès aux objets internes. Seule la racine peut être référencée de l'extérieur, ce qui simplifie la gestion du cycle de vie, préserve les invariants et assure l'intégrité des données. Ce modèle réduit la complexité et améliore la cohérence du système [11].

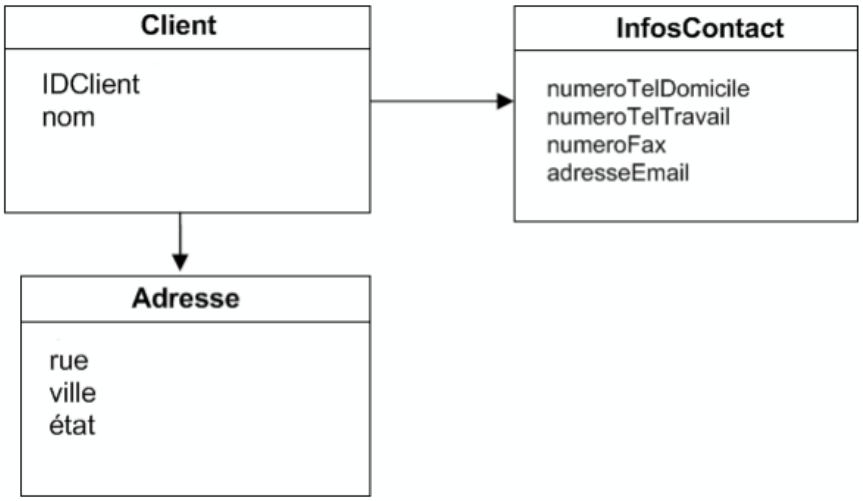

Figure II. 3 - Représentation des Agrégats [11]

4 Structurer une architecture microservices avec DDD

La structuration d'une architecture microservices à partir du Domain-Driven Design (DDD) repose sur l'identification précise des domaines métier, leur classification stratégique, et leur transformation en services indépendants. Cette section expose les étapes clés de cette démarche, depuis la compréhension du métier jusqu'au découpage en microservices cohérents, alignés sur les contextes délimités.

4.1 Identification des domaines métier

Un domaine métier représente une zone fonctionnelle ou une problématique spécifique à laquelle le système doit répondre. Dans le cadre du projet AgriCo, qui vise à connecter les coopératives agricoles via une plateforme numérique, plusieurs domaines clés ont été identifiés :

- Domaine des produits : gestion des articles agricoles (référencement, stock, caractéristiques).
- Domaine des livraisons : planification et suivi des tournées de livraison.
- Domaine des utilisateurs : gestion des profils, rôles et droits d'accès (producteur, ouvrier, commerçant, livreur, administrateur).
- Domaine des annonces et commandes : création d'annonces de vente, gestion des offres et commandes.
- Cette identification permet de clarifier les responsabilités fonctionnelles du système et d'orienter la séparation en sous-domaines.

4.2 Classification des sous-domaines

Une fois les domaines identifiés, il est essentiel de les classifier selon leur importance stratégique. Le DDD distingue trois types de sous-domaines :

- Core Domain (Domaine central) : représente le cœur du système, celui qui apporte une réelle valeur métier. Exemple : gestion des produits et des transactions commerciales dans AgriCo.
- Supporting Subdomains (Domaines de soutien) : nécessaires pour accompagner le cœur métier mais non différenciateurs. Exemple : modules de paiement ou de logistique.
- Generic Subdomains (Domaines génériques) : fonctionnalités standards communes à de nombreux systèmes, souvent réutilisables. Exemple : authentification, gestion des notifications.

Cette hiérarchisation permet de concentrer les efforts de conception et de développement sur les parties les plus stratégiques.

4.3 Collaboration avec les experts métier

Le DDD préconise une collaboration étroite entre les développeurs et les experts métier afin de construire une compréhension partagée du domaine. Cette collaboration repose sur l'utilisation d'un langage omniprésent (Ubiquitous Language), qui est un vocabulaire commun utilisé dans les échanges, la documentation et le code.

Par exemple, dans Agrico, les termes comme opérateur, réservation, matériel, ou demande de transport doivent être définis précisément et utilisés de manière cohérente à travers toute l'architecture logicielle.

4.4 Bounded Contexts (contextes délimités)

Dans une architecture microservices, ces contextes délimités correspondent souvent aux unités fonctionnelles autonomes, et sont de fait des candidats naturels pour devenir des microservices.

Exemples de Bounded Contexts dans Agrico :

Bounded Context	Responsabilités principales
Authentication	Inscription des utilisateurs, récupération de mot de passe, prévention des accès non autorisés.
Matériels Agricoles	Ajout, modification et suppression de matériels agricoles, Consultation des matériels agricoles.

Table II. 1 -Domaines métier identifiés dans l'application Agrico

Cette séparation garantit une meilleure compréhension des règles métier à l'intérieur de chaque contexte, tout en évitant les ambiguïtés dues aux termes à double sens.

4.5 Mapping des Bounded Contexts vers les microservices

Une fois les Bounded Contexts identifiés, la prochaine étape consiste à les mapper vers des microservices autonomes. Chaque microservice devient responsable d'un contexte métier spécifique, avec un haut niveau de cohésion interne et une faible dépendance externe.

Chaque microservice doit :

- Posséder son modèle métier propre, incluant ses entités, règles métier et cas d'usage.
- Disposer de sa base de données dédiée, pour garantir l'autonomie.
- Être déployé indépendamment des autres services.
- Exposer ses fonctionnalités via une interface de communication (API REST, messages, etc.).

Illustration dans Agrico :

Microservice	**Bounded Context associé**	**Exemple de responsabilités**
Product-service	Authentification	Création de produits, mise à jour des quantités, gestion des catégories
Marketplace-service	Matériels Agricoles	Publication d'annonces, définition des prix, consultation des offres

Table II. 2 -Cartographie des microservices par Bounded Context dans AgriCo

Cette cartographie permet d'aligner la structure technique (microservices) avec la réalité métier (contextes délimités), ce qui favorise la robustesse, l'évolutivité et la clarté du système.

5 Architecture Orientée Événements – Version Améliorée et Simplifiée

L'intégration de l'architecture orientée événements (EDA) dans une approche Domain-Driven Design (DDD) permet de renforcer le découplage, la réactivité et l'évolutivité des microservices. Tandis que le DDD structure le système autour des concepts métier, l'EDA assure une communication asynchrone entre les services, en se basant sur la publication et la consommation d'événements.

Cette complémentarité rend possible une modélisation fidèle du domaine tout en assurant une coordination souple et scalable entre les différents Bounded Contexts. Cette section présente les principes de l'EDA, son rôle dans une architecture distribuée, ses modes de communication, ainsi que les technologies utilisées pour sa mise en œuvre.

5.1 Présentation de l'architecture orientée événements

L'architecture orientée événements (EDA) est, en fait, axée sur l'utilisation des événements comme mécanisme central pour faciliter les interactions entre les services. Cette méthode améliore le découplage de telle manière que chaque service peut répondre aux événements publiés par d'autres services sans avoir besoin de dépendance directe à leur logique opérationnelle ou à leurs interfaces [12]. Par conséquent, les modifications apportées à un seul

service n'affectent pas directement les autres, ce qui améliore la flexibilité et permet une plus grande évolutivité.

Dans les systèmes réels, les architectures sont davantage une combinaison de communication asynchrone via des événements et d'invocation synchrone via des API [12]. L'EDA introduit également une nouvelle approche du partage de données non seulement en les demandant explicitement, mais aussi en diffusant des événements en temps réel qui signifient des changements d'état. Cette séquence d'événements permet à chaque service de réaliser l'évolution temporelle de l'information, et donc de combiner les informations provenant de diverses sources et de les présenter de la manière la plus appropriée pour son domaine spécifique et ses exigences de requête.

5.2 Avantages de l'EDA dans une architecture microservices

L'architecture orientée événements (EDA) est particulièrement adaptée aux systèmes basés sur les microservices. Elle offre plusieurs avantages essentiels qui permettent de répondre aux exigences modernes en matière de scalabilité, flexibilité et résilience.

- **Découplage entre les services**

L'un des principaux atouts de l'EDA est le faible couplage entre les composants. Les services communiquent en publiant et en écoutant des événements, sans se connaître mutuellement. Cela permet à chaque service d'évoluer indépendamment, de manière plus simple et plus rapide [12].

- **Scalabilité horizontale**

Les microservices peuvent être mis à l'échelle individuellement, selon leur charge. Grâce à la communication asynchrone, un service très sollicité peut être dupliqué sans affecter les autres. L'EDA facilite ainsi la montée en charge des systèmes distribués [12].

- **Réactivité et temps réel**

Dès qu'un événement est émis (ex. : *commande validée*), plusieurs services peuvent réagir instantanément : mise à jour du stock, envoi d'une notification, planification d'une livraison. Cela permet de concevoir des systèmes réactifs et fluides, proches du temps réel [12].

- **Évolutivité fonctionnelle**

Il est possible d'ajouter de nouveaux services sans modifier ceux qui existent déjà. Il suffit qu'un nouveau service s'abonne à un événement existant pour réagir à ce qui se passe dans le système. Cela rend l'architecture très extensible [12].

- **Résilience et tolérance aux pannes**

Comme les messages sont souvent enregistrés dans des systèmes de type journal (log), un service peut reprendre son traitement là où il s'est arrêté après une panne. Cela augmente la robustesse globale du système [12].

5.3 Fonctionnement de la communication par événements

La communication par événements permet aux services de transmettre des informations de manière asynchrone et découplée. Elle constitue un pilier central de l'EDA en assurant la réactivité et la tolérance aux pannes des microservices.

5.3.1 Les rôles

Au cœur des architectures orientées événements se trouvent les producteurs et les consommateurs, deux rôles complémentaires qui facilitent la communication asynchrone et découplée entre les services.

Un producteur produit des événements qui notifient des faits survenus dans le monde des affaires (par exemple, "commande créée", "stock mis à jour", etc.). Ces événements sont envoyés à un système de diffusion généralement sous la forme d'un journal structuré, c'est-à-dire une séquence de messages immuables ajoutée de manière séquentielle. Cette approche permet un traitement d'écriture efficace car les opérations d'ajout sont toujours séquentielles [13].

Le consommateur, de son côté, interprète ces événements à son propre rythme, en conservant un décalage (position) dans le journal qui lui permet de reprendre la lecture en cas d'interruption, de sorte que des doublons ou des pertes ne peuvent pas se produire [13].Le système est donc intrinsèquement tolérant aux pannes car une défaillance locale n'impacte jamais la source de données ni les autres consommateurs.

En conclusion, les producteurs et les consommateurs sont des parties intégrantes d'une architecture orientée événements, facilitant une communication asynchrone et découplée entre les services.

5.3.2 Les types de messages

- **Les commandes** sont des impératifs qui sont envoyés à un service pour effectuer une action. Les commandes peuvent être annulées lorsque la validation échoue. Par exemple, CreateOrderCommand est une intention de l'utilisateur de commencer un changement de domaine. Les commandes ont tendance à aller à un seul consommateur, et même si elles sont généralement envoyées via des courtiers de messages, elles peuvent être envoyées par des

canaux synchrones tels que HTTP [12].

- **Les événements** indiquent que quelque chose s'est produit dans un domaine. Les événements sont des faits et ne peuvent pas être contestés car ils annoncent une action accomplie (par exemple, OrderCreatedEvent, ProductBrandChangedEvent). Les événements sont publiés à plusieurs consommateurs, ce qui facilite l'évolution du système en introduisant de nouveaux consommateurs sans changer les producteurs [12].
- **Les documents** représentent l'état complet d'une entité et sont généralement produits lorsque des transformations se produisent au sein de l'entité. Contrairement aux événements, qui ne transmettent que les détails des modifications appliquées, les documents transmettent l'état complet (par exemple, OrderDocument). Cela permet aux consommateurs de recréer ou de synchroniser leur propre représentation locale de l'état de l'entité [12].
- **Les requêtes** sont des demandes d'information à un service, exécutées de manière à ne pas modifier l'état du système. Leur fonction principale est la récupération de données.Par exemple, la requête GetOrderById est utilisée pour récupérer l'état d'une commande à l'heure actuelle.Les requêtes sont généralement présentées via des canaux synchrones (par exemple, REST ou GraphQL), et leur succès dépend de la disponibilité en temps opportun du service ciblé [12].

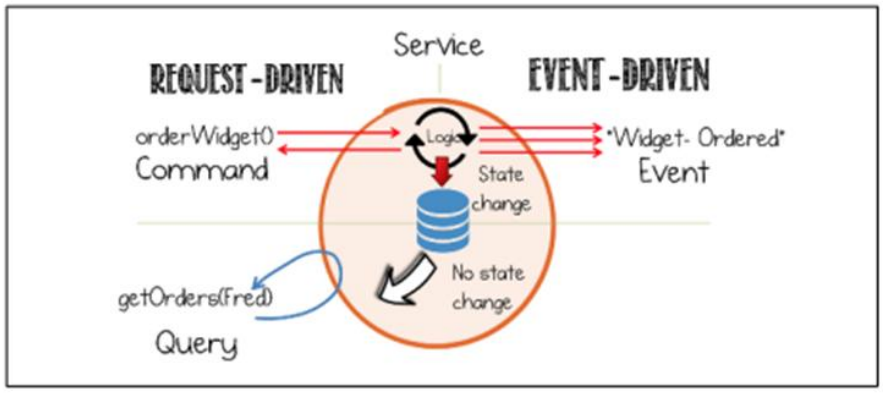

Figure II. 4 - Un résumé visuel des commandes, des événements et des requêtes [12]

5.4 Manières de faire circuler les événements

Les modes de diffusion des événements définissent comment les messages sont transmis aux services consommateurs. Cette section présente les deux approches principales utilisées en EDA : Pub/Sub et Event Streaming, chacune adaptée à des besoins spécifiques en termes de réactivité, de persistance et de scalabilité.

5.4.1 Modèle Pub/Sub (Publication – Abonnement)

L'infrastructure de messagerie publish-subscribe suit les abonnements. Lorsqu'un événement est publié, il envoie l'événement à chaque abonné. Un événement ne peut pas être rejoué après avoir été reçu, et les nouveaux abonnés ne voient pas l'événement. Utilise ça pour la définition des pub/sub [14].

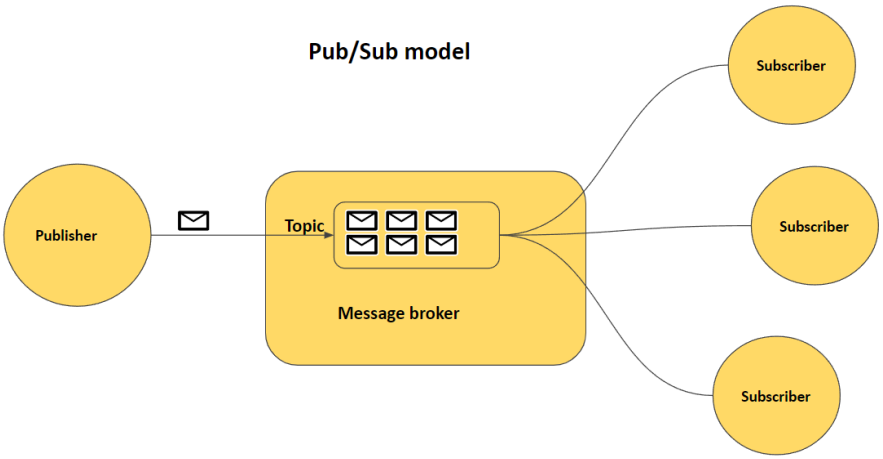

Figure II. 5 - Illustration du modèle Pub/Sub

La figure ci-dessus illustre le modèle Publish/Subscribe (Pub/Sub), utilisé pour la communication asynchrone entre services dans une architecture orientée événements.

Dans ce modèle, un Publisher (producteur) publie un message ou un événement vers un Message Broker, via un topic (canal de diffusion). Le message broker est chargé de transmettre automatiquement cet événement à tous les Subscribers (abonnés) intéressés par ce type de message.

Chaque abonné reçoit une copie du message, ce qui permet à plusieurs services de réagir simultanément à un même événement, sans que le producteur ait besoin de les connaître.

5.4.2 Event Streaming

Les événements sont écrits dans un journal. Les événements sont strictement ordonnés au sein d'une partition et sont durables. Les clients ne s'abonnent pas au flux. Au lieu de cela, un client peut lire à partir de n'importe quelle partie du flux. Le client est responsable de faire avancer sa position dans le flux. Cela signifie qu'un client peut rejoindre à tout moment et peut rejouer les événements [14].

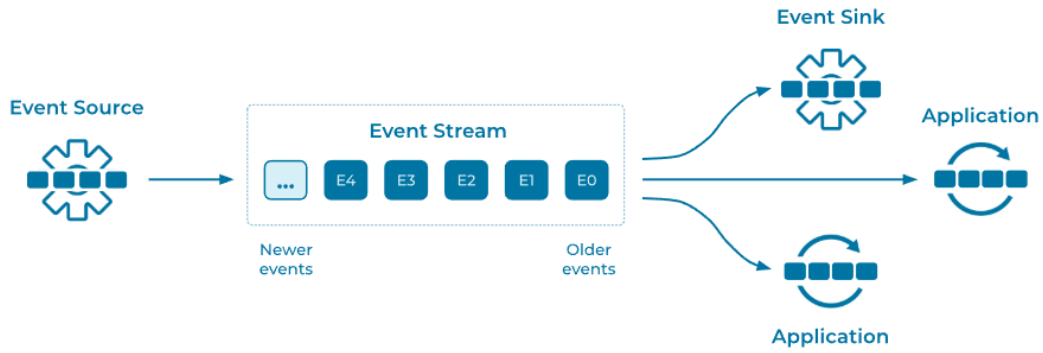

Figure II. 6 -Illustration du modèle Event Streaming

Cette figure représente le fonctionnement du modèle Event Streaming, Le processus fonctionne comme suit:

- **Event Source (Source d'événements)** : Un service ou système produit une série d'événements (par exemple, *E0, E1, E2, ... E4*), qui sont envoyés au flux.
- **Event Stream (Flux d'événements)** : Contrairement au modèle Pub/Sub, chaque événement est enregistré durablement dans un journal (ou log). Ce flux est ordonné chronologiquement, et chaque événement peut être relu autant de fois que nécessaire. Les nouveaux événements (comme *E4*) sont ajoutés à la fin du flux.
- **Event Sink (Consommateur d'événements)** : Les applications consommatrices peuvent lire les événements :
  - Soit en temps réel, dès qu'ils arrivent.
  - Soit en rejouant les événements anciens, par exemple après une panne ou pour reconstruire un état.

5.5 Mécanismes de coordination des services basés sur les événements

La coordination entre services dans une architecture événementielle peut être centralisée ou décentralisée. Cette section présente les deux approches principales chorégraphie et orchestration et explique comment elles organisent l'interaction entre microservices.

5.5.1 Chorégraphie (décentralisée)

La chorégraphie, dans les architectures de microservices, désigne un modèle décentralisé d'interaction où chaque service réagit de manière autonome aux événements reçus, sans coordination explicite ni gestion centralisée. Ce modèle repose sur un découplage fort entre les producteurs et les consommateurs d'événements, permettant à chaque service de fonctionner indépendamment et de réagir en temps réel à son environnement [15].

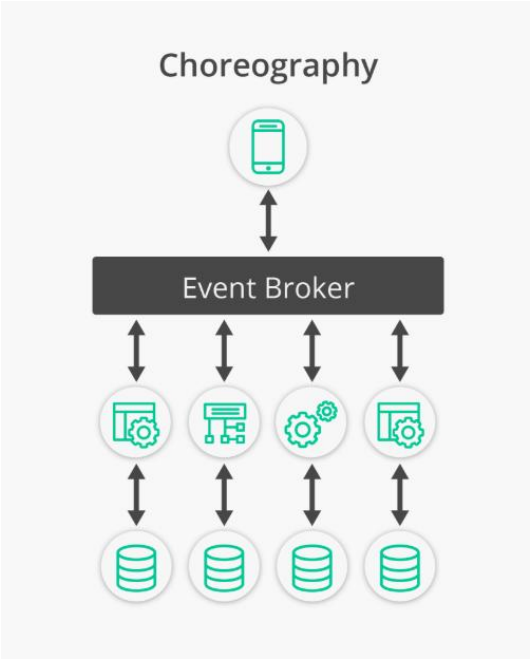

Figure II. 7 -Modèle Chorégraphie [15]

Cette figure montre une architecture où les services réagissent de manière autonome aux événements diffusés via un event broker. Il n'y a pas de coordinateur central : chaque service s'abonne aux événements qui l'intéressent et agit en conséquence.

- **Avantages :**

  Faible couplage entre services

  Meilleure évolutivité et flexibilité

- **Inconvénients :**

  Flux métier plus difficile à tracer

  Complexité croissante en cas de dépendances multiples

5.5.2 Orchestration (centralisée)

Dans les architectures orientées événements, l'orchestration désigne un modèle centralisé d'interaction dans lequel un microservice principal, appelé orchestrateur, contrôle l'exécution du flux de travail métier en émettant des commandes aux microservices participants et en recevant leurs réponses. L'orchestrateur contient toute la logique du processus, détermine la séquence des tâches et supervise l'état d'avancement global [15].

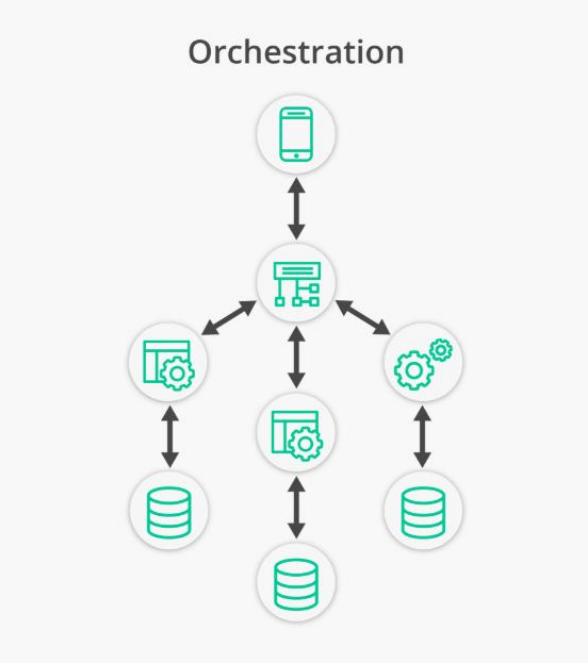

Figure II. 8 -Modèle d'orchestration [15]

Cette figure illustre une architecture où un orchestrateur central coordonne les interactions entre services. Chaque service exécute une tâche spécifique et communique uniquement via l'orchestrateur, qui pilote l'ensemble du processus métier.

- **Avantages :**

  Logique centralisée, plus facile à contrôler et à maintenir

  Séquencement clair des opérations

- **Inconvénients :**

  Forte dépendance à l'orchestrateur

  Risque de point de défaillance unique et de surcharge

5.6 Schéma d'organisation des messages

5.6.1 Topologie Médiateur

La topologie de médiateur est utile pour les événements à plusieurs étapes et nécessite une orchestration pour les traiter. Elle se compose de quatre composants principaux de l'architecture : des files d'attente d'événements, un médiateur d'événements, des canaux d'événements et des processeurs d'événements. Le flux d'événements commence par un client envoyant un événement à une file d'attente d'événements, qui est ensuite orchestrée par un médiateur d'événements. Les processeurs d'événements écoutent les canaux d'événements et exécutent une logique métier spécifique pour traiter l'événement [16].

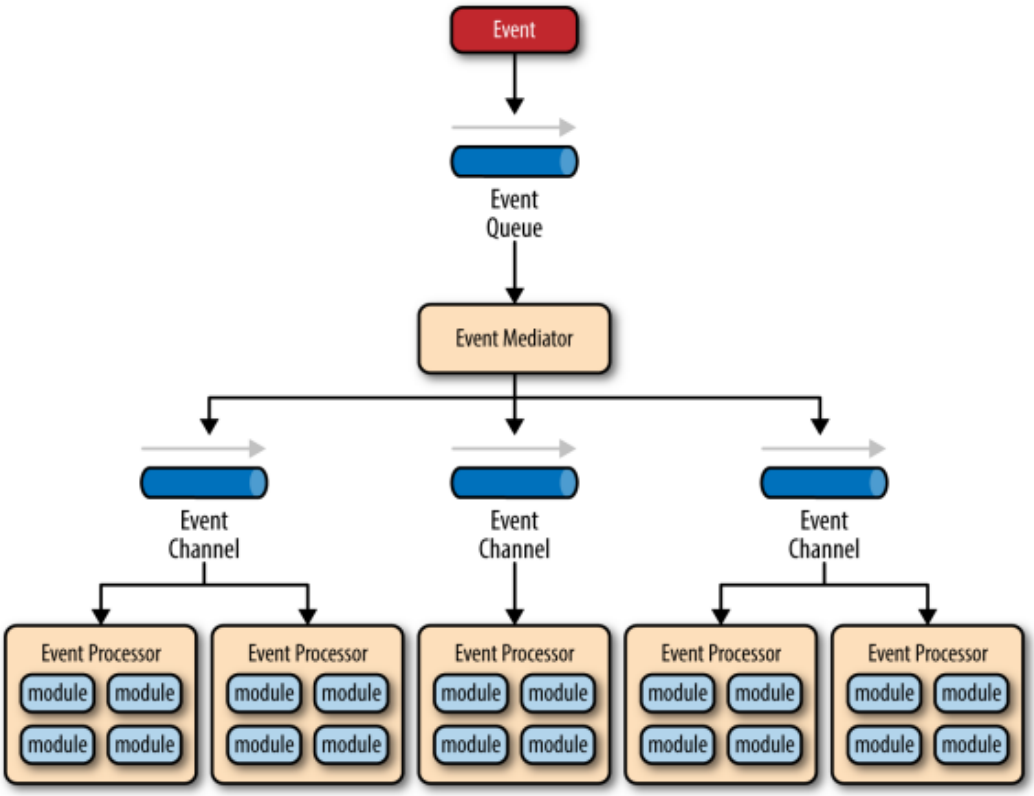

Figure II. 9 -Topologie de médiateur d'architecture orientée événements [16]

5.6.2 Topologie Courtier

La topologie de courtier est un type de système de traitement d'événements où le flux de messages est distribué entre les composants de traitement d'événements de manière chaînée via un courtier de messages léger. Il est utile pour des flux de traitement d'événements simples sans orchestration centrale des événements. Dans la topologie du courtier, il existe deux principaux types de composants d'architecture : un composant courtier et un composant de traitement des événements. Le composant du courtier peut être centralisé ou fédéré et contient tous les canaux d'événements utilisés dans le flux d'événements. La chaîne d'événements continue à travers le système jusqu'à ce qu'aucun autre événement ne soit publié pour l'événement initiateur [16].

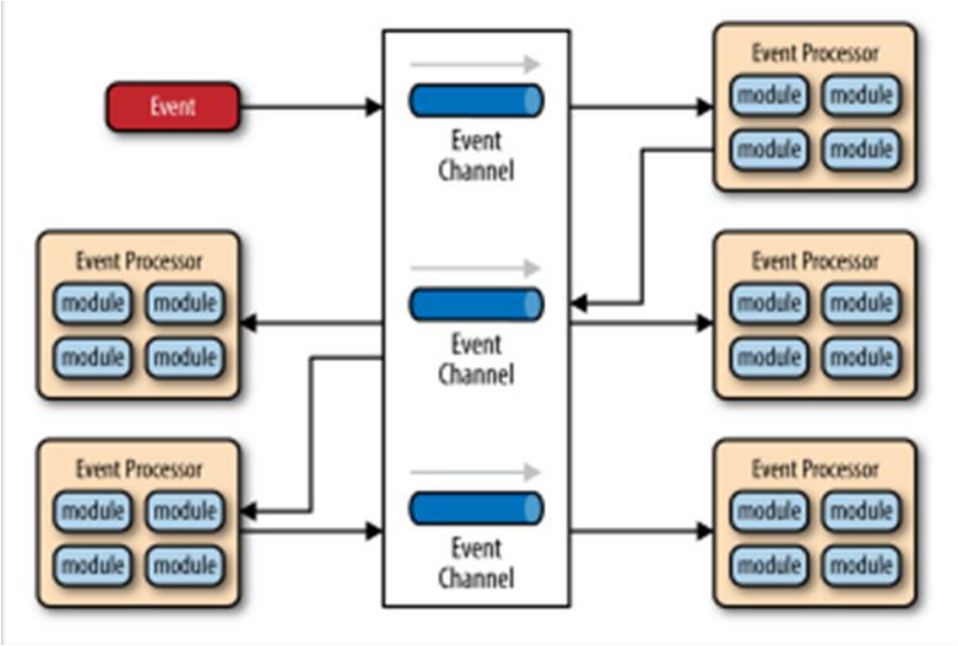

Figure II. 10 -Topologie de courtier d'architecture orientée événements [16]

5.7 Outils utilisés pour l'EDA

5.7.1 Outils de transmission des messages : les brokers

Dans une architecture orientée événements, la communication entre services repose sur des outils appelés brokers de messages. Ces composants assurent le transport, la diffusion et parfois la persistance des événements entre les différents services du système.
Parmi les brokers les plus utilisés, on distingue :

- **Apache Kafka** : spécialisé dans le streaming d'événements à grande échelle, Kafka permet de stocker les messages dans un journal persistant, de les rejouer à volonté, et de traiter de gros volumes de données en temps réel. Il est adapté aux systèmes hautement distribués et scalables.
- **RabbitMQ** : basé sur le modèle Publish/Subscribe, RabbitMQ est un broker de messagerie plus classique. Il gère des files d'attente et permet un routage flexible des messages. Il est particulièrement bien adapté aux systèmes légers ou aux flux événementiels simples.
- Ces brokers assurent plusieurs garanties essentielles :
- Une transmission fiable des messages entre services, même en cas de charge élevée.
- Une tolérance aux pannes, grâce à la persistance des messages et à la gestion des reprises.
- Une sécurité des échanges, via des mécanismes de chiffrement, d'authentification et de contrôle d'accès.
- Ainsi, Kafka et RabbitMQ jouent un rôle clé dans la mise en œuvre efficace d'une architecture orientée événements.

6 Conclusion

L'association du Domain-Driven Design (DDD) et de l'architecture orientée événements (EDA) représente une démarche puissante pour concevoir des systèmes distribués à la fois souples, cohérents et évolutifs. Le DDD permet de structurer le système autour des réalités métier grâce à des modèles explicites, des contextes délimités bien définis, et une séparation claire des responsabilités. Cette rigueur conceptuelle est essentielle pour éviter les dérives dans des environnements complexes comme ceux des microservices.

De son côté, l'EDA complète cette approche en assurant une communication asynchrone et découplée, ce qui permet aux services de réagir dynamiquement aux événements sans dépendance directe. Elle favorise la résilience, la scalabilité et l'extensibilité du système, tout en réduisant les points de défaillance. En combinant DDD et EDA, les architectes logiciels peuvent ainsi concevoir des plateformes modernes, robustes et alignées avec les besoins métier réels, comme le montre le cas de la plateforme AGRICO.DZ, qui repose sur ces principes pour structurer son fonctionnement interne et sa logique métier.

Chapitre III – systèmes de recommandation

1 Introduction

À l'ère du numérique, les utilisateurs sont confrontés à une abondance de choix dans pratiquement tous les domaines : films à regarder, produits à acheter, articles à lire, musiques à écouter. Cette surcharge informationnelle rend la prise de décision difficile et accroît le besoin de systèmes capables de filtrer, trier et personnaliser les contenus proposés. C'est dans ce contexte que les systèmes de recommandation ont émergé comme des outils essentiels, visant à orienter les utilisateurs vers des éléments pertinents en se basant sur leurs préférences, leurs comportements passés ou ceux d'utilisateurs similaires.

Présents sur des plateformes majeures telles qu'Amazon, Netflix ou Spotify, ces systèmes jouent un rôle central dans l'expérience utilisateur, en facilitant l'exploration de contenus adaptés à leurs goûts. Différentes approches existent, notamment le filtrage collaboratif, la recommandation basée sur le contenu, et les systèmes hybrides, chacun avec ses avantages et ses limites.

Ce rapport a pour objectif d'explorer ces différentes approches, de comprendre leur fonctionnement, leurs mécanismes internes et leurs domaines d'application, tout en mettant en lumière les avancées récentes et les stratégies d'hybridation.

2 Définition des systèmes de recommandation

Les systèmes de recommandation sont devenus des outils essentiels dans l'environnement numérique moderne. Cette section présente leur fonctionnement de base, leurs objectifs, ainsi que les domaines dans lesquels ils sont appliqués, tout en soulignant les défis liés à leur utilisation éthique et à leur efficacité.

2.1 Qu'est-ce qu'un système de recommandation ?

Un système de recommandation est un ensemble de techniques et d'outils conçus pour proposer des éléments pertinents à un utilisateur spécifique, fondés sur ses intérêts, son comportement précédent ou les caractéristiques similaires d'autres utilisateurs. Il est destiné à reproduire le processus naturel de recommandation personnalisée, tel que l'on pourrait l'obtenir dans la vie quotidienne d'un vendeur, d'un pair ou d'un expert [17].

Dans la vie quotidienne, de telles recommandations prennent généralement la forme de transactions simples : le conseil d'un épicier d'acheter un bon melon de saison ou le conseil d'un vendeur de disques d'acheter un CD après avoir déterminé vos goûts. Celles-ci sont généralement perçues comme utiles et sincères parce qu'elles s'inscrivent dans un cadre de confiance [17].

Dans le contexte en ligne, ces interactions sont converties en algorithmes capables de traiter

d'énormes quantités de données, en particulier des données comportementales. Cela permet la génération de recommandations basées non seulement sur les données précédentes d'un utilisateur, mais aussi sur les likes d'autres utilisateurs ayant des profils comparables ou sur la popularité générale de certains contenus. Par conséquent, la personnalisation à grande échelle est réalisable, et elle devient un élément clé des sites web en ligne [17].

2.2 Enjeux et applications des systèmes de recommandation

Les systèmes de recommandation sont omniprésents en ligne car ils résolvent un problème fondamental : comment aider les utilisateurs à choisir parmi un nombre écrasant d'options. Cette exigence est particulièrement prononcée dans des contextes caractérisés par un excès ou une offre presque infinie, comme les sites web tels qu'Amazon, Netflix ou Spotify. Dans ce sens, la valeur d'une recommandation sert d'outil stratégique pour attirer l'attention de l'utilisateur et améliorer la satisfaction ou les ventes [17].

L'une des questions les plus importantes concerne le concept de la longue traîne (long tail), qui a été inventé par Chris Anderson. Dans un espace de vente physique, seuls les produits favoris peuvent avoir de l'espace sur les étagères. Dans un espace numérique, cependant, il est possible d'offrir une quantité illimitée de contenu ou de produits, même ceux qui peuvent ne pas avoir d'attrait de masse. Les systèmes de recommandation jouent un rôle crucial en mettant ces articles de niche en avant en les connectant avec les bons utilisateurs [17]

Au-delà du commerce électronique, les applications des systèmes de recommandation sont omniprésentes dans de nombreux domaines : musique, films, articles de presse, réseaux sociaux, sites éducatifs, et même sites de rencontre. Netflix, par exemple, recommande des films en fonction des goûts personnels, et Amazon suggère des articles complémentaires à ceux actuellement regardés ou précédemment achetés. Les recommandations peuvent être non personnalisées (par exemple, les meilleures ventes), semi-personnalisées (par exemple, par région ou profil démographique), ou entièrement personnalisées [17]

Enfin, l'un des plus grands défis est de distinguer les recommandations des publicités. Les recommandations sont au bénéfice de l'utilisateur, tandis que les publicités sont principalement au bénéfice de l'annonceur. Bien que la distinction puisse parfois être subtile pour l'utilisateur, elle est essentielle pour le développement éthique des systèmes de recommandation [17].

3 Systèmes de recommandation collaboratifs

Les systèmes de recommandation collaboratifs constituent une approche incontournable dans le domaine des recommandations personnalisées. Leur force réside dans leur capacité à exploiter les préférences partagées entre utilisateurs ou objets, sans nécessiter de connaissance préalable sur les contenus à recommander. Cette méthode repose uniquement sur les interactions passées (notes, clics, achats), ce qui la rend particulièrement adaptée aux environnements riches en données.

La présente section explore les deux principales techniques de filtrage collaboratif : la recommandation basée sur les utilisateurs et celle basée sur les éléments, en détaillant leur fonctionnement à travers des exemples, le calcul des similarités, et la prédiction de notes.

3.1 Principe de fonctionnement

Le principe de fonctionnement du filtrage collaboratif repose sur l'idée que des utilisateurs ayant eu des comportements similaires dans le passé continueront à avoir des préférences proches dans le futur. Ce chapitre introduit deux méthodes classiques fondées sur la notion de voisinage : la recommandation basée sur les utilisateurs (user-based), qui identifie des utilisateurs semblables pour proposer des items, et la recommandation basée sur les éléments (item-based), qui se focalise sur les similarités entre les objets eux-mêmes. Chaque méthode sera illustrée par des exemples concrets, suivis du calcul de similarité et de la prédiction d'une note, afin d'en illustrer le fonctionnement pratique [18].

3.1.1 Recommandation du voisin le plus proche basée sur l'utilisateur

Ce type de recommandation par les utilisateurs les plus proches est l'une des premières méthodes appliquées dans les systèmes de recommandation collaborative. L'idée est simple : étant donné une base de données d'évaluations et l'ID d'un utilisateur actif, nous aimerions trouver d'autres utilisateurs (les soi-disant "voisins" ou "pairs") qui ont partagé des préférences similaires dans le passé avec l'utilisateur actif [18].

Le concept est donc de prendre les évaluations fournies par ces utilisateurs similaires et d'essayer de déterminer l'évaluation que l'utilisateur actif est susceptible de donner à un produit qu'il n'a pas encore consulté. Il y a deux grandes hypothèses ici :

- Les humains qui ont eu des goûts similaires dans le passé continueront à partager des goûts similaires à l'avenir.
- Les intérêts des utilisateurs sont statiques et ne changent pas.

Prenons le tableau simplifié suivant dans lequel Alice est l'utilisateur actuel et les autres utilisateurs sont des pairs :

	Item 1	**Item 2**	**Item 3**	**Item 4**	**Item 5**
Alice	5	3	4	4	?
User 1	3	1	2	3	3
User 2	4	3	4	3	5
User 3	3	3	1	5	4
User 4	1	5	5	2	1

Table III 1Base de données des évaluations pour la recommandation collaborative [18].

Alice a noté plusieurs items, mais pas encore l'Item5. L'objectif est de prédire si Alice aimera cet Item5 en se basant sur les notes données par des utilisateurs ayant des goûts similaires.

3.1.1.1 Calcul de la similarité

Pour mesurer la similarité entre Alice et les autres utilisateurs, on utilise souvent le coefficient de corrélation de Pearson. Cette mesure prend en compte le fait que chaque utilisateur peut utiliser différemment l'échelle de notation (certains ont tendance à mettre des notes globalement hautes, d'autres basses), en comparant les écarts par rapport à la moyenne de chaque utilisateur [18].

Par exemple, la similarité entre Alice et User1 est calculée comme suit (avec r_a la moyenne des notes d'Alice et r_b celle de User1) [18] :

$$sim(a,b) = \frac{\sum_{p\in P}(r_{a,p} - \overline{r_a})(r_{b,p} - \overline{r_b})}{\sqrt{\sum_{p\in P}(r_{a,p} - \overline{r_a})^2}\sqrt{\sum_{p\in P}(r_{b,p} - \overline{r_b})^2}} = 0.85$$

De même, on trouve pour User2, User3, User4 respectivement 0.70, 0.00, et -0.79.

On retient donc User1 et User2 comme voisins les plus proches d'Alice.

3.1.1.2 Prédiction de la note pour l'Item5

Pour prédire la note que donnera Alice à l'Item5, on utilise la formule pondérée suivante [18] :

$$pred(a,p) = \overline{r_a} + \frac{\sum_{b \in N} sim(a,b) \times (r_{b,p} - \overline{r_b})}{\sum_{b \in N} sim(a,b)}$$

Avec N l'ensemble des voisins retenus (ici User1 et User2).

En appliquant la formule :

$$pred(Alice, Item5) = 4 + \frac{0.85 \times (3 - 2.4) + 0.70 \times (5 - 3.8)}{0.85 + 0.70} = 4.8$$

Cette prédiction indique qu'Alice devrait beaucoup aimer l'Item5, il serait donc pertinent de le lui recommander.

3.1.1.3 Remarques

- Dans la pratique, les bases de données sont bien plus grandes et la matrice des notes très sparse (peu remplie), ce qui complexifie les calculs.
- La taille du voisinage k est importante : trop petit, la prédiction manque de données; trop grand, on introduit du bruit.
- Des améliorations existent, par exemple l'utilisation de pondérations pour donner plus d'importance aux items controversés ou aux voisins avec plus d'items communs.
- Le choix de la mesure de similarité peut varier, mais Pearson est souvent très efficace pour les systèmes basés sur les utilisateurs.

3.1.2 Recommandation du voisin le plus proche basée sur l'élément

La méthode de recommandation par les plus proches voisins basée sur les articles est une sous-classe de systèmes de recommandation collaborative qui repose sur la similarité des articles plutôt que sur la similarité entre les utilisateurs. La méthode a été suggérée comme un complément aux systèmes basés sur les utilisateurs, spécifiquement destinée à améliorer la scalabilité et la stabilité dans des environnements à grande échelle, comme ceux rencontrés par des entreprises telles qu'Amazon ou Netflix [18].

Le principe sous-jacent est le suivant : pour recommander un article à un utilisateur, le système effectue d'abord une recherche des articles les plus similaires à ceux qui ont été positivement évalués par l'utilisateur auparavant. Ensuite, il recommande ces articles similaires en se basant sur la similarité entre les articles et non sur la similarité entre les utilisateurs.

Cette méthode repose sur l'hypothèse :Si l'utilisateur a aimé un produit spécifique, il aimera probablement d'autres produits de même type.

3.1.2.1 Exemple

Prenons la même base de données que dans l'exemple précédent Tableau .

L'objectif est ici de prédire la note qu'Alice attribuerait à l'Item5, non pas en regardant les utilisateurs similaires à elle, mais en comparant Item5 aux autres items qu'elle a déjà évalués.

3.1.2.2 Calcul de la similarité entre items

Comme pour les utilisateurs, on peut mesurer la similarité entre deux items à l'aide de la corrélation de Pearson, ou bien du cosinus de similarité, qui est souvent préféré dans cette approche car il est moins sensible à la variance des notes entre les utilisateurs [18].

The similarity between two items a and b – viewed as the corresponding rating vectors $\vec{a}$ and $\vec{b}$ – is formally defined as follows [18]:

$$sim(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| \times |\vec{b}|}$$

Le symbole · est le produit scalaire de vecteurs. $|\vec{a}|$ est la longueur euclidienne de le vecteur, qui est défini comme la racine carrée du produit scalaire du vecteur avec lui-même [18].

Par exemple, pour calculer la similarité entre Item5 et Item1, on considère les utilisateurs ayant noté les deux items (ici User2, User3, User4), et on applique la formule du cosinus de similarité :

$$sim(I5, I1) = \frac{3 \times 3 + 5 \times 4 + 4 \times 3 + 1 \times 1}{\sqrt{3^2+5^2+4^2+1^2} + \sqrt{3^2+4^2+3^2+1^2}} = 0.99$$

3.1.2.3 Prédiction de la note pour l'Item5

Pour prédire la note que donnerait Alice à l'Item5, on utilise une moyenne pondérée des notes qu'elle a déjà données aux items similaires, pondérées par leur similarité avec Item5:

$$pred(u,p) = \frac{\sum_{i \in N} sim(i,p) \cdot r_{u,i}}{\sum_{i \in N} sum(i,p)}$$

En appliquant la formule avec les données d'Alice :

$$pred(Alice, Item5) = \frac{0.92 \times 5 + 0.85 \times 4 + 0.76 \times 4 + 0.30 \times 3}{0.92 + 0.85 + 0.76 + 0.30} \approx 4.34$$

La prédiction indique qu'Alice devrait apprécier l'Item5, ce qui justifie sa recommandation.

3.1.2.4 Remarques

- Cette approche est plus stable que la méthode centrée utilisateur, car les similarités entre items changent peu dans le temps (les items étant statiques), contrairement aux préférences utilisateurs qui évoluent souvent.
- Elle est aussi plus adaptée aux systèmes en ligne de grande taille, car il est plus efficace de pré-calculer les similarités entre les items (dont le nombre est souvent plus limité que celui des utilisateurs).
- Le principal inconvénient est que cette méthode ne fonctionne pas bien pour les items très peu notés, car il devient alors difficile de calculer une similarité fiable (effet du cold-start pour les items).
- Dans la pratique, les moteurs de recommandation basés sur les items utilisent souvent des techniques d'optimisation comme le filtrage de similarité top-N, qui ne conserve que les items les plus proches pour chaque item cible.

4 Systèmes de recommandation basés sur le contenu

La recommandation basée sur le contenu repose sur l'analyse des caractéristiques des produits ainsi que des préférences passées de l'utilisateur afin de proposer des éléments similaires à ceux déjà appréciés. Contrairement au filtrage collaboratif, cette approche nécessite une description explicite des items (par exemple : genre d'un livre, acteurs d'un film) et un profil utilisateur fondé sur ses préférences. Elle permet ainsi de recommander des produits sans avoir besoin d'une grande communauté d'utilisateurs ou d'un historique de notations riche, ce qui la rend adaptée aux situations de démarrage à froid (cold-start) pour de nouveaux utilisateurs ou items [18].

Le système identifie les produits les plus proches du profil de l'utilisateur selon leurs attributs, ce qui permet, par exemple, de recommander un roman fantastique à un utilisateur qui a manifesté une préférence pour ce genre. Toutefois, cette approche suppose la disponibilité de données descriptives de qualité, ce qui peut être un défi, notamment lorsqu'il s'agit de préférences subjectives ou esthétiques. Des initiatives comme le Music Genome Project, qui annote manuellement les chansons avec plusieurs centaines de caractéristiques, montrent l'intérêt mais aussi le coût élevé de telles démarches [18].

Les systèmes de recommandation de contenus sont historiquement associés à la recommandation de documents textuels (articles, pages web), et leur fonctionnement repose souvent sur l'extraction automatique de mots-clés depuis des descriptions non structurées. Enfin, bien que certaines définitions incluent ces systèmes dans la catégorie des systèmes à

base de connaissances, la distinction retenue dans ce contexte repose sur l'exploitation directe des descripteurs d'items, sans modèle formel de relations objectifs-moyens comme dans les systèmes à base de connaissances [18]

4.1 Principe de fonctionnement

Le filtrage basé sur le contenu repose sur l'idée que l'on peut recommander à un utilisateur des éléments similaires à ceux qu'il a appréciés par le passé, en se basant sur les caractéristiques descriptives de ces éléments. Ce paradigme de recommandation repose donc sur deux concepts fondamentaux : la représentation du contenu et la mesure de similarité entre contenus [18].

4.2 Représentation des éléments

Chaque item (livre, film, produit, etc.) est décrit à l'aide d'un ensemble explicite de caractéristiques ou attributs tels que le genre, l'auteur, le prix, les mots-clés, etc. Par exemple, pour des livres, on peut stocker ces informations dans une base de données relationnelle, comme illustré dans le tableau ci-dessous :

Titre	Genre	Auteur	Type	Prix	Mots-clés
The Night of the Gun	Mémoire	David Carr	Broché	29,90 €	Presse, drogue, mémoires, New York
The Lace Reader	Fiction, Mystère	Brunonia Barry	Relié	49,90 €	Fiction contemporaine, détective, historique

Table III 2 -Représentation des éléments dans la base de données [18].

4.2.1 Profil utilisateur

Les préférences d'un utilisateur peuvent être capturées selon les mêmes dimensions que les éléments, soit en les demandant explicitement (par exemple, en indiquant les genres préférés ou une fourchette de prix), soit en les déduisant automatiquement à partir des évaluations d'éléments. Un exemple de profil utilisateur pourrait ressembler à ceci :

Genre préféré	Auteurs favoris	Type préféré	Prix moyen	Mots-clés préféré
Fiction, Suspense	Brunonia Barry, Ken Follett	Broché	25,65 €	détective,meurtre, New York

Table III 3 -Représentation Profil utilisateur dans la base de données [18].

4.2.2 Calcul de similarité

Pour effectuer des recommandations, le système compare les éléments non encore vus avec ceux que l'utilisateur a aimés. Cette comparaison peut s'appuyer sur des métriques simples (par exemple, présence ou absence d'un genre) ou plus élaborées. Une méthode courante pour mesurer la similarité entre deux ensembles de mots-clés est le coefficient de Dice, défini par la formule suivante [18] :

$$sim(b_i, b_j) = \frac{2 \times |keywords(b_i) \cap keywords(b_j)|}{|keywords(b_i)| + |keywords(b_j)|}$$

Cette mesure est adaptée aux attributs multivalués tels que les mots-clés

4.2.3 Modèle vectoriel et TF-IDF

Pour les documents textuels, la représentation des éléments se fait souvent dans un espace vectoriel où chaque document est un vecteur de poids TF-IDF :

- **TF (Term Frequency) :** fréquence d'apparition d'un mot dans un document, normalisée par la fréquence maximale des autres mots du document [18].

$$TF(i,j) = \frac{freq(i,j)}{maxothers(i,j)}$$

- **IDF (Inverse Document Frequency)** : mesure la rareté d'un mot dans l'ensemble des documents [18].

$$IDF(i) = log\frac{N}{n(i)}$$

- **TF-IDF** : produit des deux, servant à pondérer l'importance d'un mot dans un document donné [18].

$$TF - IDF(i,j) = TF(i,j) \times IDF(i)$$

Le résultat est un vecteur pondéré qui permet d'évaluer la similarité entre documents ou entre

le profil utilisateur et les items disponibles.

4.3 Avantages et limites

Pour affiner ce modèle, diverses techniques peuvent être utilisées :

- **Suppression des mots vides (stop words)** : suppression des mots fréquents et peu informatifs comme "le", "et", "de", etc.
- **Stemming** : réduction des mots à leur racine (ex. : "mange", "mangé", "manger" → "mang").
- **Sélection des caractéristiques** : ne conserver que les *n* mots les plus informatifs pour éviter le bruit (bruit sémantique).
- **Utilisation de phrases-clés** : des expressions comme "Nations Unies" sont plus significatives que les mots isolés.

Cependant, ces méthodes présentent des limites :

- **Contexte ignoré** : la simple présence d'un mot peut induire en erreur s'il apparaît dans un contexte négatif (ex. : un restaurant décrit comme "non adapté aux végétariens" pourrait être recommandé à un végétarien à cause du mot "végétarien").
- **Ambiguïté lexicale** : des mots ayant plusieurs significations (homonymes) peuvent fausser les recommandations.

En somme, le filtrage basé sur le contenu repose sur la capacité à représenter efficacement les éléments et les préférences utilisateurs par des vecteurs sémantiques, et à évaluer la similarité entre ces représentations. Ce mécanisme constitue un fondement solide pour des recommandations personnalisées, bien qu'il nécessite souvent des ajustements pour en améliorer la précision et la pertinence.

5 Systèmes de recommandation hybrides

5.1 Motivation de l'approche hybride

Les systèmes de recommandation jouent un rôle essentiel dans les plateformes modernes, en permettant d'orienter les utilisateurs vers les contenus ou services les plus pertinents. Cependant, chaque méthode classique de recommandation présente des limites. Le filtrage collaboratif souffre du problème du démarrage à froid, car il nécessite un volume important d'interactions utilisateur pour générer des recommandations fiables. Le filtrage basé sur le contenu, quant à lui, peut manquer de diversité, car il recommande souvent des éléments très similaires à ceux déjà consultés.

L'approche hybride vise à combiner plusieurs techniques de recommandation pour tirer parti de leurs avantages tout en réduisant leurs faiblesses. Dans une application comme *Agrico*, où

les utilisateurs peuvent réserver du matériel agricole, consulter des offres de travail, ou réserver des services de transport, les préférences peuvent être complexes et variées. L'approche hybride permet donc de proposer des suggestions plus personnalisées et pertinentes, en s'adaptant aux différents types d'interactions et de contenus disponibles.

5.2 Stratégies d'hybridation

Plusieurs stratégies permettent de combiner les systèmes de recommandation, notamment :

5.2.1 Le système monolithique

Le système hybride monolithique intègre plusieurs techniques de recommandation dans une structure unique et cohérente. Contrairement à une simple juxtaposition de modèles, ici, la combinaison des approches est effectuée à l'intérieur d'un seul algorithme ou moteur de recommandation.

Par exemple, si l'on sait qu'un utilisateur aime particulièrement la science-fiction, on peut introduire artificiellement des évaluations dans la matrice de notation, représentant un "utilisateur virtuel" partageant ces préférences. Cela permet de créer des liens entre des éléments qui, bien que similaires sur le plan du contenu, n'auraient pas été associés par le filtrage collaboratif pur. Cela améliore la connectivité du système et la qualité des recommandations, surtout dans les cas de sparsité des données.

Comme le montre la Figure 5.2, plusieurs les recommandateurs contribuent virtuellement parce que l'hybride utilise des données d'entrée supplémentaires qui sont spécifiques à un autre algorithme de recommandation, ou les données d'entrée sont augmenté par une technique et exploité factuellement par l'autre.

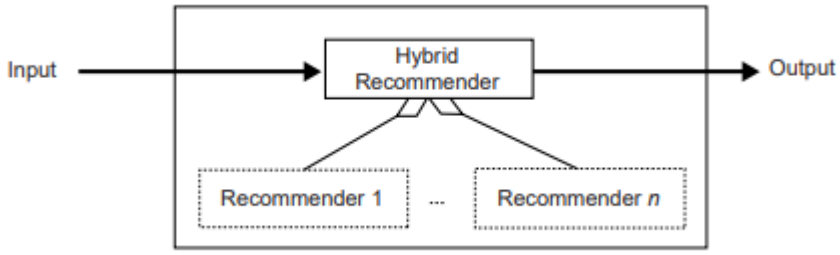

Figure III. 1Conception d'hybridation monolithique.

Comme l'expliquent Ekstrand et al., un moteur collaboratif peut être enrichi par une phase de prétraitement qui ajoute des notes fictives à la matrice, afin de relier les objets similaires selon leur contenu et ainsi améliorer les prédictions pour certains utilisateurs [17].

Avantages

- Offre une intégration forte entre les approches, ce qui permet des résultats plus homogènes.
- Peut réduire la sparsité de la matrice et améliorer la couverture des recommandations.

Inconvénients

- Implémentation complexe.
- Risque d'introduire du biais artificiel si la construction des données supplémentaires n'est pas rigoureuse.

5.2.2 Le système mixte (Mixed Hybrid)

Le système hybride mixte adopte une approche simple mais efficace : les différents moteurs de recommandation fonctionnent séparément, et leurs résultats sont présentés ensemble à l'utilisateur, sans traitement d'unification ou de pondération.

Par exemple, un site peut afficher :

- Des recommandations populaires,
- Des suggestions personnalisées via filtrage collaboratif,
- Et des produits similaires basés sur les caractéristiques de contenu.

L'utilisateur voit donc un mélange d'objets recommandés, chacun venant d'un moteur distinct. Cette approche permet de multiplier les opportunités de pertinence, même si elle ne garantit pas une cohérence globale entre les résultats affichés.

Selon les auteurs, cette méthode consiste à simplement retourner l'union de tous les résultats produits par les moteurs, ce qui permet de s'assurer que l'utilisateur reçoit *au moins* quelques recommandations, même si certaines sources d'information sont limitées [17].

Avantages

- Très facile à implémenter.
- Fournit une grande diversité de suggestions.
- Fonctionne bien pour les nouveaux utilisateurs (cold start).

Inconvénients

- Absence de logique d'unification.
- Présence possible de doublons ou de résultats incohérents entre eux.

5.2.3 Le système en ensemble (Ensemble Hybrid)

Les systèmes hybrides en ensemble sont les plus sophistiqués. Ils reposent sur l'exécution de plusieurs moteurs complets, puis sur une fusion intelligente de leurs résultats selon des méthodes précises : pondération, vote, sélection contextuelle, etc.

Plusieurs variantes existent :

- Pondération des scores : chaque moteur attribue un score aux objets, puis ces scores sont agrégés (moyenne pondérée, somme, etc.).
- Vote majoritaire : chaque moteur vote pour les éléments qu'il recommande ; les plus "votés" sont sélectionnés.
- Sélection contextuelle (switched) : le système choisit dynamiquement quel moteur utiliser selon le contexte de la demande (heure, type d'utilisateur, appareil utilisé…).

Ekstrand et ses collègues expliquent que cette approche consiste à faire tourner plusieurs moteurs complets et à combiner ensuite leurs résultats de manière contextuelle ou pondérée. Par exemple, un moteur peut être préféré en journée, et un autre la nuit, selon le comportement observé des utilisateurs [17].

Avantages

- Très flexible : chaque moteur peut être conçu et optimisé indépendamment.
- Permet une personnalisation avancée.
- S'adapte bien aux situations complexes ou hétérogènes (ex. : utilisateurs de cultures différentes, ou recommandation multi-domaines).

Inconvénients

- Coût élevé en calcul et en ressources.
- Complexité dans le réglage des pondérations ou dans la définition des règles de sélection.

Le choix de la stratégie dépend des objectifs fonctionnels de l'application et des types de données disponibles.

5.3 Exemples d'implémentation

Dans l'application *Agrico*, plusieurs cas d'usage peuvent tirer profit d'une approche hybride :

- **Suggestion de matériel agricole** : en combinant un filtrage basé sur le contenu (type de matériel, capacité, localisation, disponibilité) avec un filtrage collaboratif (réservations similaires par d'autres utilisateurs ayant un profil proche), on peut recommander des équipements adaptés aux besoins spécifiques des agriculteurs.

5.4 Avantages dans le contexte de Agrico

L'approche hybride permet à *Agrico* d'offrir une expérience utilisateur riche, proactive et personnalisée, en réduisant les délais de recherche, en améliorant la pertinence des suggestions et en augmentant la satisfaction des utilisateurs. Elle permet aussi de mieux gérer la diversité des services proposés, en s'adaptant à différents profils d'utilisateurs

(agriculteurs, chauffeurs, propriétaires de matériel) et à la complexité des interactions entre les rôles.

6 Conclusion

Les systèmes de recommandation constituent aujourd'hui une composante incontournable des plateformes numériques, en contribuant à personnaliser l'expérience utilisateur, à valoriser des contenus souvent négligés (la "longue traîne") et à accroître l'engagement et la satisfaction. Qu'ils reposent sur des approches collaboratives, sur le contenu, ou qu'ils combinent plusieurs méthodes dans une stratégie hybride, leur efficacité repose sur la capacité à extraire et exploiter intelligemment les données utilisateurs.

L'analyse présentée dans ce rapport montre que chaque type de système possède ses forces : le filtrage collaboratif excelle dans la détection de préférences sociales implicites, la méthode basée sur le contenu s'adapte bien aux nouveaux utilisateurs, tandis que l'approche hybride permet de pallier les faiblesses des autres techniques en combinant leurs avantages respectifs.

Dans un projet tel que *Agrico*, où la recommandation personnalisée peut améliorer la mise en relation entre agriculteurs, chauffeurs et prestataires, le recours à un système hybride apparaît comme particulièrement pertinent. Il offre une solution robuste et flexible, capable de répondre aux besoins complexes d'une plateforme moderne, évolutive et centrée sur l'utilisateur.

Chapitre IV – Déploiement et Gestion des Microservices

1 Introduction

Le déploiement d'applications modernes, notamment celles basées sur une architecture microservices, constitue aujourd'hui un défi majeur pour les équipes de développement et d'exploitation. Contrairement aux approches monolithiques, où le déploiement consiste à livrer une seule unité logicielle, les microservices impliquent la gestion d'un ensemble distribué de services autonomes, chacun devant être déployé, configuré, surveillé et mis à l'échelle indépendamment. Cette granularité fonctionnelle, bien qu'elle favorise l'évolutivité et la résilience, induit une complexité opérationnelle accrue.

Parallèlement à l'évolution des modèles architecturaux, les outils et processus de déploiement ont eux aussi connu une transformation radicale. La conteneurisation, portée par des technologies comme Docker ou Podman, a permis de standardiser l'environnement d'exécution des applications, facilitant ainsi leur portabilité et leur cohérence entre les phases de développement et de production. En complément, des plateformes d'orchestration telles que Kubernetes sont devenues incontournables pour automatiser le cycle de vie des conteneurs, en assurant leur déploiement, leur supervision, leur redémarrage automatique en cas de panne, et leur mise à l'échelle dynamique selon la charge.

Ce chapitre s'attache à explorer les fondements du déploiement de microservices en s'appuyant sur les outils modernes qui ont permis de surmonter les contraintes de gestion dans des systèmes distribués. Il présente les concepts de conteneurisation, les plateformes d'orchestration, ainsi que les mécanismes d'intégration et de livraison continues (CI/CD) qui soutiennent aujourd'hui la transformation DevOps dans les organisations logicielles.

2 Introduction aux Déploiement de Microservices

2.1 Évolution des Approches de Déploiement

Le Déploiement est une combinaison de deux concepts interdépendants, le processus et l'architecture :

➔ Le processus de déploiement comprend les étapes qui doivent être effectuées afin de mettre le logiciel en production.

➔ L'architecture de déploiement définit la structure de l'environnement dans lequel ce logiciel s'exécute.

Les deux aspects du déploiement ont radicalement changé depuis la fin des années 1990. Le processus manuel de production est devenu fortement automatisé. Comme le montre la figure 1, les environnements de production physiques ont été remplacés par des infrastructures informatiques de plus en plus légères et éphémères. [19]

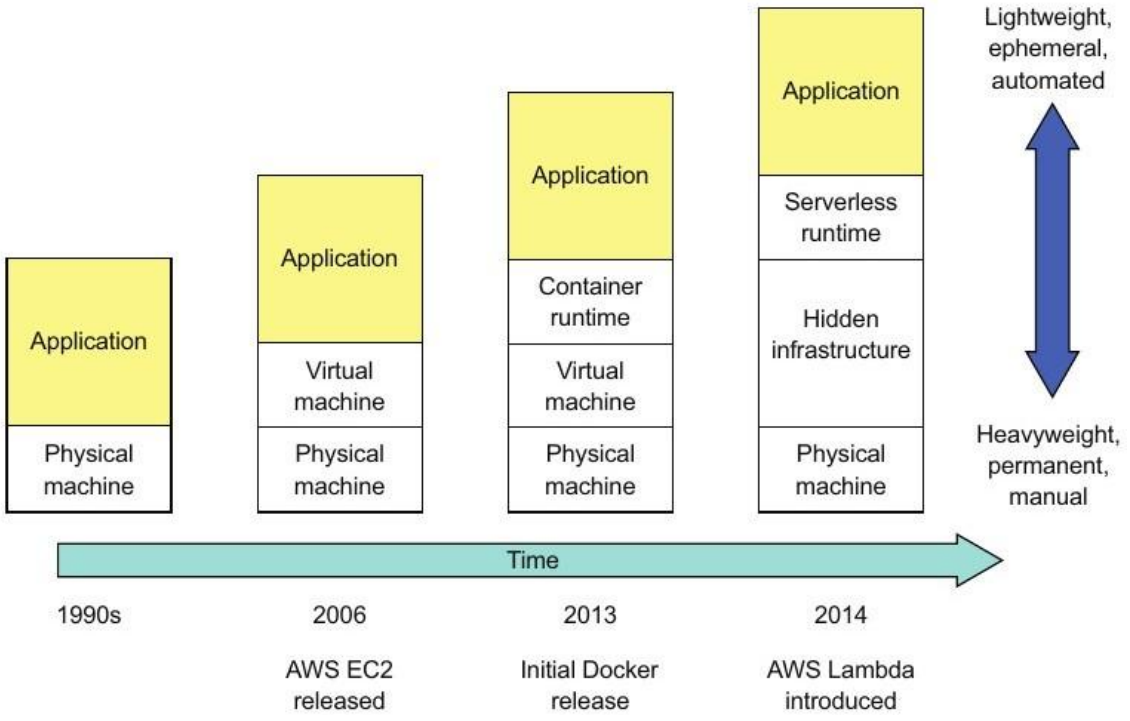

Figure IV. 1 - Heavyweight physical machines have been abstracted away by increasingly lightweight technologies. [19]

2.2 Technologies de déploiement modernes pour les microservices

L'évolution des processus de déploiement a coïncidé avec l'adoption croissante de l'architecture microservices. Cependant, ce changement architectural a également introduit de nouvelles complexités [20] :

- Augmentation du nombre d'unités déployables : chaque service nécessite son propre pipeline de déploiement.
- Communication de service : les systèmes distribués nécessitent des mécanismes robustes pour la découverte et la communication des services.
- Surcharge opérationnelle : la gestion de plusieurs services nécessite des mécanismes avancés de surveillance, d'enregistrement et de tolérance aux pannes.

Pour répondre à ces complexités, les technologies de déploiement modernes ont évolué :

- **Conteneurisation**

  Les conteneurs fournissent des environnements légers pour l'empaquetage et l'exécution cohérente de microservices tout au long du développement et de la production. [21]
- **Orchestration**

  Les plateformes comme Kubernetes automatisent le déploiement, la mise à l'échelle et la gestion des services conteneurisés. [21]
- **Intégration continue/déploiement continu (CI/CD)**

  Les pipelines d'automatisation rationalisent le processus de création, de test et de déploiement des microservices. [21]

Ces avancées ont permis de gérer des déploiements de microservices à grande échelle tout en conservant leur agilité.

3 La Conteneurisation

Lorsqu'une application n'est composée que d'un petit nombre de composants volumineux, il est tout à fait acceptable de donner une machine virtuelle (VM) dédiée à chaque composant et d'isoler leurs environnements en fournissant à chacun d'eux sa propre instance de système d'exploitation.

Mais lorsque ces composants commencent à devenir plus petits et que leur nombre commence à augmenter, on ne peut pas donner à chacun d'entre eux sa propre machine virtuelle. Et c'est là que les conteneurs entrent en action. [19]

3.1 Qu'est-ce qu'un Conteneur ?

Les conteneurs sont des groupes de processus exécutés sur un système Linux qui sont isolés les uns des autres. Les conteneurs s'assurent qu'un groupe de processus n'interfère pas avec d'autres processus du système [22], de la même manière que les machines virtuelles, mais avec beaucoup moins de surcharge.

3.2 Comparaison avec les machines virtuelles

Les conteneurs sont souvent comparés aux machines virtuelles dans la mesure où ils peuvent tous deux exécuter plusieurs applications isolées sur un seul nœud.

Lorsque on utilise des machines virtuelles, on doit gérer l'intégralité du système d'exploitation de la machine virtuelle ainsi que l'application isolée. On doit gérer le cycle de vie des différents noyaux, le système d'initialisation, la journalisation, les mises à jour de sécurité, les sauvegardes, etc. Le système doit également gérer la surcharge de l'ensemble du système d'exploitation en cours d'exécution, et pas seulement de l'application. [19]

La figure 2 montre 6 applications s'exécutant dans trois machines virtuelles différentes.

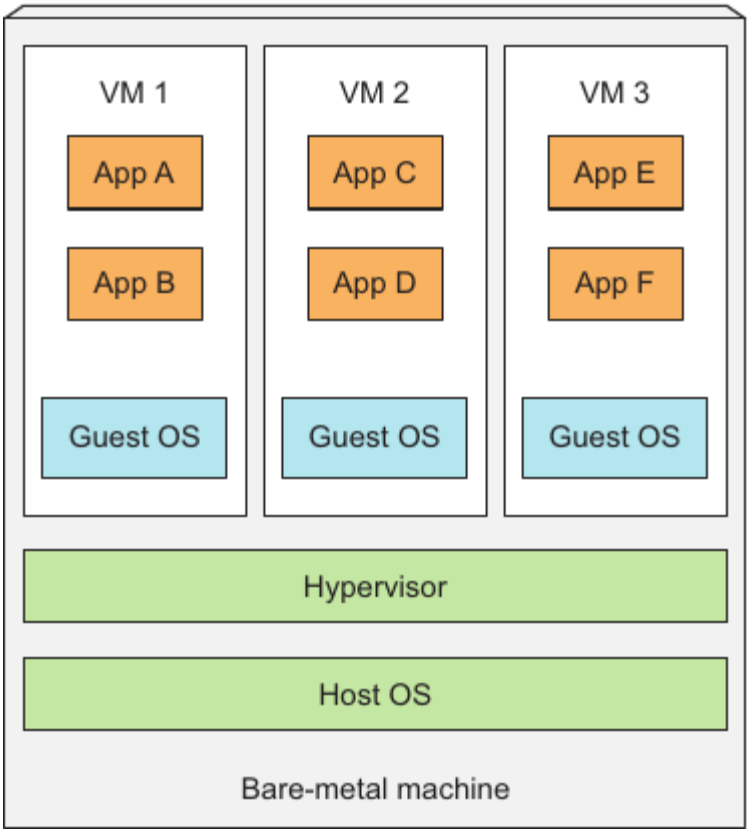

Figure IV. 2 - Machine physique exécutant 6 applications dans trois machines virtuelles [19]

Un processus s'exécutant dans un conteneur s'exécute à l'intérieur du système d'exploitation de l'hôte, comme tous les autres processus (contrairement aux machines virtuelles, où les processus s'exécutent dans des systèmes d'exploitation distincts). En général, la seule partie du système hôte avec laquelle l'application interagit est le noyau hôte [19].

La figure 3 montre 9 applications s'exécutant dans 9 conteneurs différents.

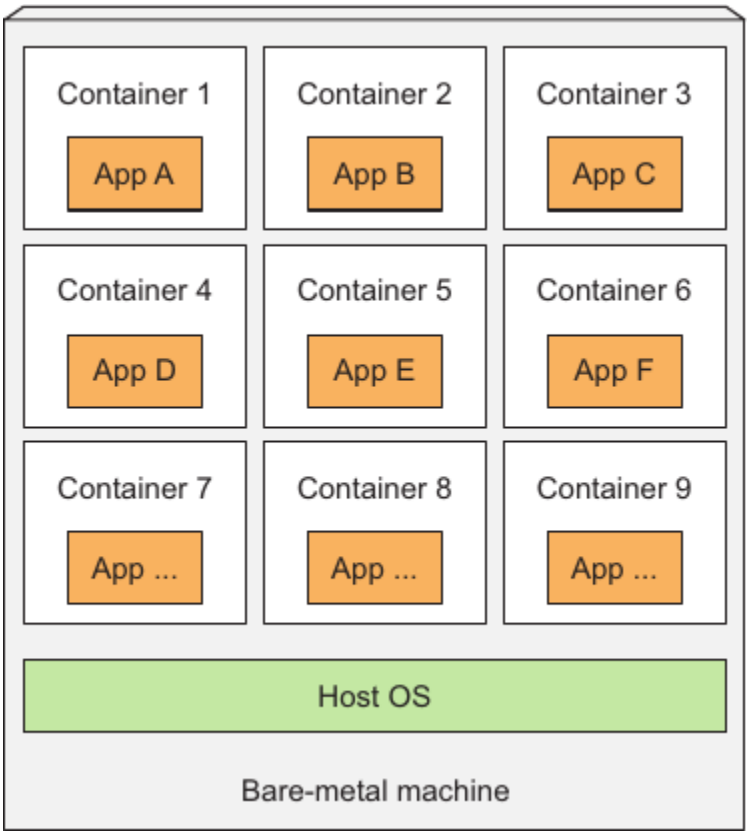

Figure IV. 3 - Machine physique exécutant 9 applications conteneurisées [19]

3.3 Les Technologies de Conteneurs

3.3.1 Docker

Docker est une plateforme open source qui automatise le déploiement, la mise à l'échelle et la gestion des applications à l'aide de la technologie de conteneurisation. Docker a été lancé pour la première fois en 2013 par Docker, Inc. (anciennement dotCloud).

Docker suit une architecture client-serveur avec plusieurs composants distincts qui fonctionnent ensemble pour créer, distribuer et exécuter des applications conteneurisées :

3.3.2 Docker Container Engine

Un moteur de conteneur (Container Engine) est un outil logiciel qui accepte et traite les demandes des utilisateurs pour créer un conteneur. [21]

Le moteur de conteneur Docker se compose de trois piliers fondamentaux :

- Docker Démon
- Docker API
- Docker CLI

Ces trois piliers sont représentés dans l'architecture suivante, comme le montre la figure 4 :

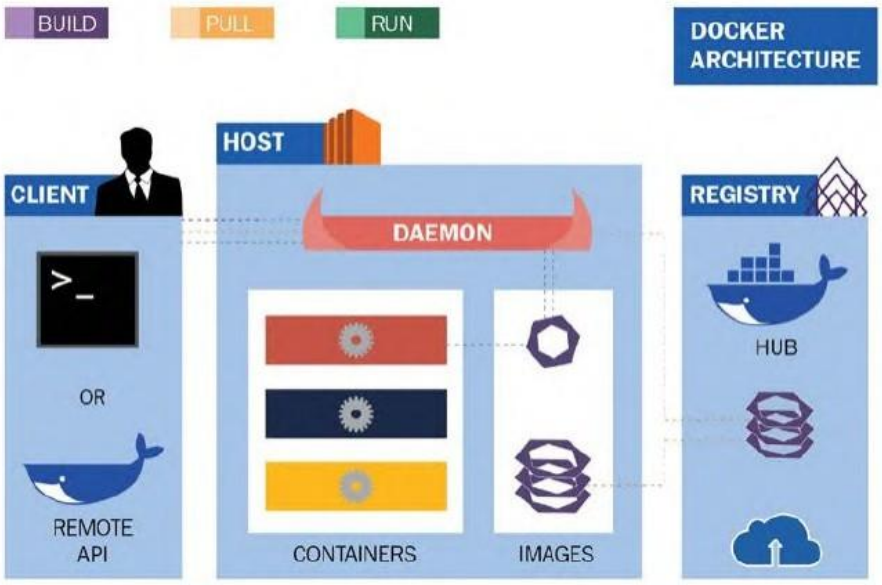

Figure IV. 4 -Docker Architecture [21]

Une fois qu'un démon Docker est en cours d'exécution, comme le montre le diagramme précédent, nous pouvons interagir avec lui via un client Docker ou une API distante. Le démon Docker est responsable de nombreuses activités de conteneur local ainsi que de l'interaction avec les registres externes pour extraire ou envoyer des images de conteneur. [21]

3.3.3 Docker Daemon

Un démon est un processus qui s'exécute en arrière-plan. Il supervise le système ou fournit des fonctionnalités à d'autres processus.

Le démon Docker est le processus d'arrière-plan qui est responsable de :

- Écoute des requêtes d'API Docker.
- Manipulation, gestion et vérification des conteneurs en cours d'exécution.
- Gestion des images Docker, des réseaux et des volumes de stockage.
- Interaction avec des registres d'images de conteneurs externes/distants.

3.3.4 Docker API

Une fois qu'un démon Docker est opérationnel, nous pouvons communiquer via un client ou directement via l'API. Grâce à l'API Docker, nous pouvons effectuer toutes les activités que nous pouvons effectuer via l'outil de ligne de commande, telles que créer, inspecter, exporter, démarrer ou arrêter un conteneur. [21]

3.3.5 Docker CLI client

Le démon Docker dispose d'un client de ligne de commande qui donne des instructions et configure. Le client CLI Docker dispose de plus de 30 commandes qui permettent à n'importe quel utilisateur Docker d'instruire et de contrôler le démon et ses conteneurs. [21]

Une fois que nous lançons le client Docker avec l'une de ces commandes, le client contacte le démon Docker, où il lui indique ce qui est nécessaire et quelle action doit être effectuée. [21]

3.3.6 Docker Images

Une image Docker est un format introduit par Docker pour la gestion des données binaires et des métadonnées en tant que modèle pour la création de conteneurs. Les images Docker sont des packages permettant d'expédier et de transférer des environnements d'exécution, des bibliothèques et tout ce qui est nécessaire pour qu'un processus donné soit opérationnel. [21]

3.3.7 Docker Registries

Un registre Docker est un référentiel d'images de conteneur Docker qui contient les métadonnées et les couches d'images de conteneur pour les mettre à la disposition de plusieurs démons Docker. Un démon Docker agit en tant que client d'un registre Docker par le biais d'une API HTTP, en envoyant et en extrayant des images de conteneur en fonction de l'action demandée par le client Docker. [21]

3.3.8 Podman : une alternative moderne

Podman, ou "POD manager", est un outil open source développé par Red Hat pour gérer des conteneurs sur des systèmes Linux. [23]

Le projet Podman décrit Podman comme « un moteur de conteneur sans démon pour le développement, la gestion et l'exécution de conteneurs OCI sur votre système Linux. Les conteneurs peuvent être exécutés en tant que root ou en mode rootless" [24]

3.3.9 Podman est sans-démon

Podman est fondamentalement différent de Docker car il est sans-démon. Podman peut exécuter toutes les mêmes images de conteneur que Docker et lancer des conteneurs avec les mêmes environnements d'exécution de conteneurs.

Cependant, Podman le fait sans avoir plusieurs démons qui s'exécutent en continu de manière rootée. [25]

3.3.10 Podman est Rootless

Podman a la capacité de fonctionner en mode rootless (non rootée). Dans de nombreuses situations, nous ne voulons pas donner un accès root complet aux utilisateurs, mais les utilisateurs et les développeurs doivent toujours exécuter des conteneurs et créer des images de conteneur [25]. Podman peut exécuter des conteneurs sans fonctionnalités de sécurité supplémentaires dans Linux autres qu'un compte de connexion standard. [22]

3.3.11 Podman est compatible avec Docker

Docker était l'outil dominant, et presque tous ceux qui avaient travaillé avec des conteneurs l'avaient fait avec son CLI. De plus, si nous devions chercher comment faire quelque chose avec un conteneur en ligne, nous obtiendrions invariablement un exemple en utilisant la ligne de commande Docker. Podman devait correspondre à la ligne de commande Docker. Une devise pour remplacer Docker par Podman a rapidement été développée : alias docker = podman.

Avec cette commande, nous pouvons continuer à taper des commandes Docker, mais Podman exécute les conteneurs [25].

3.3.12 Podman est compatible avec Kubernetes

La documentation officielle de Kubernetes indique que « un pod est un groupe d'un ou plusieurs conteneurs, avec des ressources de stockage/réseau partagées, et une spécification sur la façon d'exécuter les conteneurs » [26]. Podman fonctionne avec

un seul conteneur à la fois, comme Docker, ou il peut gérer de groupes de conteneurs ensemble dans un pod. Les pods nous permettent de regrouper plusieurs services pour former un service plus vaste géré comme une seule entité [25].

Nous pouvons utiliser Podman pour exécuter des pods et des conteneurs sur un seul hôte et utiliser Kubernetes pour prendre ces pods et conteneurs et les exécuter sur plusieurs machines et dans toute l'infrastructure [4]. D'autres projets, comme kind [27], expérimentent l'exécution de pods avec Podman sous la direction de Kubernetes.

4 Orchestration des Conteneurs

4.1 Rôle de l'orchestration dans le déploiement.

L'orchestration de conteneurs consiste à automatiser le déploiement, la gestion, la mise à l'échelle et la surveillance des conteneurs au sein d'un cluster de serveurs. Cette automatisation est cruciale pour déployer des applications modernes, notamment en production, où les environnements sont souvent complexes. [20]

Google Cloud définit l'orchestration comme le moyen d'automatiser la mise en place et la gestion des applications conteneurisées sans se préoccuper de l'infrastructure sous-jacente [28].

4.2 Introduction aux Kubernetes

4.2.1 Qu'est-ce que Kubernetes ?

Kubernetes, souvent abrégé en K8s, s'est imposé comme la solution d'orchestration de conteneurs de référence au cours des dernières années [20].

Son histoire est étroitement liée à Google, qui l'a initialement développé en 2013 avant de le rendre open source en 2014. [20]

4.2.2 Fonctionnalités de Kubernetes

Kubernetes permet fondamentalement de déployer des applications conteneurisées sur n'importe quelle infrastructure informatique et de gérer de façon centralisée les différentes ressources dont elles ont besoin [20].

Ces ressources peuvent inclure des capacités de calcul, de stockage, de bases de données, de réseau, et bien d'autres composants essentiels au fonctionnement des applications modernes.

L'architecture de Kubernetes s'organise autour du concept de cluster, la figure présente la vue la plus simple possible d'un système Kubernetes.

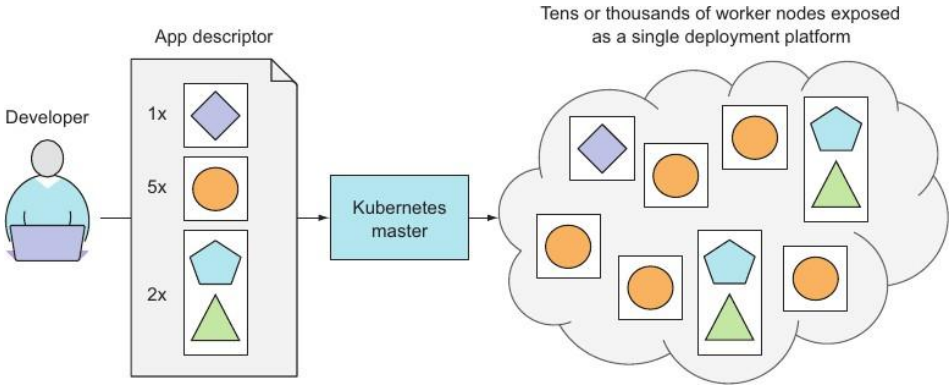

Figure IV. 5 -Vue d'un système Kubernetes [20]

Le système est composé d'un nœud maître et d'un nombre quelconque de nœuds travailleurs. Lorsque le développeur soumet une liste d'applications au maître, Kubernetes les déploie sur le cluster de nœuds travailleurs.

4.2.3 Exécuter une application dans Kubernetes

Pour exécuter une application dans Kubernetes, nous devons d'abord la packager en une ou plusieurs images de conteneurs, pousser ces images vers un registre d'images, puis envoyer une description de l'application au serveur API de Kubernetes. La description inclut des informations telles que l'image ou les images de conteneurs qui contiennent les composants de l'application, la manière dont ces composants sont liés les uns aux autres, et lesquels doivent être exécutés de manière co-localisée (ensemble sur le même nœud) et lesquels ne le doivent pas.

Pour chaque composant, nous pouvons également spécifier combien de copies (ou réplicas) nous voulons exécuter. De plus, la description indique également lesquels de ces composants fournissent un service à des clients internes ou externes et doivent être exposés via une seule adresse IP et rendus détectables par les autres composants. [20]

4.2.4 Maintenir les conteneurs en fonctionnement

Une fois l'application en cours d'exécution, Kubernetes s'assure en permanence que l'état déployé de l'application correspond toujours à la description que nous avons fournie. Par exemple, si nous spécifions que nous voulons toujours cinq instances d'un serveur web en fonctionnement, Kubernetes maintiendra exactement cinq instances en cours d'exécution. Si l'une de ces instances cesse de fonctionner correctement, Kubernetes la redémarrera automatiquement.

De manière similaire, si un nœud de travail entier tombe en panne ou devient inaccessible, Kubernetes sélectionnera de nouveaux nœuds pour tous les conteneurs qui étaient en cours d'exécution sur ce nœud et les exécutera sur les nœuds nouvellement sélectionnés. [20]

4.2.5 Ajustement dynamique du nombre d'instances

Tant que notre application est active, nous pouvons augmenter ou réduire le nombre d'instances selon nos besoins, et Kubernetes démarre ou arrête les conteneurs nécessaires. Nous pouvons également déléguer cette tâche à Kubernetes, qui ajuste automatiquement le nombre d'instances en fonction de métriques en temps réel, comme l'utilisation du processeur, la mémoire ou les requêtes par seconde, pour garantir des performances optimales. [28]

4.2.6 Gérer des conteneurs en mouvement

Lorsque Kubernetes déplace nos conteneurs, par exemple en cas de panne d'un nœud, Kubernetes les regroupe sous une adresse IP unique et stable, accessible via DNS ou variables d'environnement. Grâce à kube-proxy, les connexions sont équilibrées entre les conteneurs, assurant un accès continu pour les clients, même si les conteneurs changent d'emplacement dans le cluster. [28]

4.2.7 Avantages clés de Kubernetes

- **Optimisation des ressources matérielles :** Allocation dynamique des applications selon les besoins (ex : SSD/HDD) et regroupement intelligent des composants pour une utilisation maximale des serveurs [20].
- **Déploiement simplifié :** Les conteneurs autonomes s'exécutent sur n'importe quel nœud sans intervention manuelle. Les développeurs déploient sans connaître l'infrastructure sous-jacente [20].
- **Autoréparation et résilience :** Redémarrage automatique des applications sur des nœuds sains en cas de panne, réduisant l'intervention humaine [20].
- **Mise à l'échelle automatique :** Ajustement du nombre d'instances et de la taille du cluster (sur le cloud) en temps réel selon la charge [20].
- **Environnement unifié dev/prod :** Réduction des bugs tardifs grâce à des environnements identiques, et simplification de fonctions complexes (découverte de services, élection de leader) via l'API Kubernetes [20].
- **Livraison continue sécurisée :** Rollouts progressifs avec blocage automatique en cas d'erreur, accélérant les déploiements [20].

4.2.8 Les Défis de l'Orchestration avec Kubernetes

Malgré ses nombreux avantages, l'orchestration de conteneurs avec Kubernetes présente des défis significatifs [20] :

- L'équilibrage optimal des ressources entre les différentes applications
- L'adaptation dynamique aux charges de travail fluctuantes
- L'optimisation énergétique des clusters (green computing)
- La gestion des applications sensibles au temps (time-sensitive)
- La sécurité des déploiements distribués

Noter que de nombreux travaux récents proposent des améliorations aux algorithmes de planification de Kubernetes [26].

5 Conclusion

Le déploiement de microservices représente un écart significatif par rapport aux approches monolithiques traditionnelles en raison de l'accent mis sur l'évolutivité et la flexibilité indépendantes au prix d'une complexité opérationnelle accrue.

Nous avons donné un aperçu de l'évolution des approches de déploiement, ainsi que des modèles clés et des exigences d'infrastructure qui sous-tendent les mises en œuvre réussies.

Nous avons aussi exploré comment la conteneurisation facilite les déploiements cohérents dans tous les environnements et comment les plateformes d'orchestration comme Kubernetes rationalisent les opérations dans les systèmes distribués, permettant aux organisations de réaliser pleinement le potentiel de l'architecture des microservices tout en maintenant l'efficacité opérationnelle à grande échelle.

Chapitre V – Analyse et Conception

1 Introduction

Ce chapitre présente l'analyse et la conception de notre application. Il expose l'approche adoptée pour découper le système en sous-domaines fonctionnels, en s'appuyant sur les principes de la conception orientée domaine (DDD). Il détaille également la modélisation des entités, les cas d'utilisation, ainsi que les interactions entre les composants sous forme de microservices interconnectés à l'aide d'une architecture orientée événements (EDA). L'objectif est de poser une base solide avant l'implémentation, en assurant cohérence, extensibilité et compréhension claire du système.

Dans cette section, nous détaillons la méthode suivie pour concevoir l'architecture logicielle de notre application. Nous avons opté pour une approche Domain-Driven Design (DDD), qui consiste à identifier les Contextes Métiers clés du domaine, et à modéliser chaque sous-système comme un Bounded Context (DDD stratégique). Chaque domaine est ensuite traduit en microservice indépendant, intégrant ses propres entités, règles métier et interfaces (DDD tactique).

La communication entre microservices repose en grande partie sur des événements métiers émis via un système de messagerie, selon une architecture orientée événements (EDA).

Nous présentons également les diagrammes UML utilisés (diagrammes de cas d'utilisation, de classes, de séquence) afin d'illustrer la structure et le fonctionnement global du système.

2 Analyse du domaine et des Besoins (DDD Stratégique)

Avant de concevoir l'architecture microservices de l 'application, il est essentiel de comprendre le domaine global dans son ensemble. Cette vue d'ensemble sert de point de départ pour identifier les différents sous-ensembles fonctionnels. Une fois cette analyse effectuée, l'objectif est de décomposer ce domaine complexe en plusieurs contextes bien délimités, chacun portant sur un aspect métier cohérent. C'est dans cette logique que nous nous appuyons sur les principes du Domain-Driven Design (DDD) stratégique. Nous pouvons ensuite construire une carte des relations entre eux (context map) qui facilite la structuration du système et la répartition claire des responsabilités.

2.1 Contexte global de l'application

Notre application s'inscrit dans un contexte agricole où plusieurs acteurs interagissent autour d'activités complémentaires (mise en location des matériels, recherche de main-d'œuvre/transport, paiements, etc.). Ces interactions impliquent des rôles variés (agriculteurs, ouvriers, propriétaires de matériel, etc.) et des processus distincts.

2.1.1 Identifications des Acteurs

Agriculteur :

L'agriculteur représente un utilisateur final consommateur de services au sein de la plateforme. Contrairement aux autres acteurs, il n'offre pas de prestations, mais bénéficie des services mis à disposition par les différents intervenants (propriétaires de matériel, opérateurs, transporteurs, propriétaires de terrains).

Fonctions principales :

- Consulter les annonces de matériel agricole disponibles, avec possibilité d'appliquer des filtres.
- Envoyer des demandes de réservation pour du matériel ou un transport.
- Consulter les annonces de terrains agricoles à louer, avec possibilité d'appliquer des filtres.
- Envoyer une demande de location de terrain.
- Consulter les profils transporteurs et opérateurs affichés sur la carte interactive.
- Recevoir une estimation du prix basée sur la distance (calculée automatiquement).
- Évaluer les services reçus après leur exécution.

Propriétère de Matériel Agricole :

Le propriétaire de matériel agricole est un prestataire de services qui met à disposition des équipements pour location via la plateforme.

Fonctions principales:

- Créer et publier des annonces détaillant les équipements agricoles disponibles.
- Gérer les demandes de réservation reçues.
- Accepter ou refuser une demande, en fonction de la disponibilité ou des préférences.
- Consulter les informations du client (nom, numéro, profil) une fois la demande acceptée.

Propriétaire de Terrain Agricole :

Le propriétaire de terrain met à disposition une ou plusieurs parcelles agricoles à louer temporairement via la plateforme. Ce rôle permet de valoriser les terrains inutilisés ou saisonniers, en les rendant accessibles aux agriculteurs ayant besoin d'espace supplémentaire

Fonctions principales :

- Créer une annonce de location de terrain avec ses caractéristiques (surface, type, prix, disponibilité, etc.).
- Gérer les annonces existantes (mise à jour, désactivation, suppression).
- Recevoir les demandes de location de la part des agriculteurs.
- Accepter ou refuser les demandes selon les préférences et la période.
- Suivre les paiements associés aux locations validées.

Opérateur de Matériel Agricole :

L'opérateur est un prestataire de terrain chargé de manipuler les machines agricoles pour le compte des agriculteurs.

Fonctions principales :

- Renseigner et maintenir son profil professionnel.
- Activer sa localisation pour apparaître sur la carte interactive de la plateforme.
- Recevoir des demandes d'intervention émanant des agriculteurs.
- Être contacté directement via l'application.
- Être évalué par les utilisateurs à l'issue de chaque mission.

Transporteur :

Le transporteur assure le déplacement de marchandises ou d'équipements agricoles.

Fonctions principales:

- Activer sa visibilité géographique sur la carte interactive.
- Recevoir des demandes de transport contenant les détails logistiques (lieu de départ, destination, charge à transporter).
- Proposer un tarif, basé sur la distance estimée automatiquement.
- Évaluer les clients après chaque livraison.

2.1.2 Identifications des Fonctionnalités

Cette section présente les fonctionnalités principales offertes par l'application Agrico. Chaque fonctionnalité est étroitement liée à un sous-système métier défini précédemment dans la section 2.1.1, et répond à un besoin spécifique des acteurs du domaine agricole (agriculteurs, opérateurs, propriétaires, transporteurs).

1. Service d'Authentification

Ce service permet aux utilisateurs de créer un compte, de se connecter à la plateforme et de sécuriser leurs accès via un système de vérification.

Objectif : garantir l'authentification sécurisée et la confidentialité des identifiants.

2. Service de Gestion des Profils

Il permet la modification des informations personnelles, l'attribution du rôle (agriculteur, opérateur, transporteur, etc.) et la gestion de la visibilité sur la carte interactive.

Objectif : personnaliser les profils utilisateurs et adapter leur visibilité selon le contexte métier.

3. Service de Consultation des Profils

Ce service affiche les profils des prestataires (opérateurs et transporteurs), avec leurs détails, disponibilités et évaluations.

Objectif : faciliter la sélection de prestataires adaptés aux besoins agricoles.

4. Service de Gestion du Matériel Agricole

Il permet la création, modification ou suppression des annonces de location d'équipements agricoles.

Objectif : mettre à disposition des matériels pour la location via une interface simple et accessible.

5. Service de Consultation du Matériel Agricole

Ce service propose aux utilisateurs une interface de recherche des équipements disponibles à la location, avec filtres et visualisation sur carte.

Objectif : faciliter la recherche rapide de matériel adapté aux besoins spécifiques.

6. Service de Gestion des Terrains Agricoles

Il permet aux utilisateurs (propriétaires) de publier des annonces de terrains agricoles à louer, avec toutes les informations nécessaires (photos, prix, superficie).

Objectif : proposer un espace de publication structuré pour les terrains agricoles.

7. Service de Consultation des Terrains

Ce service permet aux utilisateurs de consulter les terrains disponibles à l'aide de filtres ou via une carte interactive.

Objectif : optimiser la recherche de terrains selon des critères géographiques et de surface.

8. Service de Notifications

Il envoie des alertes automatiques concernant les demandes, les paiements, les validations ou relances.

Objectif : assurer une communication fluide et réactive entre les utilisateurs.

9. Service de Paiement

Ce service gère le calcul des frais et commissions, la génération de factures et le suivi de l'état des paiements.

Objectif : automatiser le traitement des transactions financières de manière fiable et traçable.

10. Service de Recommandation

Il analyse le profil, l'historique et les interactions de l'utilisateur pour proposer des annonces ou profils pertinents.

Objectif : améliorer l'expérience utilisateur par la personnalisation des suggestions.

11. Service d'Évaluation

Ce service permet aux utilisateurs de laisser une note et un commentaire après chaque service reçu.

Objectif : renforcer la transparence et la fiabilité de la plateforme à travers un système de retour utilisateur.

12. Service de Réservation

Il prend en charge la création, la modification et l'annulation des demandes et réservations, qu'il s'agisse de matériel, de terrain ou de services.

Objectif : centraliser et simplifier la gestion des réservations tout en assurant leur traçabilit

La figure ci-dessous représente le schéma du Contexte Métier global de notre application qui va nous aider à mieux comprendre nos composants métiers :

Figure V. 1 -Contexte global de l'application

2.2 Identification des sous-domaines métier (Bounded Contexts)

Afin de décomposer notre modèle métier global en plusieurs sous modèles métier On va regrouper toutes les fonctionnalités métiers qui appartiennent aux mêmes contextes dans un modèle.

La figure ci-dessous représente le schéma des Contexte Bornés (Bounded Contextes) :

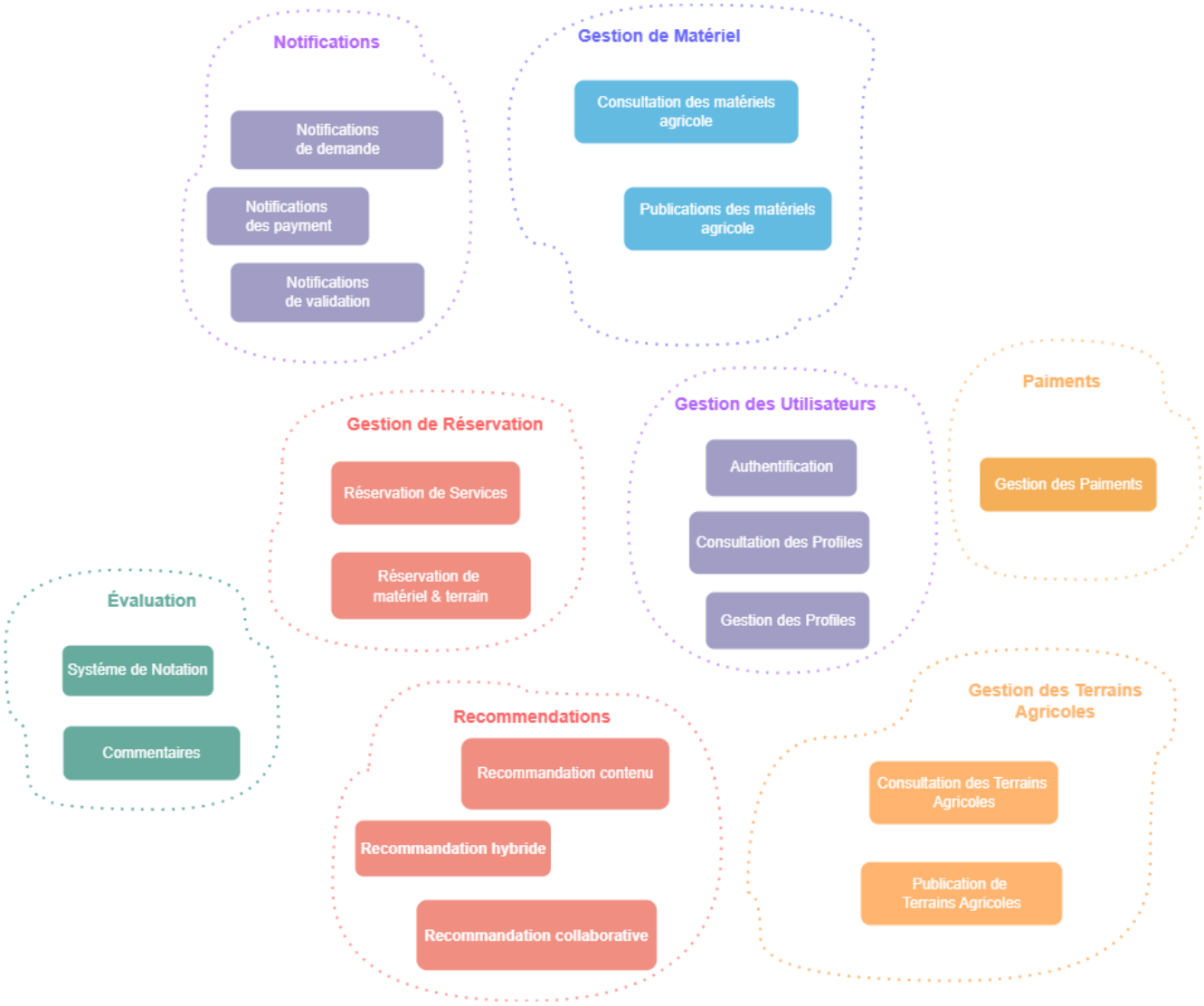

Figure V. 2 -Les Contextes Bornés (Bounded Contextes)

2.2.1 Contexte de Gestion des Utilisateurs.

Ce contexte regroupe les fonctionnalités essentielles à l'identification et à l'accès sécurisé des utilisateurs à la plateforme. Il permet la création de comptes, la connexion via des identifiants sécurisés, ainsi que la gestion des informations personnelles telles que le nom, le mot de passe ou la visibilité du profil. Il offre également la possibilité de consulter les profils d'autres utilisateurs afin de faciliter les interactions et la sélection de services pertinents.

Authentification

- Création d'un compte pour accéder à la plateforme.
- Connexion sécurisée avec un identifiant et un mot de passe.

Diagramme de flowcase

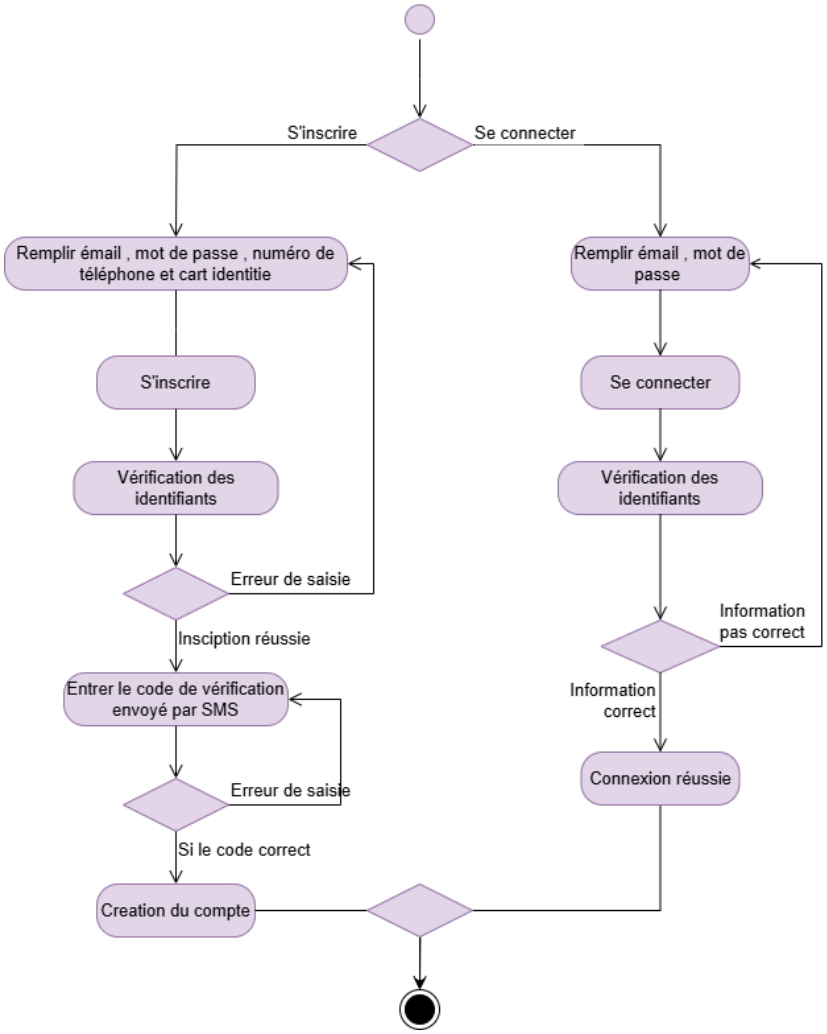

Figure V. 3 -Diagramme Flowcase : Authentification de l'utilisateur

Gestion des Profiles

- Modifier ses informations personnelles (nom, téléphone, mot de passe, etc.).
- Rendre son profil visible ou invisible sur la carte.

Diagramme de cas d'utilisation

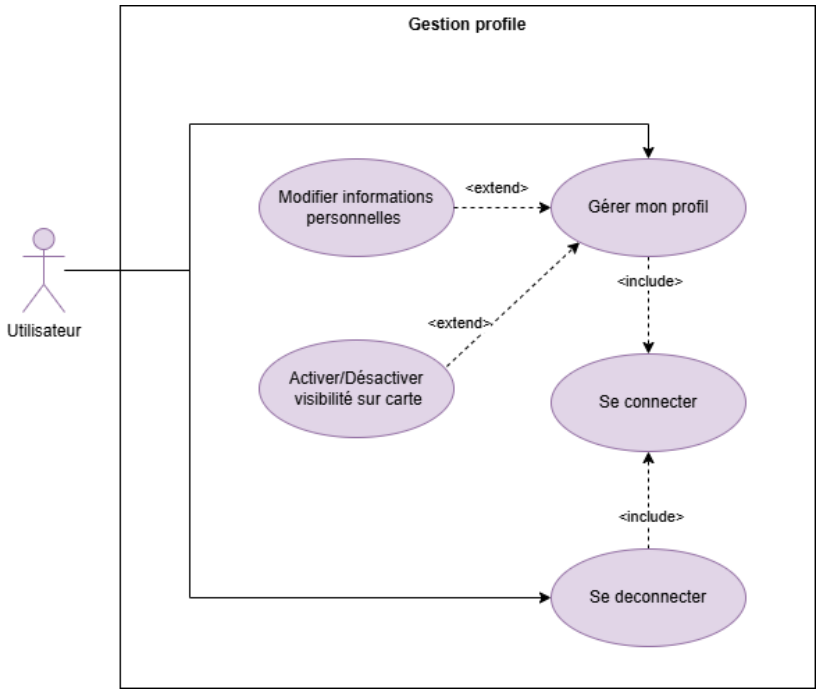

Figure V. 4 -Diagramme de cas d'utilisation : Gestion du profil utilisateur

Consultation des profils

- Voir la liste des opérateurs et transporteurs disponibles.
- Consulter les détails de leur profil (services, localisation, etc.).

Diagramme de cas d'utilisation

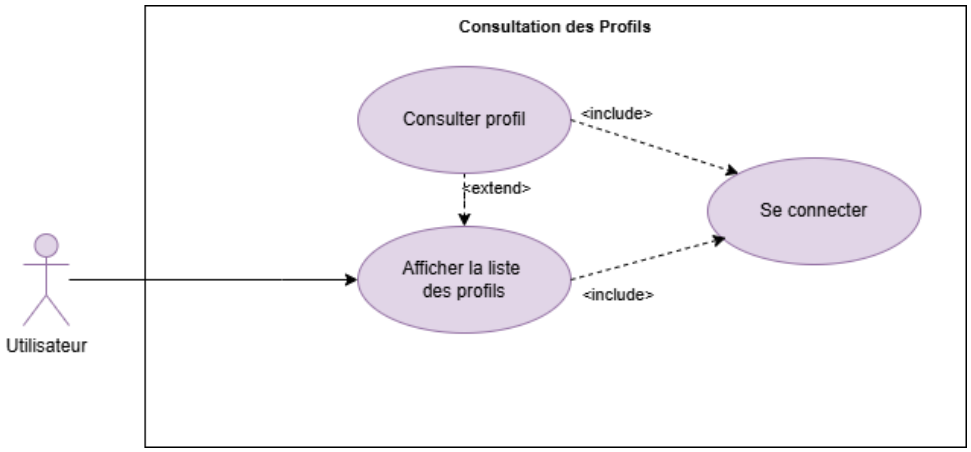

Figure V. 5 -Diagramme de cas d'utilisation : Consultation des profils utilisateurs

2.2.2 Contexte de Gestion de Matériel.

Ce contexte permet aux utilisateurs de publier, modifier et consulter des annonces de location de matériel agricole. Il facilite la mise en relation entre les propriétaires de matériel et les demandeurs via des outils de recherche, de filtrage et de présentation détaillée des équipements proposés, avec un affichage interactif sur carte.

Publication de matériels agricoles

- Publier, modifier ou supprimer une annonce de matériel.
- Ajouter des photos, une description, la localisation.

Diagramme de cas d'utilisation

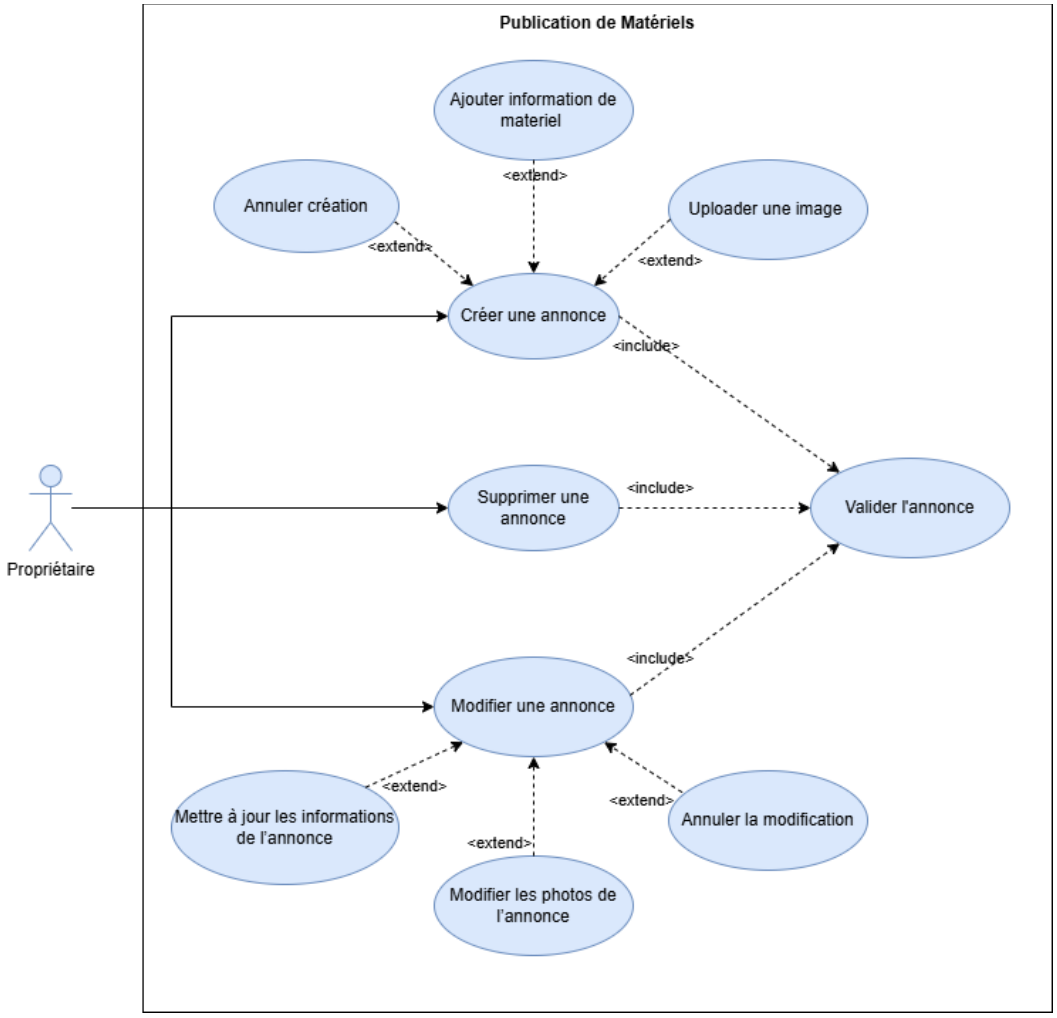

Figure V. 6 -Diagramme de cas d'utilisation : Publication de matériels agricoles

Consultation de matériels agricoles

- Rechercher du matériel à l'aide de filtres (type, prix, localisation, etc.).
- Afficher les résultats avec la possibilité de consulter les détails complets de chaque annonce.

Diagramme de cas d'utilisation

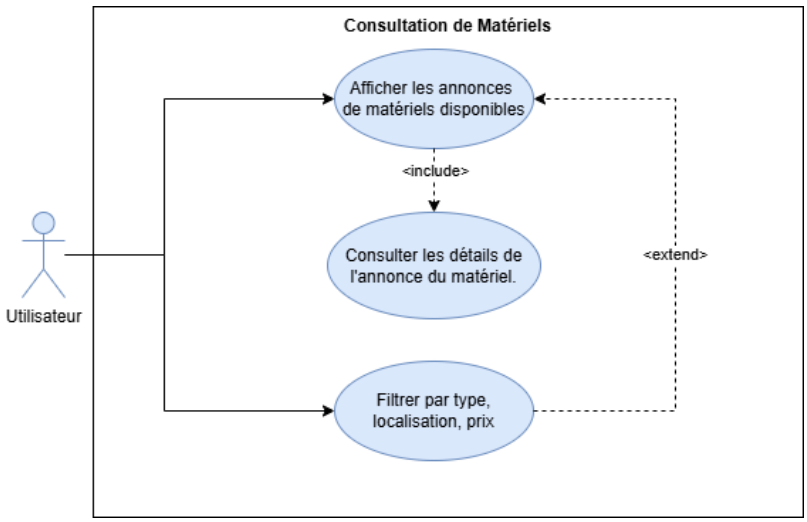

Figure V. 7 - Diagramme de cas d'utilisation : Consultation de matériels agricoles

Diagramme global de cas d'utilisation du Contexte

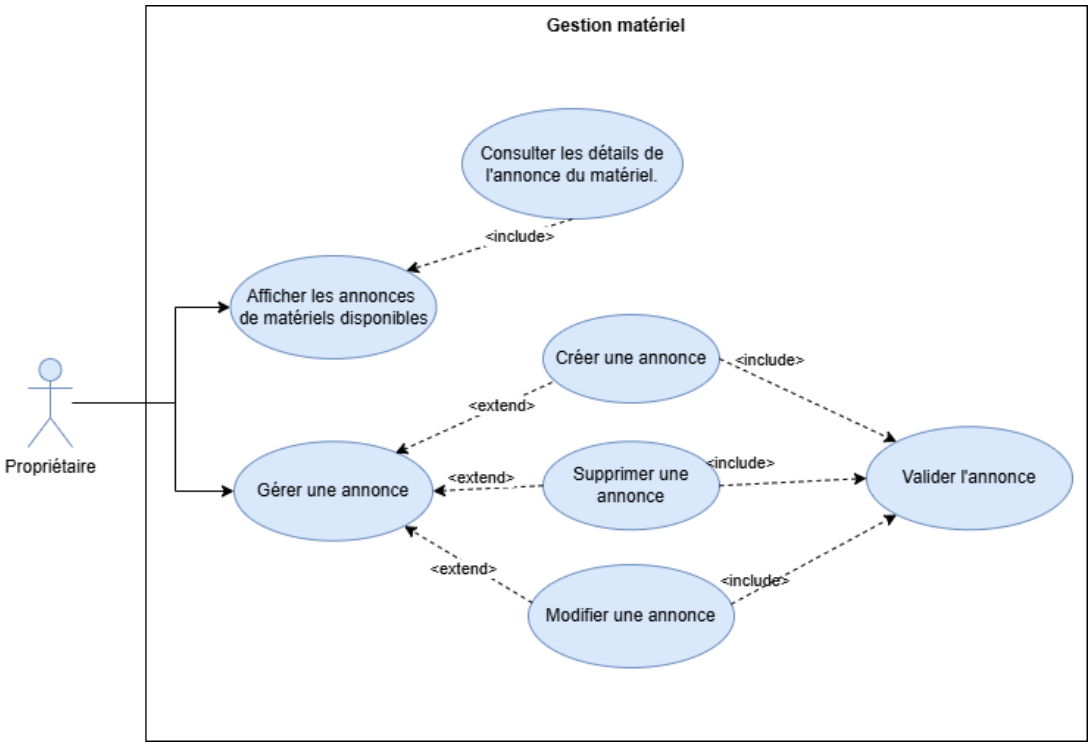

Figure V. 8 -Diagramme de cas d'utilisation : Gestion du matériel agricole

2.2.3 Contexte de Gestion des Terrains Agricoles.

Ce contexte permet aux utilisateurs de publier et de consulter des annonces de terrains agricoles disponibles à la vente ou à la location. Il facilite la diffusion d'informations détaillées sur les terrains et permet aux intéressés de les rechercher selon différents critères, avec un affichage interactif sur carte.

Publication de terrains

- Créer, modifier ou supprimer une annonce de terrain.
- Ajouter des photos, la superficie, la localisation, le prix et d'autres caractéristiques.

Diagramme de cas d'utilisation

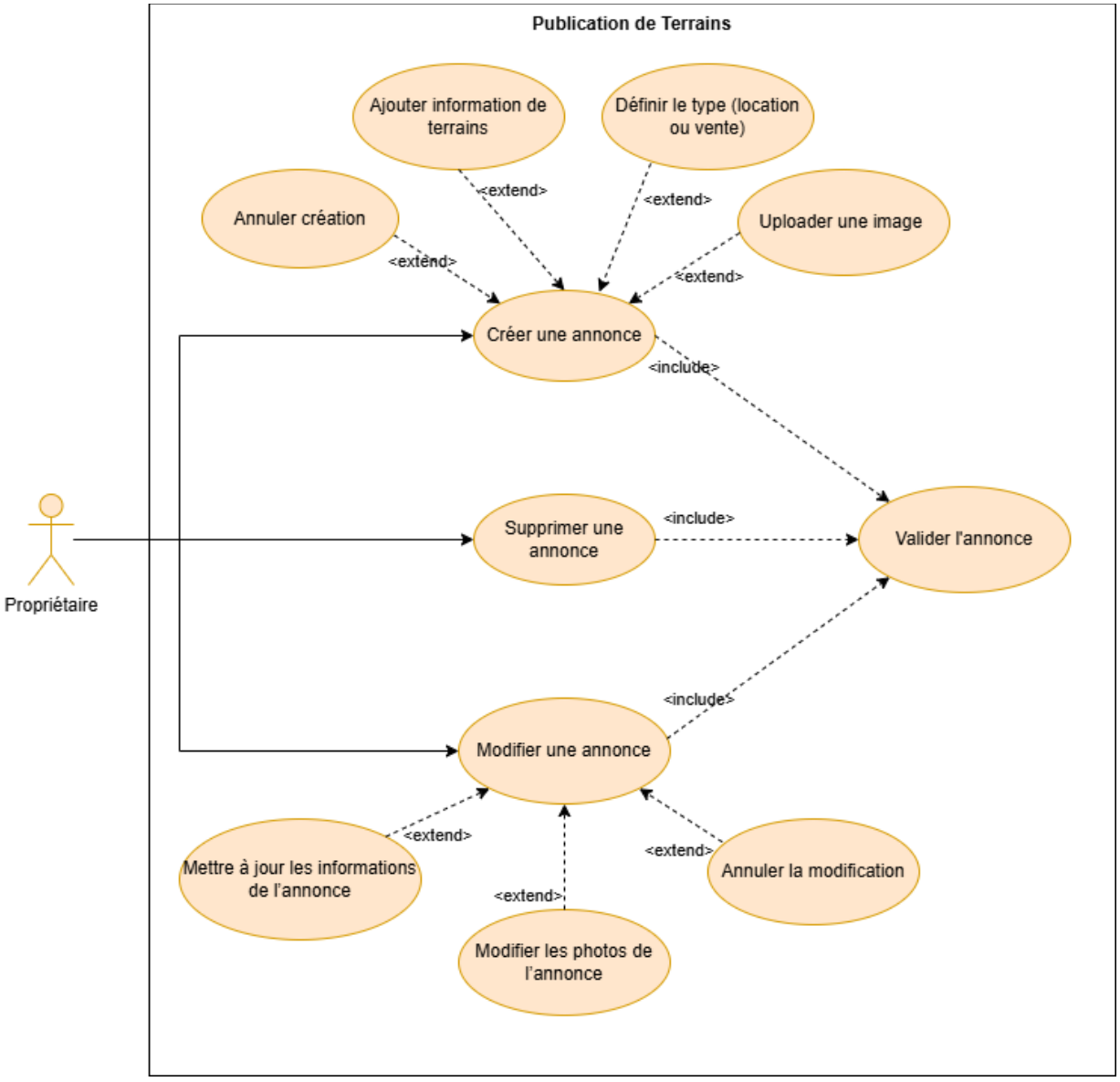

Figure V. 9 -Diagramme de cas d'utilisation : Publication de terrains agricole

Consultation de terrains

- Rechercher des terrains à l'aide de filtres (superficie, localisation, prix, etc.).
- Afficher les résultats avec les informations complètes pour chaque terrain.

Diagramme de cas d'utilisation

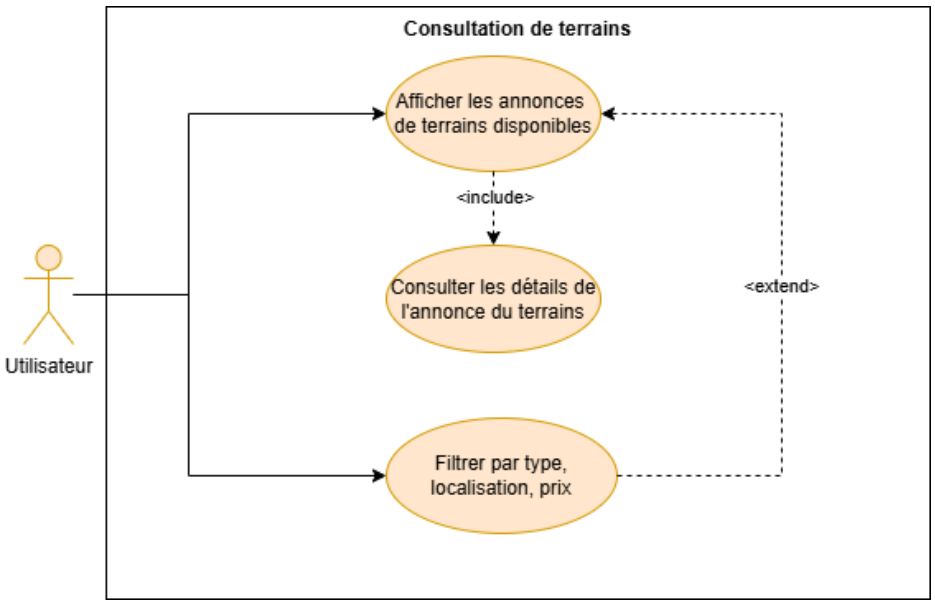

Figure V. 10 -Diagramme de cas d'utilisation : Consultation de terrains agricoles

Diagramme global de cas d'utilisation Gestion des Terrains Agricoles

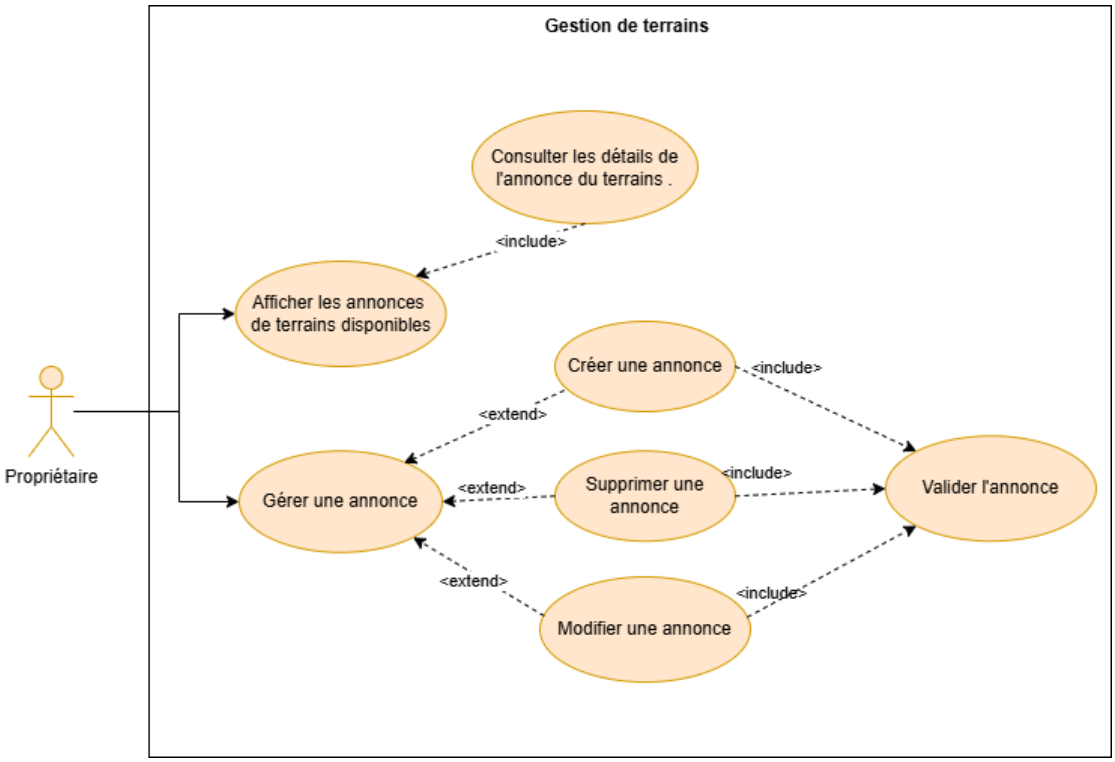

Figure V. 11 -Diagramme de cas d'utilisation : Gestion des Terrains Agricoles

2.2.4 Contexte de Gestion des Réservations.

Ce contexte permet aux utilisateurs de réserver des services, du matériel ou des terrains. Il intègre la création, la modification et l'annulation des demandes de réservation, tout en assurant un suivi détaillé de l'historique et des actions possibles sur les réservations et les commandes.

Réservation de matériel et de terrains.

- Créer, modifier ou annuler une demande ou une réservation.
- Consulter l'historique des réservations avec possibilité de voir les détails.
- Afficher la liste des commandes avec option de modification ou de suppression.

Diagramme de cas d'utilisation

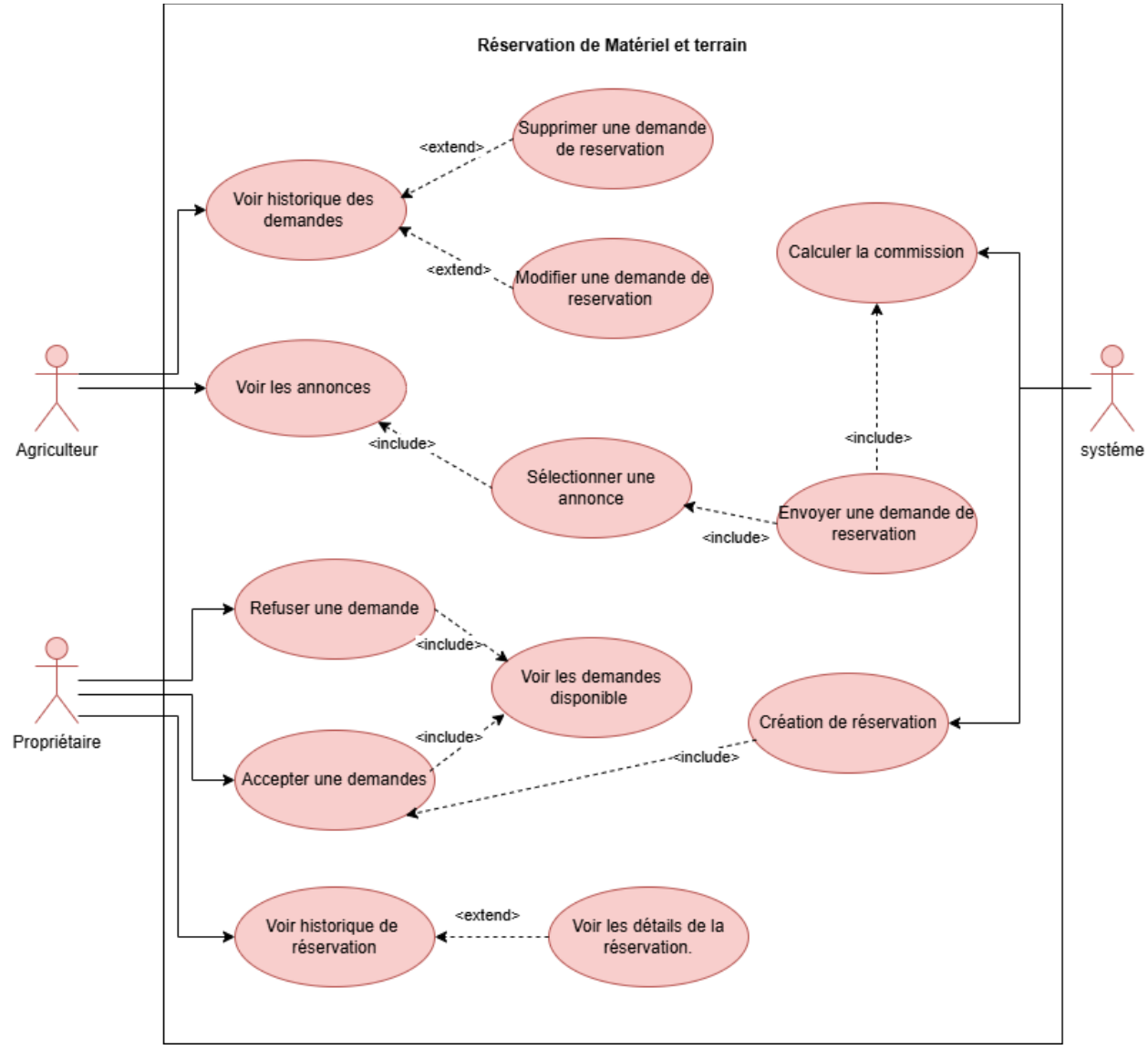

Figure V. 12 -Diagramme de cas d'utilisation : Réservation de matériel et de terrains agricole

Réservation de services.

- Créer, modifier ou annuler une demande ou une réservation.
- Consulter l'historique des réservations avec possibilité de voir les détails.
- Afficher la liste des commandes avec option de modification ou de suppression.

Diagramme de cas d'utilisation

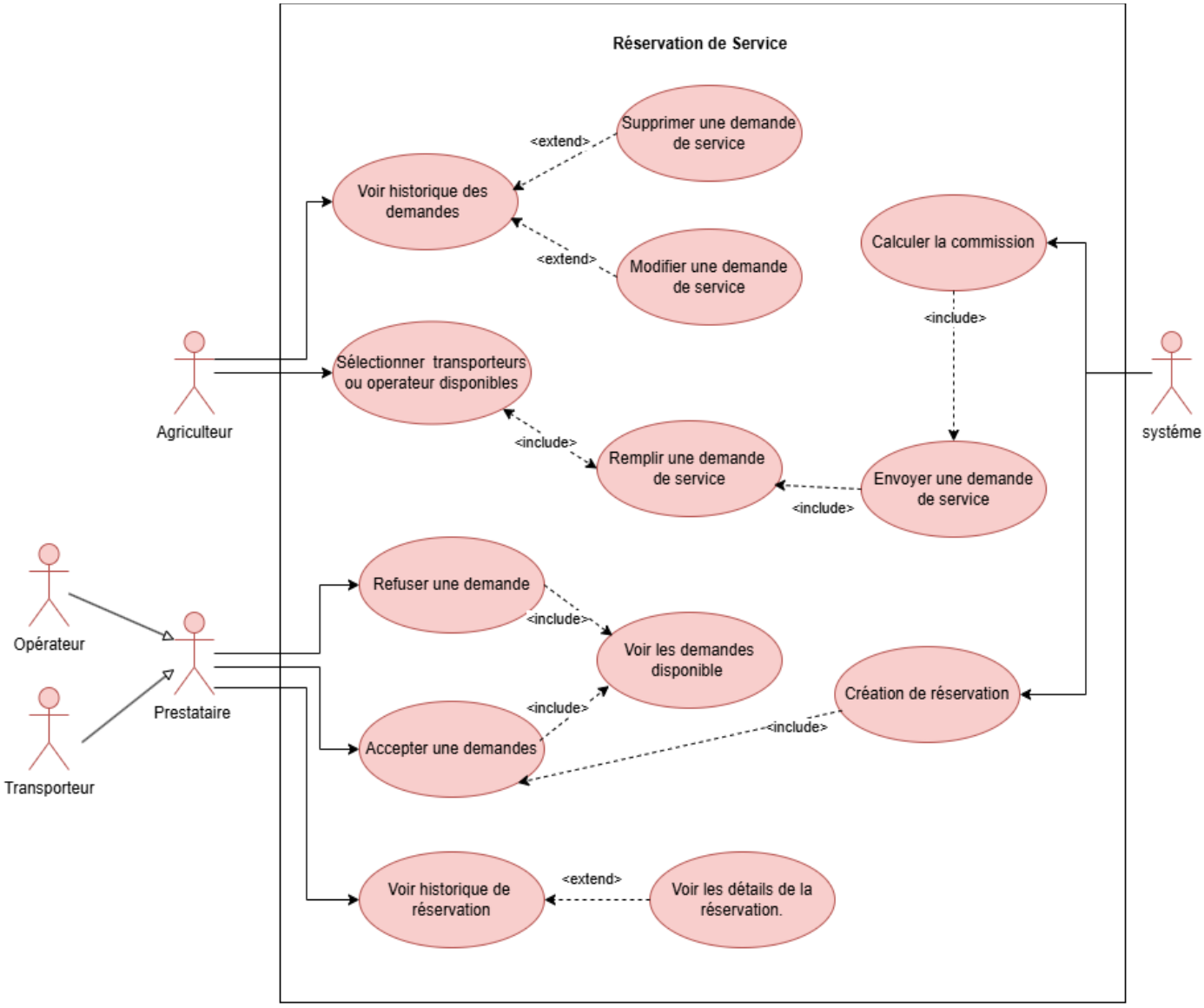

Figure V. 13 -Diagramme de cas d'utilisation : Réservation de services

2.2.5 Contexte de Notifications.

Ce contexte permet d'informer les utilisateurs en temps réel en leur envoyant des notifications liées aux différentes actions effectuées sur la plateforme, telles que les demandes, les paiements ou la fin d'un service.

Notifications liées aux demandes.

- Envoyer une notification à chaque étape d'une demande (création, annulation, etc.).

Notifications liées aux paiements.

- Informer l'utilisateur en cas de confirmation ou d'échec d'un paiement.

Notifications liées aux validations.

- Envoyer une confirmation à la fin d'un service ou d'une réservation.

Diagramme de flowcase

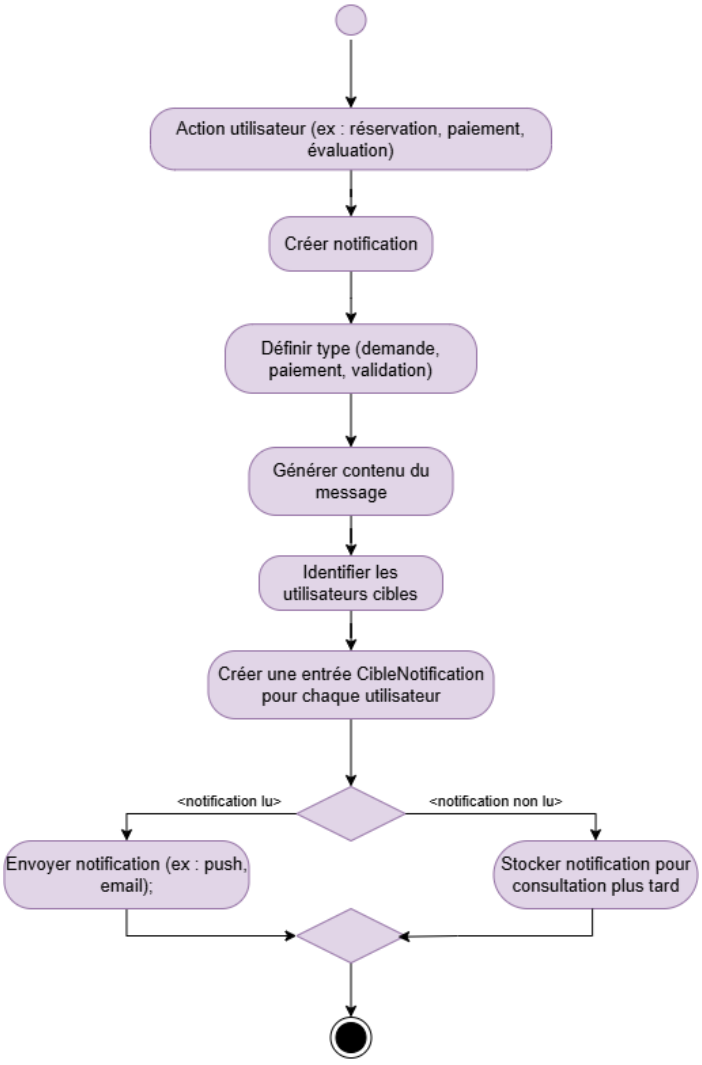

Figure V. 14 -Diagramme flowcase : Gestion des notifications

2.2.6 Contexte de Paiements.

Ce contexte encadre le processus de gestion financière entre les utilisateurs de la plateforme. Il assure le calcul automatique des frais applicables, la génération de factures et le suivi de l'état des paiements de manière sécurisée et transparente.

Gestion des paiements.

- Calculer automatiquement les frais de service et les commissions applicables.
- Générer des factures à partir des transactions réalisées.
- Suivre l'état des paiements (en attente, validés, échoués).

2.2.7 Recommandation

Ce contexte permet de recommander aux utilisateurs des terrains et matériels agricoles adaptés à leurs besoins, en s'appuyant sur des techniques d'analyse du comportement. Il repose sur trois systèmes de recommandation distincts et complémentaires : basé sur le contenu, collaboratif (user-based et item-based), et hybride.

Système de recommandation basé sur le contenu.

- Extraire les caractéristiques des annonces consultées ou appréciées.
- Comparer ces caractéristiques avec celles des annonces disponibles.
- Calculer un score de similarité entre contenus.
- Proposer des annonces similaires à celles déjà explorées par l'utilisateur.

Système de recommandation collaboratif.

a) Approche item-based :
   - Identifier les annonces fréquemment consultées ou réservées ensemble.
   - Établir des corrélations entre annonces en fonction des comportements de l'ensemble des utilisateurs.
   - Recommander des annonces proches de celles déjà sélectionnées par l'utilisateur.

b) Approche user-based:
   - Analyser le comportement global de l'utilisateur (clics, recherches, réservations).
   - Trouver d'autres utilisateurs au profil comportemental similaire.
   - Recommander des annonces appréciées par ces profils comparables.

Exemples des techniques de recommandation

1. Filtrage collaboratif basé utilisateur (User-Based)

Données de notation :

Utilisateur	Matériel A	Matériel B	Matériel C	Matériel D
Ahmed	**5**	**4**	**4**	**?**
Amina	**3**	**2**	**3**	**5**
Yacine	**5**	**4**	**4**	**4**

Table IV. 1Données de notation pour le filtrage collaboratif basé utilisateur (User-Based)

Étape 1 : Moyennes

Moyenne Ahmed : (5+4+4) /3 = 4.33

Moyenne Amina : (3+2+3+5) /4 = 3.25

Moyenne Yacine : (5+4+4+4) /4 = 4.25

Étape 2 : Similarité (Pearson) entre Ahmed et Amina

$$sim(ahmed, amina) = \frac{\sum_{p \in P}(r_{a,p} - \bar{r_a})(r_{b,p} - \bar{r_b})}{\sqrt{\sum_{p \in P}(r_{a,p} - \bar{r_a})^2}\sqrt{\sum_{p \in P}(r_{b,p} - \bar{r_b})^2}} \approx 0.31$$

Étape 3 : Prédiction

$$pred(ahmed, D) = \bar{r_a} + \frac{\sum_{b \in N} sim(a,b) \times (r_{b,p} - \bar{r_b})}{\sum_{b \in N} sim(a,b)} = 4.55$$

Résultat : Matériel D est recommandé à Ahmed avec une note estimée à 4.55.

2. Filtrage collaboratif basé sur les annonces (Item-Based)

Prédire la note d'Ahmed pour Matériel D, en se basant sur la similarité entre les annonces qu'il a déjà notées et D.

Similarité cosinus entre D et les autres :

Supposons:

- sim (D, A) = 0.9
- sim (D, B) = 0.7
- sim (D, C) = 0.6

Prédiction:

$$pred(ahmed, D) = \frac{\sum_{i \in N} sim(i,p) \cdot r_{u,i}}{\sum_{i \in N} sum(i,p)} = 4.41$$

Matériel D est recommandé avec une note estimée à 4.41.

3. Filtrage basé sur le contenu

Recommander les annonces dont les caractéristiques sont les plus proches de celles déjà vues.

Annonces vues par Ahmed:

- Tracteur A: 90 CV, 6500 DA/jour
- Tracteur B: 100 CV, 7000 DA/jour

Moyenne: 95 CV, 6750 DA/jour

Nouvelle annonce C:

- Tracteur C: 93 CV, 6800 DA/jour

Similarité (cosinus simplifié) entre profil Ahmed et annonce C :

Soit les vecteurs:

- Ahmed: [95, 6750]
- Tracteur C: [93, 6800]

Produit scalaire et Normes:

$$||ahmed|| = \sqrt{95^2 + 6750^2} \approx 6749.5$$

$$||C|| = \sqrt{93^2 + 6800^2} \approx 6800.6$$

$$sim = 0.99988$$

La similarité est très proche de 1 → l'annonce C est hautement recommandée.

4. Recommandation hybride

Formule de fusion :

$$Score\ total = 0.5 * Score\ contenu + 0.3 * Score\ user + 0.2 * Score\ item = 0.9493$$

L'annonce reçoit un score final de 0.9493 → elle est classée parmi les recommandations prioritaires pour l'utilisateur.

2.2.8 Contexte d'Évaluation

Ce contexte permet aux utilisateurs d'évaluer les services ou les prestataires avec lesquels ils ont interagi. Il favorise la transparence et la qualité des services proposés à travers un système de notation et de commentaires publics, associés aux profils ou aux annonces.

Notation

- Attribuer une note (système d'étoiles) à un utilisateur ou un service.
- Calculer automatiquement la moyenne des évaluations reçues.

Commentaries

- Rédiger un avis public pour exprimer un retour d'expérience.
- Associer automatiquement le commentaire au profil concerné ou à l'annonce correspond

3 Modélisation des Contextes Délimités (DDD Tactique)

Cette partie détaille la modélisation interne de chaque sous-système métier de l'application Agrico, selon l'approche DDD tactique. Chaque contexte correspond à un microservice et repose sur un modèle structuré contenant des entités, objets de valeur et agrégats. Les diagrammes de classes illustrent les relations internes à chaque modèle.

3.1 Contexte de Gestion des Utilisateurs

3.1.1 Modèle du Domaine

Ce modèle couvre l'inscription, la connexion, la gestion des profils et la visibilité sur la carte.

- Utilisateur : identifiant, mot de passe, nom, prénom, téléphone, photo, date de naissance.
- Adresse : localisation avec rue, ville, wilaya, code postal.
- Document : fichiers justificatifs liés au profil.

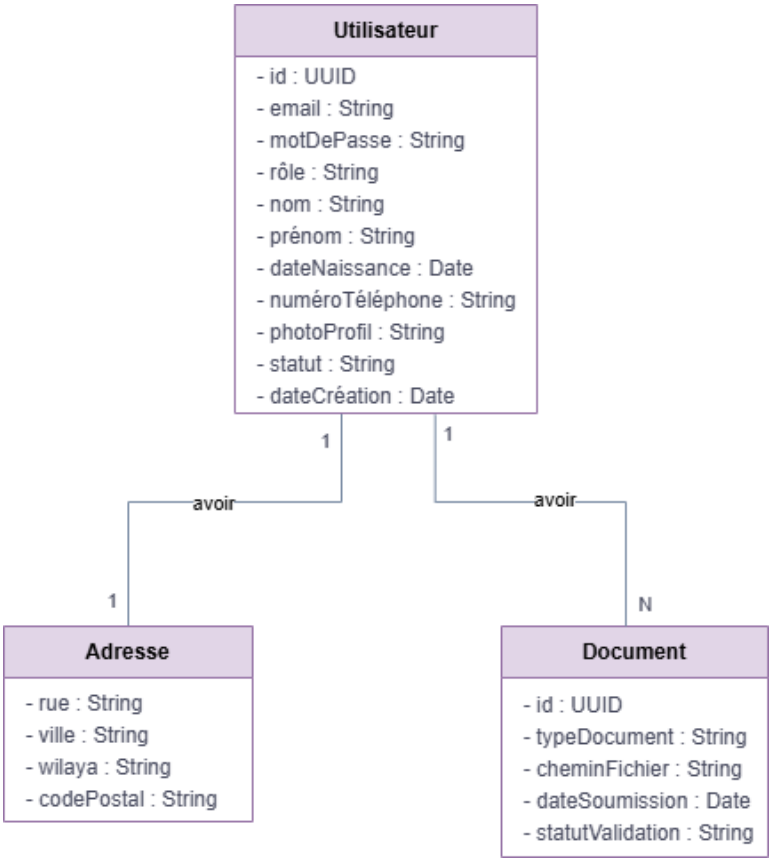

Figure V. 15 -Diagramme de classes – Gestion des Utilisateurs

3.2 Contexte de Gestion du Matériel Agricole

3.2.1 Modèle du Domaine

Ce contexte gère la création, modification et consultation des annonces de matériel.

- Équipement : type, marque, modèle, images, localisation.
- AnnonceMateriel : annonce liée à un équipement, avec prix, disponibilité, description.

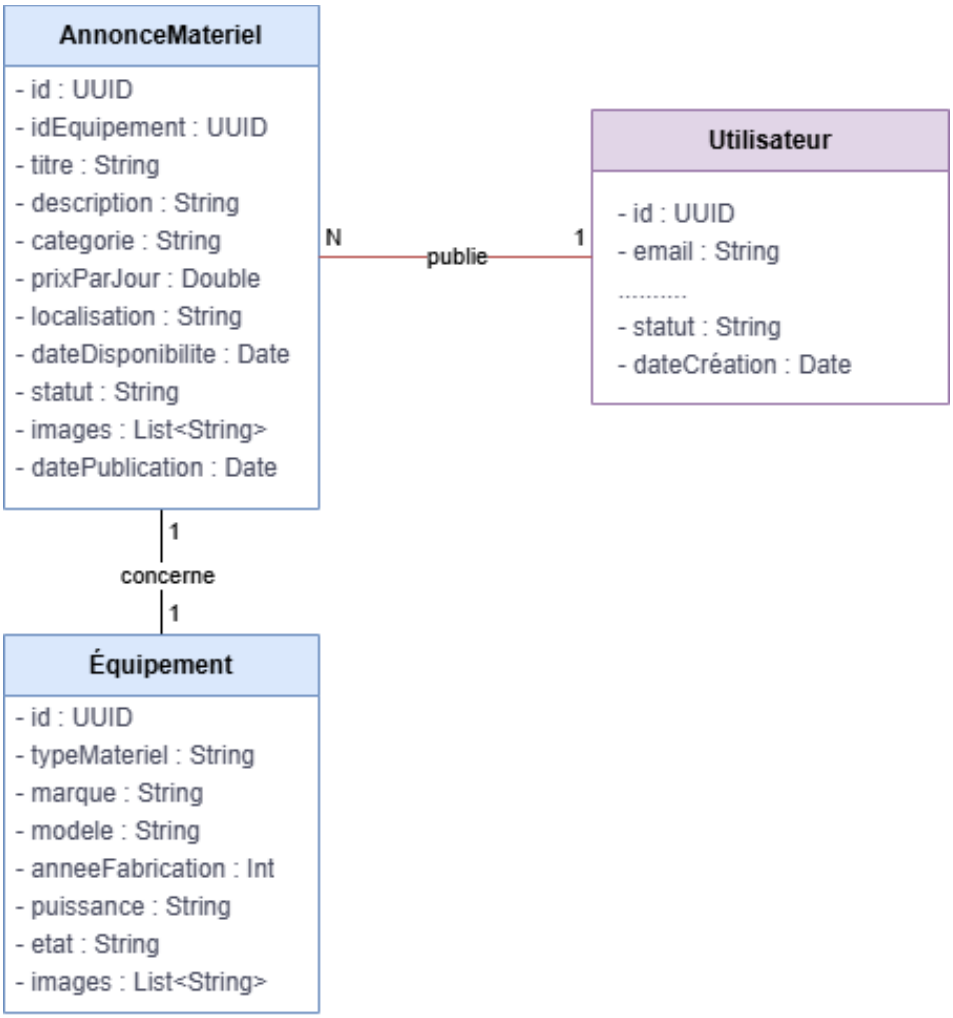

Figure V. 16 -Diagramme de classes – Gestion du Matériel Agricole

3.3 Contexte de Gestion des Terrains Agricoles

3.3.1 Modèle du Domaine

Ce modèle permet la gestion des annonces de terrains agricoles et leur consultation via une carte.

- Terrain : description, superficie, localisation, images.
- AnnonceTerrain : publication associée au terrain, avec prix et disponibilité.

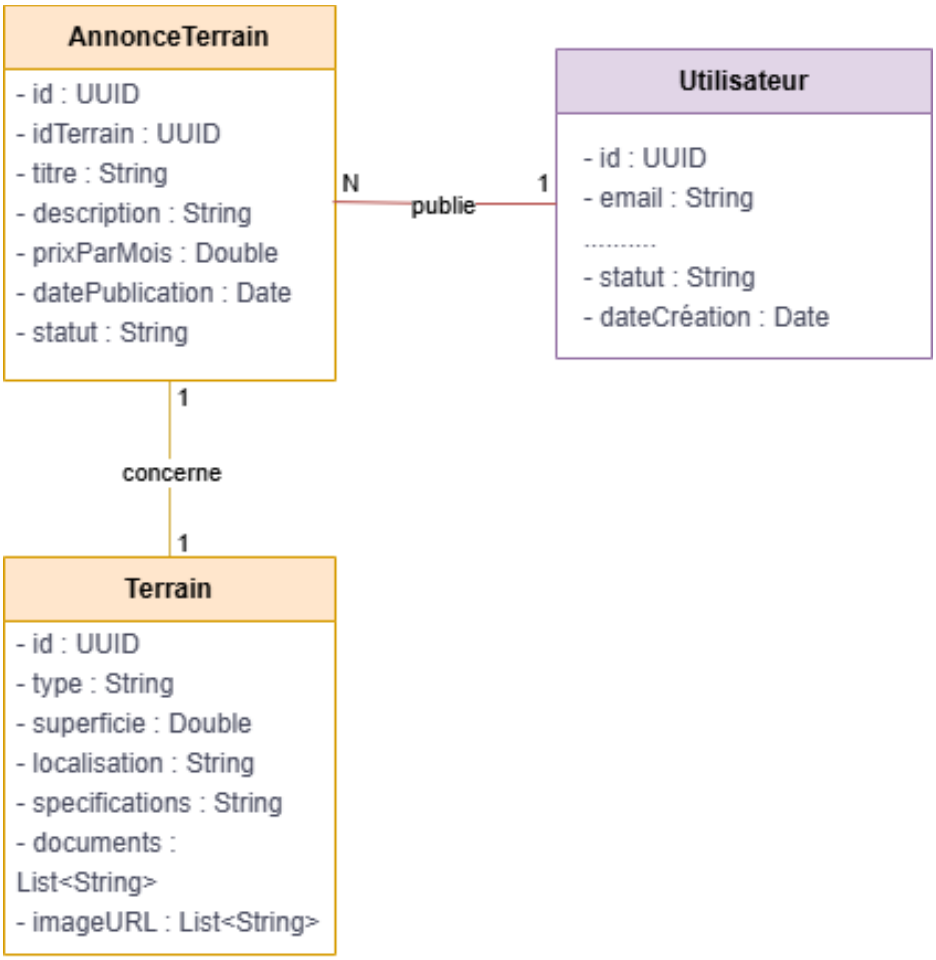

Figure V. 17 -Diagramme de classes – Gestion des Terrains Agricoles

3.4 Contexte de Notifications

3.4.1 Modèle du Domaine

Ce modèle permet de notifier les utilisateurs en temps réel sur les actions importantes.

- Notification : type (demande, paiement, validation), contenu, date, statut (lu/non lu).
- CibleNotification: identifie l'utilisateur concerné.

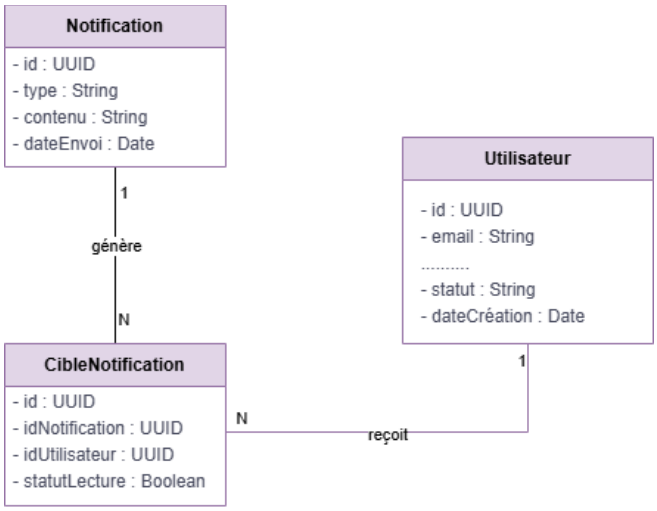

Figure V. 18 -Diagramme de classes – Système de Notifications

3.5 Contexte de Paiement

3.5.1 Modèle du Domaine

Ce modèle gère les frais de service, la génération de factures et le suivi des transactions.

- Paiement : montant, statut (en attente, validé, échoué), date, mode.
- Facture : document généré à partir d'une réservation.

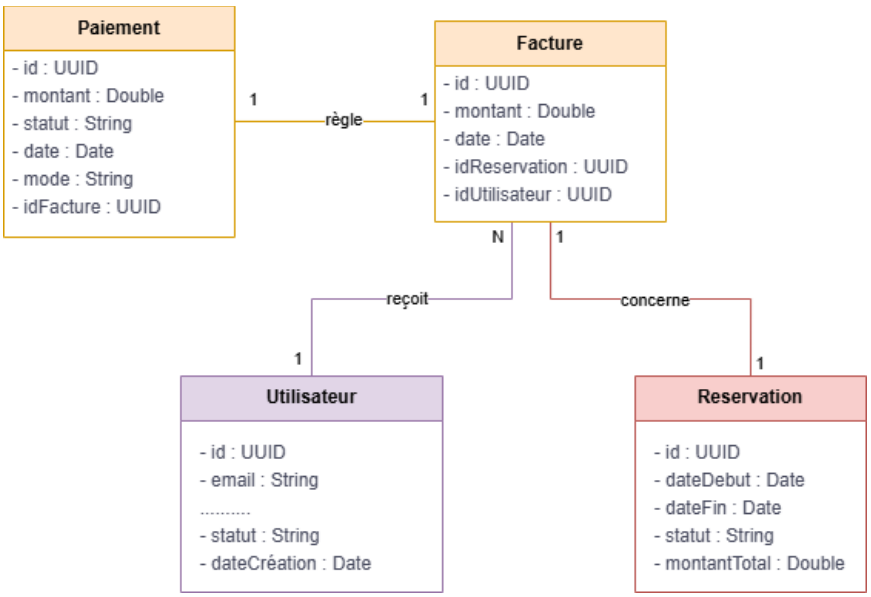

Figure V. 19 -Diagramme de classes – Système de Paiement

3.5 Contexte de Recommandation

3.5.1 Modèle du Domaine

Ce modèle propose des contenus personnalisés à l'utilisateur.

- ProfilUtilisateur : base de la recommandation.
- InteractionHistorique: clics, réservations passées.
- Suggestion : liste d'annonces ou de profils proposés.
- Score : pondération des résultats selon pertinence.

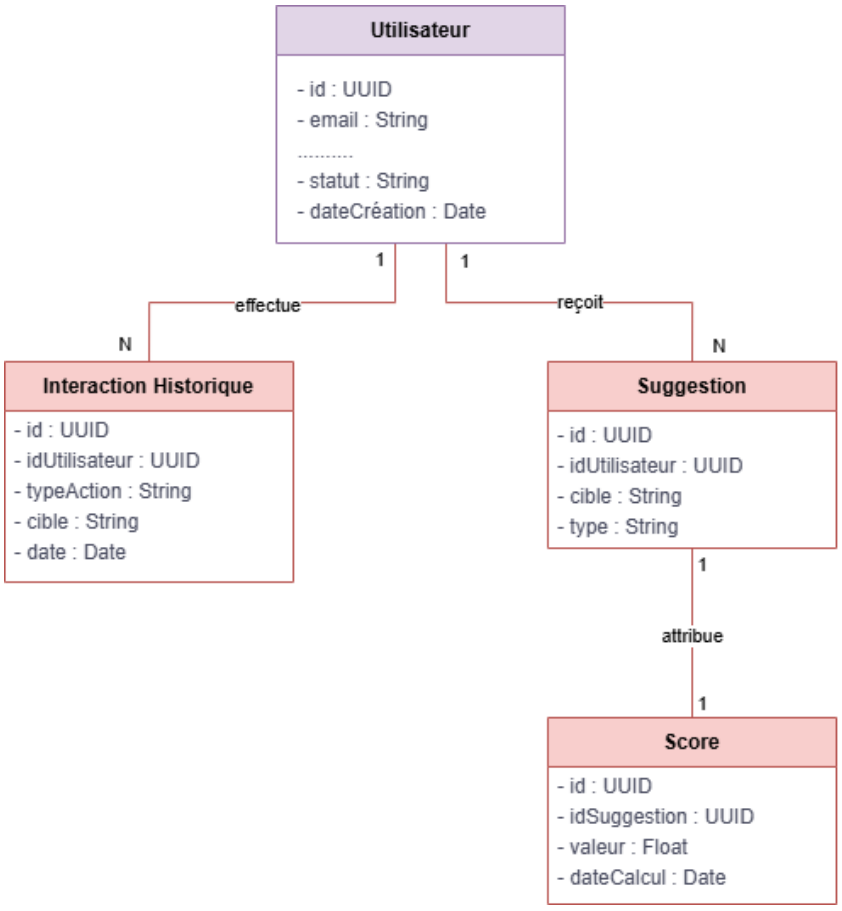

Figure V. 20 -Diagramme de classes – Système de Recommandation

3.6 Contexte d'Évaluation

3.6.1 Modèle du Domaine

Ce modèle permet de noter les prestations et de laisser des commentaires visibles publiquement.

- NoteEvaluation : note sur 5.
- Commentaire : texte de retour.

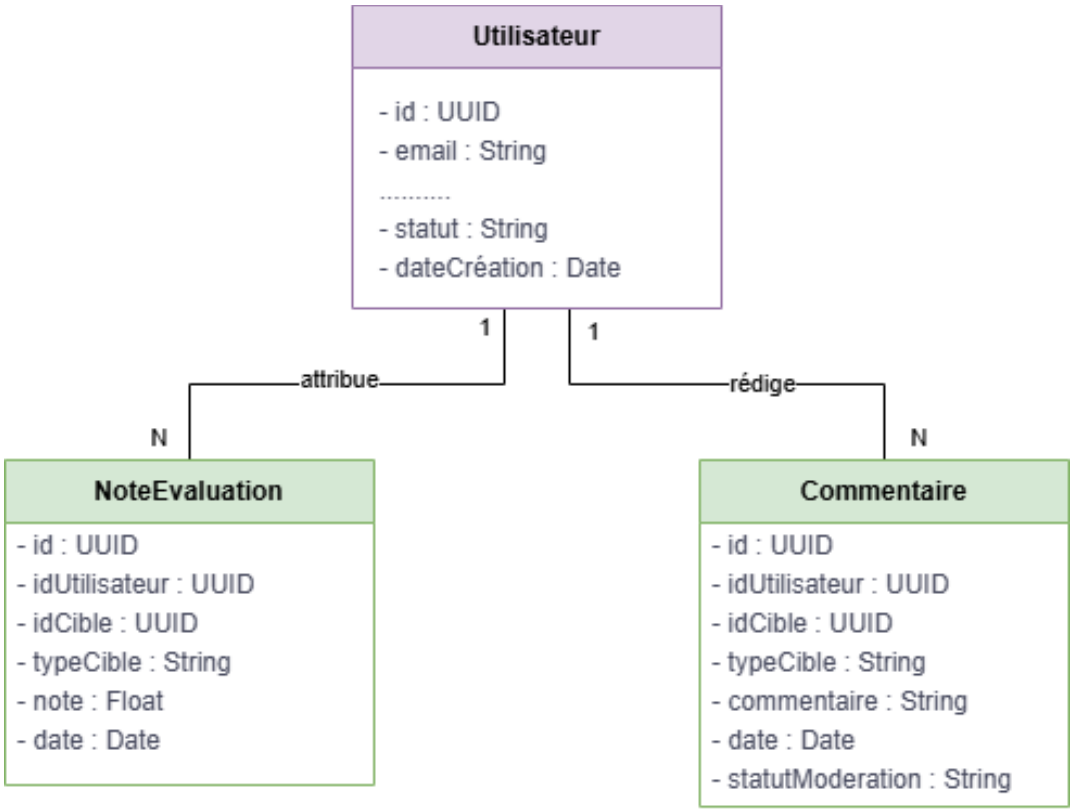

Figure V. 21 -Diagramme de classes – Système d'Évaluation

3.7 Contexte de Gestion des Réservations

3.7.1 Modèle du Domaine

Le contexte de gestion des réservations regroupe l'ensemble des processus liés à la demande et la réservation de ressources : matériels, terrains et services (opérateurs, transporteurs). Ce domaine distingue deux types de demandes:

- DemandeReservation : concerne les annonces de location de matériels ou de terrains.
- DemandeService : concerne les prestations humaines liées à l'utilisation ou au
- HistoriqueDemande : pour les modifications, refus, annulations.
- HistoriqueReservation : pour les confirmations, clôtures, litiges, paiements...

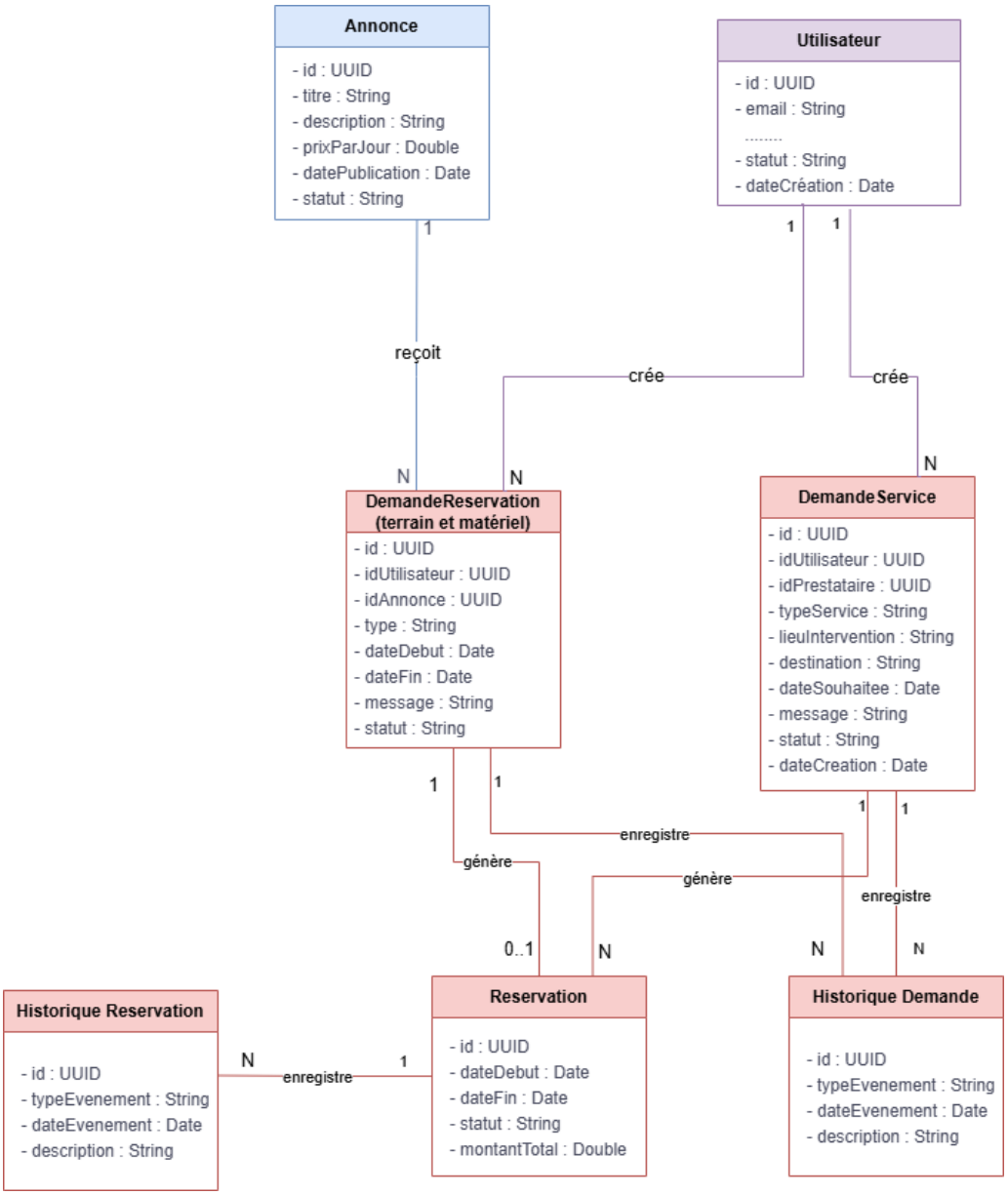

Figure V. 22 -Diagramme de classes – Gestion des Réservations

3.8 Contexte Global de l'Application Agrico

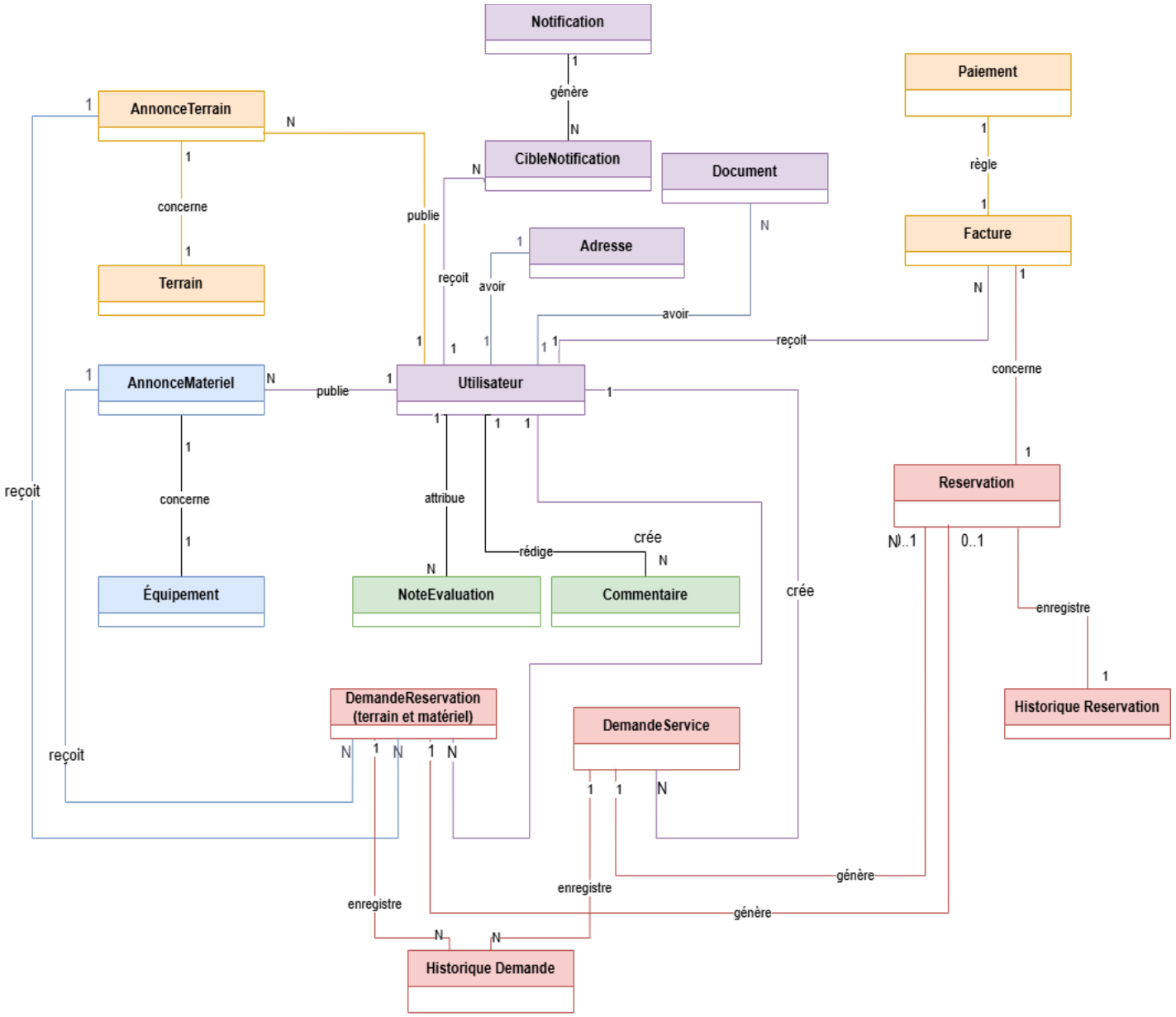

Figure V. 23 -Diagramme de classes : Vue d'ensemble des contextes de l'application Agrico

4 L'EDA et La Communication Inter-Contextes

Dans notre application, l'adoption de l'EDA permet aux divers contextes de fonctionner de manière autonome tout en collaborant efficacement. Plutôt que de s'appeler directement, les services publient des événements lorsqu'un changement d'état se produit, et d'autres services intéressés peuvent s'abonner à ces événements pour réagir en conséquence. Cette approche réduit les dépendances directes, améliore la résilience du système et facilite l'évolutivité.

Les diagrammes de séquence UML suivants illustrent les flux d'événements clés entre les différents contextes de l'application :

4.1 Flux d'Inscription et de Mise à Jour du Profil Utilisateur

Ce diagramme (Figure 1) illustre le processus d'inscription d'un nouvel utilisateur et la mise à jour de son profil. Il met en évidence l'interaction entre l'Utilisateur, le Contexte de Gestion des Utilisateurs et le Contexte de Notifications.

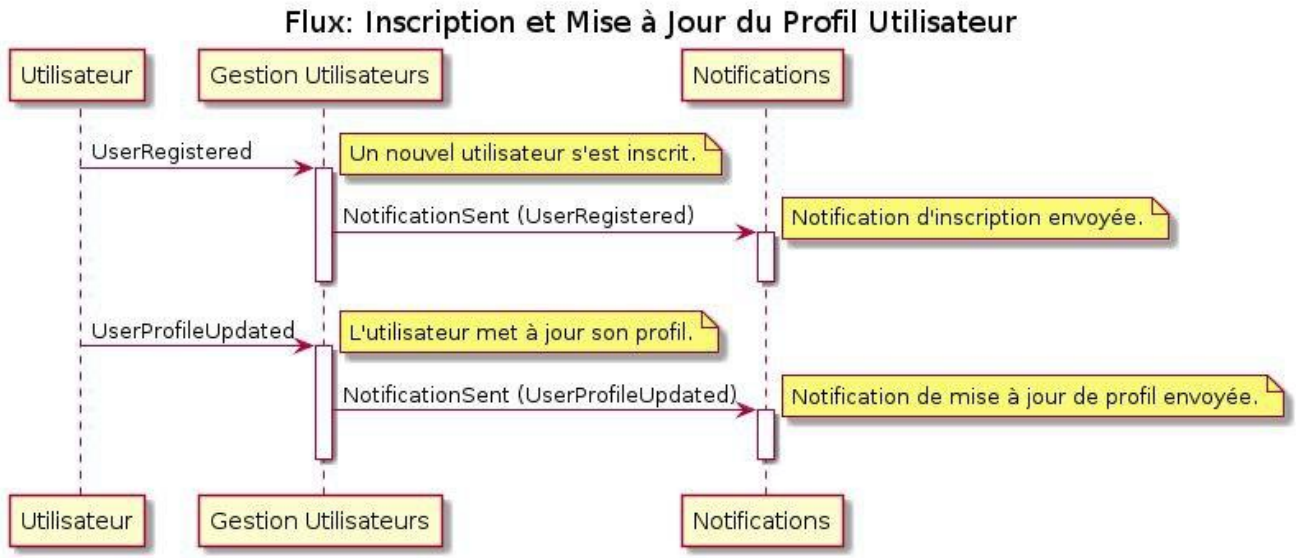

Figure V. 24 -Flux d'Inscription et de Mise à Jour du Profil Utilisateur

4.2 Flux de Demande et Acceptation de Réservation d'Équipement

Ce diagramme (Figure 2) détaille le processus de demande et d'acceptation d'une réservation d'équipement agricole. Il montre les interactions entre l'Agriculteur, le Contexte de Gestion d'Équipement, le Contexte de Gestion des Réservations et le Contexte de Notifications.

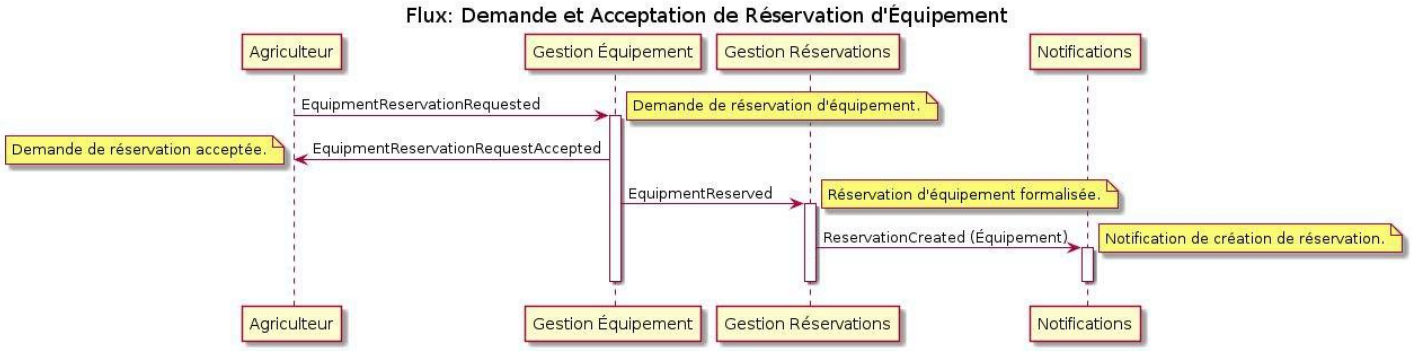

Figure V. 25 -Figure : Flux de Demande et Acceptation de Réservation d'Équipement

4.3 Flux de Demande et Acceptation de Location de Terrain

Ce diagramme (Figure 3) représente le flux de demande et d'acceptation d'une location de terrain agricole. Il implique l'Agriculteur, le Contexte de Gestion des Terrains, le Contexte de Gestion des Réservations et le Contexte de Notifications.

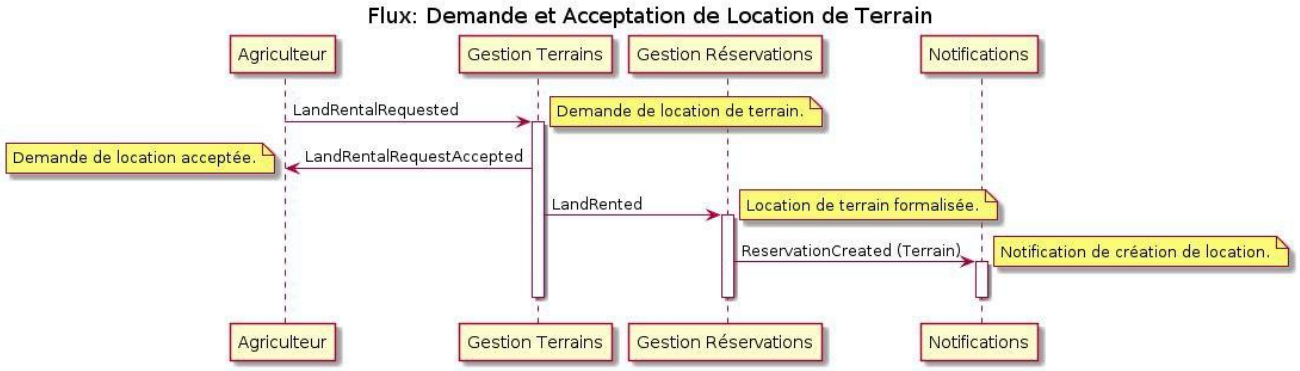

Figure V. 26 -Flux de Demande et Acceptation de Location de Terrain

4.4 Flux de Demande et Acceptation de Service (Transport/ Opération)

Ce diagramme (Figure 4) illustre les flux de demande et d'acceptation pour les services de transport et d'opération. Il met en scène l'Agriculteur, le Contexte de Gestion des Services, le Contexte de Gestion des Réservations et le Contexte de Notifications.

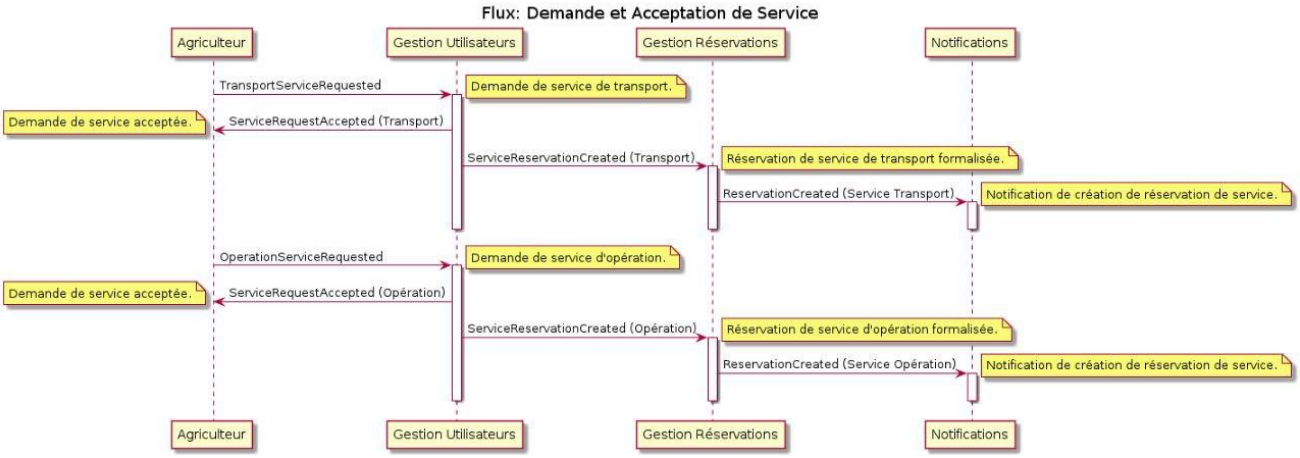

Figure V. 27 -Flux de Demande et Acceptation de Service (Transport/ Opération)

4.5 Flux de Paiement

Ce diagramme (Figure 5) décrit le processus de paiement, de la création de la commande à la notification du succès ou de l'échec du paiement. Il implique l'Agriculteur, le Contexte de Gestion des Réservations, le Contexte de Paiements et le Contexte de Notifications.

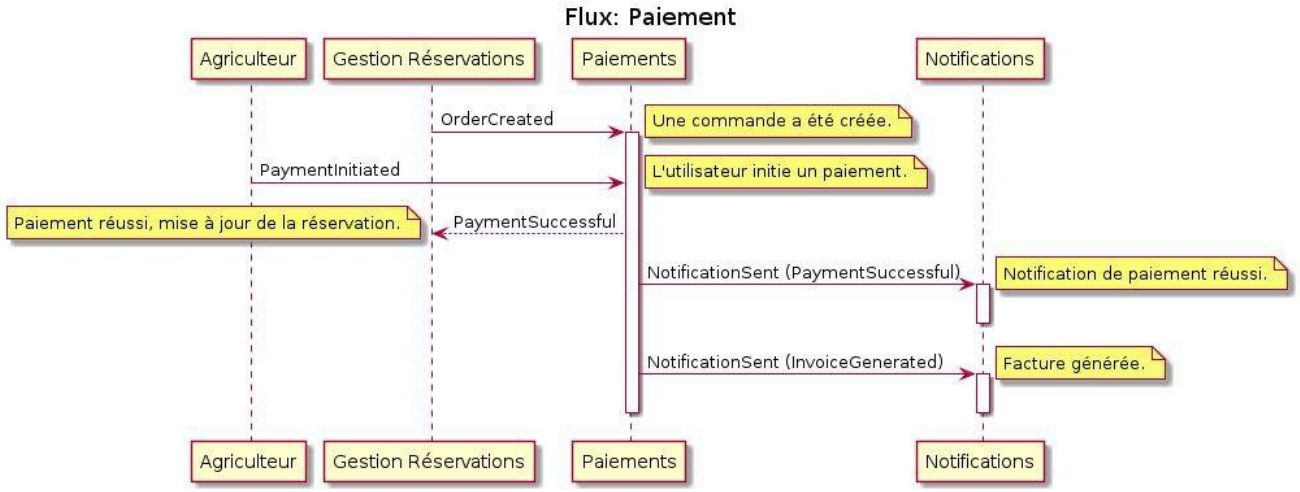

Figure V. 28 -Flux de Paiement

4.6 Flux d'Évaluation de Service

Ce diagramme (Figure 6) présente le flux d'évaluation des services après leur complétion. Il montre les interactions entre l'Agriculteur, les contextes de Gestion d'Équipement, Gestion des Terrains, Gestion des Services, le Contexte d'Évaluations et le Contexte de Notifications.

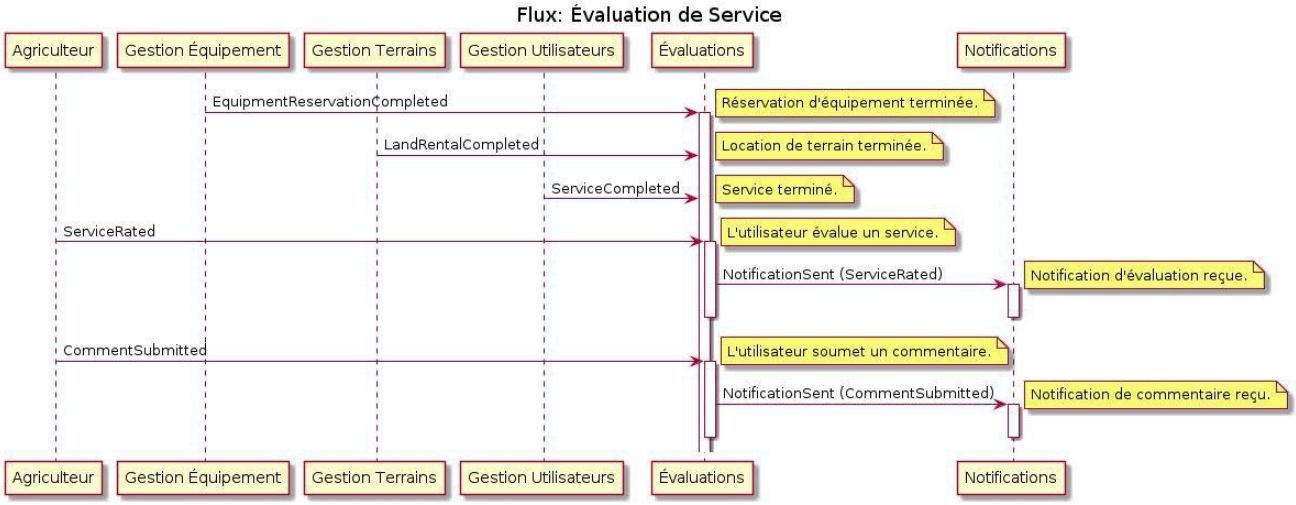

Figure V. 29 -Flux d'Évaluation de Service

4.7 Flux de Recommandation

Ce diagramme (Figure 7) illustre comment les données de complétion de services et les interactions utilisateur alimentent le moteur de recommandation. Il implique l'Agriculteur, les contextes de Gestion d'Équipement, Gestion des Terrains, Gestion des Services et le Contexte de Recommandations.

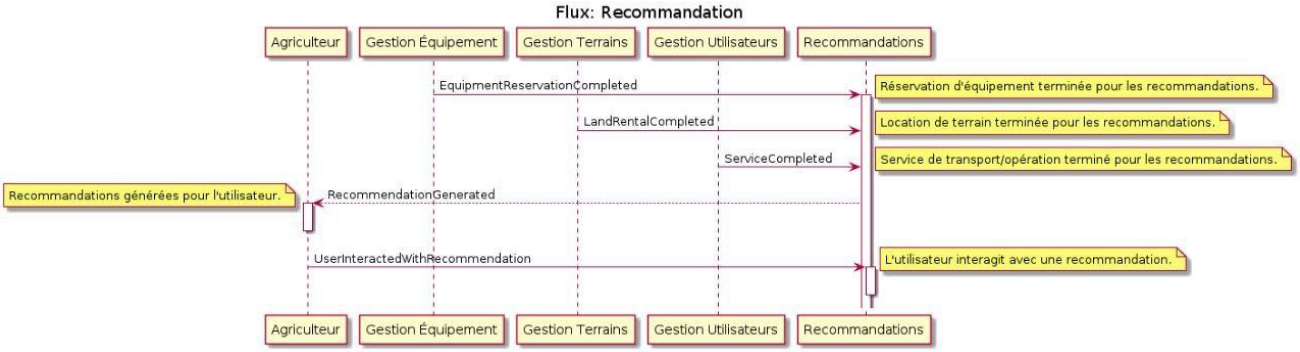

Figure V. 30 -Flux de Recommandation

5 Architecture Globale des Microservices

5.1 De la Modélisation des Contextes aux Microservices

L'architecture de l'application Agrico repose sur les principes du Domain-Driven Design (DDD). Cette approche consiste à diviser le système en contextes délimités, chacun représentant un ensemble cohérent de règles métiers et de fonctionnalités spécifiques. Cette structuration permet de mieux organiser la complexité fonctionnelle du domaine agricole et facilite la transition vers une architecture microservices.

Les contextes suivants ont été identifiés :

- **Gestion des utilisateurs** : inscription, authentification, gestion et consultation des profils.
- **Gestion du matériel agricole** : publication, modification, consultation et localisation des équipements.
- **Gestion des terrains agricoles** : création et consultation d'annonces de terrains, avec filtres et carte interactive.
- **Gestion des réservations** : traitement des demandes pour les matériels, les terrains et les services, suivi des statuts et historiques.
- **Service de notifications** : envoi automatique de notifications pour les demandes, les paiements et les validations.
- **Service de paiement** : calcul des commissions, génération de factures, suivi des paiements.
- **Service d'évaluation** : ajout de notes et commentaires, calcul de moyennes, modération.
- **Service de recommandation** : système hybride basé sur le profil et l'historique de l'utilisateur, affichage de résultats personnalisés.

Chacun de ces contextes est implémenté sous forme de microservice indépendant, possédant sa propre logique métier et sa base de données dédiée. Cette séparation logique assure une scalabilité optimale, une maintenabilité renforcée, et une flexibilité dans le déploiement et l'évolution du système.

Le schéma ci-dessous illustre cette architecture : le client mobile communique avec une API Gateway qui distribue les requêtes vers les différents microservices. Chaque service interagit avec sa base de données spécifique.

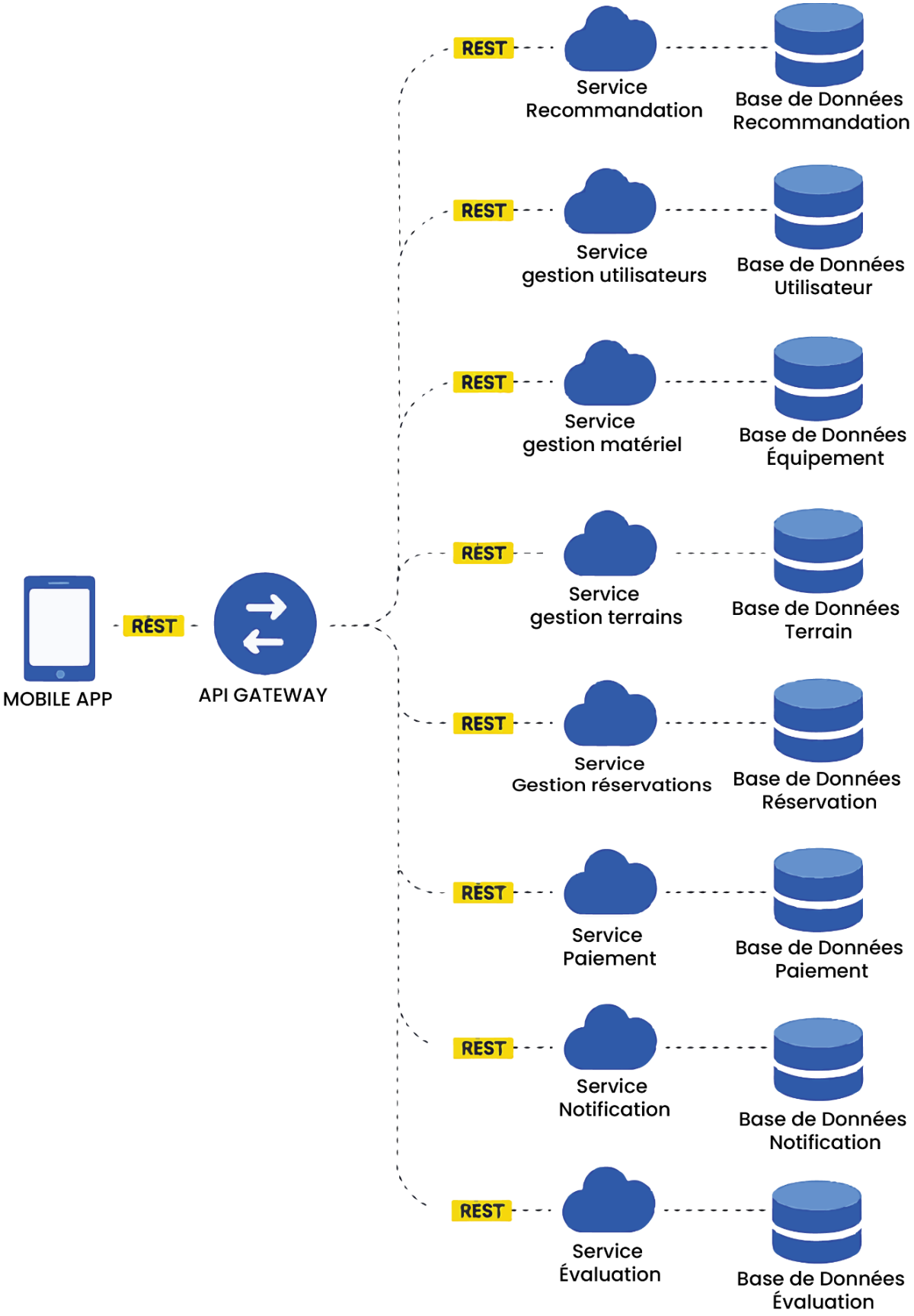

Figure V. 31 -Architecture globale des microservices de l'application Agrico

Chapitre VI – Implémentation

1 Introduction

Ce chapitre détaille les choix technologiques et les stratégies d'implémentation adoptées pour concrétiser l'architecture de l'application. Il couvre les outils et frameworks utilisés pour le développement des microservices et des interfaces utilisateur, ainsi que les approches de conteneurisation et de déploiement qui garantissent la robustesse, la scalabilité et la maintenabilité du système.

2 Outils et Technologies de Développement

Le développement de l'application a reposé sur un ensemble d'outils et de technologies modernes, choisis pour leur performance, leur flexibilité et leur adéquation avec les principes de l'architecture microservices et orientée événements.

Chaque composant a été sélectionné afin d'optimiser le cycle de développement et d'assurer une communication efficace entre les services.

2.1 Microservices avec Spring Boot

Pour l'implémentation des microservices, le framework Spring Boot a été privilégié. Spring Boot est une solution robuste et largement adoptée pour le développement d'applications Java autonomes et prêtes pour la production. Sa capacité à simplifier la configuration et le déploiement des applications en fait un choix idéal pour une architecture microservices. [29]

Figure VI. 1Logo officiel de Spring Boot

Chaque microservice a été développé en suivant une approche axée sur le domaine, en accord avec les contextes délimités identifiés lors de la phase de conception. Cela implique que chaque service est responsable d'un ensemble spécifique de fonctionnalités métier et gère ses propres données. La communication inter-services s'effectue principalement via le broker d'événements, mais des appels REST synchrones peuvent être utilisés pour des requêtes spécifiques nécessitant une réponse immédiate.

Un microservice typique développé avec Spring Boot inclut :

- **Contrôleurs REST :** Pour exposer les API du service.
- **Services Métier :** Contenant la logique métier spécifique au domaine.
- **Référentiels de Données :** Pour interagir avec la base de données du service.
- **Producteurs/Consommateurs d'Événements :** Des composants dédiés à la publication d'événements vers le broker et à la consommation d'événements pertinents provenant d'autres services.

2.2 Bases de Données Relationnelles avec PostgreSQL

Pour la persistance des données de chaque microservice, PostgreSQL a été choisi comme système de gestion de base de données relationnelle (SGBDR). PostgreSQL est reconnu pour sa robustesse, sa conformité aux standards SQL, sa richesse fonctionnelle et sa capacité à gérer des charges de travail complexes et des volumes de données importants. [30]

Figure VI. 2 -Logo de PostgreSQL

Chaque microservice possède sa propre instance de base de données PostgreSQL, renforçant ainsi le principe d'autonomie et de découplage des données, essentiel dans une architecture microservices.

2.3 Frontend Mobile avec Expo et React Native

Le développement de l'application mobile a été réalisé en utilisant Expo et React Native. Cette combinaison permet de construire des applications mobiles natives pour iOS et Android à partir d'une seule base de code JavaScript/TypeScript, accélérant ainsi le développement et la maintenance. [31]

Figure VI. 3 - Logo de React Native et Expo

2.4 Authentification et Gestion de Session avec Clerk

Pour la gestion de l'authentification et des sessions utilisateur, la solution Clerk a été intégrée. Clerk est une plateforme de gestion des utilisateurs (User Management Platform) qui offre des fonctionnalités complètes et sécurisées pour l'inscription, la connexion, la gestion des profils et des sessions. [32]

Figure VI. 4 - Logo de Clerk

2.5 Broker de Messages avec CloudAMQP (RabbitMQ)

Pour la mise en œuvre du broker d'événements, CloudAMQP a été utilisé. CloudAMQP est un service hébergé qui fournit des instances de RabbitMQ [33], un broker de messages open-source largement reconnu pour sa fiabilité et sa flexibilité. RabbitMQ implémente le protocole AMQP (Advanced Message Queuing Protocol) [34], offrant un mécanisme robuste pour la communication asynchrone entre les microservices.

Figure VI. 5 -Logo de CloudAMQP et RabbitMQ

Chaque microservice interagit avec CloudAMQP pour publier les événements qu'il génère et pour s'abonner aux événements pertinents émis par d'autres services, renforçant ainsi le découplage et la réactivité de l'architecture.

3 Conteneurisation et Déploiement

La conteneurisation et l'orchestration sont des piliers essentiels d'une architecture microservices moderne, garantissant la portabilité, la scalabilité et la gestion efficace des applications. Cette section détaille l'utilisation de podman et podman-compose pour la conteneurisation locale, et de Kubernetes pour le déploiement et l'orchestration en production.

3.1 Conteneurisation avec podman

On considère l'application spring boot (microservice) et la base de données PostgreSQL comme deux services différents. Pour que l'application et la base de données soient déployées on aura besoin de deux images conteneurs. Ceux-ci devront fonctionner sur le même réseau pour pouvoir communiquer entre eux. Nous présentons quelques étapes et commandes de conteneurisation de l'application et sa base de données.

3.1.1 Construction des images conteneurs

En premiere étape, nous avons élaboré un Dockerfile placé à la racine du répertoire de

chaque microservice. Le Dockerfile type que nous avons appliqué pour nos microservices (par exemple, user-service) est le suivant.

Dockerfile

```
1  FROM docker.io/eclipse-temurin:21-jdk
2
3  WORKDIR /app
4
5  ARG JAR_FILE=target/*.jar
6  COPY ${JAR_FILE} app.jar
7
8  EXPOSE ${SERVER_PORT:-8080}
9
10 ENV SPRING_PROFILES_ACTIVE=prod
11
12 ENTRYPOINT ["sh", "-c", "java -jar /app/app.jar
   --spring.config.additional-location=optional:/config"]
```

Figure VI. 6 -Dockerfile Type pour les applications spring-boot

Pour chaque microservice, La commande <podman build> était ensuite exécutée, en spécifiant le tag de l'image pour maintenir un contrôle sur les versions des images utilisées dans les différents environnements de développement et de test.

```
fadis@fadis-pc:~/Desktop/deploiment/microservices/users-service$ podman build -t user-service:latest .
STEP 1/7: FROM docker.io/eclipse-temurin:21-jdk
STEP 2/7: WORKDIR /app
--> Using cache 980c5ef15abf67951b47f79843d3240232d2803b2185ec9b5f6a8d2825fa14f1
--> 980c5ef15abf
STEP 3/7: ARG JAR_FILE=target/*.jar
--> Using cache ae1bec1647674b6a3a76ed866886f1adc3d087699749abbd0a4ed58bec9171b5
--> ae1bec164767
STEP 4/7: COPY ${JAR_FILE} app.jar
--> Using cache 29c2a737cfb6d2481de41c027d0e2df92db90f70daab4e16914faeeacfa309a0
--> 29c2a737cfb6
STEP 5/7: EXPOSE ${SERVER_PORT:-8080}
--> Using cache 1d7848d5ae3945169bb74bd4cb2917cf70517afaa8bd7594d6f57850e122e806
--> 1d7848d5ae39
STEP 6/7: ENV SPRING_PROFILES_ACTIVE=prod
--> Using cache a88f3596b4e7bac1896fef3ea41167126a0498ebeec2052a653fbd31e66f1607
--> a88f3596b4e7
STEP 7/7: ENTRYPOINT ["sh", "-c", "java -jar /app/app.jar --spring.config.additional-location=optional:/confi
g"]
--> Using cache 8203f91407a2231344a5090b90f6a0ca0ec6a7a1e6a01fdcecf965768720a0aa
COMMIT user-service:latest
--> 8203f91407a2
Successfully tagged localhost/user-service:latest
8203f91407a2231344a5090b90f6a0ca0ec6a7a1e6a01fdcecf965768720a0aa
fadis@fadis-pc:~/Desktop/deploiment/microservices/users-service$
```

Figure VI. 7 -La commande principale pour construire une image à partir d'un Dockerfile.

Cette procédure était répétée pour chacun de nos microservices. À l'issue de cette phase, nous disposions d'un ensemble d'images conteneurs locales, chacune encapsulant un microservice

spécifique, prêtes à être orchestrées.

```
fadis@fadis-pc:~/Desktop/deploiment/microservices$ podman image list localhost/*-service
REPOSITORY                        TAG         IMAGE ID      CREATED      SIZE
localhost/reviews-service         latest      8203f91407a2  2 hours ago  506 MB
localhost/paiment-service         latest      8203f91407a2  2 hours ago  506 MB
localhost/recommendation-service  latest      8203f91407a2  2 hours ago  506 MB
localhost/notification-service    latest      8203f91407a2  2 hours ago  506 MB
localhost/terrain-service         latest      8203f91407a2  2 hours ago  506 MB
localhost/reservation-service     latest      8203f91407a2  2 hours ago  506 MB
localhost/equipment-service       latest      8203f91407a2  2 hours ago  506 MB
localhost/user-service            latest      8203f91407a2  2 hours ago  506 MB
```

Figure VI. 8 -Ensemble d'images conteneurs locales

3.1.2 Orchestration Locale avec podman-compose

Une fois chaque microservice encapsulé dans son image conteneur respective, l'étape suivante consistait à orchestrer ces composants pour qu'ils puissent interagir et former une application fonctionnelle dans notre environnement de développement local.

Nous créerons un fichier podman-compose.yml pour définir les services qui composent l'application, et compléter/surcharger certaines des propriétés définies dans leurs Dockerfiles respectives. Voici un exemple de notre fichier podman-compose.yaml, illustrant la configuration de plusieurs microservices et de leurs bases de données associées.

podman-compose.yml

```yaml
version: '3.8'

services:
  user_db:
    image: docker.io/postgres
    container_name: user_db
    environment:
      POSTGRES_DB: userdb
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: admin
      PGDATA: /var/lib/postgresql/data
    volumes:
      - user_db_volume:/var/lib/postgresql/data
    ports:
      - 5434:5432
    networks:
      - agrico-net
    restart: always

  user_service:
    image: localhost/user-service:latest
    container_name: user_service
    ports:
      - 8004:8080

    environment:
      - DB_URL=jdbc:postgresql://user_db:5432/userdb
      - DB_USER=admin
      - DB_PASSWORD=admin
      - SERVER_PORT=8080
```

Figure VI. 9 -Partie du fichier podman-compose.yaml

Pour démarrer les application conteneuri en arrière-plan (mode détaché), nous utilisions la commande <podman-compose up-d>.

Après le démarrage, nous vérifiions l'état de tous les conteneurs pour nous assurer qu'ils étaient bien en cours d'exécution. La commande <podman-compose ps> nous fournissait un aperçu clair de l'état de chaque service :

```
fadis@fadis-pc:~/Desktop/deploiment/microservices$ podman-compose ps
podman-compose version: 1.0.6
['podman', '--version', '']
using podman version: 4.9.3
podman ps -a --filter label=io.podman.compose.project=microservices
CONTAINER ID  IMAGE                                COMMAND     CREATED      STATUS         PORTS                   NAMES
33ac7132b08e  docker.io/library/postgres:latest  postgres    4 hours ago  Up 19 seconds  0.0.0.0:5434->5432/tcp  user_db
a6c9b9fb1551  localhost/user-service:latest                   4 hours ago  Up 15 seconds  0.0.0.0:8004->8080/tcp  user_service
```

Figure VI. 10 -État des services (conteneurs) en cours d'execution

Cette approche a considérablement simplifié la gestion de notre environnement de développement, et elle a jeté les bases solides pour la transition vers un environnement Kubernetes.

3.2 Déploiement des Microservices

Après avoir maîtrisé la conteneurisation individuelle et l'orchestration locale avec Podman Compose, l'étape suivante de déploiement a été de préparer nos microservices (ou bien conteneurs) pour un environnement Kubernetes.

3.2.1 Creation des Manifestes Kubernetes

Le fichier kubernetes-manifests.yaml contenait des définitions pour chaque service et base de données, incluant les Deployments (pour gérer les Pods), les Services (pour l'accès réseau aux Pods), et les PersistentVolumeClaims (pour la persistance des données des bases de données).

- **Services**

kubernetes-manifests.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    io.podman.compose.project: podman-compose
    io.podman.compose.service: users-service
  name: users-service
spec:
  ports:
    - name: "8080"
    port: 8080
    targetPort: 8080
  selector:
    io.podman.compose.project: podman-compose
    io.podman.compose.service: users-service
    status: loadBalancer: {}
```

Figure VI. 11 -Service - kubernetes-manifests.yaml

- **Deployments**

```yaml
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    io.podman.compose.project: podman-compose
    io.podman.compose.service: users-service
  name: users-service
spec:
  replicas: 1
  selector:
    matchLabels:
      io.podman.compose.project: podman-compose
      io.podman.compose.service: users-service
    strategy: {}
    template:
      metadata:
        creationTimestamp: null
          labels:
            io.podman.compose.project: podman-compose
            io.podman.compose.service: users-service
        specs:
          containers:
            - image: users-service:latest
            name: users-service
```

Figure VI. 12 -Deloyment - kubernetes-manifests.yaml

3.2.2 Déploiement sur le Cluster Kubernetes Local

Une fois les manifestes Kubernetes vérifiés, le déploiement de notre application sur le cluster Kubernetes local était une opération directe. Nous avons utilisé l'outil en ligne de commande Kubectl pour interagir avec notre cluster local.

On utilise la commande kubectl apply-f Cette commande instruit Kubernetes de créer ou de mettre à jour les ressources définies dans le fichier (Deployments, Services, etc.).

```
service/user_service created
deployment.apps/user_service created
service/user_db created
deployment.apps/user_db created
service/equipment_service created
deployment.apps/equipment_service created
service/equipment_db created
deployment.apps/equipment_db created
service/terrain_service  created
deployment.apps/terrains_db created
```

Figure VI. 13 -La création des ressources Kubernetes avec kubectl apply

3.2.3 Vérification du Déploiement

La commande kubectl get pods nous a permis de lister tous les Pods en cours d'exécution et de vérifier leur statut.

```
$ kubectl get pods
NAME                                          READY   STATUS    RESTARTS   AGE
db-gestion-utilisateurs-6c7b8d9f-abcde        1/1     Running   0          5m
db-gestion-materiel-5a6b7c8d-fghij            1/1     Running   0          5m
db-gestion-terrains-7d8e9f0a-klmno            1/1     Running   0          5m
db-gestion-reservations-8e9f0a1b-pqrs         1/1     Running   0          5m
db-gestion-paiements-9f0a1b2c-tuvw            1/1     Running   0          5m
db-gestion-evaluations-0a1b2c3d-xyza          1/1     Running   0          5m
db-gestion-recommandations-1b2c3d4e-bcde      1/1     Running   0          5m
db-gestion-notifications-2c3d4e5f-fghi        1/1     Running   0          5m
gestion-utilisateurs-service-9d8e7f6g-klmno   1/1     Running   0          4m
gestion-materiel-service-8h7g6f5e-pqrst       1/1     Running   0          4m
gestion-terrains-service-7i6h5g4f-uvwx        1/1     Running   0          4m
gestion-reservations-service-6j5i4h3g-yzab    1/1     Running   0          4m
gestion-paiements-service-5k4j3i2h-cdef       1/1     Running   0          4m
gestion-evaluations-service-4l3k2j1i-ghij     1/1     Running   0          4m
gestion-recommandations-service-3m2l1k0j-klmn 1/1     Running   0          4m
gestion-notifications-service-2n1m0l9k-opqr   1/1     Running   0          4m
```

Figure VI. 14 -Vérification des pods

4 Présentation du système

4.1 Espace Connexion et Inscription

Cet espace permet à l'utilisateur d'accéder à la plateforme.

- Un utilisateur déjà inscrit peut se connecter à son compte avec son e-mail et son mot de passe.
- Un nouvel utilisateur peut créer un compte en remplissant un formulaire avec ses informations personnelles.
- Après l'inscription, un e-mail de vérification est envoyé. Le compte n'est activé qu'après avoir cliqué sur le lien reçu.

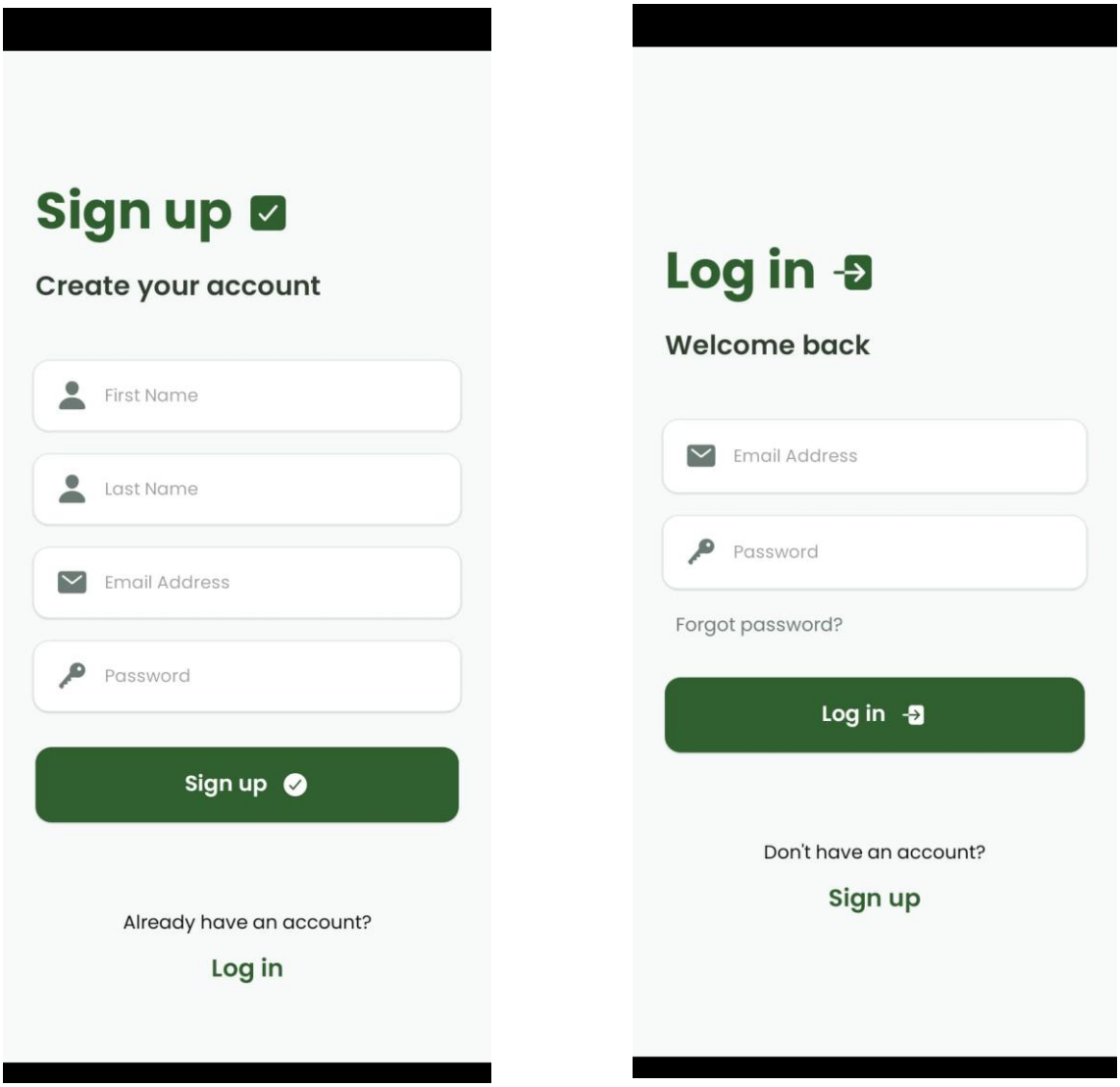

Figure VI. 15 -Interface de la page de connexion et page d'inscription

4.2 Page d'Accueil

Après la connexion, l'utilisateur est redirigé vers la page d'accueil de l'application, qui présente les informations principales de manière simple et rapide.

- La page d'accueil montre les données importantes de la plateforme.
- L'utilisateur peut voir le nombre total d'équipements, de terrains, de travailleurs et de demandes.
- Un code promo est mis en avant pour encourager l'utilisation de l'application.
- Un menu situé en bas de l'écran permet de naviguer vers les autres pages comme les équipements, les terrains ou les travailleurs.

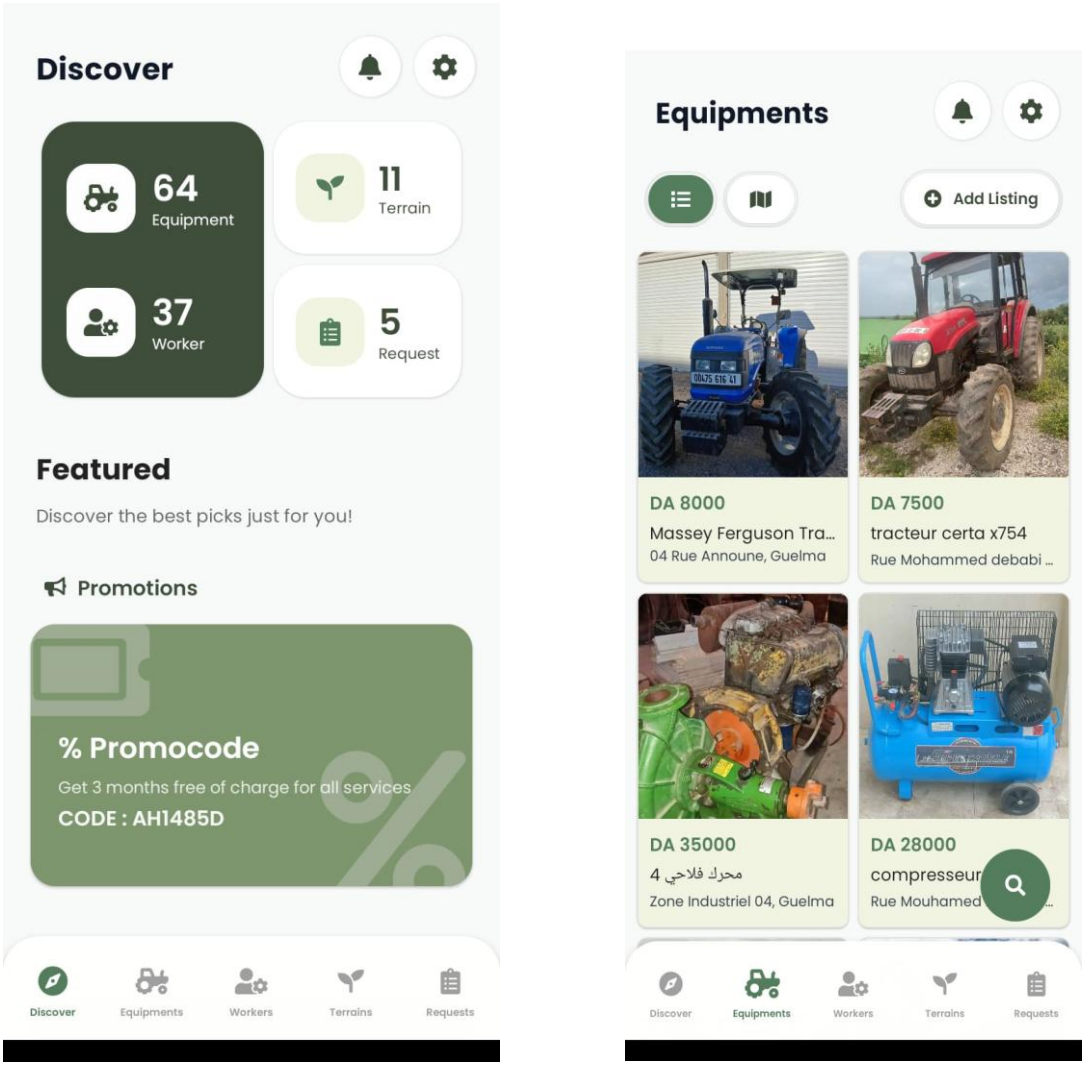

Figure VI. 16 -Page d'accueil de l'application

4.3 Page des Équipements

Cette page permet à l'utilisateur de consulter les annonces de matériel agricole disponibles.

- L'utilisateur accède à cette page en cliquant sur l'icône "Equipments" dans le menu en bas.
- La liste des équipements à louer est affichée sous forme de cartes contenant une image, un prix, une courte description et l'adresse.
- Un bouton permet d'ajouter une nouvelle annonce.
- L'utilisateur peut changer l'affichage entre vue liste et vue carte.
- En cliquant sur une annonce, il accède à la page de détail du matériel.
- Il peut alors faire une demande de réservation.

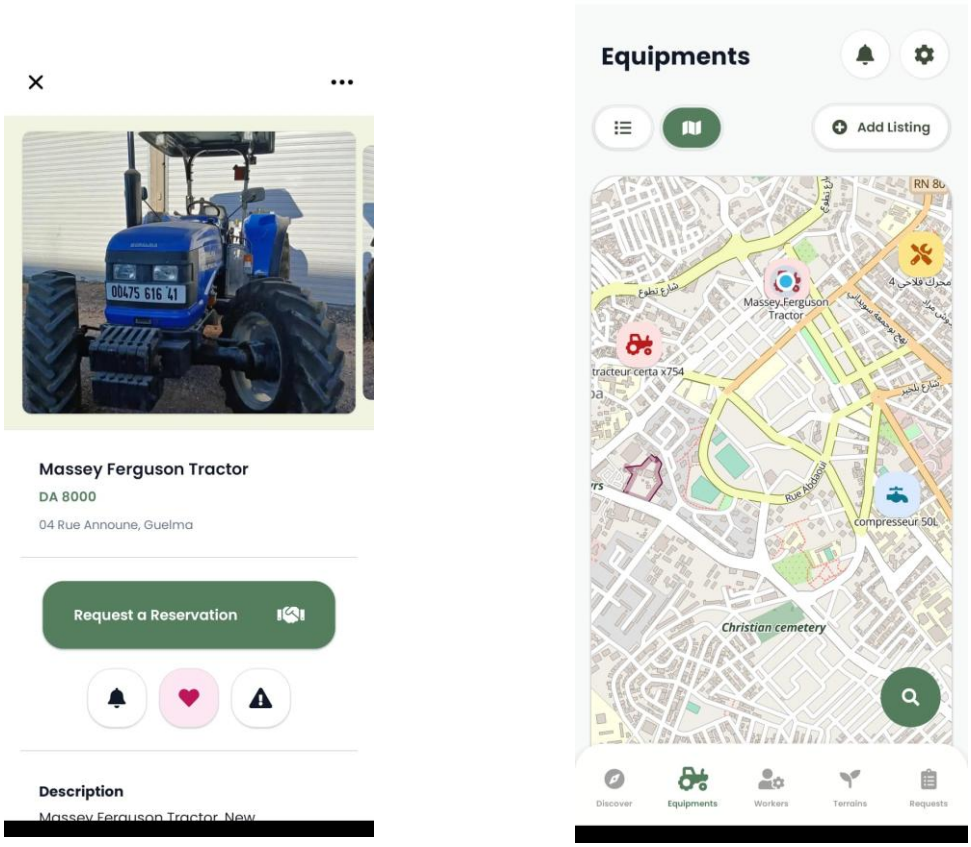

Figure VI. 17 -Page des équipements et détails d'un matériel

4.4 Page de Création d'une Annonce de Matériel

Cette page permet à l'utilisateur de publier une nouvelle annonce pour un équipement agricole.

- Il peut ajouter des photos du matériel.
- Il saisit une description, un titre, une catégorie et un prix.
- Il indique la localisation du matériel à l'aide d'une carte interactive.
- Il peut utiliser sa position actuelle ou entrer une adresse manuellement.
- Il choisit une date de disponibilité.
- Enfin, il clique sur le bouton Post pour publier l'annonce.

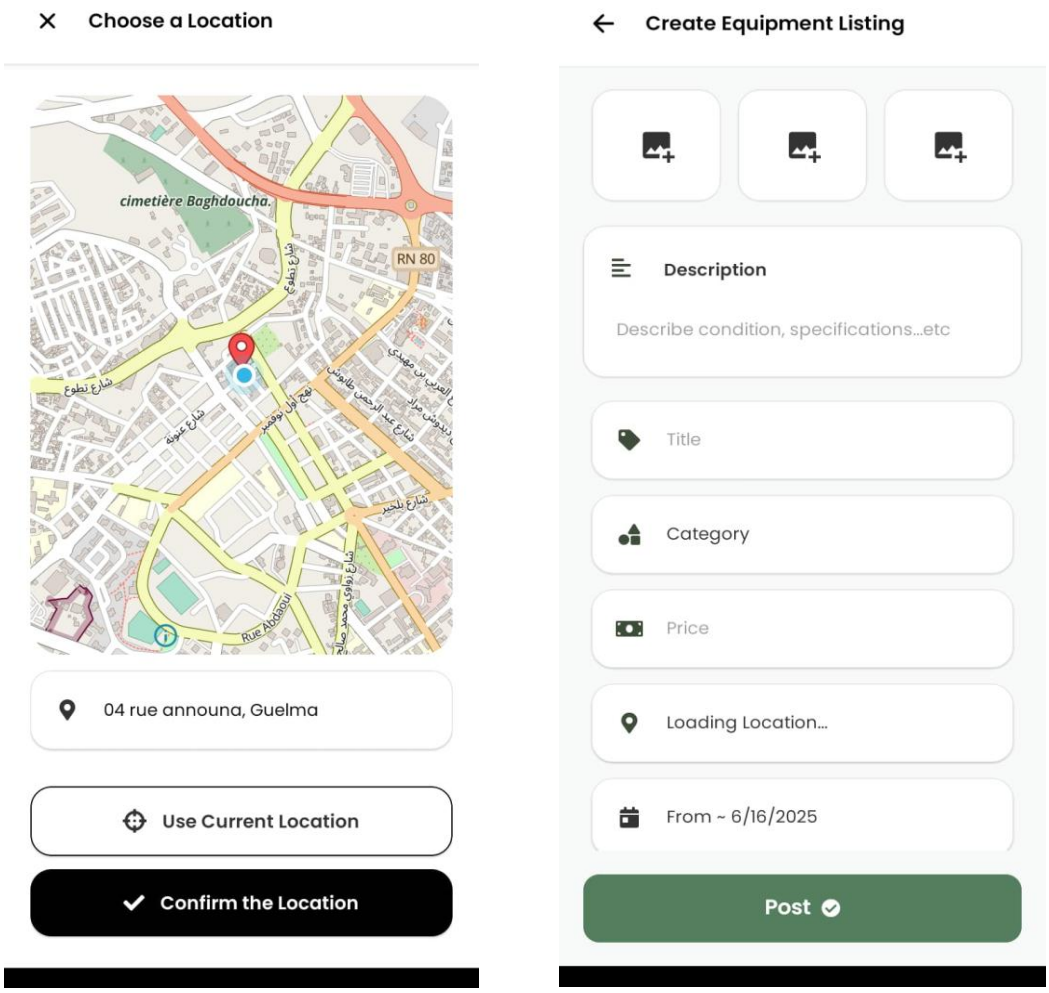

Figure VI. 18 -Formulaire de création d'une annonce de matériel

4.5 Page des Terrains

Cette page permet à l'utilisateur de consulter et gérer les annonces de terrains agricoles.

- L'utilisateur accède à cette page en cliquant sur l'icône "Terrains" dans le menu du bas.
- Les terrains disponibles sont affichés sous forme de cartes avec photo, prix, description et localisation.
- Un bouton permet d'ajouter une nouvelle annonce de terrain.
- L'utilisateur peut changer l'affichage entre vue liste et vue carte.
- En cliquant sur une annonce, il est redirigé vers la page de détail du terrain.
- Il peut alors faire une demande de réservation.

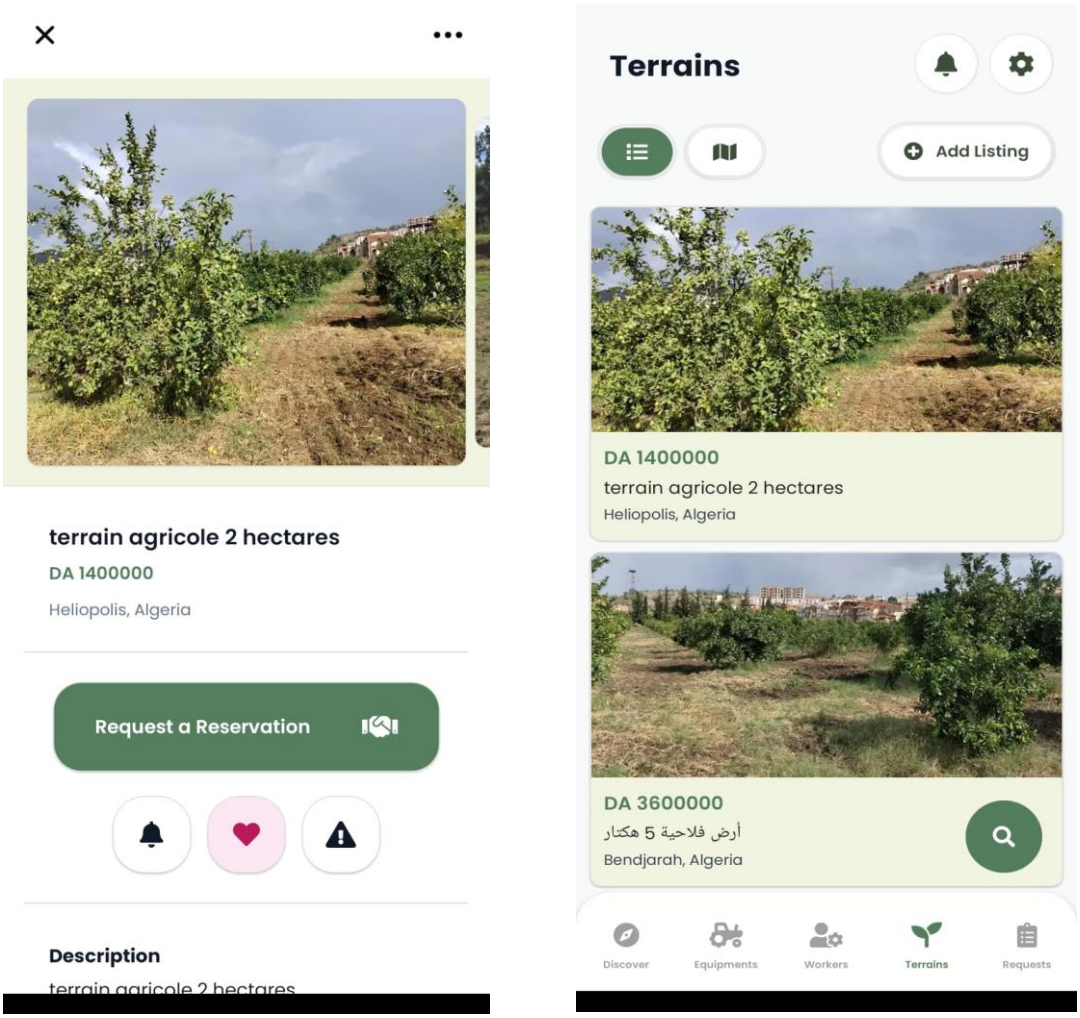

Figure VI. 19 -Page des terrains et détails d'un terrain

4.6 Page de Création d'une Annonce de Terrain

Cette page permet à l'utilisateur de publier une nouvelle annonce pour un terrain agricole.

- Il peut ajouter des photos du terrain.
- Il saisit une description, un titre, le type de terrain, les spécifications, les documents et la superficie.
- Il indique la localisation du terrain à l'aide d'une carte interactive.
- Il peut utiliser sa position actuelle ou entrer une adresse manuellement.
- Enfin, il clique sur le bouton Post pour publier l'annonce.

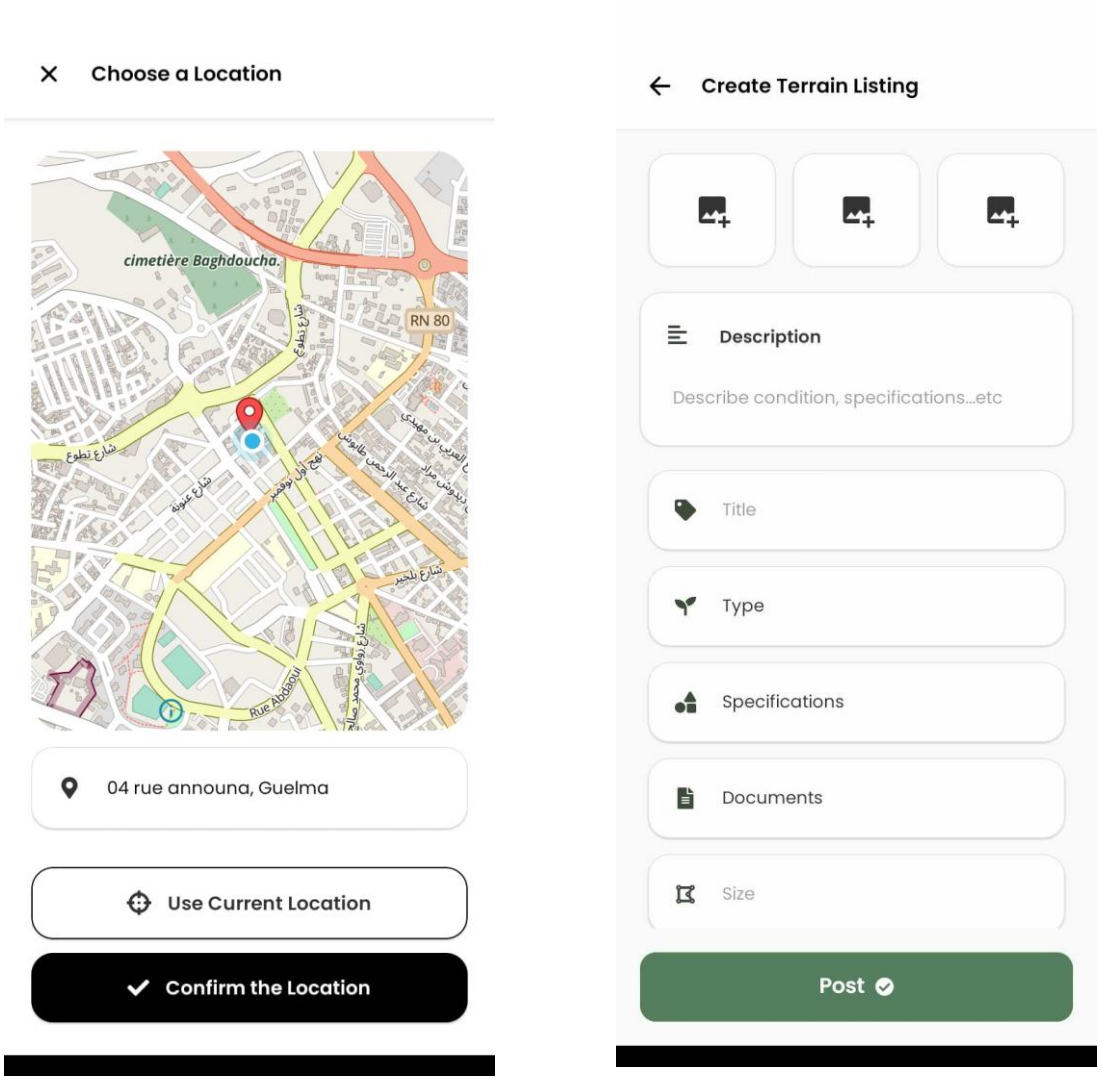

Figure VI. 20 -Formulaire de création d'une annonce de terrain

4.7 Page des Travailleurs

Cette page permet à l'utilisateur de consulter la liste des opérateurs et transporteurs disponibles sur la plateforme.

- L'utilisateur accède à cette page en cliquant sur l'icône "Workers" dans le menu en bas.
- La liste affiche les noms, rôles (équipements ou transport) et notes des travailleurs.
- Il peut activer ou désactiver son propre statut de disponibilité (bouton "Inactive").
- Il est possible de basculer entre l'affichage en liste et carte.
- Un bouton de recherche permet de filtrer les profils visibles.

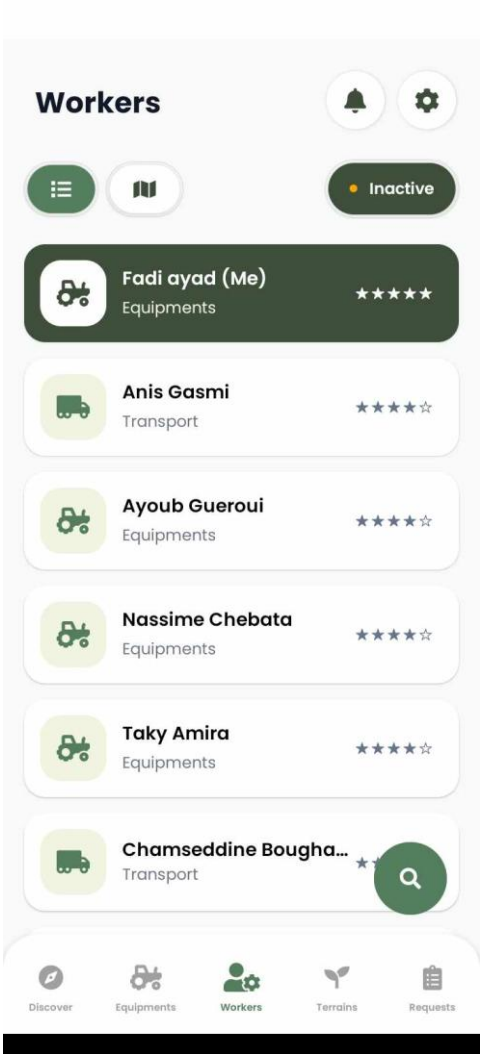

Figure VI. 21 -Page de consultation des travailleurs

4.8 Page des Demandes de Réservation

Cette page permet à l'utilisateur de suivre toutes les demandes liées aux services proposés ou réservés.

- L'utilisateur accède à cette page en cliquant sur l'icône "Requests" dans le menu en bas.
- Il peut visualiser les demandes entrantes (Incoming) ou sortantes (Outgoing).
- Chaque demande affiche le type de service (ex. terrain), la date, l'état (en attente, accepté, etc.), et un bouton pour consulter les détails.
- Dans la fiche détaillée, l'utilisateur peut voir les informations du service demandé et du propriétaire.

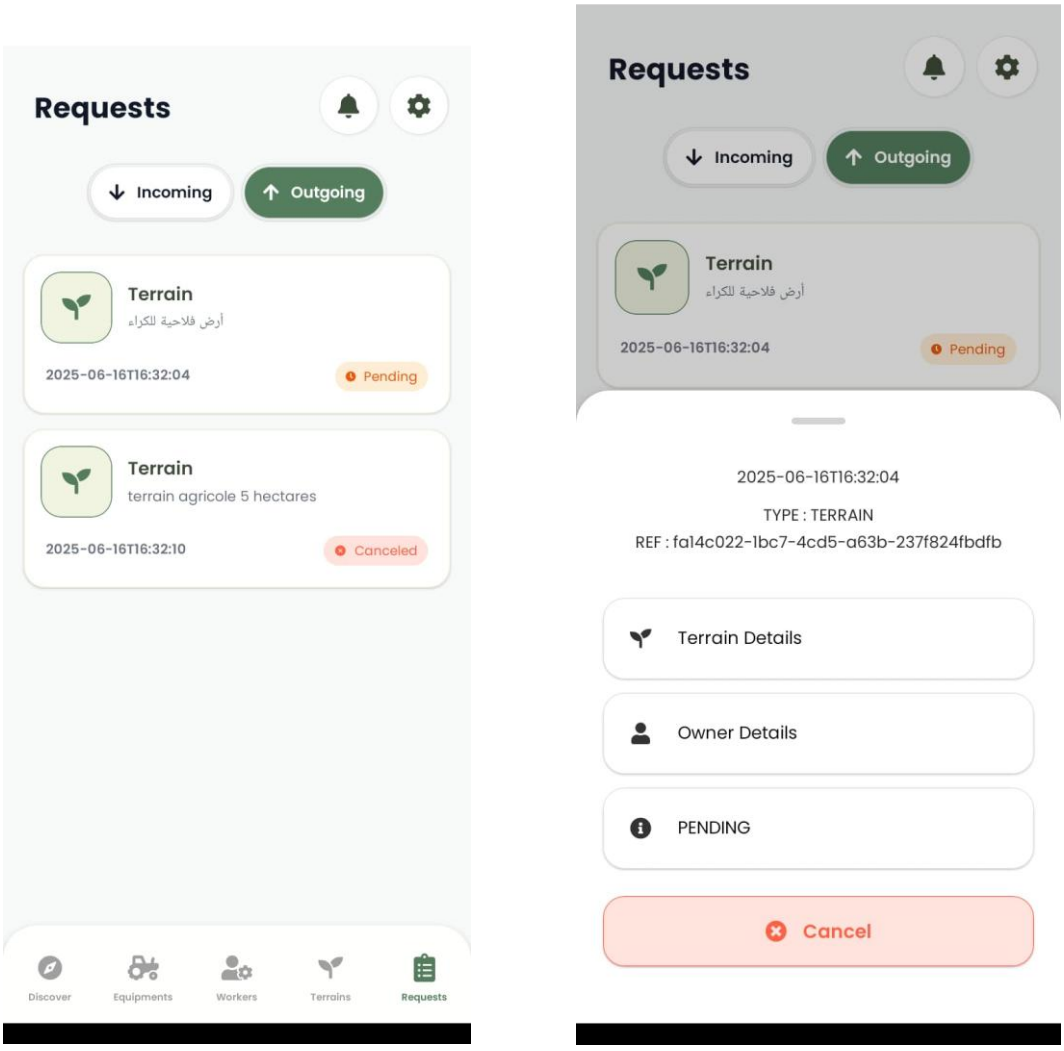

Figure VI. 22 -Interface de gestion des demandes de réservation

4.9 Page des Paramètres

Cette page permet à l'utilisateur de gérer son compte et ses préférences.

- Il peut consulter et modifier son profil personnel.
- Il accède à ses propres annonces (My Listings).
- Il peut voir ses favoris, son activité récente et ses documents.
- Il peut consulter l'historique des paiements.
- Il peut changer la langue de l'application.
- Il a accès à une section d'aide (FAQ) et peut signaler un problème.
- Enfin, il peut se déconnecter en cliquant sur Log out.

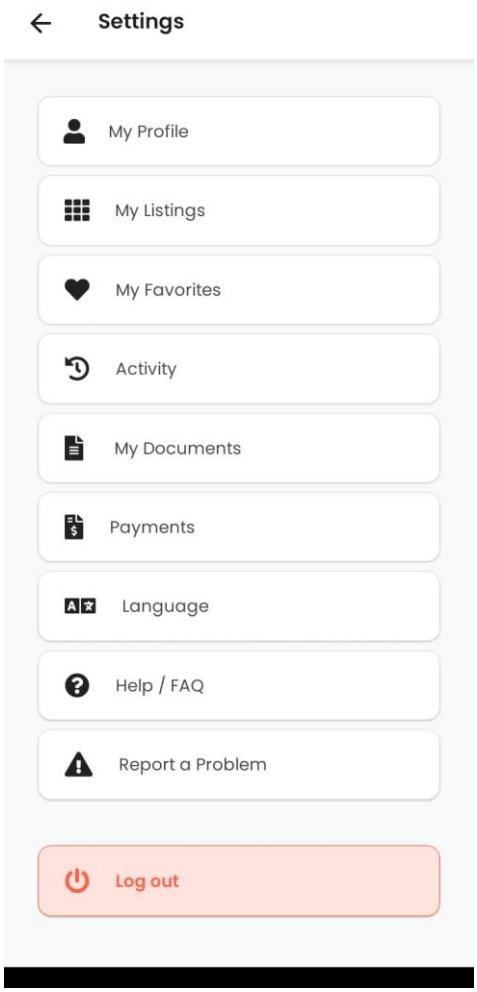

Figure VI. 23 -Interface des paramètres de l'application

5 Conclusion

Nous avons présenté dans ce chapitre les différentes technologies, environnement et langages de programmation utilisés dans la réalisation de ce travail. Ensuite, nous avons montré la simulation de notre application, après avoir présenté les différentes étapes de configurations à suivre pour ce faire.

Conclusion Générale

Dans un contexte où le secteur agricole algérien cherche à renforcer sa productivité et à moderniser ses pratiques, le projet AGRICO.DZ s'impose comme une initiative numérique novatrice, pensée pour répondre aux besoins réels des agriculteurs et des prestataires de services ruraux. Face aux nombreuses difficultés structurelles — logistique défaillante, organisation informelle des services, manque de visibilité des offres — cette plateforme vise à structurer un marché encore peu digitalisé, en facilitant la mise en relation entre les différents acteurs via une interface moderne, intuitive et adaptée au terrain.

Tout au long de ce mémoire, nous avons mis en évidence les choix technologiques et méthodologiques qui ont guidé la conception d'AGRICO.DZ. L'adoption d'une architecture microservices a permis de garantir modularité, évolutivité et indépendance fonctionnelle des services. Le recours au Domain-Driven Design (DDD) a structuré la modélisation des sous-domaines métier de manière claire et cohérente, tandis que l'intégration d'une architecture orientée événements (EDA) a renforcé la fluidité des interactions entre les services. En complément, un système de recommandation intelligent a été conçu pour offrir une expérience personnalisée, en valorisant les profils fiables et en orientant l'utilisateur vers les services les plus pertinents.

L'implémentation de la plateforme s'est appuyée sur des technologies robustes et actuelles : Spring Boot pour les microservices, React Native pour le frontend mobile, PostgreSQL pour la gestion des données, RabbitMQ pour la messagerie interservices, et Podman/Kubernetes pour la conteneurisation et l'orchestration. Ce socle technologique assure une base solide pour le déploiement et l'évolution future du système.

Au-delà de ses aspects techniques, AGRICO.DZ se positionne comme un levier de transformation sociale et économique, en participant à la digitalisation des zones rurales, à l'autonomisation des agriculteurs et à la structuration d'un écosystème agricole collaboratif. Les résultats obtenus démontrent la faisabilité de ce type de solution dans le contexte algérien, tout en ouvrant la voie à de futures améliorations.

Parmi les perspectives envisageables, on peut citer l'intégration d'une intelligence artificielle plus avancée pour la recommandation, le développement de services offline, l'élargissement du système à d'autres filières agricoles (élevage, irrigation, etc.), ainsi que la mise en place de partenariats institutionnels pour favoriser son adoption à large échelle. Ainsi, AGRICO.DZ n'est pas seulement une application, mais une contribution concrète à la modernisation durable de l'agriculture nationale.

Références

[1] M. &. L. J. Fowler, Microservices: A Definition of This New Architectural Term, 2014.

[2] I. M. a. P. L. P. D. Francesco, Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption, International Conference on Software Architecture (ICSA), Gothenburg, Sweden, 2017, pp. 21-30, doi: 10, 2017 .

[3] C. Richardson, Microservices Patterns: With Examples in Java. Manning Publications, 2018.

[4] R. Burdiuzha, Useful Cheat Sheet: 9 Key Components of a Production Microservices Application, Apr. 15, 2025. .

[5] stephen-sumner, API gateways - Azure Architecture Center..

[6] jamesmontemagno, Asynchronous message-based communication - .NET., Apr. 15, 2025.

[7] S. Newman, Building Microservices: Designing Fine-Grained Systems. O'Reilly Media., (2015). .

[8] E. Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software, Boston: Addison-Wesley, 2003.

[9] V. Vernon, Implementing Domain-Driven Design, Boston: Addison-Wesley, 2013.

[10] E. Wolff, Microservices: Flexible Software Architecture, Addison-Wesley Professional, 2016.

[11] A. A. &. F. Marinescu, Domain-Driven Design Vite fait, 27 mai 2013.

[12] Rocha, Practical Event-Driven microservices architecture : Building Sustainable and Highly Scalable Event-Driven Microservices, Apress, 2021.

[13] B. Stopford, Designing Event-Driven Systems, O'Reilly, 2018.

[14] Microsoft, «Event-driven architecture style».

[15] «Microservices Choreography vs. Orchestration: The Benefits of Choreography,» October 28, 2022.

[16] M. Richards, Software Architecture Patterns, O'Reilly Media, 2015.

[17] K. Falk, Pratical Recommender Systems, 2019.

[18] M. Z. A. F. G. F. Dietmar Jannach, Recommender Systems_ An Introduction, Cambridge University Press, (2010).

[19] C. Richardson, Microservices Patterns: With examples in Java. Manning,, 2018..

[20] M. Luksa, Kubernetes in Action_. Manning, 2018..

[21] S. R. Goniwada, Cloud Native Architecture and Design: A Handbook for Modern Day Architecture and Design with Enterprise-Grade Examples, Apress, , 2021..

[22] C. Binnie and R. McCune, "What Is A Container?," in Cloud Native Security,, Wiley, 2021, .

[23] D. Walsh, Podman in Action: Secure, rootless containers for Kubernetes, microservices, and more., Simon and Schuster, , 2023..

[24] "Podman." Accessed: Apr. 15, 2025. [Online]. Available: https://podman.io/.

[25] G. S. a. B. J. B. A. Arrichiello, Podman for DevOps: Containerization reimagined with Podman and its companion tools., Packt Publishing Ltd, 2022..

[26] Kubernetes Scheduling: Taxonomy, Ongoing Issues and Challenges https://dl.acm.org/doi/10.1145/3539606.

[27] “kind – Rootless.” Accessed: Apr. 15, 2025. [Online]. Available: https://kind.sigs.k8s.io/docs/user/rootless.

[28] Learning Kubernetes - Red Hat https://www.redhat.com/fr/topics/containers/learning-kubernetes-tutorial.

[29] Spring, "Spring Boot," [Online]. Available: https://spring.io/projects/spring-boot. [Accessed: Jun. 17, 2025].

[30] PostgreSQL Global Development Group, "PostgreSQL: The World's Most Advanced Open Source Relational Database," Postgresql.org, [Online]. Available: https://www.postgresql.org. [Accessed: Jun. 17, 2025]..

[31] Expo, "Expo – The fastest way to build an app," Expo.dev, [Online]. Available: https://expo.dev. [Accessed: Jun. 17, 2025]..

[32] Clerk, "Authentication and User Management for Modern Web Apps," Clerk.dev, [Online]. Available: https://clerk.dev. [Accessed: Jun. 17, 2025]..

[33] CloudAMQP, "RabbitMQ as a Service," Cloudamqp.com, [Online]. Available: https://www.cloudamqp.com. [Accessed: Jun. 17, 2025].

[34] RabbitMQ, "Messaging that just works," Rabbitmq.com, [Online]. Available: https://www.rabbitmq.com. [Accessed: Jun. 17, 2025]..